Real-Time Air Quality Index Application for the City of Cape Town

An Event-Driven System with Kafka, CQRS and Clojure

Presented by:
Subha Singh
BSc. Electrical Engineering (Mechatronics)

Prepared for:
Dr. S Winberg
Department of Electrical Engineering
University of Cape Town

Submitted to the Department of Electrical Engineering,
University of Cape Town, in fulfilment of the academic requirements
for the degree
Master of Science
Faculty of Engineering and the Built Environment
UNIVERSITY OF CAPE TOWN

June 23, 2018

Keywords: Event Sourcing, Clojure, Kafka, CQRS, Air Quality Index,
City of Cape Town

Declaration

"I know the meaning of plagiarism and declare that all the work in the document, save for that which is properly acknowledged, is my own. This thesis/dissertation has been submitted to the Turnitin module (or equivalent similarity and originality checking software) and I confirm that my supervisor has seen my report and any concerns revealed by such have been resolved with my supervisor."

Signature: Signed by candidate

Subha Singh

Date: **June 23, 2018**

Acknowledgments

"The single greatest cause of happiness is gratitude." - Auliq-Ice

Thank you:

- *To my family and friends for their support.*
- *To my mother for her encouragement, support, love and understanding throughout this project.*
- *Antoine Chesnais, Ashiv Dhondea, Oliver Powell and my supervisor, Dr. Simon Winberg, for their help, guidance and time.*
- *UCT/CSIR and Ada & Levenstein funding foundation for their support.*

Abstract

The World Health Organization estimates around 3.7 million premature deaths worldwide were due to ambient air pollution in 2012, 88% of which occurred in low to middle income countries such as South Africa. This project focuses on the development of an event-driven real-time air quality index application for the City of Cape Town. Event streams are being more commonly adopted in data centric applications that aim to produce trend analyses and prediction models. Event-driven systems store immutable raw event data, providing both a history of what has happened in the database, and the current state, thereby assisting with debugging and providing audit trail support. In addition to increasing public accessibility to the city's air quality information, the application has been designed for scalability, extensibility and data analysis through the incorporation of the Command Query Responsibility Segregation, Event Sourcing and Model-View-Controller architectural patterns. The design of the application itself serves as a basic reusable template for any new applications that may require the scalability, extensibility and is inherently data-centric nature as is found in this implementation. By taking advantage of the city's existing air quality monitoring sensor network, the real-time application has the capability to highlight problematic areas within the City of Cape Town with regard to high pollution levels, create greater public awareness, and lays the foundation for the future development of predictive air quality models and pollution forecasts.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

"Technology, and applications of this technology, will continue to improve and evolve, providing unprecedented, global access to information, individuals, training, and opportunities."

-Maynard Webb

This dissertation contains the work done to research, design, develop and test a real-time air quality web application for the City of Cape Town. The application is an event-driven system that makes use of the Clojure functional programming language, Apache Kafka, the MVC architectural pattern and CQRS. The application focuses on scalable real-time air quality monitoring with the aim of providing real-time air quality index updates to the public, thereby enhancing public awareness and the accessibility to air quality information, and taking a proactive step in ensuring the safety of public health with respect to ambient air pollution.

1.1 Project Background

South Africa is currently ranked 47th out of 99 countries on the Pollution Index by Country for 2016.[1] The WHO Global Urban Ambient Air Pollution Database includes the fine particle matter (Particulate Matter 2.5 ($PM_{2.5}$) and Particulate Matter 10 (PM_{10})) measurements in micrograms per cubic meter ($\mu g/m^3$) for over 3000 cities worldwide. The WHO 2016 ambient air pollution update showed that the small urban area, Hartebeespoort

in the North West province, has the dirtiest air in South Africa as it is ranked number one on the list of most polluted South African urban areas, with pollutant concentration levels much higher than the WHO's guidelines, mostly due to it's close proximity to mining operations. Additionally, the major cities within the Gauteng province, the most populated urban region in South Africa, also have pollutant concentration levels above the WHO guideline thresholds. Cape Town itself, is ranked 13th out of the 15 South African urban areas featured in the update.

Ambient air pollution poses a serious environmental health risk to the public. A reduction in air pollution would reduce the prevalence of diseases such as stroke, heart disease, lung cancer, as well as chronic and acute respiratory diseases. The WHO estimates around 3.7 million premature deaths world-wide were due to ambient air pollution in 2012, 88% of which occurred in low to middle income countries.[2] According to the WHO guidelines, a reduction of particulate matter from 70 to 20 micrograms per cubic meter ($\mu g/m^3$) would decrease the number of air pollution-related deaths by 15% world-wide. These statistics highlight the need to reduce public exposure to ambient air pollution. The first proactive step towards combating ambient air pollution, and its resultant health issues, is to setup a system that allows all the stakeholders involved to monitor the levels of ambient air pollution in real-time, so any one can develop effective ambient air pollution reduction strategies in key areas within a city or country. These strategies may be as simple as avoiding physical exertion outdoors, or wearing a surgical mask, when Air Quality Index (AQI) levels are dangerously high. Many government institutions, such as in China and India for example, have taken advantage of the readily available technology at their fingertips to create real-time air quality index web and mobile applications for their cities to help ensure the safety and protection of public health.

The City of Cape Town has modelled itself as a progressive city and actively encourages developments that enables it to meet global standards. One such development is atmospheric data monitoring. The city has setup an air quality monitoring sensor network and made the air quality data collected freely available to the public. However this data is not released to the public in a form that can be accessed in real-time, rather it is shared publicly only six to twelve months after it is collected, and is thus of limited use to the citizens of the City of Cape Town. The proposed real-time air quality index application for the City of Cape Town will allow the city to align itself to current international trends. The major benefit of real-time monitoring is efficiency. In real-time systems you are provided with information as soon as it is available, therefore you can respond immediately. You are always aware of the changes in the system state and are able to make

decisions based on the most up-to-date information. In the context of this application, a real-time air quality monitoring system will:

1. Highlight problematic areas, with respect to ambient air pollution, within the city which can then be managed more economically and effectively;
2. Be a tool for strategic urban planning e.g. road usage limitations to reduce carbon emissions by motor vehicles in specific high risk areas;
3. Increase public awareness with respect to the levels of air pollution within the city;
4. Be extended to highlight seasonal allergen trends which will provide huge health benefits in itself, especially to those who suffer from hay-fever; and
5. Allow for the development of predictive models and trend analysis that can provide a pollution forecast to citizens in the future.

Event-Driven Architecture (EDA) is gaining in popularity as more organizations desire to extract value from their system data and use it to leverage their offerings, whilst keeping the system's architecture maintainable, scalable and extensible. The fundamental concept in EDA is that all changes to the application's state are modelled and stored as a sequence of immutable events. Unlike traditional Create, Read, Update and Delete (CRUD) systems[3], the events cannot be updated or deleted, and therefore the state history of an entity is preserved rather than only its current state. This architectural approach puts the application's data first, is scalable and extensible, and makes it easier for the data to be mined at a later stage for the purposes of data science and predictive modelling, which is why it will form the foundation of the real-time air quality index application for the City of Cape Town.[4]

1.2 Project Objectives

This section outlines the main objectives, sub-objectives and the motivations behind these objectives to achieve the main objective of the project.

The main objective of the study is to design and develop an air quality index application for the City of Cape Town that provides real-time air quality index updates to the client and informs the user about the WHO key pollutants, what the AQI is, it's thresholds and health risks, and the precautionary measures to be taken.

The following list outlines the sub-objectives of the project:

1. Compile a literature review of research conducted with respect to air quality data, the air quality index application architecture and tools and testing for the application.
2. Develop the application using an architectural approach that is scalable, extensible and maintainable as the project has the potential to be scaled to a national or international level.
3. Develop a user interface that provides a good user experience and is informative with regard to AQI data, ambient air pollution health risks and precautionary measures.
4. Provide real-time air quality updates to the client to increase public awareness and safety with regard to their current environment.
5. Ensure the application acts as the foundation for future work that includes the development of air quality prediction models and trend analysis.
6. Conduct functionality tests to verify that the application performs as intended, in addition to performance and robustness testing.
7. Draw meaningful conclusions from the results obtained from testing.
8. Provide recommendations for further research and development of the application.

1.3 Problem Description

Presently, the City of Cape Town has made the air quality data collected from their sensor network available to the public. However, it is not released in real-time and can be several months old, which does not make it very useful to ordinary members of the public who would like to know the quality of the air in their current environment.

The project aims to provide a real-time air quality index web application with EDA for the City of Cape Town. A real-time air quality index application will be able to increase public awareness with respect to the pollution levels within the city, and their environment, by providing real-time updates on the air quality status of their current environment, in addition to potentially assisting local government with environmental regulation and safety policies; and setting the foundation for future air quality prediction modelling and

trend analysis for the City of Cape Town. The event-driven system combined with the CQRS architectural pattern places the air quality data at the centre of the application.

1.3.1 Research Questions to be Investigated

The following main research questions are to be investigated in the project:

1. What are the key ambient air pollutants the WHO has scientific evidence for and what are the effects of those pollutants on human health?
2. What are the guidelines of ambient air pollution set out by the WHO?
3. What is an AQI?
4. What is the USEPA AQI method, how is the USEPA AQI calculated and what air quality data attributes are required for the AQI calculation?
5. What effect do meteorological factors have on ambient air pollution?
6. What has been done with regard to air quality monitoring internationally and within the City of Cape Town?
7. What is EDA? What are it's benefits? What is required to develop an event sourced system?
8. What is the CQRS architectural pattern? What are it's benefits and how is it implemented?
9. What is the MVC architectural pattern and what are it's benefits?
10. What type of database will work well in an event sourced system?
11. What is functional programming and what are the benefits of using Clojure List Programming (LISP) language?
12. What tools can be used to provide real-time updates to the client and how to use those tools?
13. What are the aspects of a user interface that ensure a good user experience for all web devices?

14. How the air quality data should be modelled in the database such that it aids extensibility?

15. What tests need to be conducted to test for robustness, scalability, extensibility and performance of the application architecture?

The research questions will be answered through a process of literature review and experimental investigation via software development. In particular, the literature will be used to answer questions 1-9, and the answers to 10-15 will be answered through first initial insights from the literature followed by application implementation and testing.

1.4 Terms of Reference

This section outlines the requirements and sub-requirements required to achieve the project objectives outlined in section 1.2.

R1 Research with regard to the problems to be investigated in the previous section needs to be carried out and a concise literature review containing only information related to this application and it's scope needs to be compiled.

R2 Create a data generator to simulate air quality data. This is required because the City of Cape Town municipality does not provide ordinary members of the public access to the raw data collected by their real-time air quality monitoring sensors spread throughout the city, or the system's data API. Additionally, it would be beneficial to anyone wishing to use this application with a real data source to ensure that replacing the data generator with the City of Cape Town's air quality API, or an API from another source, is as simple as possible.

- **R2.1** Generate air quality data from the same geographical locations as the real-time sensors the City of Cape Town municipality has distributed within the City of Cape Town.
- **R2.2** Data generated includes data attributes needed to serve as a international API because of the potential for scaling and extensibility.
- **R2.3** The data generator pushes data out periodically, mimicking a Push API which has been assumed for the air quality data API of the City of Cape Town.

R2.4 The pollution data measured is based on pollutants for which the WHO has scientific evidence.

R2.5 The simulated pollution data fits into data ranges extracted from the City of Cape Town's air quality data archives.

R3 Create the application using EDA, in combination with CQRS, as the architectural patterns allow for scalability, extensibility and maintainability. Event Sourcing allows for greater control over the processing of data and the flow of the data through the system. Both these architectural patterns are outlined in section 2.3 of the literature review.

R3.1 The read and write model are decoupled for the implementation of CQRS to allow for the separation of concerns. The write model follows the principles of EDA. The MVC architectural pattern, discussed in section 2.3, is implemented in the read model.

R3.2 Provide audit support for tracing and debugging.

R3.3 The write model of the application architecture includes validating the data received from the air quality data API, which in this case is the data generator.

R4 Provide real-time air quality index updates to the client so that users are aware of their current environmental conditions.

R4.1 The application provides real-time air quality index updates to the user using the MVC architectural pattern. Real-time updates are discussed in section 2.3.4 of the literature review.

R5 Provide ambient air pollution information to the client for context and understanding of the health risks.

R5.1 The application web page provides information about air quality data, precautionary measures and the implications of ambient air pollution for basic user understanding. It's layout and navigation should be informative, yet simple so it may be intuitively understood by the user.

R6 Ensure the application acts as the foundation for future work and enables the development of air quality prediction models and trend analysis.

R6.1 The application makes use of an immutable data store in the event-driven architectural pattern explained further in section 2.3.1 of the literature review.

R6.2 Users are able to query and download air quality data in CSV file format though asynchronous web communication.

R6.3 The application's write model can be easily scaled to accommodate more data attributes, and data sources, and allow for more complex air quality event processing.

R7 Performance, robustness and functionality tests need to be performed to assess the effectiveness and capability of the application.

1.4.1 Acceptance Testing

The following table provides a breakdown of the tests to be performed in the acceptance testing.

Table 1.1: Breakdown of sub-tests to be performed in the acceptance testing of the application.

Test Number	**Description**	**Functions Checked**	**Requirements Tested**
1	Testing the fault tolerance of the Kafka multi-broker cluster for robustness.	Kafka multi-broker cluster.	Forms part of R3: the implementation of EDA.
2	Testing that the Clojure Spec data validation works as expected for robustness and reliability.	Clojure Spec data validation.	Forms part of R3: the data validation of data received from the data generator.
3	Profiling high level functions in the read model and eliminating potential bottlenecks that will affect the user experience.	Profiling for bottlenecks in the read model.	Performance optimization of the read model.
4	Testing for memory leaks within the application for robustness.	Memory leak detection testing.	Performance and robustness of application.
5	Ensuring the real-time AQI updates displayed on the Google map in the user interface functions as expected.	Google map AQI real-time update.	Functionality testing R4.
6	Ensuring the air quality data CSV file download feature in the user interface works as expected.	Air quality data CSV file download.	Functionality testing R6.
7	Testing the page response time of the application's web page for performance analysis.	Page response time testing.	Performance testing web page of application.
8	Load and stress testing the application for performance analysis.	Load and stress testing.	Performance testing the application read model.
9	Testing the user interface with volunteers using the application and filling out a survey rating their user experience.	User experience testing.	User experience testing the user interface as a part of R5.

1.5 Scope and Limitations

The project's research scope is limited to that of air quality, web application development with Clojure, EDA, MVC and CQRS architectural patterns. Any other related fields are not included within the scope.

The project will make reference to various reliable websites, blogs and open-source documentation, in addition to journal articles, GitHub libraries and books. Although these sources of information are not all necessarily peer-reviewed, in the field of web development they form sources of credible information. Peer-reviewed documentation is not published as rapidly as technology is evolving and the latest advancements in the field of web development are showcased in web development related conference talks, websites and blogs.

Project scope and limitations are listed below:

1. The application is limited to the usage of the Datomic Free Database for development. Datomic Starter and Pro editions are not suited for the development of a low-cost open-source projects with the potential for future work, as Datomic Pro is expensive to maintain and Datomic Starter usage is limited to one year.
2. The project is limited by the denial of access to the City of Cape Town's real-time air quality data API. No other existing open-source air quality data API for the City of Cape Town exists but the application endeavours to work around this limitation with the implementation of a data generator, which can be easily replaced with a real raw air quality data API in the future, to simulate a real-time air quality data Push API. As a result of this limitation, no data accuracy tests will be performed, however; unit testing will be conducted for the USEPA air quality index calculation method used for each pollutant to ensure the correct air quality index results are produced given certain inputs.
3. Optimization with respect to performance of the application is included within the scope of the project but is limited to the read model of the application which affects the user experience.
4. Robustness testing and user experience testing is included within the scope of this project.
5. Performance testing is limited to the read model of the application.

6. Server optimization is not included within the scope of this project as the application will not be put into production.

7. The web application scope, although cross platform in nature, does not include cross platform optimization and testing.

8. Web application security is not included within the project scope as no sensitive information is stored in the application's database.

Due to the nature of the project the results of functionality tests will be determined by the correct functioning of the application's features with respect to the intended process by which the application should perform the task.

1.6 Document Outline

Chapter 1 introduces the research project. It provides the background to the topic, the relevance of the project and it's motivation. It outlines the objectives, research questions, terms of reference and scope and limitations by which the project will be investigated, designed, developed and tested.

Chapter 2 presents a review of the relevant research conducted. It covers the topics of ambient air pollution, air quality index application architecture and tools and application testing. It begins with a discussion on ambient air pollution, the key pollutants the WHO has scientific evidence for, what the air quality index is and how it is calculated using the USEPA method. It then moves to the topic of the air quality index application architecture and tools which describes what Event Sourcing is, it's characteristics and advantages. It includes an explanation on architectural design concepts such as the CQRS and MVC architectural patterns and the tools needed to develop an application incorporating those architectural patterns. Lastly, the chapter focuses on application testing and describes the types of testing that may be performed on the application to ensure it behaves as expected.

Chapter 3 details the project's research methodology. The chapter opens with an explanation of the software development method used. A flow diagram is used to illustrate how the research questions and project objectives were investigated and each step is described in detail. A high level application design is provided which includes the architectural patterns and design concepts discussed in Chapter 2. An acceptance testing

outline is provided based on the project's terms of reference and lastly, a Gantt chart is given to illustrate the project time-line and the time period required for each project phase.

Chapter 4 discusses the detailed design of the application, building on the high level design presented in Chapter 3. It begins by providing details about the hardware available. An overview of the detailed application design is provided, which includes the back-end and front-end requirements extracted from the terms of reference. Aspects of the detailed design are then discussed in more detail. The aspects include the design of the application's data generator, read and write model and the database design.

Chapter 5 details the development environment and implementation of the application detailed design discussed in the previous chapter. It begins by providing details of the development environment in terms of the software and third-party user accounts required. The implementation of the data generator, the air quality data type validation with Clojure Spec, Event Sourcing with Apache Kafka, the Datomic database and application queries, application logging, web interface, application life-cycle management and user interface are covered in the chapter. Lastly, the chapter concludes with an explanation on how to extend the application's dataset in the future with examples.

Chapter 6 reviews the acceptance tests performed on the application and the results obtained. It begins with providing an acceptance testing strategy, which lists the acceptance tests to be performed, extracted from the terms of reference and outlined in section 3.4. The details of the tools used during the acceptance testing are provided. The rest of the chapter focuses on how each acceptance test was conducted, the results of each test and the analysis and interpretation of those results.

Chapter 7 provides concluding remarks with regard to the application prototype developed based in reference to project objectives and requirements. The reader is reminded of the main aim and requirements of the project and how that has been achieved is briefly discussed. It presents a concise summary of the results from the acceptance testing and the major highlights of the application design in terms of scalability, extensibility, reusability and laying the foundation for future predictive modelling work.

Chapter 8 makes mention of recommendations for future research and development of the application. It includes recommendations for the both the application's architecture and the application's user interface. The architectural recommendations are with respect

to the back-end of the application and how it may be improved. The user interface recommendations focuses on how the user interface may be improved in order to provide a better user experience. The user interface recommendations are based on user feedback obtained from the user experience testing discussed in section 6.8.

Chapter 2

Literature Review

The literature review is a concise presentation of the relevant research conducted. The scope is limited to the topics of air quality, Clojure web application development and testing, EDA, the CQRS and MVC architectural patterns. Various websites, blogs and open-source software documentation, in addition to journal articles, books and conference proceedings, were consulted during the compilation of the literature review.

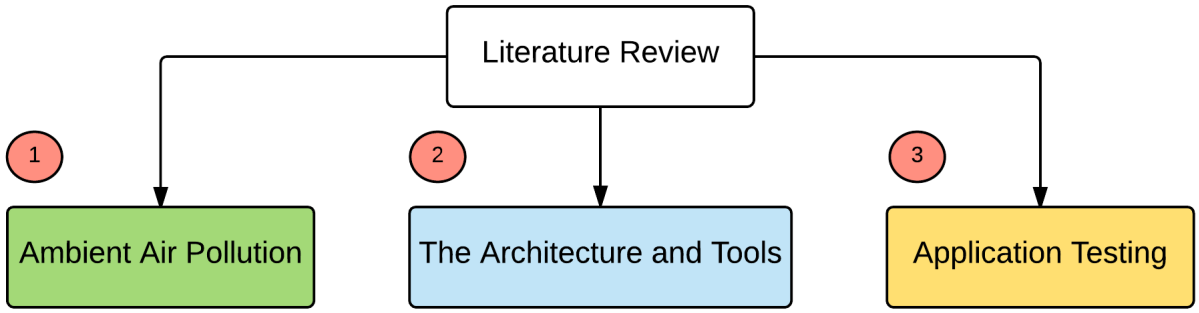


Figure 2.1: Map of Literature Review.

Figure 2.1 illustrates the literature review is divided into three broad main sections: ambient air pollution, the architecture and tools and application testing. The ambient air pollution section briefly discusses the impact of ambient air pollution on human health and provides information on some of the key pollutants, as well as what the AQI is, how it is calculated and what has been done with regard to air quality internationally, in the commercial field and within South Africa itself. The architecture and tools section describes what Event Sourcing and EDA is and its characteristics and advantages. It

explains architectural design concepts such as the CQRS and MVC architectural patterns and the tools needed to develop an application incorporating those architectural patterns. The last section, application testing, describes the tests that may be performed on the application to help ensure it behaves as expected, is robust and reliable, as well as guidelines on user interface design that lead to the provision of a good user experience.

2.1 Ambient Air Pollution

The high mortality rate due to contracted diseases from ambient air pollution is largely attributed to particulate matter of 10 microns or less in diameter. The WHO air quality guidelines outlines global concentration limits for key air pollutants. These WHO guidelines are based on scientific evidence for particulate matter, ozone, nitrogen dioxide and sulphur dioxide. The following information about each pollutant was extracted from the WHO ambient air quality and health 2016 fact sheet[2]:

Particulate Matter

Particulate matter is considered to be the most dangerous to human health. It consists of both solid and liquid particles suspended in the air. These particles are small enough in size to penetrate and accumulate in the lungs, increasing the risk of developing cardiovascular or respiratory diseases and lung cancer.

Ozone

Ozone is a major component of photochemical smog. Photochemical smog is formed by the reaction of sunlight with vehicle and industry emissions. High levels of ozone in the air we breathe can result in breathing problems, such as asthma, decreased lung function and result in lung diseases.

Nitrogen Dioxide

Short-term exposure of high levels of nitrogen dioxide is toxic. It is also a component of particulate matter 2.5 and ozone. Studies have indicated a positive link to long-

term exposure to nitrogen dioxide and symptoms of bronchitis in asthmatic children and reduced lung function. The major sources of nitrogen dioxide emissions are combustion processes, such as vehicle emissions and power generation.

Sulphur Dioxide

Studies have indicated disturbances in pulmonary function and respiratory symptoms due to short term exposure to high levels of sulphur dioxide in people with asthma. Sulphur dioxide is a by-product of burning fossil fuels and the smelting of certain mineral ores. It affects the respiratory system, functioning of the lungs and can irritate the eyes. It can also result in deforestation when it reacts with water to produce sulphuric acid.

2.1.1 The Air Quality Index

The AQI is an index for daily air quality reporting that provides an indication of the human health effects due to short-term exposure to each pollutant. There are many different air quality indices in existence, some give an overall assessment of air quality and incorporate the synergistic effects of the major air pollutants, whilst others do not. The calculation of the AQI requires raw pollution concentration readings from air quality monitoring sensors over a specified averaging period. Together the time and pollutant concentration represent the effects of the pollutant on human health that has been established by epidemiological research.[5] The severity of each pollutant on human health varies and so does the formulae used to convert from the pollutant concentration to AQI. Air quality values are grouped into ranges with a colour code, descriptor and standard public health advisory. Figure 2.2 is an example of the AQI scale taken from the World Air Quality Index Project as defined by the USEPA 2016 standard.

AQI	Air Pollution Level	Health Implications	Cautionary Statement (for PM2.5)
0 - 50	Good	Air quality is considered satisfactory, and air pollution poses little or no risk.	None.
51 -100	Moderate	Air quality is acceptable; however, for some pollutants there may be a moderate health concern for a very small number of people who are unusually sensitive to air pollution.	Active children and adults, and people with respiratory disease, such as asthma, should limit prolonged outdoor exertion.
101-150	Unhealthy for Sensitive Groups	Members of sensitive groups may experience health effects. The general public is not likely to be affected.	Active children and adults, and people with respiratory disease, such as asthma, should limit prolonged outdoor exertion.
151-200	Unhealthy	Everyone may begin to experience health effects; members of sensitive groups may experience more serious health effects.	Active children and adults, and people with respiratory disease, such as asthma, should avoid prolonged outdoor exertion; everyone else, especially children, should limit prolonged outdoor exertion.
201-300	Very Unhealthy	Health warnings of emergency conditions. The entire population is more likely to be affected.	Active children and adults, and people with respiratory disease, such as asthma, should avoid all outdoor exertion; everyone else, especially children, should limit outdoor exertion.
300+	Hazardous	Health alert: everyone may experience more serious health effects.	Everyone should avoid all outdoor exertion.

Figure 2.2: World Air Quality Index Project AQI scale showing AQI ranges, colour codes, range descriptors and public health advisory.

The 2016 WHO Air quality guidelines offer global guidance on concentration thresholds and limits for key air pollutants that pose severe health risks. The thresholds for the key air pollutants, extracted from the WHO ambient air pollution and health 2016 fact sheet, are the following[2]:

Table 2.1: WHO key pollutant concentration thresholds. The pollutant threshold is given as a mean value over a specific time period listed in the table below.

WHO Key Pollutant Concentration Thresholds.

Pollutant	Threshold [$\mu g/m^3$]	Measurement Time Period [Hours]
$PM_{2.5}$	25	24
PM_{10}	50	24
Ozone (O_3)	100	8
Sulphur Dioxide (SO_2)	20	24
Nitrogen Dioxide (NO_2)	200	1

2.1.2 Calculating the AQI

The AQI for each pollutant is calculated by the following mathematical expression using the USEPA method[6]:

$$I_p = \frac{I_{Hi} - I_{Lo}}{BP_{Hi} - BP_{Lo}}(C_p - BP_{Lo}) + I_{Lo} \tag{1}$$

Where:

I_p	The index for pollutant p
C_p	The truncated mean concentration of pollutant p obtained from the sensor readings over the time period stated in Table 2.1.
BP_{Hi}	The concentration breakpoint that is greater than or equal to C_p
BP_{Lo}	The concentration breakpoint that is less than or equal to C_p
I_{Hi}	The AQI value corresponding to BP_{Hi} from the table of breakpoints in Figure 2.3.
I_{Lo}	The AQI value corresponding to BP_{Lo} from the table of breakpoints in Figure 2.3.

Eq. (1) is used to obtain the index for each pollutant using their respective mean concentrations, breakpoints and associated AQI values, as stated in the USEPA table of pollutant breakpoints shown in Figure 2.3.[6] The pollutant breakpoints in the USEPA table indicate the high and low pollutant concentration thresholds for each AQI range.

AQI Calculation example:
A sensor records a 24-hour average $PM_{2.5}$ concentration of 11.0 $[\mu g/m^3]$. The resultant AQI using the USEPA mathematical expression and breakpoint table is calculated as follows,

$$\begin{aligned}
C_p &= 11.0 \quad (PM_{2.5} \text{ concentration}) \\
BP_{Hi} &= 12.0 \quad (PM_{2.5} \text{ high concentration breakpoint from breakpoint table}) \\
BP_{Lo} &= 0 \quad (PM_{2.5} \text{ low concentration breakpoint from breakpoint table}) \\
I_{Hi} &= 50 \quad (\text{AQI high breakpoint from breakpoint table}) \\
I_{Lo} &= 0 \quad (\text{AQI low breakpoint from breakpoint table})
\end{aligned}$$

O_3 (ppb)	O_3 (ppb)	$PM_{2.5}$ (µg/m^3)	PM_{10} (µg/m^3)	CO (ppm)	SO_2 (ppb)	NO_2 (ppb)	AQI	AQI
C_{low} - C_{high} (avg)	C_{low} - C_{high} (avg)	C_{low}- C_{high} (avg)	C_{low} - C_{high} (avg)	C_{low} - C_{high} (avg)	C_{low} - C_{high} (avg)	C_{low} - C_{high} (avg)	I_{low} - I_{high}	Category
0-54 (8-hr)	-	0.0-12.0 (24-hr)	0-54 (24-hr)	0.0-4.4 (8-hr)	0-35 (1-hr)	0-53 (1-hr)	0-50	Good
55-70 (8-hr)	-	12.1-35.4 (24-hr)	55-154 (24-hr)	4.5-9.4 (8-hr)	36-75 (1-hr)	54-100 (1-hr)	51-100	Moderate
71-85 (8-hr)	125-164 (1-hr)	35.5-55.4 (24-hr)	155-254 (24-hr)	9.5-12.4 (8-hr)	76-185 (1-hr)	101-360 (1-hr)	101-150	Unhealthy for Sensitive Groups
86-105 (8-hr)	165-204 (1-hr)	55.5-150.4 (24-hr)	255-354 (24-hr)	12.5-15.4 (8-hr)	186-304 (1-hr)	361-649 (1-hr)	151-200	Unhealthy
106-200 (8-hr)	205-404 (1-hr)	150.5-250.4 (24-hr)	355-424 (24-hr)	15.5-30.4 (8-hr)	305-604 (24-hr)	650-1249 (1-hr)	201-300	Very Unhealthy
-	405-504 (1-hr)	250.5-350.4 (24-hr)	425-504 (24-hr)	30.5-40.4 (8-hr)	605-804 (24-hr)	1250-1649 (1-hr)	301-400	Hazardous
-	505-604 (1-hr)	350.5-500.4 (24-hr)	505-604 (24-hr)	40.5-50.4 (8-hr)	805-1004 (24-hr)	1650-2049 (1-hr)	401-500	

Figure 2.3: USEPA table of breakpoints for each pollutant.

This leads to the substitution of values in the formula as follows:

$$\frac{50-0}{11.0-0.0}(12.0-0.0)+0=45.8$$

An AQI of 45.8 is in the "Good" range of the AQI scale shown in Figure 2.2.

2.1.3 Meteorological Factors that Affect Ambient Air Pollution

Meteorological factors, such as atmospheric temperature and pressure, change the behaviour of atmospheric gases. Air quality studies have indicated that the following factors are highly correlated with the air pollution forecast in different cities and affect the air pollution index [7, 8]:

1. Wind speed;
2. Atmospheric pressure;
3. The previous day's AQI; and
4. Atmospheric temperature.

The most significantly correlated meteorological variable to the pollution forecast by AQI across the AQI studies was found to be the atmospheric temperature.[7, 8] The previous day's AQI is significant as it provides a basis upon which to assess the current day's air quality. A study conducted in Italy concluded that the previous day's AQI, minimum temperature, and the forecast maximum temperate and wind speed, make it possible for an air pollution forecast to be made in the morning of the current day.[7]

Meteorological factors like those listed above need to be taken into consideration, in addition to the synergistic effect pollutants have on each other, when developing a reliable predictive model for ambient air pollution within the City of Cape Town, or when mapping the movement of ambient air pollution within the city. However, there does not seem to be a consensus on exactly which meteorological factors (with the exclusion of atmospheric temperature) affect ambient air pollution within a city and which do not. In fact studies have produced different conclusions for different geographical areas.[7, 8] Therefore an investigation into which meteorological factors highly correlate to the pollution forecast

by AQI specifically for the City of Cape Town would be advisable prior to the development of a reliable and accurate ambient air pollution predictive model.

2.2 What Has Been Done

This section outlines what has be done with regard to air quality monitoring commercially and within government institutions internationally, and within South Africa.

Internationally

As mentioned previously in section 1.1, the government institutions of many countries, such as China and India for example, have created web and mobile applications for real-time monitoring of air quality within their major, and most at risk cities, to create awareness and ensure the safety of their citizens. AQI updates are also broadcast around the clock on news channels to further increase awareness.

With the impact of climate change being increasingly felt world-wide, countries and cities are pledging to act by setting greenhouse-gas reduction targets, preparing risk assessments and looking at how interaction may be improved in urban environments. Additionally, many cities around the world encourage the use of public transport, and car and bike sharing programmes, in an effort to reduce the air pollution within their cities produced by the daily usage of privately owned vehicles. Others have banned the usage of cars with certain engine types and have experimented with alternating the banning of cars with odd or even number plates. In Germany, citizens are offered cheaper housing and other lifestyle perks as an incentive to live without a car. Both consumers and cities are slowly switching to 'cleaner' vehicle options, such as electric or compressed natural gas vehicles, with the aim of reducing their contribution to the ambient air pollution within their cities.[9]

Commercially

In addition to many international government organizations creating air quality index web applications that are freely available to the public, many companies in the private sector have taken up the cause and have begun collecting air quality data to support

their own air quality index web and mobile applications. These applications are available to the public. Many of them allow members of the public to purchase their own private air quality sensors and connect it to their company's web application. Some examples of these non-government linked web applications are Plume Labs, AirNow and Airpocalypse.

Within South Africa

Currently, the City of Cape Town municipality has set up 14 real-time air quality monitoring sensors around the city. They have made the collected air quality sensor network data available to the public, however the data is released to the public in a form that cannot be accessed in real-time, rather it is shared publicly only six to twelve months after it is collected, and is thus of limited use to the citizens of the City of Cape Town. In terms of the country as a whole, there does exist an air quality information system. The South African Air Quality Information System (SAAQIS) provides a platform for managing air quality information in the country and it does make the data available to the public, however, it is once again not real-time data. The SAAQIS is still in the process of shifting to a national real-time air quality monitoring system for country-wide monitoring.

2.3 Air Quality Index Application Architecture and Tools

The section explores the architectural patterns and design concepts of the proposed application. The concepts covered include Event Sourcing and EDA, the CQRS and MVC architectural patterns, immutable databases, Representational State Transfer (REST) APIs and application routing for real-time updates.

2.3.1 Event Sourcing and Event-Driven Architecture

Event Sourcing is an architectural pattern that is gaining in popularity. In an event sourced system all changes to the application's state are stored as an immutable sequence of events. An event contains information about something has happened e.g. a new user was created or an account was updated. Events are always past tense. These events are immutable. If an event contains a mistake, it can only be corrected by generating an

event that will have the reverse effect. The 'mistake' event will still exist along with the correction event in the entity's event history.[10]

Event Sourcing indicates how a certain state was reached, whereas traditional relational databases overwrite data in the database and consequently, can only show the current state. Every time an entity's state is changed a new event is appended to the entity's sequence of events. The current state of the entity is then reconstructed from it's event history. The immutability of the events makes it possible to rebuild the entire database from scratch using the event log, in comparison to having to retrace and rebuild the database from production or development logs as is the case with many traditional databases.[10]

Command Query Responsibility Segregation (CQRS)

CQRS is a system architectural pattern that separates the system into two clear parts, one is used for writing (commands that update the system's state) and the other part is used for reading (querying for information without changing the system's state). The separation of the read and write model allows each model to be scaled independently according to performance requirements.[11]

Event Sourcing and CQRS complement each other and combine well. New read models may be added to the application, long after the deployment of the write model, whilst still being able to take into consideration the events stored thus far. Alternatively, how a stream of events is converted into a representation may be changed without affecting the read model.

In Bobby Calderwood's *Clojure/conj 2015* presentation, *From REST to CQRS*[12], he suggests defining the three HTTP endpoints given in Table 2.2 to easily implement CQRS within a system.

Table 2.2: CQRS HTTP endpoints.

Endpoint	**Purpose**
/command	Writes
/query	Reads
/update	Real-time updates e.g. WebSockets or Server-Sent Events (SSE)

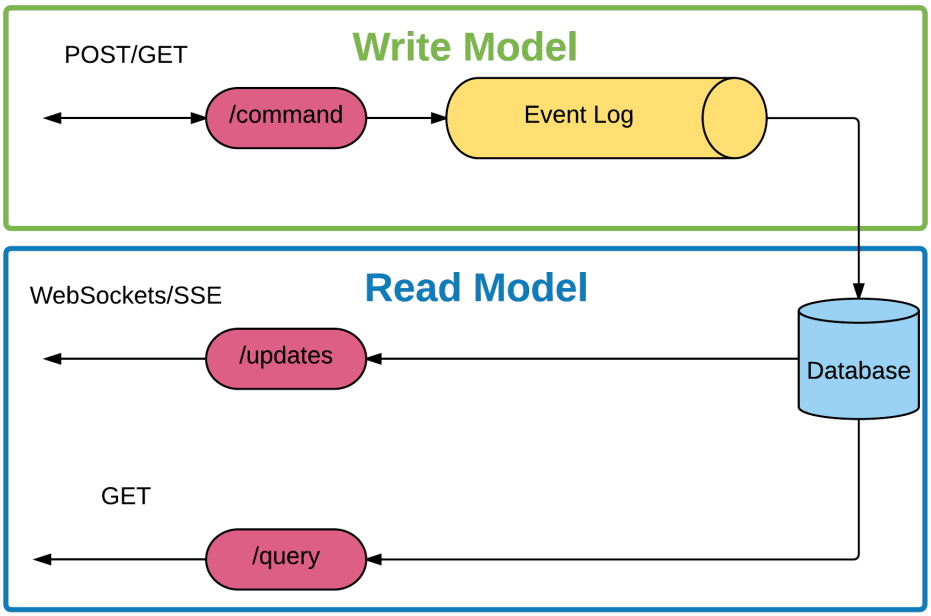


Figure 2.4: Event Sourcing with CQRS architectural pattern.

Architectural patterns that combine Event Sourcing with CQRS usually have a part of the application that models writes to an event log as illustrated in Figure 2.4 above. An event handler subscribes to the event log, transforms the event data if necessary and writes the data to a read store (database) which may be queried against.[13]

Immutable Data Stores

Relational databases have been used in software since the 1980's and are still commonly used today. Traditionally mutable relational databases have been used for transactional processing. They were born in 'an era when storage was expensive and all systems were transactional.' The *Update* function can change the content of any cell within CRUD-based relational database management system and the *Delete* function can remove the cell completely. 'Today storage is inexpensive and enterprises are collecting more and more data to leverage their businesses offerings. They want to minimize latency caused by overwriting data and to keep the written data in a full-fidelity form' so it can be analyzed and learnt from.[14]

According to Martin Kleppmann, a developer at LinkedIn, mutable state is the enemy. Today storage is inexpensive which makes immutable data storage at scale feasible. He believes a 'database should be an always-growing collection of immutable facts, rather than a technology that can overwrite any cell.' Overwriting data was more appropriate

during a time when data was predictable. He notes that conventional database replication mechanisms already rely on streams of immutable events. Immutable data stores are designed for analysis and the application of data science techniques.[14]

Event Queue/Channel

An event queue is a persistent repository where the events of an application are serially logged prior to being processed by the system as illustrated in Figure 2.5. A log or queue is akin to a table but the records are sorted by time. When a record changes, the change is stored as a separate timestamped event.[14] They are usually used in the context of an enterprise messaging system or distributed streaming platform. The term *message* is used interchangeably with the term *event*.[15]

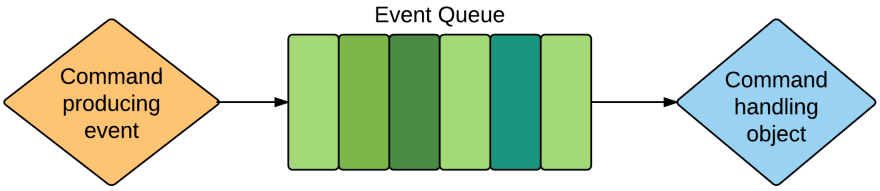


Figure 2.5: Illustration of an event queue.

The Event Store (Canonical Database)

The event store is a mechanism for storing the system's events, usually a database, and returning a stream of events associated with an aggregate instance, allowing you to replay the events to determine the state of the aggregate that is required, as illustrated in Figure 2.6a.[16]

Aggregated Database

The aggregated database stores the aggregated data. It only contains information about the current state which has been built up by the event information recorded in the event store as illustrated in Figure 2.6b.

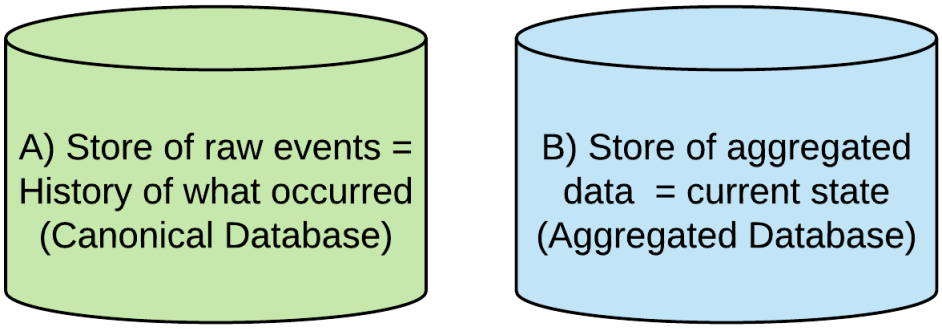


Figure 2.6: a) Event store containing raw events. Raw events are in the ideal form to write data. b) Aggregated database containing aggregate data. Aggregate data is in the ideal form to query from the database.

Characteristics of Event-Driven Architectures

EDA system models tend to be expressive, whilst displaying an aura of simplicity because they are modelled after real world events. EDAs usually include the following set of characteristics[17]:

1. **Broadcast communications**
   Systems publish/broadcast events to a messaging system/event queue. Any interested party(ies) can subscribe to that event queue and process it's event(s).

2. **Timeliness**
   'Systems publish the events as they occur rather than storing them locally and waiting for the processing cycle (e.g. nightly batch cycle) to begin.'

3. **Asynchrony**
   'The publishing system(s) does not wait for the consumer system(s) to process the event(s).'

4. **Fine Grained Events**
   Systems publish individual events rather than a single aggregated event.

5. **Ontology**
   The system defines how events are classified. Consumer systems may be interested 'in individual events or categories of events.'

6. **Complex Events Processing**
   The system is aware of 'and monitors the relationships between events.'

Advantages of Event Sourcing

Using EDA within a system model can be highly beneficial when used in the appropriate context. A few of the benefits of incorporating, or using an EDA system model, are listed below.[17, 18, 10]

1. **System state recreation**
   In event-driven systems each state corresponds to one or more events. The system state can be recreated from all the system's previous events, even if the individual state was originally lost. Additionally, this allows for 100% audit logging and assists with debugging. Events may also be replayed but with changes inserted into the past event streams (the event history) which is invaluable when testing how the system behaves in different real-life scenarios.

2. **Easy to execute temporal queries**
   As event-sourced systems maintain the history of each entity, it's relatively easy to implement temporal queries and reconstruct the history of an entity.

3. **Read and write performance**
   The reads and writes are separated which allow for fast reads and fast writes because they no longer use the same schema.

4. **Scalability**
   Event streams are relatively easy to scale across multiple machines because they are simple abstractions. They allow you to breakdown your application into producers and consumers of streams which operate independently.

5. **Loosely coupled components**
   In event-based systems the interaction between components is restricted to the exchange of events. Communication through events 'allows components to be decoupled to the extent that the "caller" is no longer aware of what function is executed next nor which component is executing it,' as is the case in a call stack oriented interaction. Loose coupling is needed in situations that require independent variability, for example in distributed systems.

6. **Consumer keeps state**
   Systems do not query other systems for information, they keep their own copy of the required data and receive updates as they occur. This shift in responsibility of keeping state from the component to the consumer means that the component does not have to be concerned by the needs of the data's consumers. The consumers are

able to listen to event channels from multiple sources, keep the relevant state and combine information from multiple events into new events, or marginalized views.

Disadvantages of Event Sourcing

Although EDA can be advantageous when used in the appropriate context, it does present some practical challenges. The trade-offs of EDA need to be considered before incorporating it within a system's architecture. A few disadvantages of EDA are listed below.[19]

1. **Snapshots and scaling**
   Handling entities with long and complex lifespans are a point of complexity in an event sourced system. If entities are defined by frequent changes in state it produces a large number of events that need to be processed to determine the current state of the entity. This problem is usually addressed by creating snapshots that summarize the state up to a certain point in time, thereby reducing the query load. However, the solution of snapshots give rise to another form of complexity though - the question of when and how snapshots should be created? This is not a trivial question as snapshot creation usually requires an asynchronous process to create a snapshot before any expected query load which is difficult to predict in the real world.

2. **Schema changes**
   Although events are immutable, the structure of future events may change as the structure of the data itself changes. This leads to a choice between updating all the existing events or handling multiple schemas in the code.

3. **Complex real world domains**
   Event sourcing is suited to relatively simple domains, defined by one or two events. The processing logic of EDA can become quite complicated and unwieldy when aggregating multiple types of event streams. To combat this many event sourcing systems maintain a separation between a working copy of the data and the underlying event store by exposing a derived state of the data to consumers, but the difficulty of determining how best to expose the working copy of the data so that the data is timely and relevant remains.

The Architecture of Existing Air Quality Monitoring Systems

The architecture of existing open source air quality monitoring web applications, such as AirCasting and EuroMonitorWeb, differ from that of EDA in that they do not make use of an event queue to serially log air quality data readings prior to being processed by the system. Instead these applications process the data as it is received and store that data in a new timestamped row in a relational database table. The drawbacks of not using a queue to serially log the data received from sensors are that:

1. The components in the system are more tightly coupled;
2. At scale it's harder to reason about the application, whereas with the use of queues the components of the system can be broken down into consumers and producers;
3. It becomes more difficult to spread out the work to multiple processes at scale because processors cannot simply just subscribe to a queue of air quality data. Although that being said there are many ways to scale up an application, a queue is just one of them, and it depends entirely on the context of the application; and
4. It's easier to be notified about changes within the system with a queue. Without a queue one would have to poll the database for updates. This added benefit depends once again on the context of the application, in the case of a small application, polling the database is sufficient.

2.3.2 Model-View-Controller (MVC) Architecture

MVC is an architectural pattern that is widely used in the read model of web applications. It separates the domain logic from the user input logic and the presentation of information to the client. MVC architecture is composed of the following three components illustrated in Figure 2.7:

1. **Model:** Manages the data of the application. It is independent of the user interface.
2. **View:** Responsible for the presentation of information to the user. It is triggered by the controller.
3. **Controller:** Handles communication between the model and view components when it receives user input.

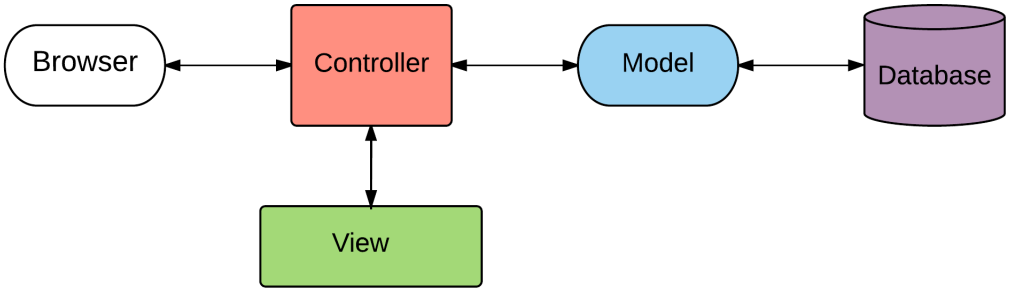


Figure 2.7: MVC architectural pattern.

In the MVC architectural pattern, the controller receives all the application requests. Depending on the request it can either render a view for the client immediately, if no interaction with the database is required, or it can interact with the database via the model. When the model interaction is completed the controller will render the required view and return it to the browser.

A few advantages of MVC architecture adapted from *MVC Architecture Driven Design and Agile Implementation of a Web-Based Software System* are listed on the following page[20]:

1. The separation of concerns and low dependency between components makes modification and maintenance tasks easier, which in turn makes scalability easier.
2. Programs are able to run on different platforms without having to change the source code.
3. The application's work is divided i.e. the web designer and the web developer for the application can work on their respective parts of the application independently.

2.3.3 Functional Programming

Functional programming is a programming paradigm like that of object-oriented programming and procedural programming. It is a way of constructing software based on the following principles: building software through the creation and composition of pure functions, whilst avoiding shared state, mutable data and side effects to the greatest extent possible. Functional programming is declarative. The application state is not shared with object methods, as is the case in object-oriented programming, rather the application state flows through pure functions.[21]

To get a better understanding of how these two programming paradigms differ, an example of how each might create a simple Shopping List program that helps keep track of items to be bought at the grocery store. Classes and Objects are the main units of programs in object-oriented programming, therefore a simple solution for the Shopping List program may be a Shopping List class and a Shopping Item class, as shown in Listing 2.1. Each class has its own member variables to hold the data and internal methods to operate on that data. On the other hand, functional programming operates on simple structures, so one can assume that the Shopping List will be represented by an array. We can then define the *addItem* function that returns a copy of the list with the new item appended to it, as shown in Listing 2.2.[22]

In object-orient programming a new object is created for each new piece of data. It is necessary to define what object each piece of data represents. In functional programming data is represented in simple structures, such as a hash with name and value keys in the shopping list example. It is not necessary to define what each piece of data is and what it can do, instead importance is placed on the raw data itself and what operations and transformations can be performed on it.[22]

Listing 2.1: Simple implementation of Shopping List program in object-oriented programming.

```
1  //Object-Oriented Programming
2
3  class ShoppingList{
4     constructor(items) {
5        this.items = items
6     }
7     addItem(item){
8        this.items.push(item)
9     }
10 }
11
12 class ShoppingItem {
13    constructor(name, price) {
14       this.name = name
15       this.price = price
16    }
17 }
18
19 //Adding shopping list items in initial list
```

```
20
21 var myShoppingList = new ShoppingList([
22    new ShoppingItem ("bread", 5.99),
23    new ShoppingItem ("butter", 57.99),
24 ])
25
26 //Adding new items to myShoppingList
27 myShoppingList.addItem(
28    New ShoppingItem("bananas", 1.99))
```

Listing 2.2: Simple implementation of Shopping List program in functional programming.

```
1  //Functional Programming
2
3  function addItem(list, item) {
4     return list.concat(item)
5  }
6
7  //Adding shopping list items to initial list
8
9  const myShoppingList = [
10    {name: "bread", price: 5.99},
11    {name: "butter", price: 57.99}
12 ]
13
14 //Adding new items to the myShoppingList at a later point
15 const nextWeeksShoppingList = addItem(myShoppingList,
16                           {name: "bananas", price: 1.99})
```

Clojure Functional Programming Language

Clojure is a dynamic programming language created by Rich Hickey.[23] It is a LISP language hosted on the Java Virtual Machine (JVM). Clojure addresses the need for a 'general purpose language that is suitable in those areas where Java is suitable,' whilst providing the succinctness, flexibility and productivity of dynamic languages. It is a LISP language that integrates seamlessly with the JVM and has been designed for concurrency. It solves this need through its following key features[23]:

1. It, like other LISPs, treats code-as-data and extends this approach to vectors and maps;
2. The reader supports a superset of Extensible Data Notation (EDN);
3. Lazy sequences;
4. Java Interoperability;
5. Immutable data structures; and
6. Built-in support for concurrency.

Clojure's programming approach allows for the majority of the applications' code to be written as pure functions, with each function operating on immutable data values. The advantage of pure functions is that they have no side-effects, ensuring they are easy to understand, test and are thread-safe.[23]

Clojure's LISP based syntax is also very concise, resulting in less time spent typing and more time spent on designing, as demonstrated in Listing 2.3 and Listing 2.4 with the same basic *Hello World* program example given in both the Java and Clojure programming languages. Adrian Cockcroft, currently a technology fellow at Battery Ventures and the previous chief architect of Netflix's cloud services, has said the following on Clojure in an interview with The New Stack: *"The really cool things I'm seeing being built, some of them are in Go, and the other one is Clojure. A lot of the best programmers and the most productive programmers I know are writing everything in Clojure and swearing by it, and then just producing ridiculously sophisticated things in a very short time. And that programmer productivity matters."*[24]

Listing 2.3: Hello World program example written in the Java object-oriented programming language.

```
1 //Java Hello World Program - Object-Oriented Programming
2
3 public class HelloWorld {
4     public static void main(String[] args) {
5         // Prints "Hello, World!" to the terminal window.
6         System.out.println("Hello, World!");
7     }
8 }
```

Listing 2.4: Hello World program example written in the Clojure functional programming language.

```
1 //Clojure Hello World Program - Functional Programming
2
3 (ns hello-world.core)
4
5 (defn -main [& args]
6   // Prints "Hello, World!" to the terminal window.
7   (println "Hello, World!"))
```

A few advantages of Clojure's distinctive features are listed below[25, 23]:

1. Functional programming separates calculation from state and identity, therefore parallel programming is easy to implement and programs are easier to understand, write, test and optimize.
2. Clojure's Java interop gives the programmer access to Java platform semantics, providing the performance and semantics equivalent to that of Java.
3. LISP separates the reading from evaluation. It also provides a compiler and macro system at runtime.
4. Clojure's time model separates values, identities, state and time. It can perceive and remember information without fear of erasing the past as its data structures are immutable. Clojure's immutability also makes it easier to write thread-safe code as mentioned previously.

2.3.4 RESTful APIs and Application Routing

"RESTful web APIs" refer to those APIs that are implemented using HTTP and conform to the REST API design architecture. REST defines a set of architectural principles for interoperability between computer systems on the internet. REST web services allow requesting systems to access and manipulate web resources using a standard set of predefined operations. It uses HTTP to make requests between servers, is platform and language independent and standards based.[26] Roy Fielding, a principal author of the HTTP specification, describes REST *"as a coordinated set of architectural constraints that attempts to minimize latency and network communication, while at the same time maximizing the independence and scalability of component implementations."*[27]

REST includes some of the following design features [27]:

1. Client-server separation: improves the portability of the user interface across multiple platforms. Both the front-end and back-end of the application can be replaced, or scaled, independently of each other.
2. Statelessness: the client stores session data and no client data is stored on the server between requests.
3. Cacheability: clients may cache responses to improve responsiveness of the application.

RESTful Application Routing

Application routing is the process of using Uniform Resource Locator (URL)s to drive the user interface. Users navigate an application by clicking through URLs. The URL is essentially the point of entry for the user to access the application. The URL is sent to the server via HTTP and the server responds appropriately via a server-side router. "RESTful" application routes map HTTP verbs to route handlers in an application. Handlers are functions that accept a request map and return a response map.

Real-Time Updates, WebSockets and AJAX

Originally, a server-push was simulated in earlier versions of web services. The client had to make a long request that remained open until the server was ready to push a message. Once the message was received the connection was closed and then a new request would be made. This method involved implementing a number of techniques to ensure no messages were missed, while the connection was open, and its resource requirements were expensive.

In 2006, Opera introduced Server-Sent Events (SSE). SSE made it possible to stream events from the web server to the browser. It was developed further and in 2011, the WebSocket protocol was standardised. WebSockets allows for a persistent two-way connection between the client and server, providing the ability to push data to the clients whenever data changes on the server. The client no longer had to commit manual refresh actions to update the information on the web page. This greatly improved the

responsiveness of an application with changing content and many concurrent connections. [28]

Asynchronous JavaScript and XML (AJAX) makes use of a combination of server- and client-side technologies to also allow parts of a web page to be updated without having to reload the page. Unlike WebSockets though, AJAX does not allow for a persistent connection or two-way communication, and is therefore not suited for providing data updates to the client as soon as it is available. An example of where AJAX may be used is to provide an Internet Movie Database (IMDB) movie's star rating and have it show up immediately on the web page, or request to download a file of data from the server.[28]

Front-End Framework

A front-end framework is composed of packages containing files and folders of pre-written, standardized HyperText Markup Language (HTML) and Cascading Style Sheet (CSS) code. The framework provides a foundation upon which one can build and customize. Front-end frameworks usually include the following components[29]:

1. A grid to organize the design elements.
2. Font styles and sizing that vary based on function (e.g. heading, paragraph, section, etc.).
3. Pre-written website components such as buttons, menus, etc.

Examples of some front-end frameworks are Bootstrap, Foundation and Semantic UI.

2.4 Application Testing

This section discusses how testing is performed in the development environment of the application, the types of application testing that may be performed to help ensure the application behaves as expected and the significance of each test. The tests focus on testing the robustness, reliability, performance, functionality and user experience of the application.

2.4.1 Local Test Server

A local server is hosted locally on your computer. A local server allows one to test all the application's API without a connection to internet or network. This isolates one from the issues that may arise from both a network or internet connection. Running a local server does not impair the correct functioning of browsers or browser extensions, nor will it affect the 'integrity of a live site.' A local server is useful for the purposes of efficiently testing the functionality and appearance of an application. It may also be used during the process of familiarization of new software, allowing one to test and experiment and view the changes in the development environment of the application.[30]

2.4.2 Functionality Testing

Functionality testing is used to verify that the application performs as it was designed to. It is also used to check the usability of the application by ensuring the navigational functions are working as required and intended.

2.4.3 Profiling

Application profiling is a performance analysis technique. During testing the application is analyzed dynamically to assist with program optimization and reveal performance bottlenecks. Profiling can be used to measure, the memory or time complexity of a program, the usage of particular instructions and the duration of function calls for example. Profiling is conducted by measuring or monitoring the program source code or its binary executable form with a tool called a profiler. Profilers 'insert instructions into the application under test to obtain frequency, memory usage and elapsed execution time of method calls.'[31] Profilers make use of many techniques, such as event-based, statistical, instrumented and simulation methods. The profiling of Clojure code may be performed with any Java profiler, such as *YourKit Java Profiler* and *YourKit .NET Profiler*. Alternatively, *ptaoussanis/timbre* and *ptaoussanis/tufte* are two Clojure libraries one can use for simple profiling.[31]

2.4.4 Memory Leak Detection

A memory leak occurs when a program fails to release memory it has consumed for temporary use, resulting in a gradual loss of available computer memory. The available memory is thus used up and the program can no longer function. This can terminate frequently or continuously run programs, or terminate the whole system. A memory leak arises from a programming bug making it very important to test for memory leaks in the development phase of the application or program.

The GC process is used to recycle memory used by a Java based application. The garbage collector is used to reclaim the memory of objects that are no longer used. The JVM determines when to run the garbage collector. While the application or program is running the JVM tracks which objects in memory are still being used and which are not. The unused objects are discarded and the memory is recovered and reused.

A heap is an area of memory used for dynamic memory allocation. When an application requires memory temporarily, it can allocate the memory from the heap and when it is no longer needed the memory is released for future allocations. If there is no memory space available for creating new objects in a heap, the JVM produces an out of memory error because a bug in the program prevents it from releasing memory that is no longer needed. There are various tools available for memory profiling, such as *Java Flight Recorder*, *JProfiler* and *Eclipse Memory Analyzer*.[32]

2.4.5 Load Testing

Load testing is putting simulated demand on software or an application in order to determine its behaviour under various conditions. It makes it possible to single out bottlenecks, bugs and component limitations. The primary goal of load testing is to determine the maximum work or volume the application can handle without a significant decrease in system performance.[33]

2.4.6 Stress Testing

Stress testing is performed as a part of performance testing. It is the process of determining the ability of a computer, network, program or device to maintain a certain level of effectiveness under unfavourable conditions. The system is overloaded during the stress

testing to measure it's ability to sustain the stress and determine its boundary conditions and points of failure.

2.4.7 User Experience Testing

The user experience is the experience the application creates for the people who use it. It includes everything the user experiences when they interact with the application. It is an important factor in an application's design as it can often be the difference between a successful application or failure. If users do not want to use the application then it becomes redundant technology. Consequently, it is important to conduct user experience tests, so the feedback received may be used to implement changes that will allow the application to provide the best user experience possible.

Ten general usability principles for user interface design are outlined on the following pages.[34, 35] Not all the heuristics will apply to an application, as it will depend on the application's context, but all aim to enhance the user experience.

1. **Visibility of system status**
   The system keeps the user informed about what is happening via timely feedback.

2. **Match between system and real world**
   The system uses language that is familiar to the user and the information displayed follows a natural and logical order.

3. **User control and freedom**
   Users are able to move between states easily. If a user accidentally enters an unwanted state by selecting the 'wrong' system function they are able to easily navigate out of that state.

4. **Consistency and standards**
   The system is consistent in design and layout. It conforms to platform conventions, shows similar information in the same place on each web page, uses consistent definitions throughout and makes use of universal commands.

5. **Error prevention**
   The system is designed to take steps to prevent errors from occurring. If error prone conditions cannot be eliminated, the system ensures users are presented with a confirmation message before they commit to the action.

6. **Recognition rather than recall**
   Ensure the system options, objects and actions clearly visible to minimize the user's memory load. Ensure the user does not have to remember information from one part of the interaction to another and that instructions for use are clearly visible or retrievable.

7. **Flexibility and ease of use**
   The system incorporates accelerators that allow experienced users to navigate the system interaction faster than the inexperienced users. The system itself should be efficient to use. The user interface should be customizable and provide users with the option to cancel or re-order tasks.

8. **Aesthetic and minimalist design**
   The system user interface interaction should be concise and not contain irrelevant information. The addition of unnecessary information reduces the visibility of the important information.

9. **Help users recognize, diagnose and recover from errors**
   The error messages displayed should be in plain language with no error codes and concisely indicate the problem and suggest a possible solution.

10. **Help and documentation**
    It's considered better if a system can be used without documentation but it may be necessary to provide additional help. The documentation should be easy to search for, concentrated on the user's task and list concrete steps to be performed.

Chapter 3

Methodology

This chapter discusses the project development and how the research was conducted. The chapter includes an explanation of the project software development method used. The project research approach is illustrated in the form of a flow diagram which is discussed in detail. A high level application design that incorporates the design concepts and architectural patterns discussed in the literature review is provided, as well as an outline of the acceptance testing based on the project terms of reference. The chapter concludes with a Gantt chart of the project phases.

3.1 Software Development Method

The software development method used in the project combined aspects of the Agile and Waterfall methods of development. The project followed the linear project phases in Figure 3.1 and stuck to the original requirements and objectives outlined in the beginning of the project, both aspects of the Waterfall development method, but also allowed for adaptation, iteration and re-design when necessary, which are characteristics inherent to the Agile development method.

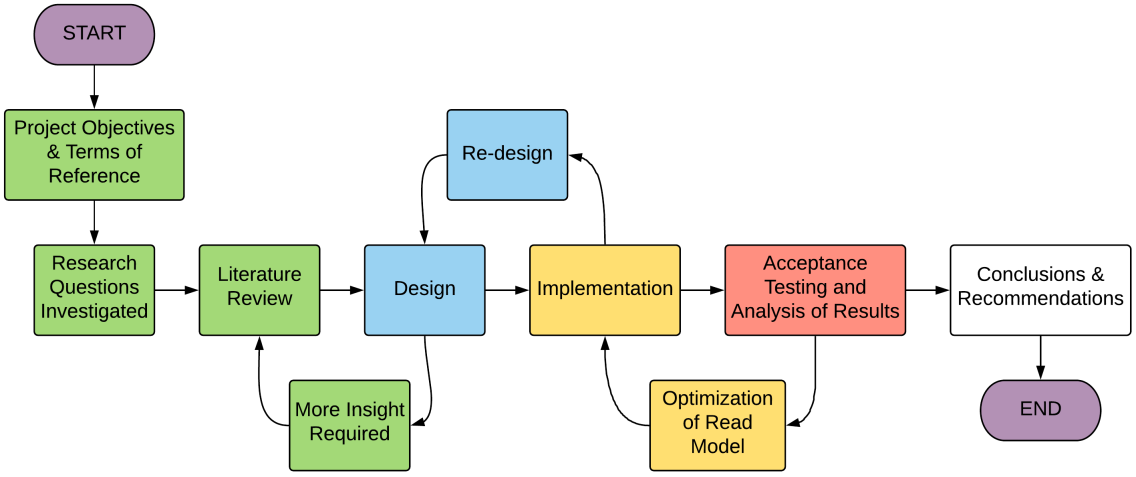


Figure 3.1: Flow chart of project phases.

3.2 Research Approach

The research methodology followed in the project is outlined in the flow diagram in Figure 3.2. It indicates how the research questions and project objectives are to be investigated. The subsequent sections discuss each stage in the flow diagram.

3.2.1 Project Objectives & Terms of Reference

This project stage comprised of the steps taken to try establish if a real-time air quality monitoring system existed for the City of Cape Town, or South Africa, and then requesting access to the raw sensor data and the data API. After communicating with the SAAQIS and City of Cape Town environmental department, it was established that the SAAQIS did not have a real-time air quality system in place. The response from the City of Cape Town was that it does not allow members of the public access to 'unverified data' and therefore access to their air quality data API was not granted. What the City of Cape Town meant by 'unverified data' was not mentioned in their response. As a result the application design included a data generator in the terms of reference requirements to mimic that of a Push API which can easily be replaced with a real air quality data API in the future. The objectives and terms of reference will be used to evaluate if the project can be deemed successful or unsuccessful.

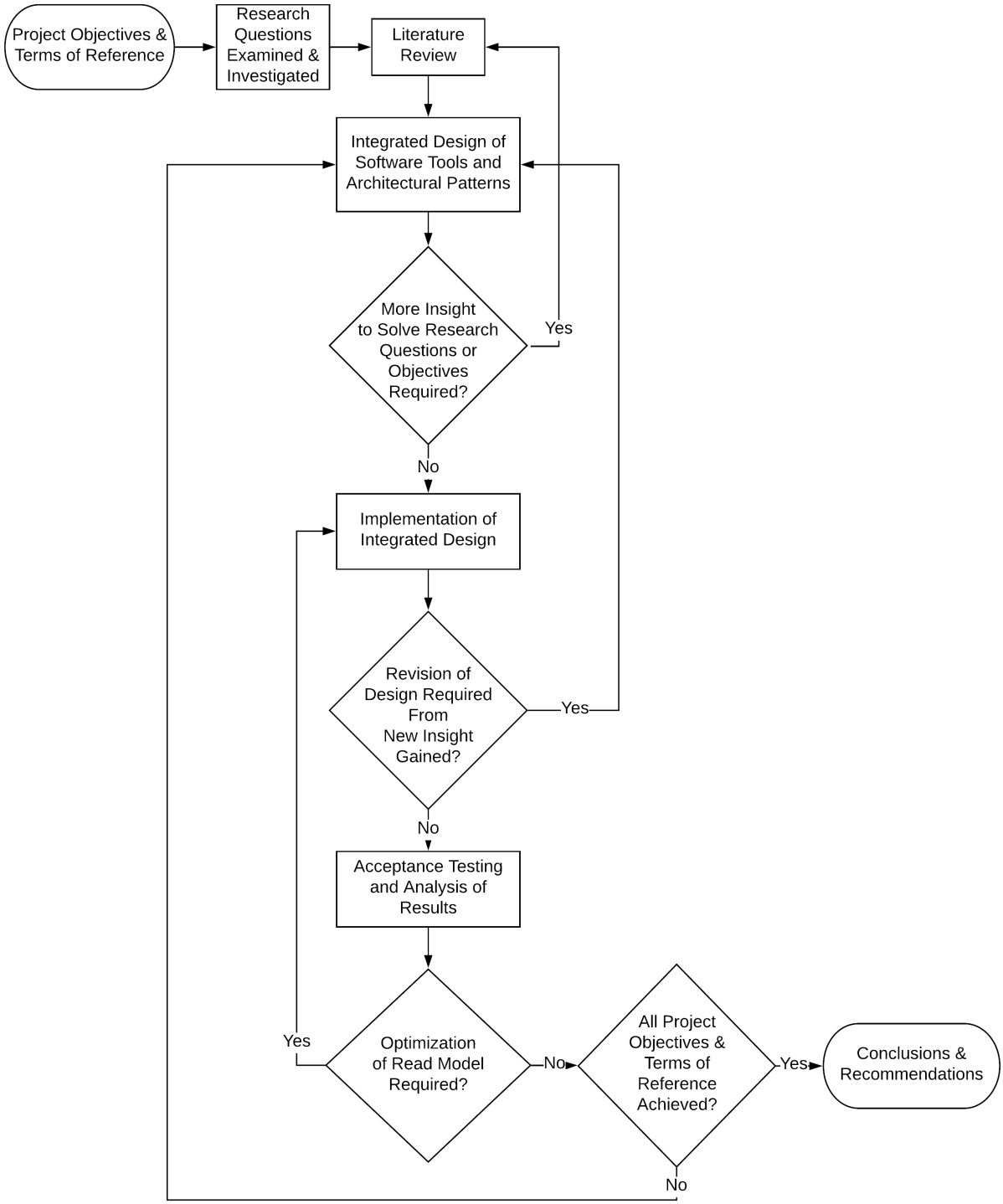


Figure 3.2: Flow diagram indicating how the research questions and project objectives are to be investigated.

3.2.2 Research Questions Examined & Investigated

A comprehensive set of project research questions to be investigated are listed in section 1.3.1. They have been established from the project objectives and terms of reference to guide the research.

3.2.3 Literature Review

The literature examined during the investigation into the project research questions is narrowed down to only that which provides the necessary information to meet the project objectives and terms of reference, as well as stay within the bounds of the project scope and limitations. Research papers, journal articles, Clojure, Clojure Spec, Apache Kafka, Datomic, WebSockets documentation, books and various reliable blog posts, GitHub libraries, conference talks and websites were reviewed to better understand the project requirements and proposed system architecture. A few of the main research topics included:

1. Ambient air pollution and the WHO ambient air guidelines;
2. The USEPA AQI calculation method;
3. EDA and CQRS architectural patterns;
4. MVC architecture;
5. Functional programming and Clojure;
6. Immutable databases;
7. "RESTful" APIs and application routing;
8. Real-time updates;
9. Front-end frameworks; and
10. Application testing.

3.2.4 Integrated Design of Software Tools & Architectural Patterns

The various design aspects in the literature review and application requirements in the terms of reference were analyzed in detail during this stage to develop an integrated conceptual design for the air quality index application. These aspects included:

1. Analyzing how the air quality data will flow through the system;
2. The calculation of the AQI using the USEPA method;
3. Separating the system architecture into the read model with MVC architecture and write model with EDA for the implementation of CQRS;
4. Design of the required data models and the model associations, database schema and Datalog query structure;
5. Establishing the REST API and application routes required for functionality;
6. The front-end design components which included a WebSocket server, AJAX requests, Semantic UI framework components and the user interface layout; and
7. Establishing what tests should be conducted, and the testing tools required, for robustness, performance, functionality and user experience.

3.2.5 More Insight to Solve Research Questions or Objectives Required?

If further insight was required to find solutions to project objectives, or answers to research questions, for the integrated design of the air quality index application software tools and architectural patterns, the project looped back to the literature review stage. Once no further information was needed to solve for the project objectives and research questions in the integrated design, the project moved onto the next stage in the flow diagram.

3.2.6 Implementation of Integrated Design

During this stage the application development environment was setup. The software required for development was installed and the software libraries were selected to implement the established integrated application design. Clojure Code was written to implement the following:

1. Data models with the desired associations and parameters for scalability and extensibility;
2. The data generator and the Clojure Spec data validation;
3. The Datomic database was setup and the database schema;
4. A multi-broker Kafka cluster was setup for the EDA in the write model;
5. Batch processing of Kafka queue messages;
6. MVC architecture and database queries for the read model;
7. The application web server, routes, route handlers, WebSocket server and AJAX request; and
8. The necessary HTML and CSS views for the user interface.

3.2.7 Revision of Design Required From New Insight Gained?

The project underwent an iterative re-design and implementation process largely due to new insight into the original application design discovered during implementation, after the initial integrated design stage. If new insight was gained that required changes to be made to the integrated design, the project looped back to the integrated design of software tools and architectural patterns stage. Once no further changes to the application design were required and the implementation of the integrated design was completed, the project moved onto the next stage in the flow diagram.

3.2.8 Acceptance Testing & Analysis of Results

During this stage the application was tested in development environment using a local test server. The acceptance testing included robustness testing, functionality testing of the

application's features, performance testing and testing of the user interface as outlined in the terms of reference. The robustness testing included testing the fault tolerance of the implemented Kafka cluster and the implemented Clojure Spec data validation to ensure both work as expected, profiling the read model for the identification of bottlenecks and verifying the application does not have any memory leaks. The functionality tests were conducted to verify the application features performed as intended during development and that it met specified the project requirements. The performance testing comprised of page response time testing and load testing the application. During the user interface testing potential users of the application were asked to use the application and provide feedback on their user experience, by rating various aspects of the user interface in a survey.

3.2.9 Optimization of Read Model Required?

Optimization of the application's read model is included within the project scope and limitations. At this stage it was decided if optimization of the read model was required based on the results obtained from the acceptance testing. If optimization was necessary than the project looped back to the implementation of the integrated design stage to allow for the mitigation of problem areas in the read model implementation of the application. Otherwise, the project continued to the next stage in the flow diagram.

3.2.10 All Project Objectives & Terms of Reference Achieved?

If only a subset of the project objectives and terms of reference have been achieved at this stage in the project life-cycle then it is redirected back to the integrated design of software tools and architectural patterns stage, so as to modify the application design to satisfy all the project objectives and terms of reference. However, if all the project objectives and terms of reference have been achieved, then the project life-cycle moves onto the final stage in the flow diagram.

3.2.11 Conclusions & Recommendations

Conclusions are drawn based on the results obtained from acceptance testing and the feedback received from those individuals who participated in the user interface test.

The overall system is analyzed to assess whether all the project objectives and terms of reference have been satisfactorily met and whether or not the project may be considered a success. Recommendations for future work are then given based on the feedback received from user experience testing, application performance and future application features that were not included in the project scope.

3.3 High Level Application Design

The application architectural design is composed of the design concepts and architectural patterns discussed in the literature review. The conceptual high level design of the application is shown in the flow diagram in Figure 3.3. The air quality data moves through the system's architecture as illustrated in Figure 3.3, which involves the following steps:

1. The data generator simulates a Push API and produces air quality data that is fed to the application periodically.
2. The air quality data produced by the data generator is validated for the purposes of data type checking, as would be the case with data received from an external API to ensure the data was valid and fit the application data models.
3. Event Sourcing is implemented to write the data to the application's immutable database.
4. An immutable database is used to store the air quality data after processing.
5. MVC architecture is implemented in the read model for application queries and to provide air quality index real-time updates (the bidirectional arrow between the immutable database and MVC architecture indicate database queries).
6. The client displays the air quality index information to the user and allows the user to submit air quality data requests (the bidirectional arrow between the client and the MVC architecture indicates communication between the client and the application's web server).

In accordance with requirement 3 of the research project (discussed in the terms of reference), the application architecture uses the CQRS architectural pattern to separate the read and write model for the separation of concerns and so each may be scaled

independently according to independent performance requirements. EDA is implemented in the write model and the MVC architectural pattern is implemented in the read model.

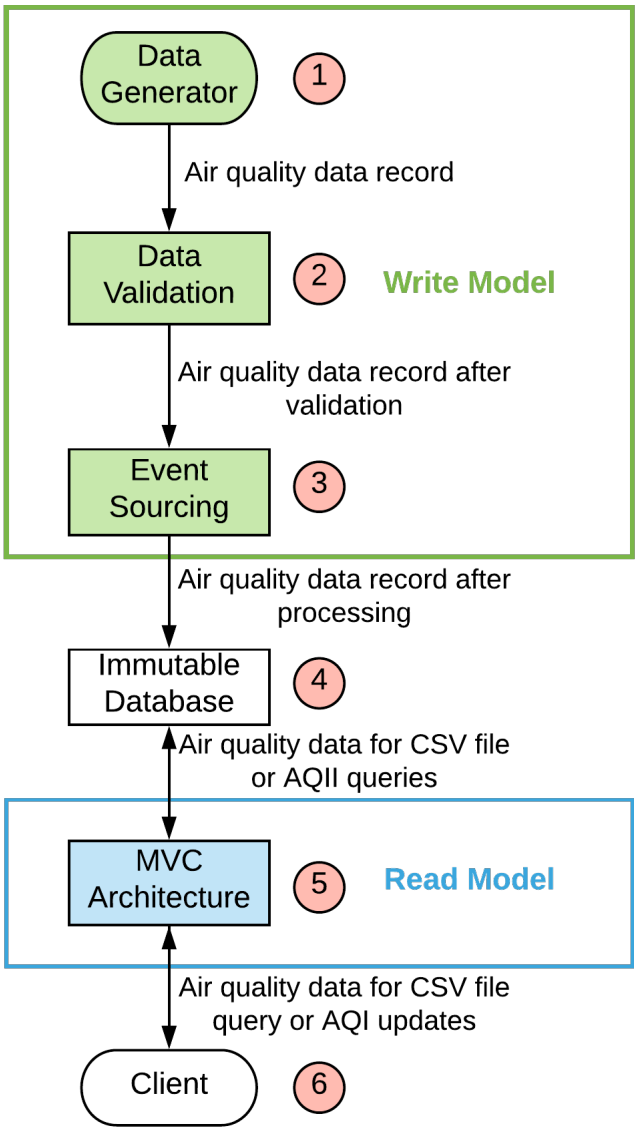


Figure 3.3: The high level design of the application.

3.4 Acceptance Testing

This section provides more detail with regard to the acceptance tests, outlined in the terms of reference, that will be conducted in order to satisfy requirement 7 of the research project.

3.4.1 Kafka Fault Tolerance

Apache Kafka is used for the implementation of EDA in the write model of the application, fulfilling part of requirement 3 of the research project. The test will be conducted to ensure the Kafka multi-broker cluster is fault tolerant. It is a test to ensure the system is robust enough to continue normal operation in the event of broker failures. If the leader broker for a partition in the cluster suddenly failed for example, one of the other brokers in the cluster should become the new leader for that partition and normal broker operation should continue, otherwise the processing of the air quality data by the Kafka consumer will be negatively affected and data may be lost. The test will be carried out by killing the process of the leader broker in the cluster and observing the behaviour of the cluster thereafter.

3.4.2 Clojure Spec Data Validation

The Clojure Spec data validation of the air quality data received from the data generator forms part of requirement 3 of the project research. The test aims to ensure the validation of the data received from the data generator is performed as expected, that the data is checked for validity, and any invalid data received is detected and appropriately logged. Both invalid and valid air quality datasets will undergo the data validation process and the results of the data validation will be analyzed.

3.4.3 Bottleneck Identification

A profiler will be used in the read model of the application to identify bottlenecks and areas that require optimization to ensure the read model provides a satisfactory response time and does not have a negative impact on the user experience. The high level functions of the read model will be profiled and the time complexity of each function will be analyzed and optimized where necessary. This test is a component of requirement 7's performance testing.

3.4.4 Memory Leak Detection

The test will be used to determine if the application contains any memory leaks which have a negative effect on the performance and operation of the application. Software monitoring tools will be used to monitor the memory usage of the application whilst in operation in real-time, as well as over a set period of time, and the results will be analyzed for the possibility of a memory leak. This test is a component of requirement 7's performance and robustness testing.

3.4.5 Functionality

The implementation of Google map real-time AQI updates to the client and the air quality data CSV download satisfy the research project's requirements 4 and 6 respectively. The Google map AQI real-time updates and the AQI CSV data file download functionality of the application will be tested by running the application using a local server in the development environment, imitating user behaviour and sending browser requests to the application server for each feature. The functionality test for each will focus on the application processing the browser request and returning the expected response. These tests are necessary for requirement 7's functionality testing.

3.4.6 Web Page Response Time

Page response time testing is essential to ensure the application is responsive and provides a good user experience. It is a measure of how long it takes for a web page to load. The page response time testing will be conducted by running the application in the development environment and measuring the page response time of the application using the Chrome Developer Tools without CPU throttling, on a high-end device (2x slower) and a low-end device (5x slower) and analyzing the results. These tests is a component of requirement 7's performance testing.

3.4.7 Load and Stress Testing

Load and stress testing will be conducted, with a load generator software tool, as a measure of the application's performance under various concurrent user conditions and

to identify the application's boundaries with respect to the number of concurrent users it can sustain before failing to respond to user requests. The application will be run using a local server in the development environment and the Gatling load testing tool will be used to mimic browser behaviour and simulate users using the application concurrently and the results will be analyzed. These tests is a component of requirement 7's performance testing.

3.4.8 User Experience

The satisfaction of requirement 5 in the terms of reference required the development of an application user interface. The quality of the user interface will be determined through user experience testing. Monitored in-person usability testing will be performed by running the application on a local server in the development environment. Volunteers will fill out a survey rating their user experience with respect to various aspects of the user interface and the results will be analyzed.

3.5 The Project Time-line

The approximate project time-line is outlined in the Gantt chart in Figure 3.4. The Gantt chart is composed of the linear project phases from Figure 3.1 characteristic of the Waterfall software development method. The approximate time period for each linear project phase is outlined in Table 3.1.

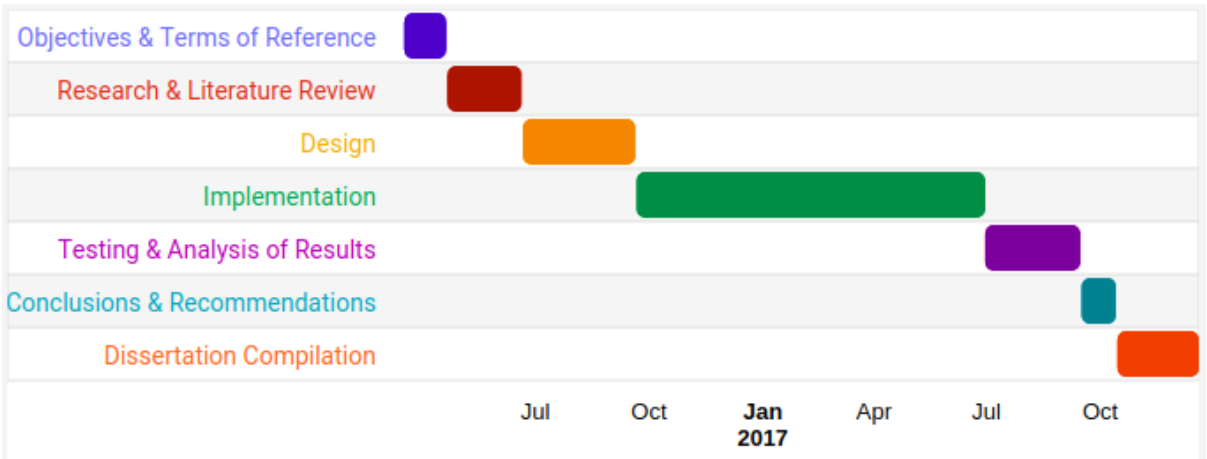


Figure 3.4: Gantt chart outlining the major project phases and duration.

The examination and investigation of the research questions is included in research and literature review in the Gantt chart above.

Table 3.1: Time period of each project phase.

Project Phase	**Time Period**
Objectives and Terms of Reference	February - March 2016
Research & Literature Review	March - May 2016
Design	May - August 2016
Implementation	August 2016 - May 2017
Testing & Analysis of Results	May - August 2017
Conclusions & Recommendations	August - September 2017
Dissertation Compilation	September - November 2017

Chapter 4

Detailed Application Design

This chapter builds on the high level application design of the preceding chapter and provides more detail with regard to the application design. The chapter begins by detailing the available hardware. An overview of the detailed application design is provided, as well as the back-end and front-end requirements extracted from the terms of reference. The design of the application's data generator, read and write model and database are also covered in detail in this chapter.

4.1 Available Hardware

The application was developed on a laptop computer with the specifications outlined in Table 4.1.

Table 4.1: Laptop specifications.

Make and model	Acer Aspire 5750G
Processor type	Intel Core i7(2nd Gen)-2670QM
Max turbo speed	3.1 GHz
Number of cores	4 (Quad-Core)
RAM installed size	4 GB
Memory max supported size	8 GB
Hard drive	500 GB HDD
Display	15.6" HD LED LCD
Graphics	NVIDIA GeForce GT 630M
Battery	6-cell Li-on
Operating system	Linux Mint 18 Cinnamon 64-bit

4.2 Overview of Detailed Application Design

The application architecture was designed to satisfy the requirements and functionality outlined in the terms of reference.

4.2.1 Back-End Architecture Requirements

The back-end architecture needed to satisfy the following requirements and contain the following functionality:

1. Create a data generator to simulate air quality data. The air quality data must be from the same geographical locations as the real-time air quality monitoring sensors setup by the City of Cape Town municipality. The data generator needs to push data out periodically. The air quality data measured must be for pollutants the WHO has scientific evidence for and within the data ranges extracted from the City of Cape Town air quality data archives. This is requirement 2 of the project.

2. The data received from the data generator needs to be validated in the write model, prior to being placed on the event queue. This is a component of project requirement 3.

3. The application write model must implement EDA and the MVC architectural pattern must be implemented in the read model. Each model should be decoupled for the implementation CQRS to allow for the separation of concerns and independent scalability, extensibility and maintainability. This is a component of project requirement 3 and project requirement 6.

4. Be data-centric and make use of an immutable data store, not just for the implementation of EDA, but to ensure the application acts as the foundation for future work and enables the development of air quality prediction models and pollution forecasts in the future. This is a component of project requirement 6.

4.2.2 Front-End Requirements

The user interface of the application needed to satisfy the following requirements and provide following functionality:

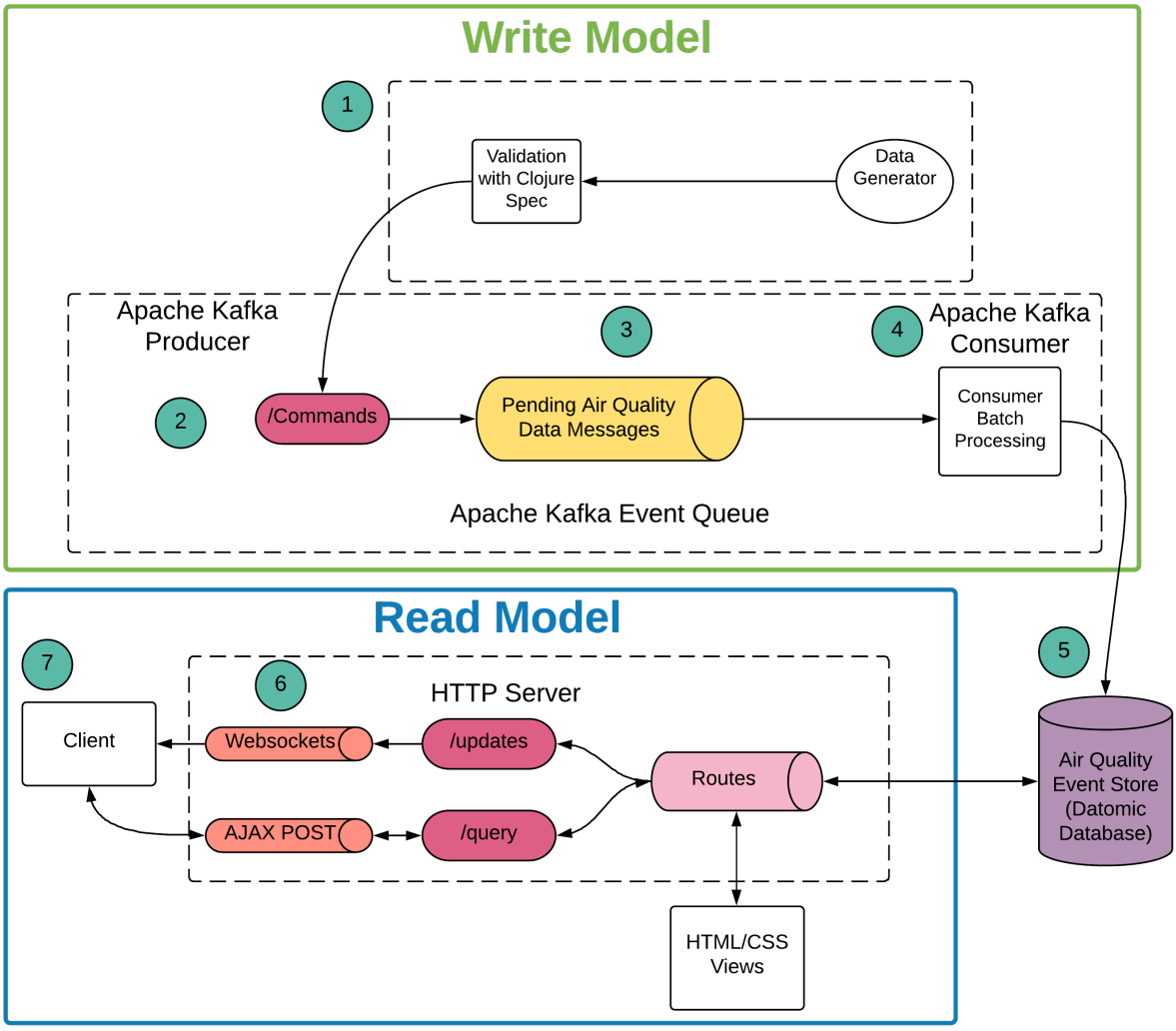
1. Display real-time air quality index data, based on the USEPA 2016 standard discussed in section 2.1, for each suburb within the City of Cape Town that the municipality has placed air quality monitoring sensors in. This is project requirement 4.

2. Allow the user to be able to download air quality data in a CSV file format via asynchronous web communication. This is a component of project requirement 6.

3. Be informative for the user such that it provides a basic understanding of what the air quality index is, it's impact on health and precautionary measures. This is project requirement 5.

4.2.3 Detailed Application Architectural Design

The design of the application is outlined in Figure 4.1. The detailed application architectural design builds on the design concepts, and the flow of the air quality data through the system, previously discussed in the high level application design.



1. Air quality data collected from data generator and validated with Clojure Spec.
2. Air quality data messages placed on Apache Kafka queue.
3. Single node multi-broker Apache Kafka cluster.
4. Kafka consumer batch processing events before transacting air quality data to the database.
5. Datomic database to air quality events and event data (Event Store).
6. Websockets is used for real-time updates to user's web browser and AJAX request used to retrieve air quality data required for CSV file download.
7. User's web browser.

Figure 4.1: Detailed design of system architecture showing the components that form the read and the write model. Both the models are connected via the air quality event store.

The application design in Figure 4.1 separates the read and write models of the application for the implementation of CQRS and the exploitation of its benefits (project requirement 3). Bobby Calderwood's suggested CQRS HTTP endpoints, discussed previously in section 2.3.1, are clearly shown in the diagram above. EDA is used for the write model of the application and the air quality data updates received from the data generator are modelled (*/commands*) as events. Air quality data is supplied periodically by the data

generator (project requirement 2). The data is validated using Clojure Spec (project requirement 3.3). The validated data is then passed to the Kafka producer which places the data onto the Kafka queue as an air quality event. In the application the air quality data update received from the data generator is modelled as an event and written it to a central log, the air quality Kafka topic (project requirement 3.1). The air quality data events are consumed from the Kafka queue in batches of 100 for higher throughput, processed and then transacted to the Datomic air quality event store (project requirement 3.2 and 6.1). WebSockets allow for full duplex communication between the client and the back-end of the application through HTTP requests and responses. A WebSocket server is used to provide real-time air quality updates to the client (*/updates*) (project requirement 4) and AJAX POST requests are used to query the air quality data required for the CSV file for the client (*/query*) (project requirement 6.2).

In the application design in Figure 4.1 the read model is composed of the MVC architectural pattern (project requirement 3.1), as previously mentioned in the high level application design. The Datomic data models are the model component of the MVC architectural pattern and are used for interaction between the controller and the air quality event store database. The controller component of the MVC architectural pattern is composed of the REST API: WebSocket server, application routes and route handlers; and the MVC view component is the application's web page displayed in the client's browser.

The application's back-end architecture was also designed to be a generic basic event-driven architectural template in the sense that, with a few modifications to the Clojure Spec data validation codes, the batch processing code and the database schema to accommodate a different dataset, the back-end architecture (minus the data generator) can be re-used in any application that collects data from an API and uses it to provide a service.

4.3 Data Generator

Access to the City of Cape Town's air quality real-time data API was not granted on the basis that 'unverified' air quality data may not be shared with members of the public, as previously discussed in section 3.2.1. As a result a data generator was created to mimic the air quality data API of the City of Cape Town, or any other air quality data API that returns air quality data. If the application was to be used by the City of Cape Town in the future then the data generator could simply be replaced by the data API

as illustrated in Figures 4.2 and 4.3 by point 1 (highlighted in blue). It was assumed that the sensors measured the concentration levels for all the pollutants and sent the data to an API endpoint as a collection rather than sending the data for each pollutant separately.

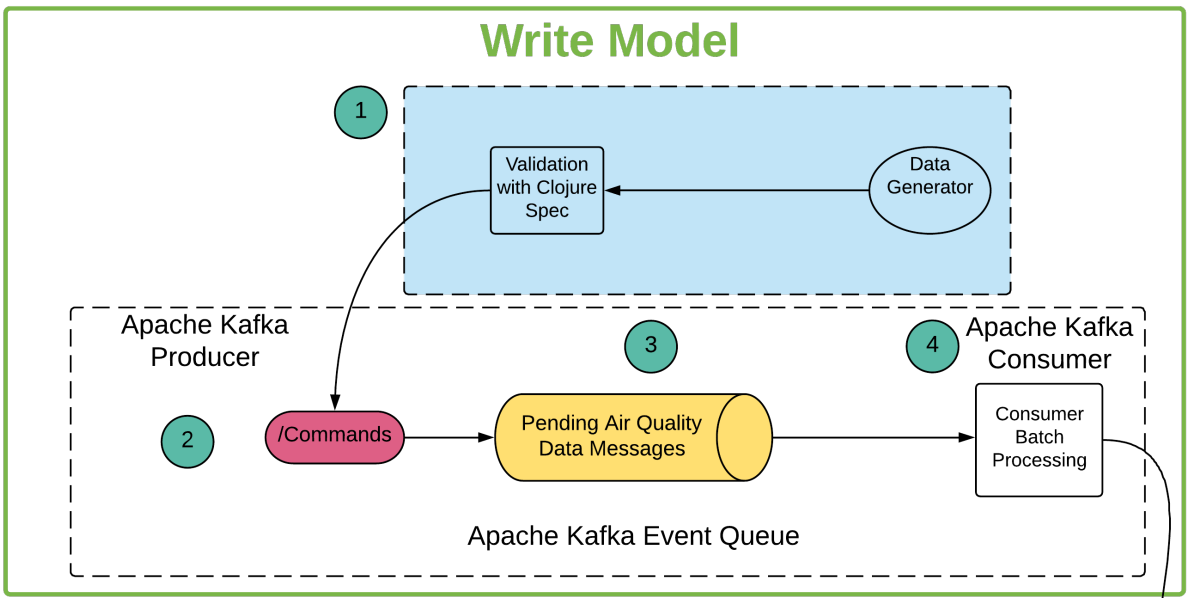


Figure 4.2: Application design with data generator to mimic an air quality data API and simulate air quality data.

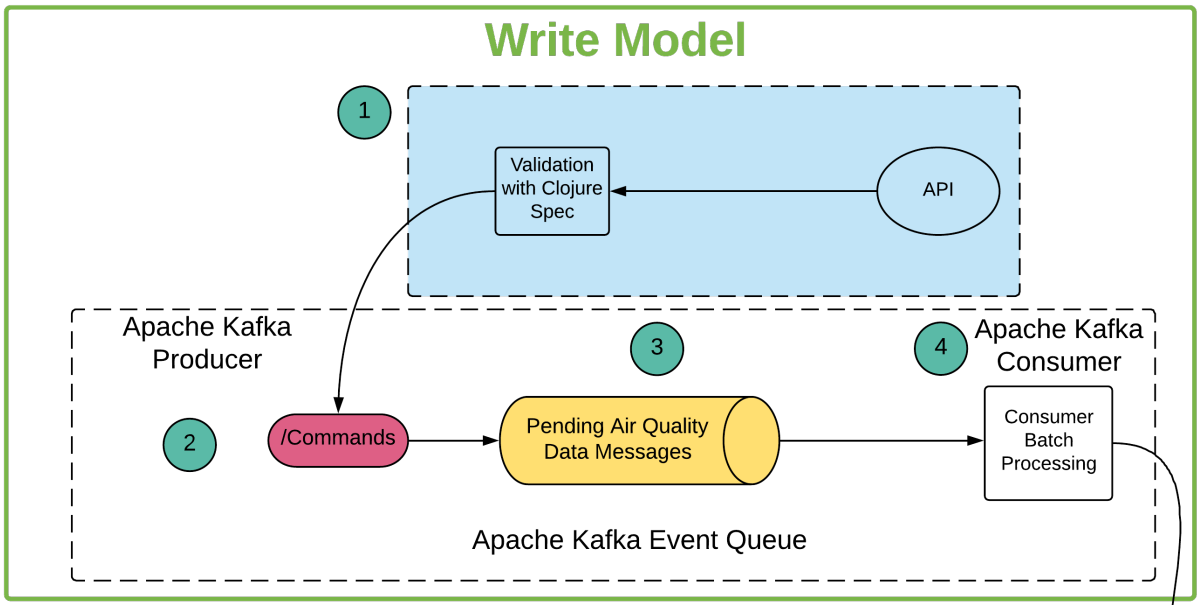


Figure 4.3: Application design receiving data from a real air quality data API without the need for data generator to simulate air quality data.

4.4 The Write Model

The write model of an application consists of the components that update the system's state. In this context it is composed of the elements that write the air quality data to the database. The following sections expand on each element within the write model of the detailed application design.

4.4.1 Data Validation with Clojure Spec

The process by which the data will be validated prior to being placed in the Kafka event log is outlined in flow diagram in Figure 4.4. The air quality data received from the data generator is type checked and validated using Clojure Spec codes, thereafter it is sent to the Kafka producer and placed on the Kafka event queue.

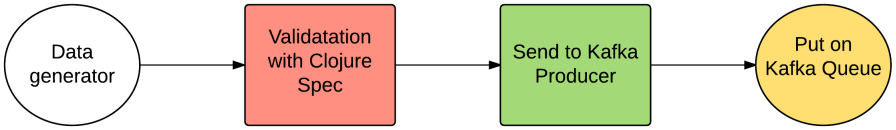


Figure 4.4: Process of air quality data validation with Clojure Spec.

Clojure is a dynamically typed programming language with a core principle being that information is represented as data, therefore data type specifications are not required for the code to run. The Clojure documentation strings may be used for communication with humans but they cannot be used by programs or tests. Essentially, Clojure does not have a standard, expressive and integrated system for specification and testing. Consequently, more powerful specifications are required to validate data types than the Clojure language itself provides.

Clojure Spec, a new core library, is a solution to this problem. It provides support for data and function specifications in Clojure. Specs are a logical composition of predicates, or self-created expressions, that data is validated against for conformity. If a value fails to conform to a spec it is possible to call the *explain* function to report why a value does not conform to a spec and exactly which predicate failed, which is helpful for debugging. Logical operations, such as *spec/and* and *spec/or*, allow for specs to be combined in a logical manner. Clojure Spec provides tools for validation, error reporting, conform support, parsing or destructuring, instrumentation, test-data generation and generative testing.[36, 37]

4.4.2 Apache Kafka

Kafka is a distributed publish-subscribe messaging system reworked as a commit log. It acts like a message queue. It is fast, scalable, durable and battle-tested at scale. As a result it is adept at being the backbone for event logging, or storage, in an event sourced application architecture. 'It maintains feeds of messages in topics. Producers write data to topics and consumers read data from topics.'[11] Topics can be partitioned and replicated across multiple nodes. Each topic is treated as a set of ordered messages and each message is assigned a unique offset. Kafka does not track which messages were read by the consumer and which were not, instead it keeps all messages for a set amount of time and consumers track their location in each log using the messages' unique offsets. As a result, Kafka can maintain a large number of consumers and large amounts of data at little cost. Kafka is ideal for building real-time streaming data pipelines that reliably move data between systems, applications or real-time streaming applications that transform, or react, to the streams of data.[38, 11]

4.5 Datomic Database

Datomic is a functional database. It focuses on largely information management rather than just information storage. Traditional databases 'forget' old information as soon as it has been updated and data is overwritten. Conversely, Datomic is accumulate only and keeps a record of all the entity's information and its previous history. This allows the programmer to 'time travel' and view the system history within a certain period of time. Because Datomic stores both the past states and current state of an entity, it is therefore capable of acting as both a canonical and aggregate database. As previously mentioned in the advantages of event sourcing, events may also be added to the database in order to make predictions and determine what would happen if a certain sequence of events were executed, which is useful for testing the application in different scenarios. One of the most defining features of Datomic is that it has many indices, such as attribute first and then value second, or entity first and then attribute second and so on. The different components of the stored data can be indexed in many different ways, depending on the type of questions that the system needs to ask, making it very versatile.[39]

4.5.1 Overview of Datomic Architecture

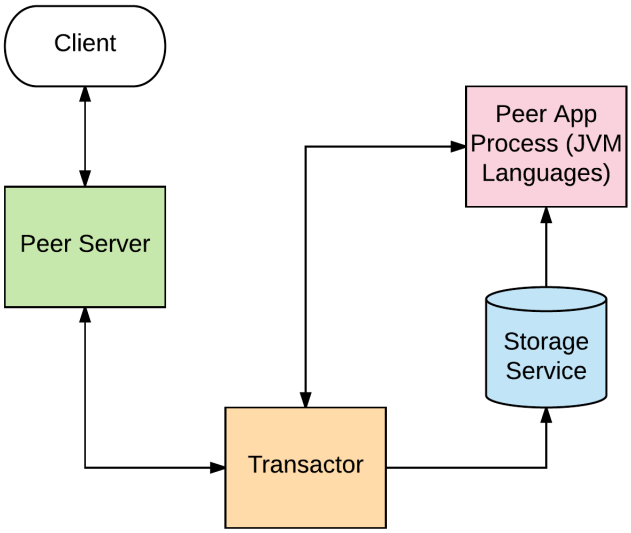


Figure 4.5: Simplified overview of the Datomic Architecture.

The Datomic Architecture is composed of the following key components as illustrated in Figure 4.5[39]:

Peer App Process A Peer is a component library that is embedded in applications. The Peer App Process is a process that uses the Peer component library, embedded in the application, to manipulate a database. Peers send transactions to and receive changes from the transactor. The library includes communication components for interacting with the transactor and storage services. It allows data access, caching and query capability from the storage service.

Client A library used to connect lightweight processes to Datomic. It sends transactions and queries to the Peer Server, supports all query capabilities to the Peer and can support non-JVM languages, but does not perform local caching.

Peer Server A JVM process that provides an interface for the Datomic Client library. It receives queries and transactions from Datomic Clients and sends transactions to, and receives changes from, the transactor. It allows data access, caching and query capability from the storage service.

Transactor The transactor receives transactions, processes them serially, commits the result to storage and transmits the changes to the Peers.

Storage Services Provides an interface to reliable, redundant storage and is available from third-parties, such as Amazon DynamoDB.

4.5.2 Advantages of Datomic

A few advantages of using a Datomic database for the storage of information within a system are listed below[39]:

1. Uses the EDN format and integrates seamlessly with Clojure.
2. Its entity API allows the programmer to walk the database graph, allowing for expressive code and queries.
3. Allows for architectural flexibility from lightweight microservice Clients or Peers for complex analytics processes.
4. Accumulate only, allowing you to track changes of entities over time and providing 100% audit logging.
5. Datomic uses the declarative query language, Datalog, which provides SQL power but with a simpler pattern-based syntax.
6. Designed to be ready for use with Amazon Web Services cloud storage.
7. Allows provisioning of storage services like Cassandra, SQL and DynamoDB.
8. Datomic peers provide built-in caching.
9. The Peer/Client model allows for horizontal scaling of queries without affecting other queries or the transaction throughput.
10. Atomic, Consistent, Isolated and Durable (ACID) consistent throughout.
11. Provides a flexible schema. It is capable of handling different data models such as row-oriented, column-oriented, graph and hierarchical data. This is because the schema defines the characteristics of attributes and not which attributes can be associated with which entities. The application makes the decision about which attributes apply to which entities. This can be seen in the simple Datomic schema and entity example in Figure 4.6. This provides applications with a greater flexibility to adapt over time. For example, if the application needs to model a person entity, the decision about whether or not the person is a student or lecturer does not have to be made at the beginning. The application can associate a mixture of student attributes and lecturer attributes to the same person entity. The application can determine if an entity represents a certain abstraction, student or lecturer, by looking at the presence of the suitable attributes.

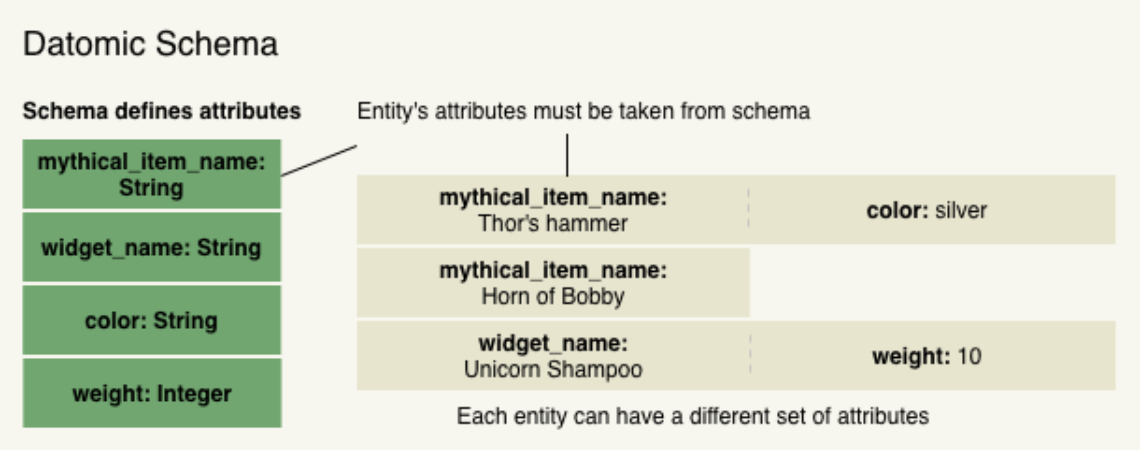


Figure 4.6: Simple example indicating how the Datomic schema defines the characteristics of attributes and not which attributes can be associated with which entities. An entity's attributes must be taken from the schema but each entity can have various attributes. For example in the figure above a mythical item entity may possess any of the attributes defined in the schema such as name or color.[40]

The application requires a single Datomic database to act as the air quality data event store and aggregate database, as Datomic is capable of being both (previously discussed in section 4.5). The database stores the air quality data consumed from the Kafka "Air Quality" event queue.

4.5.3 Database Design

Datoms are used to represent the facts that a Datomic database stores. 'A datom is an addition or retraction of a relation between an entity, an attribute, a value, and a transaction. The set of possible attributes a datom may specify is defined by the database schema.'[39] The Datomic database requires a unique schema determined by the air quality data, the database entities and their attributes. The data model tables for the database and the model associations are explained in the following section.

The Air Quality Event Store Database

The air quality event store database stores the air quality event data consumed from the Kafka event queue after batch processing. All the data stored in the database is modelled as events. It consists of three data models, namely the event model, sensor model and

pollutant model. These three models do associate with each other in the database.

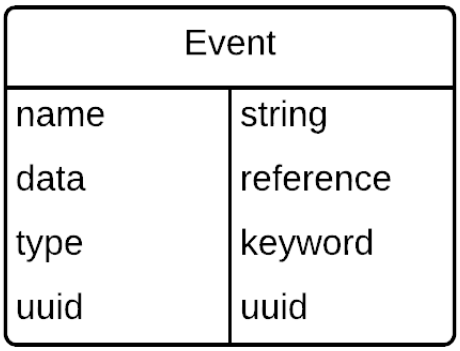


Figure 4.7: Event data model.

Table 4.2: Event data model.

Event Data Model	
Attribute	**Explanation**
name	The name of the event e.g. Received Air Quality Data or Received Humidity Data.
data	Reference to the data associated with the event. In this application it is a reference to the sensor data.
type	The type of event e.g. data or query event.
uuid	The sequential universal unique identifier for the event.

The events model shown in Figure 4.7 includes the name and type attributes for extensibility. The addition of new data streams may result in new event types and names. The value of the *type* attribute is enumerated (e.g. :data or :query) for greater query performance and filtering. The sequential universal unique identifier gives each event a unique identifier and ensures events are ordered in time. Datomic automatically gives each transaction a unique ID so one would think this attribute is unnecessary, however; it ensures the data can be exported to, and used in, another non-Datomic database because it is not dependent on Datomic internals for ordering and unique identifying data. An explanation of each attribute within the event model is provided in Table 4.2.

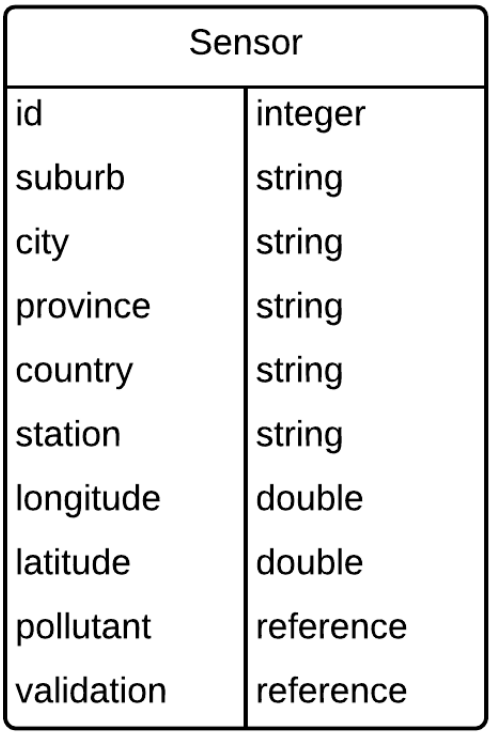


Figure 4.8: Sensor data model.

Table 4.3: Sensor data model.

Sensor Data Model	
Attribute	**Explanation**
id	The ID number of the sensor.
suburb	The suburb in which the sensor is placed.
city	The city in which the sensor is placed.
province	The province where the sensor is placed.
country	The country in which the sensor is placed.
station	The name of the weather/air quality station to which the sensor is associated.
longitude	The longitudinal co-ordinate of the sensor's geographical location.
latitude	The latitudinal co-ordinate of the sensor's geographical location.
pollutant	Reference to the pollutant data measured by the sensor.
validation	Reference to the Clojure Spec validation of the data received from the API.

Although the AQI application is for the City of Cape Town, the sensor model in Figure 4.8 includes certain attributes, such as city, province and country, so that the application may be scaled to a national or international level. The geographical co-ordinate attributes in the sensor model allow for the possibility of placing more than one air quality monitoring sensor in a single suburb. An explanation of each attribute within the sensor model is provided in Table 4.3.

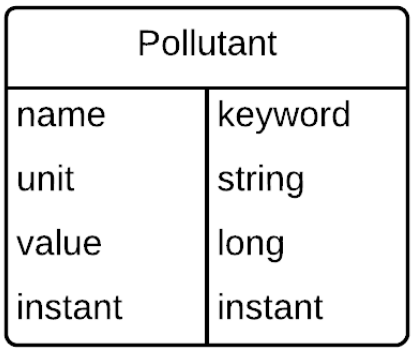


Figure 4.9: Pollutant data model.

Table 4.4: Pollutant data model.

Pollutant Data Model	
Attribute	**Explanation**
name	The name of the pollutant.
unit	The unit of measurement of the pollutant.
value	The pollutant concentration measured by the sensor.
instant	The time instant at which the sensor took the measurement.

The data generator is designed to produce pollution data measured in micrograms per cubic meter [$\mu g/m^3$] (the WHO standard of measurement for pollution concentration, as well as the unit of measurement used by the City of Cape Town for pollutant concentration) but the pollution model in Figure 4.9 includes the unit attribute for extensibility, as pollution concentration may also be measured in other units, such as parts per million [ppm] or parts per billion [ppb]. The pollutant names are represented as a set of enumerated values in the schema (e.g. :name/NO2, :name/SO2 etc.). Keywords are used for the pollutant names for greater query efficiency. An explanation of each attribute within the pollutant model is provided in Table 4.4.

Air Quality Data Model Associations

Figure 4.10 on the following page illustrates how the event, sensor and pollutant data models are associated in the air quality event store database.

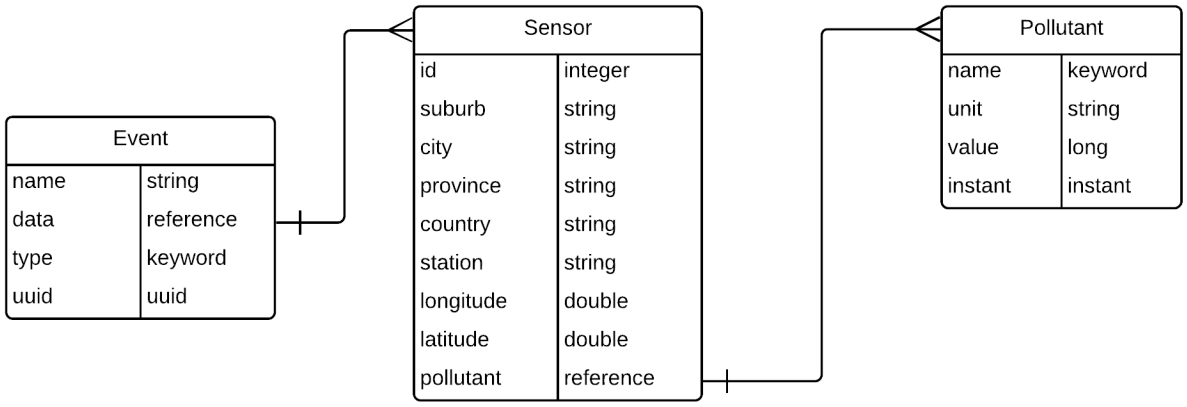


Figure 4.10: Diagram showing the air quality data model associations.

The event model provides information about the data event, such as the name of the event, the type (e.g. query or data event), and the event's data. The sensor model contains information about the sensor's geographical location and it's unique ID number. The pollutant model contains information about the pollutant's name, concentration, unit of measurement and the time instant of concentration measurement.

The cardinality of the data models, as indicated in Figure 4.10, is the following:

Table 4.5: Cardinality of data models.

Models	**Cardinality**
Event - Sensor	one to many
Sensor - Pollutant	one to many

Listing 4.1 is an example of the valid air quality data transacted to the air quality event store. It contains the data attributes of all four data models.

Listing 4.1: Example of valid air quality data transacted to the air quality event store after batch processing. The air quality event data is transacted to the "airQualityData" partition of the database as shown in line 1.

```
1  {:db/id (d/tempid :airQualityData)
2   :event/name "example"
3   :sensor/id       1
4   :sensor/suburb   "Athlone"
5   :sensor/city     "Cape Town"
6   :sensor/province "Western Cape"
```

```
7       :sensor/station  "AQM"
8       :sensor/latitude -33.96526
9       :sensor/longitude 18.50179479999997
10      :sensor/country  "South Africa"
11      :pollutant/name  :name/NO2
12      :pollutant/value 10
13      :pollutant/unit  "ug/m3"
14      :pollutant/instant #inst "2017--21T14:27:52.195-00:00"
15      :event/type :type/data
16      :event/uuid (d/squuid)}
```

Air Quality Event Store Database Partition All the entities created within a database are stored within a partition. Partitions group data together. Entities should be grouped together based on how they will be used. Entities that will be queried together often should be stored in the same partition to increase query performance. Each Datomic database entity is associated with a specific partition via the specified *tempid* prior to transaction to the database. An "airQualityData" partition is created within the air quality event store database to store all the air quality event related data to increase the air quality data query performance. This is shown in Listing 4.1 line 1.[39]

4.6 The Read Model

The read model of an application consists of the components that query the system's state without changing it. In this context it is composed of the elements that allow air quality data to be read from the database and sent to the client. The following sections expand on each element within the read model of the detailed application design. The MVC architectural pattern explained in section 2.3.2 is implemented in the for the web interface.

4.6.1 Application Routes

A "RESTful" web API is used in the application and it requires the three application routes shown in Table 4.6.

Table 4.6: HTTP application routes.

HTTP	Path	Purpose
GET	"/"	Web page of the application.
GET	"/data"	Real-time air quality updates.
POST	"/csv"	JavaScript Object Notation (JSON) format air quality data from the database, based on user date range input, required for the creation of a CSV file on the front-end that is downloaded by the user.

The WebSocket server is used to push real-time AQI updates to the client. An AJAX POST request is used to return the air quality data from the database to the client for the CSV file download, as per the user input on the front-end of the application.

4.6.2 User Interface

The application user interface design consists of a single web page. The user interface is designed to be as informative as possible without cluttering the web page. It is designed to be aesthetically pleasing and minimalistic by not providing too much information (implementation of point 8 of the usability principles listed in the literature review). The layout is designed to be familiar to the user, as the same layout has been used often in web design, thereby allowing the user to intuitively navigate the web application. The user interface is created with HTML, CSS and JavaScript. The basic layout design of the web page can be seen in Figure 4.11.

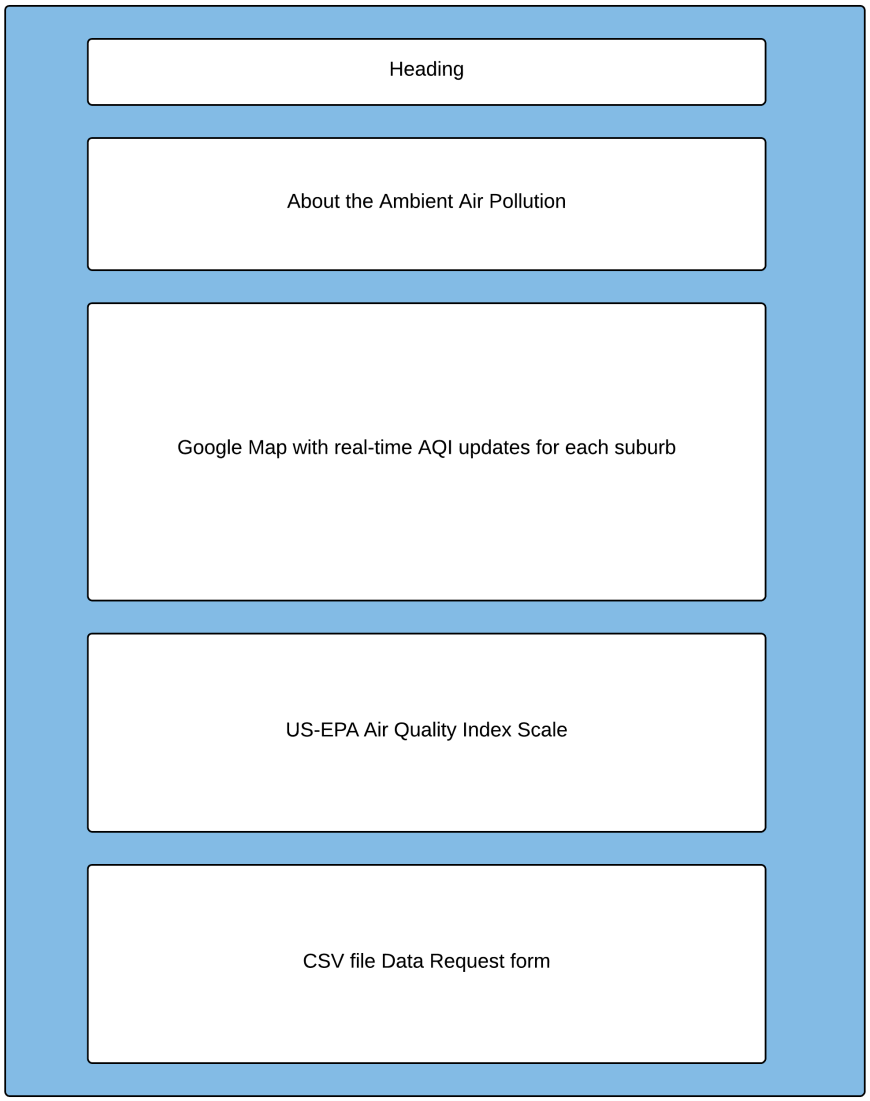


Figure 4.11: Basic layout design of user interface.

The first two sections in the Figure 4.11 layout, the heading and about the ambient air pollution, are presented to the user first. These sections aim to introduce the subject to the user and provide a basic understanding about the topic of ambient air pollution. The following section is designed to display the real-time AQI updates for all the suburbs to the user in a neat and intuitive way through the use of a Google map and map pins for each sensor's geographical location. The fourth section is composed of the USEPA AQI scale which provides information about the health implications and cautionary measures associated with the AQI values displayed in the Google map. The last section is the CSV air quality data file request form. It appears last on the web page as the majority of users, namely ordinary members of the public, will not use this feature of the application: it will be most likely used by those who need air quality data for the City of Cape Town for research purposes.

Chapter 5

Implementation

This chapter provides details on how the detailed application design from the preceding chapter was implemented. The topics covered in the chapter include the development environment in terms of the software and third-party user accounts required, the implementation of the data generator, the data validation with Clojure Spec, Event Sourcing with Apache Kafka, the Datomic database and application queries, application logging, web interface, application life-cycle management, user interface and how to extend the application's dataset in the future.

5.1 Development Environment

This section outlines the necessary components required to setup the development environment to allow for the implementation of the application's detailed design.

5.1.1 Software Required

Table 5.1 contains the software installed to setup the development environment and is based on the requirements of the application's detailed design.

Table 5.1: Software installed for the development environment.

Software	**Usage**
Leiningen 2.7.1	Build automation and dependency management tool for the simple configuration of Clojure (see section 2.3.3) software projects.
Datomic-Free 0.9.5407	The application database (see section 4.5).
Emacs	Text editor used for creating application code.
Apache Kafka 0.8.2.1	Used for creating real-time data pipelines and streaming applications (see section 4.4.2).
Semantic UI 2.2	Easy to use user interface development framework to create a neat, minimalistic, responsive layout that is aesthetically pleasing using simple human-friendly HTML.

The application code was created using the *Emacs* text editor and the *Clojure Interactive Development Environment that Rocks (CIDER)*. A Clojure web development framework was not used.

The software libraries that were used in the implementation of the application is given in Table 5.2.

Table 5.2: Software libraries used in implementation.

Library	**Usage**
Compojure 1.5.2	Application routes.
Cheshire 5.8.0	JSON encoding and decoding.
Clj-time 0.13.0	Date and time library for Clojure.
Clojure 1.9.0-alpha	LISP language for the development of the application.
Clojure Spec-alpha	Specifies the structure of data, validates or destructures it.
Clojure/tools.nrepl 0.2.12	Network REPL that provides a REPL server and client.
Datomic API	Database and queries.
Google Maps API 3.28	Front-end AQI updates based on air quality monitoring sensor locations.
HTML5 WebSockets	Real-time AQI updates.
Http-kit 2.2.0	Web server.
Immutant 2.1.6	Periodic scheduling of tasks.
JQuery 3.2.1	JavaScript library required for user interface development.
Apache Kafka _2.11 0.10.1.0	Java libraries to setup Kafka producer and consumer.
Ring 1.6.0	Application route handler and middleware.
Ring-cors 0.1.10	Allow cross-origin resource sharing.
Stuartsierra/component 0.3.2	Manages life-cycle of application components that share runtime state.
Ptaoensso/timbre 4.10.0	Application logging.

5.1.2 Accounts Required

The implementation of the application did not require the usage of any third party services that involved the creation of a user account.

5.2 Data Generator

A periodic scheduler was created, using the Immutant scheduling library, to generate air quality data approximately every 10 minutes and send it to the Kafka producer to be placed on the Kafka event queue. The data generator's air quality data attributes were modelled around the feed sample of the World Air Quality Index Project (an international air quality initiative). It's API was selected as a guide because of the application's potential for scaling to an international level. The pollutants measured were selected from the WHO's guidelines on ambient air pollution, which is based on pollutants for which the WHO has scientific evidence (PM_{10}, $PM_{2.5}$, O_3, SO_2 and NO_2). The ambient air quality data produced by the data generator is randomized between certain pollutant specific data ranges. The data ranges for each type of pollutant was extracted from archived air quality data released by the City of Cape Town. The City of Cape Town data portal archives contain air quality data for the following suburbs:

1. Athlone
2. Atlantis
3. Bellville
4. Bothasig
5. Central Business District
6. Foreshore
7. Goodwood
8. Khayelitsha
9. Oranjezicht
10. Plattekloof
11. Milnerton

12. Somerset West
13. Table View; and
14. Kraaifontein.

An example of the air quality data produced by the data generator for all the suburbs listed above may be viewed in Appendix A. As previously mentioned in section 4.5, Datomic is ACID consistent throughout, therefore air quality data that does not comply with the schema will not be transacted to the database to ensure the state of the database remains consistent and that no malformed data is stored.

5.3 Clojure Spec Data Validation

The implementation of the Clojure Spec for air quality data validation, previously discussed in section 4.4.1, is shown in Listing 5.1 lines 1-13 and 24-36. The Clojure Spec codes are used to type check each of the air quality data attributes received from the data generator. Each Spec code specifies a predicate (the expected structure of the data value) to test against and type check the air quality data. If the data attribute fails the predicate check it is of the incorrect data type and Clojure Spec will produce a error for that specific code, the opposite occurs if the data attribute passes the predicate check.

Lines 15-21 in Listing 5.1 illustrate how the Spec type checking works using the *sensorID* Spec code defined in line 2. In line 16, the Spec helper method *valid?* is called to verify input data passed to the *sensorID* Spec code and it returns *true*, indicating the data type is valid and conforms to the *sensorID* Spec code. The opposite occurs in line 19 when an invalid data input is passed to the *sensorID* Spec code. When a value does not conform to a Spec code, the *explain* method can be used to report why. In line 17 *explain* has been used for a valid data input and returns *Success!* In line 20 *explain* has been used for an invalid data input and it returns a detailed report as to why the value does not conform. The report includes the input value, the namespaced keyword of the Spec code and its specification.

Specs give meaning to individual attributes (lines 1-13), then assembles them into maps using a set of semantics (lines 24-36), rather than defining attribute specifications within the scope of the map. This allows for the assignment and sharing of semantics at the attribute level across libraries and applications. The Clojure Spec codes that are combined into an entity map *sensorData* in line 25 can be applied to a map of data. An

entity map in Clojure Spec is defined with *keys*. The map spec does not specify the value specification for the attributes, only which attributes are required or optional. In this application the *sensorData* Spec is defined with required attributes defined in lines 1-13. When conformance is checked on a map, the inclusion of the required attributes are checked and then the value of every registered key is checked.[36]

Table 5.3 provides a description of each Clojure Spec code specification.

Listing 5.1: Clojure Spec Codes.

```
1  //Spec codes
2   (s/def ::sensorID int?)
3   (s/def ::areaName (s/and string? #(re-matches #"[a-z\sA-Z]+\S")))
4   (s/def ::cityName (s/and string? #(re-matches #"[a-z\sA-Z]+\S")))
5   (s/def ::provinceName (s/and string? #(re-matches #"[a-z\sA-Z]+\S")))
6   (s/def ::countryName (s/and string? #(re-matches #"[a-z\sA-Z]+\S")))
7   (s/def ::stationName (s/and string? #(re-matches #"[a-z\sA-Z]+\S")))
8   (s/def ::longitude double?)
9   (s/def ::latitude double?)
10  (s/def ::pollutantName keyword?)
11  (s/def ::value int?)
12  (s/def ::unit string?)
13  (s/def ::time inst?)
14 /*-------------------------------------------------------------------------*/
15 //sensorID example
16 (s/valid? ::sensorID 1) //=> true
17 (s/explain ::sensorID 1) //=> Success!
18
19 (s/valid? ::sensorID 2.1) //=> false
20 (s/explain ::sensorID 2.1)
21 // => val: 2.1 fails spec: :data-generator.data-generator/sensorID predicate:
       int?
22
23 /*-------------------------------------------------------------------------*/
24 //Composition of Spec codes for map validation
25  (s/def ::sensorData (s/keys :req [::sensorID
26                                    ::areaName
27                                    ::cityName
28                                    ::provinceName
29                                    ::countryName
30                                    ::stationName
31                                    ::longitude
```

```
                                ::latitude
                                ::pollutantName
                                ::value
                                ::unit
                                ::time]))
```

Table 5.3: Clojure Spec code descriptions.

Keyword	**Description**
::sensorID	Checks the data value is of data type int
::areaName	Checks the data value is of data type string and matches the regex
::cityName	Checks the data value is of data type string and matches the regex
::provinceName	Checks the data value is of data type string and matches the regex
::countryName	Checks the data value is of data type string and matches the regex
::stationName	Checks the data value is of data type string and matches the regex
::longitude	Checks the data value is of data type double
::latitude	Checks the data value is of data type double
::pollutantName	Checks the data value is of data type keyword
::value	Checks the data value is of data type int
::unit	Checks the data value is of data type string
::time	Checks the data value is of data type instant

Listing 5.2: Data validation function with Clojure Spec Codes.

```
(defn validate-sensor-data
   "validate data from sensor"
  [sensorID areaName long lat pollutantName value]
  (let [data_check   {::sensorID    sensorID
                      ::areaName    areaName
                      ::cityName    "Cape Town"
                      ::provinceName "Western Cape"
                      ::countryName "South Africa"
                      ::stationName "AQM"
                      ::longitude   long
                      ::latitude    lat
                      ::pollutantName pollutantName
                      ::value         value
                      ::unit          "ug/m3"
                      ::time          (java.util.Date. (+ (* 7200000)
                         (System/currentTimeMillis)))}
```

```
16
17        data       {:sensorID    sensorID
18                    :areaName    areaName
19                    :cityName    "Cape Town"
20                    :provinceName "Western Cape"
21                    :countryName "South Africa"
22                    :stationName "AQM"
23                    :longitude   long
24                    :latitude    lat
25                    :pollutantName pollutantName
26                    :value         value
27                    :unit          "ug/m3"
28                    :time          (java.util.Date. (+ (* 7200000)
                          (System/currentTimeMillis)))}]
29
30    (if (s/valid? ::sensorData data_check)
31        (timbre/log "SUCCESS - Data validation passed")
32        (timbre/error "FAILURE - Data validation failed: " (s/explain-str
              ::sensorData data_check)))
```

Listing 5.2 provides the complete Clojure data validation function used in the application. In lines 4-15, the air quality data attributes are extracted into a Clojure Spec entity map from the map of data received from the data generator and validated against *sensorData* in line 30. Based on the outcome of the data validation certain actions are executed. If the data passes all the Spec codes a success message is logged with *Timbre*. If the data does not pass all the Spec codes, a failure message is logged with *Timbre* in the application logs along with the Clojure Spec error (like that shown in line 21 of Listing 5.1). The air quality data is then passed to the Kafka producer.

5.4 Event Sourcing with Apache Kafka

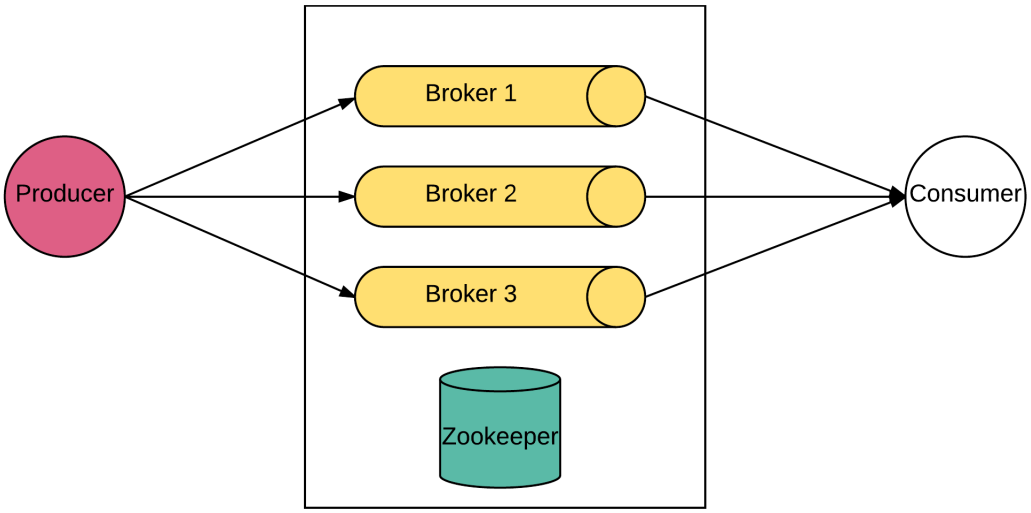


Figure 5.1: Illustration of multi-broker Kafka cluster.

A multi-broker Kafka cluster with a topic replication factor of three, as illustrated in Figure 5.1, was setup to protect the air quality data events against host failures. Kafka uses Zookeeper (an open-source, high performance coordination service for distributed applications) for storing and using configurations across clusters in a distributed manner. Kafka topics are divided into a number of partitions which can be spread out across brokers, as illustrated in Figure 5.2 on the following page, allowing Kafka topics to be parallelized. A number of partitions are held by each broker and each partition can either be a leader or a replica for a topic. All the reads and writes go through the leader and the leader updates the replicas with the new data. If the leader fails, a replica takes over as the new leader. In this application, the topic data is split into three partitions across each of the threebrokers to increase the message throughput.

The air quality data received from the data generator is sent to the Kafka producer after Clojure Spec validation and serialized. The producer then places the data on the Kafka queue as a string data type. Messages are then read by the Kafka consumer and processed according to a batch size of 100 and a batch time-out limit of approximately 1 hour to increase the throughput of the application. During the batch processing the air quality data is deserialized from strings to data maps and transacted to the air quality event store database.

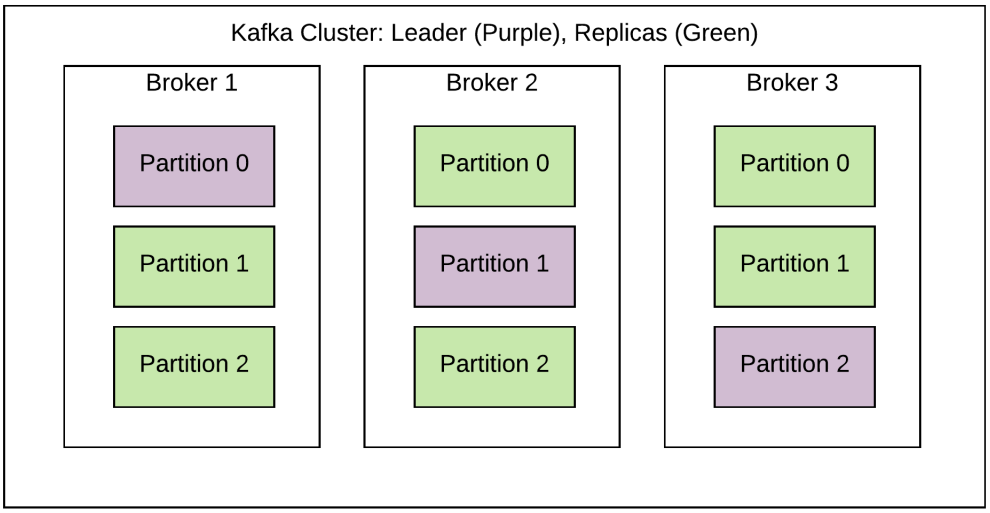


Figure 5.2: Illustration of a Kafka topic divided into a number of partitions spread out across three brokers.

5.5 Datomic Database and AQI Queries

This section describes the datomic database that was used during the implementation of the application and the motivation for doing so, as well as how the AQI queries were implemented for the real-time AQI updates.

5.5.1 Datomic

Datomic is a propriety software with many editions, namely Free, Starter, Pro and Enterprise, each offering different features, services and pricing. The Datomic Free database edition was selected for the development of the application. Datomic Free is recommended for open-source applications because the components are re-distributable. It also allows for the migration of data to the Datomic Pro edition. A simplified overview of the Datomic architecture was shown previously in section 4.5.1.[39]

A single Datomic Free database was created for the air quality event store. The full schema of this database may be viewed in Appendix B.

5.5.2 AQI Queries and Calculations

This section focuses on how the AQI for each pollutant was calculated during the implementation phase of the application. The AQI for each pollutant was calculated using the following steps based on the USEPA AQI method (the following steps are numbered according to the steps to be followed for the implementation of the AQI queries and calculations):

1. **USEPA Concentration Values for Calculating AQI**

   The air quality database is queried for the average pollutant concentration measured over the time periods listed in Table 2.1 for each pollutant and is truncated as indicated in Table 5.4 in accordance with USEPA AQI calculation method:

Table 5.4: Truncation of average pollution concentration values to calculate AQI according to USEPA method.

Pollutant	**Truncation**
O_3 *[ppb]*	Integer
SO_2 *[ppb]*	Integer
NO_2 *[ppb]*	Integer
$PM_{2.5}$ $[\mu g/m^3]$	1 decimal place
PM_{10} $[\mu g/m^3]$	Integer

2. **Concentration Unit Conversion**

   Archived air quality data indicated that the concentration of ambient air pollutants is measured in micrograms per cubic meter $[\mu g/m^3]$ in the City of Cape Town. The WHO also states the threshold concentration for each pollutant in micrograms per cubic meter $[\mu g/m^3]$ in the WHO air pollution and health fact sheet 2016 summarized in Table 2.1, however; some pollutant concentrations are stated in parts per billion $[ppb]$ in the USEPA table of pollutant breakpoints in Figure 2.3 in the literature review, therefore a conversion between the two units of measurement is necessary before calculating the AQI for each pollutant. The conversion between micrograms per cubic meter $[\mu g/m^3]$ and parts per billion $[ppb]$ for the necessary pollutants is given in Table 5.5 (Particulate Matter concentration is stated in micrograms per cubic meter $[\mu g/m^3]$

in the USEPA table of breakpoints and is therefore excluded from this process). The conversion assumes an ambient air pressure of 1 atmosphere and a temperature of 25 degrees Celsius as meteorological data is not included in the air quality dataset of this application.

Table 5.5: Conversion between micrograms per cubic meter $[\mu g/m^3]$ and parts per billion $[ppb]$ for pollutant concentrations stated in parts per billion $[ppb]$ in USEPA table of breakpoints.

Pollutant	**1 $[ppb]$**
Sulphur Dioxide	2.62 $[\mu g/m^3]$
Nitrogen Dioxide	1.88 $[\mu g/m^3]$
Ozone	2.00 $[\mu g/m^3]$

The general equation for converting between micrograms per cubic meter $[\mu g/m^3]$ and parts per billion $[ppb]$ is the following[41]:

$$\text{Concentration } [ppb] = \frac{24.45 \times \text{Concentration } [\mu g/m^3]}{\text{Molecular Weight}}$$

3. **Calculating the AQI**

   Once the pollutant concentration values had been converted to the appropriate unit of measurement they were substituted into the USEPA method mathematical expression for calculating the AQI, as discussed in section 2.1.2, along with the necessary breakpoints, from the USEPA table of breakpoints, to determine the AQI for each pollutant. The code used to implement this calculation for each pollutant is similar (only the mean concentration value truncation and breakpoint ranges differ per pollutant), and therefore only the code for PM_{10} is shown in Listing 5.3.

Listing 5.3: Code used to calculate the AQI for PM_{10} using the USEPA method.

```
1   //Datomic Query for Average Particulate Matter 10 Concentration
        over a period of 24 hours
2   (defn PM10-conc-avg [suburb-name]
3     (let [db (d/db data-conn)]
4       (int   //convert value to integer as per USEPA method
            truncation
5           (or (some->> (d/q '[:find (avg ?value). //Return average
                value of pollutant concentration
6                              :in $ ?suburb ?pollutant ?instant
                                   //Input suburb name, pollutant name,
                                   time period
7                              :where
8                              [?e :sensor/suburb ?suburb]
9                              //Filter entities based on suburb name
10                             [?e :pollutant/name ?pollutant]
11                             //Filter resultant entities based on
                                   pollutant name
12                             [?e :pollutant/instant ?in]
13                             //Obtain timestamps of resultant entities
14                             [ (<= ?instant ?in)]
15                             //Filter resultant entities with
                                   timestamps within time period
16                             [?e :pollutant/value ?value]]
17                             //Obtain concentration values of
                                   filtered entities
18                         //query inputs - database, suburb name,
                               pollutant, averaging time period
19                         (-> db
20                          (d/since (twenty-five-hours-ago)))
21                           suburb-name
22                           :name/PM10
23                           (twenty-four-hours-ago)))
24                0))))
25
26
27
28  //USEPA Method AQI Formula Function
29  (defn AQI-formula [I-Hi I-Lo BP-Hi BP-Lo C-p]
30      (int (+ (* (/ (- I-Hi I-Lo) (- BP-Hi BP-Lo)) C-p) I-Lo)))
31
```

```
32    //Function to determine USEPA AQI breakpoint range
33    (defn PM10-AQI [suburb-name]
34      (let [avg-conc (PM10-conc-avg suburb-name) //average
          concentration value obtained from Datomic Query stored in
          avg-conc
35            key-name (keyword (clojure.string/join (flatten
                [(clojure.string/split suburb-name #" ")"PM10"])))]
36        (list key-name
37              (cond
38              //Evaluate avg-conc value against conditional statements
                  to determine the USEPA AQI breakpoint range and then
                  use the AQI formula to produce the AQI
39                (and (>= avg-conc 0.0) (<= avg-conc 54)) (AQI-formula
                     50 0 54 0.0 avg-conc)
40                (and (>= avg-conc 55) (<= avg-conc 154)) (AQI-formula
                     100 51 154 55 avg-conc)
41                (and (>= avg-conc 155) (<= avg-conc 254)) (AQI-formula
                     150 101 254 155 avg-conc)
42                (and (>= avg-conc 255) (<= avg-conc 354)) (AQI-formula
                     200 151 354 255 avg-conc)
43                (and (>= avg-conc 355) (<= avg-conc 424)) (AQI-formula
                     300 201 424 355 avg-conc)
44                :else                                    0))))
```

5.6 Application Logging

The open-source pure Clojure logging library, Timbre, was used for the application logging. It's an alternative to Java logging, as well as fast, flexible and easy to configure and does not require any Extensible Markup Language (XML). In addition to console logging, the library has been configured with a file appender to save all the application logging to a local file in the root folder of the application.

5.7 Web Interface

This section focuses on the implementation of the elements required to enable the interaction between the user and the software running on a web server, namely the HTTP Kit web server used in the application and the application routes.

5.7.1 HTTP Kit Server

A HTTP kit web server was used in the implementation of this application. It is a scalable, lightweight thread-based server as it uses an event-driven, non-blocking model. The server has been designed to conform with the Clojure web server Ring Spec with asynchronous and WebSocket extensions. Thread-based servers associate each request with a new thread, as opposed to handling each request with a new process. The HTTP Kit default thread setting of four threads was used in the web server implementation.

5.7.2 Application Routes

The application routes, given in Listing 5.4, were created using the Compojure routing library, which return Ring handler functions. Handlers are just functions that accept a request map and return a response map. The application only uses synchronous handlers.

The Cheshire library encoding supports Clojure data structures, Java classes and allows for custom encoding. In the application it is used to JSON encode the application response maps that are sent to the client.

Listing 5.4: Application routes.

```
1 (defroutes routes
2   (GET "/data" [] ws-handler)
3   (GET "/" [] (redirect "/dashboard.html"))
4   (POST "/csv" request (data-request-handler request))
5   (route/resources "/")
6   (route/not-found "Page not found"))
```

Prior to being processed, web requests or responses go through a series of middleware: special functions used to perform additional validation or enrich the request or response. Middleware wraps all or part of the application to modify a request or response, and they add additional functionality to route handlers. Table 5.6 provides the Ring middleware that is added to the routes in the application implementation.

Table 5.6: Ring middleware.

Ring Middleware	**Purpose**
wrap-session	Reads in the current HTTP session map, and adds it to the *:session* key on the request.
wrap-flash	If a *:flash* key is set on the response by the handler, a *:flash* key with the same value will be set on the next request that shares the same session.
wrap-cookies	Parses the cookies in the request map, then associate the resulting map to the *:cookies* key on the request.
wrap-multipart-params	Parse a map of multipart parameters from the request.
wrap-params	Parse url encoded parameters from the query string and form body.
wrap-nested-params	Converts a flat map of parameters into a nested map.
wrap-keyword-params	Converts the string-keyed *:params* map to one with keyword keys before forwarding the request to the given handler.
wrap-reload	Reload namespaces of modified files before the request is passed to the supplied handler.
wrap-cors	For easy cross-origin resource sharing.

WebSocket Server and Google Map API

The Google map API was used to display the geographical locations of the air quality monitoring sensors spread throughout the City of Cape Town as map location pins. Google map information windows were used to display the WebSocket server real-time updates of pollution levels measured at the specific location by the sensor. All the sensor pin locations are visible on the Google map when the web page is loaded. When the user clicks on a map pin, the map is set to zoom in on that location and an information window with the AQI data for that area is displayed in the browser. The WebSocket server is set to push AQI updates to the client every 10 minutes. The frequency of the AQI updates is customizable and may be extended, or shortened. An interval of 10 minutes between

updates was chosen because the data generator periodically simulates new air quality data approximately every 10 minutes, therefore new AQI values may only be calculated once new air quality data is received.

AJAX POST Request

An AJAX POST request was used to seamlessly request the air quality data required for the air quality data CSV file, based on the user date range input, without having to refresh the web page.

5.8 Application Life-cycle Management with Component Library

The *Component* library 'is a Clojure framework used to manage the life-cycle and dependencies of software components which have runtime state. A component is a collection of functions, or procedures, that share the same runtime state' and is similar to that of an object in object-oriented programming. For example, a component may be a web server: 'functions to handle different routes sharing all the runtime state of the web application, or database access: query and insert functions sharing the same database connection.' Components are composed into systems. A system is a component that knows how to start and stop other components and is responsible for injecting dependencies into components that require them. Furthermore the start and stop methods of each component may be defined in an idempotent manner and therefore will only have an effect if the component has not already been started or stopped, allowing for more robust system state management. [42]

The *Component* library is used to manage relationships between the different application components that have to be started and stopped in a particular order within the application. It allows for the mutable state within the application to be grouped together into a single "system" object that can easily be inspected and allows for a logical way to set up and break down all the application state, allowing for a simple way to start and stop the application without having to restart the JVM.

The components created in the application were the web server, Kafka producer periodic scheduler and Kafka consumer, because these components needed to be started

and stopped in a specific order, and together they formed the "system object" of the application that is used to start and stop the application.

5.9 User Interface

This section discusses the implementation of the user interface: the implementation of the application's web page. It focuses on the front-end framework used, the application's site navigation and the layout of the web page, as well as the motivation behind certain aspects of the layout.

5.9.1 Semantic User Interface

The Semantic UI front-end development framework was used to design the web page of the application. The development framework was used to create a neat, minimalistic, responsive layout that is aesthetically pleasing using simple human-friendly HTML. The simplicity of the HTML and CSS in the framework allow for easy modification of the user interface in the future.

5.9.2 Site Navigation

The front-end consisted of a single web page with all the relevant air quality data and information to make it quick and easy for the user to find the information most relevant to them, in addition to providing the user with a basic understanding of the air quality information.

5.9.3 Web Page Layout

Unfortunately, due to the web page length, the entire application web page could not be captured in a single figure. The layout of the application's web page is shown in the Figures 5.3 - 5.7. The figures have been ordered in the same order as the information appears to the user in the browser.

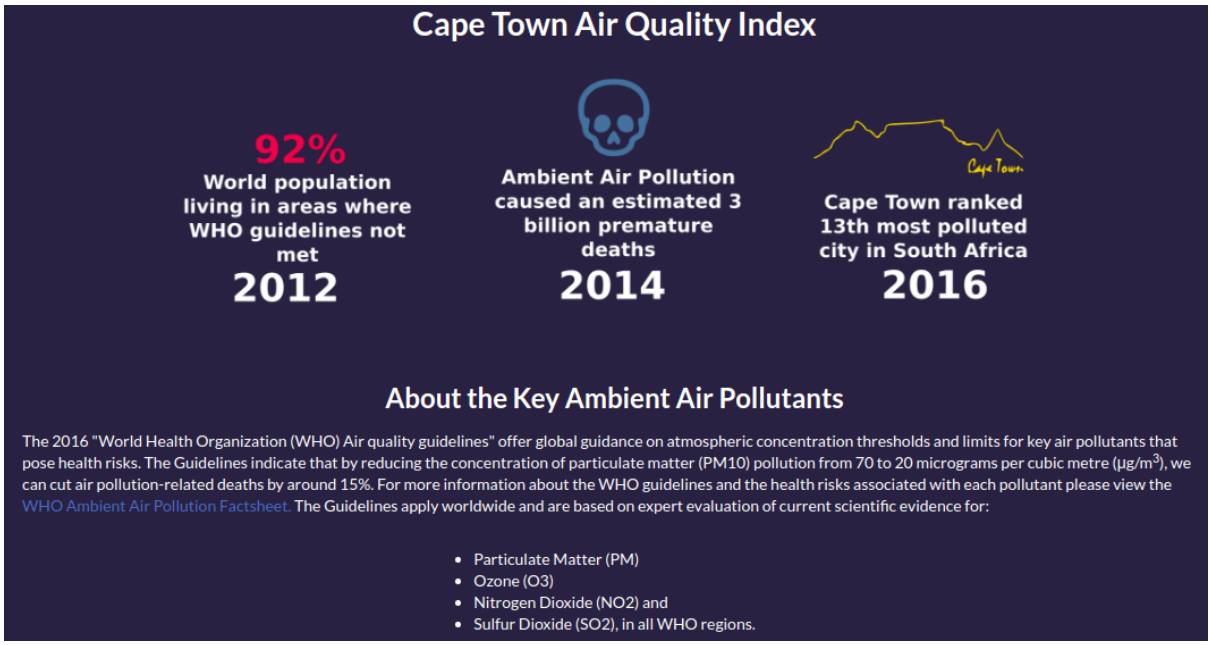


Figure 5.3: Air quality information on web page of application.

Figure 5.3 shows the first piece of information presented to the user. The information presented aims to provide the user with some basic context with regard to ambient air pollutants and the effects of ambient air pollution on human health which has been backed by scientific evidence. It includes the Cape Town Air Quality Index web page heading, statistics about the devastating effects of air pollution on human health to highlight the significance of the problem, information from the WHO 2016 guidelines, and it's key pollutants, and a link to the *WHO Ambient Air Pollution Factsheet 2016* (should the user wish to find out more about the WHO's key pollutants).

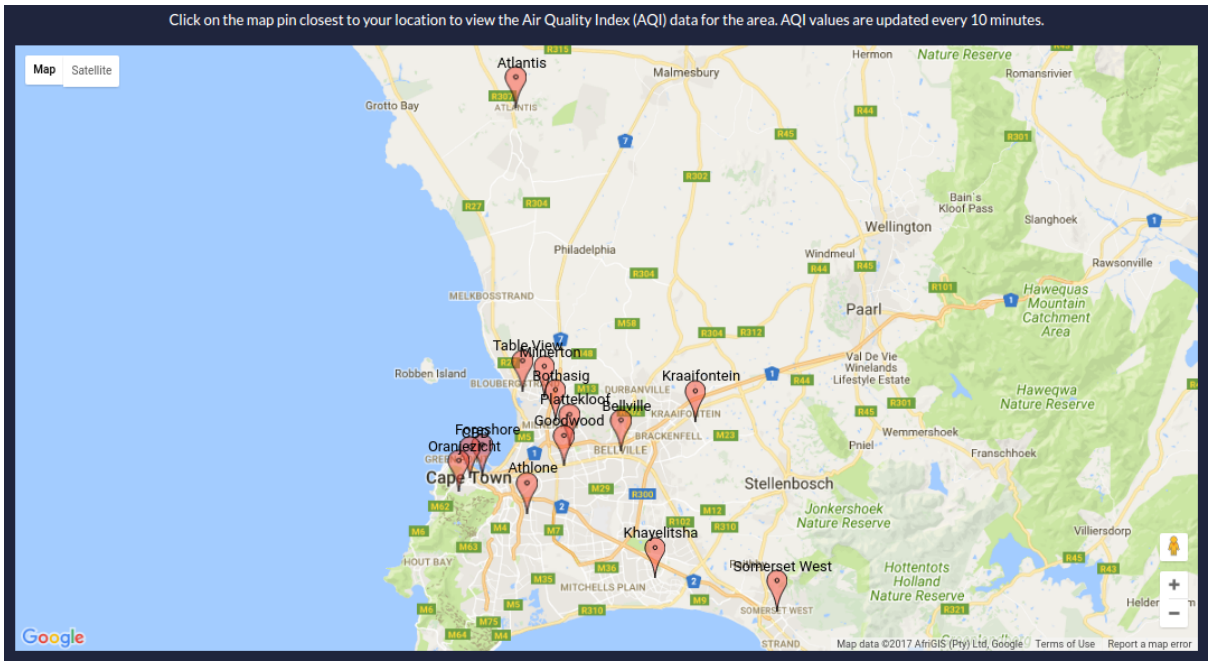


Figure 5.4: Google map showing the geographical locations of the air quality sensors across the City of Cape Town on web page of application.

Figure 5.4 shows the Google map of the geographical locations of the air quality sensors across the City of Cape Town. The real-time AQI update feature of the application is presented through the use of the Google map, as it is a neat and clean way to present all the AQI information and it is easy for the user to intuitively navigate. The HTML text above the map instructs the user to click on the map pin closest to their location to view the AQI data for that location and informs the user that the AQI values are updated every 10 minutes as mentioned in section 5.7.2.

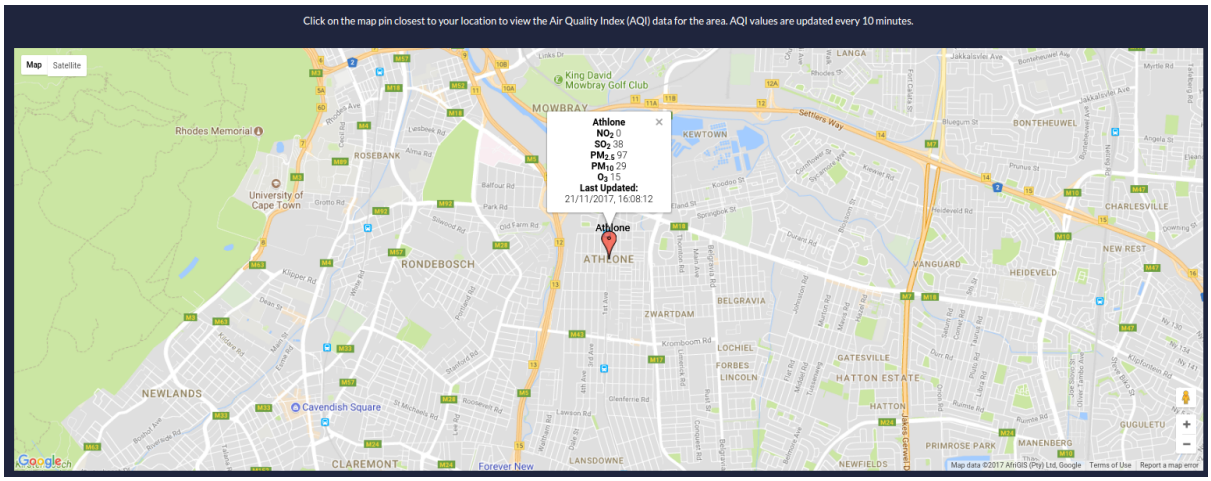
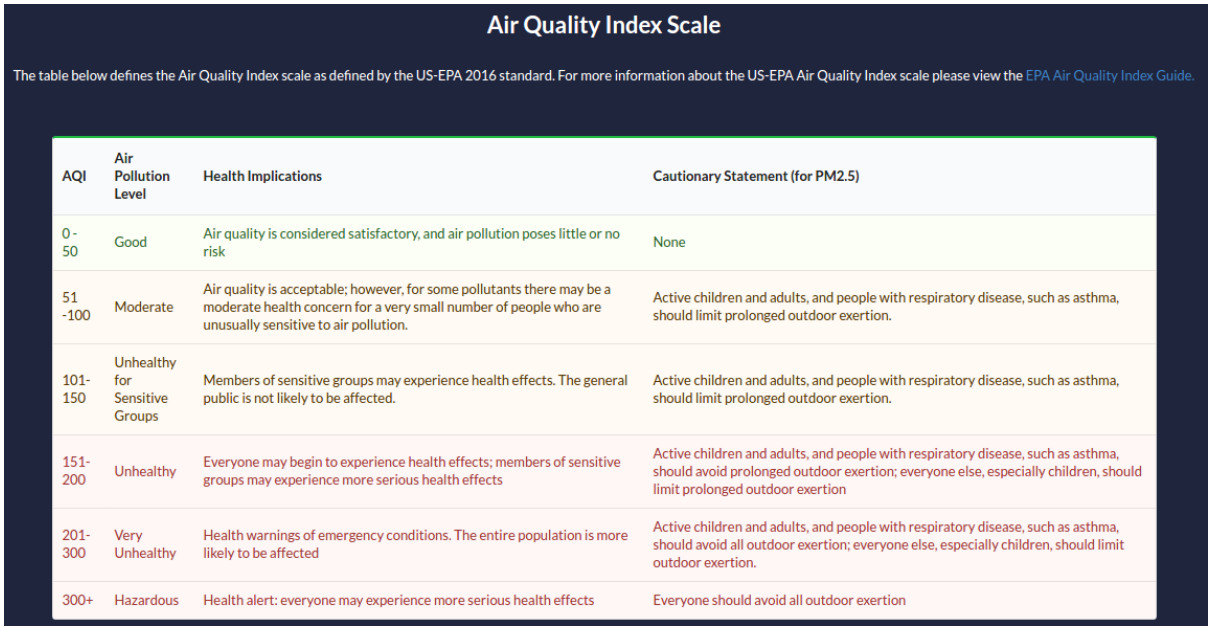


Figure 5.5: Google map information window of the Athlone map pin showing the AQI information for the Athlone area.

Figure 5.5 shows what happens when a user clicks on a Google Map pin. The map zooms in on the location and an information window with the AQI data for that location is displayed to the user.



Air Quality Index Scale

The table below defines the Air Quality Index scale as defined by the US-EPA 2016 standard. For more information about the US-EPA Air Quality Index scale please view the EPA Air Quality Index Guide.

AQI	Air Pollution Level	Health Implications	Cautionary Statement (for PM2.5)
0 - 50	Good	Air quality is considered satisfactory, and air pollution poses little or no risk	None
51 -100	Moderate	Air quality is acceptable; however, for some pollutants there may be a moderate health concern for a very small number of people who are unusually sensitive to air pollution.	Active children and adults, and people with respiratory disease, such as asthma, should limit prolonged outdoor exertion.
101-150	Unhealthy for Sensitive Groups	Members of sensitive groups may experience health effects. The general public is not likely to be affected.	Active children and adults, and people with respiratory disease, such as asthma, should limit prolonged outdoor exertion.
151-200	Unhealthy	Everyone may begin to experience health effects; members of sensitive groups may experience more serious health effects	Active children and adults, and people with respiratory disease, such as asthma, should avoid prolonged outdoor exertion; everyone else, especially children, should limit prolonged outdoor exertion
201-300	Very Unhealthy	Health warnings of emergency conditions. The entire population is more likely to be affected	Active children and adults, and people with respiratory disease, such as asthma, should avoid all outdoor exertion; everyone else, especially children, should limit outdoor exertion.
300+	Hazardous	Health alert: everyone may experience more serious health effects	Everyone should avoid all outdoor exertion

Figure 5.6: AQI scale table with precautionary measures on web page of application.

Figure 5.6 shows the AQI scale based on USEPA standards so that they user may compare the AQI values from the Google map information windows to the table to determine the

health implications, and take the necessary precautionary measures. A link to more information about the *EPA Air Quality Index Guide* is also provided.

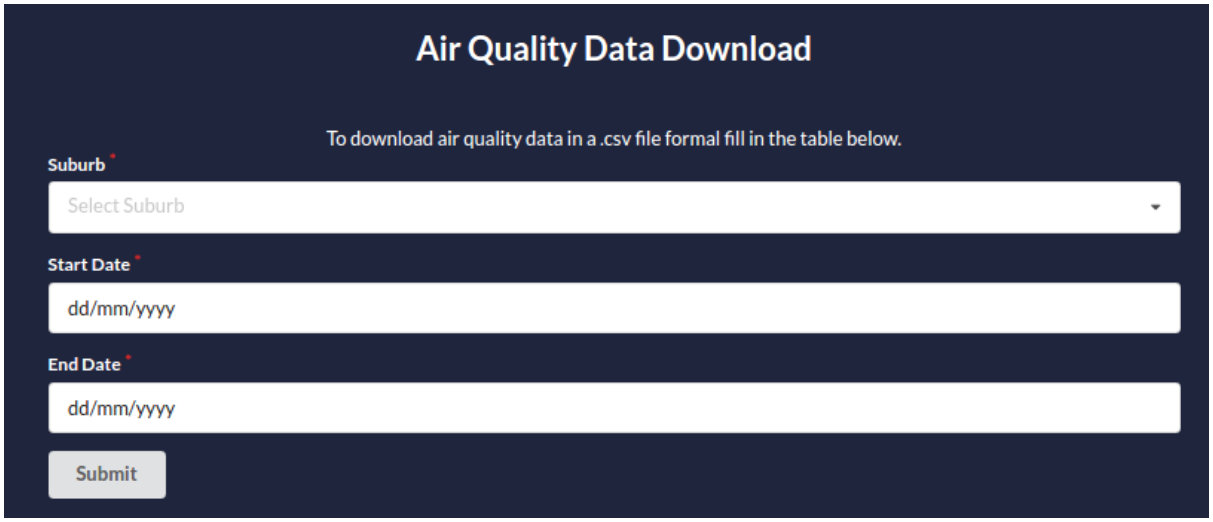


Figure 5.7: Air quality information CSV file download request form on web page of application.

Figure 5.7 shows the last part of the application's web page: the CSV air quality data download form. The form is presented to the user last because the majority of users will not need to use this feature of the application: most users will just want to view the AQI for their area, determine the implications and then leave the web page, only users who need to analyze air quality data for the City of Cape Town will need to download a CSV file of the data.

5.10 Extending the Dataset

This section focuses on how to extend the air quality dataset by either including another pollutant or adding a meteorological data attribute to the dataset, such as raw ambient air temperature readings from a sensor network.

5.10.1 Adding Another Pollutant to the Dataset

It is relatively simple to include another pollutant to the dataset, such as carbon dioxide, as it would fit the existing pollutant data model. The data validation, database schema

and batch processing code already caters for the pollutant data model. Only changes to the AQI queries and the Google map JavaScript on the web page would be necessary. An AQI query for carbon dioxide will have to be created, the CSV data request query will have to be modified to include a new pollutant in the dataset and the JavaScript in the user interface of the application will need to be modified to display the new pollutant's real-time AQI data.

5.10.2 Adding a Meteorological Attribute to the Dataset

The process of adding meteorological data to the current dataset is more complicated. The existing Clojure Spec data validation, database schema, batch processing code and database queries do not cater for meteorological data and will require modification. The process can be broken down into the following steps using the example of ambient air temperature and the assumption that the temperature and pollutant data will be included in the same dataset received from the data API:

1. **Clojure Spec Data Validation**

   The Clojure Spec Data Validation code will have to be modified to include the type checking of the temperature data. The following Spec code may be added to Listing 5.1 in the previous chapter:

   Listing 5.5: Clojure Spec code modification for the inclusion of raw ambient air temperature data.

   ```
   1  (s/def ::temperature double?) //Measured temperature by
          sensor
   2  (s/def ::temperatureUnit string?) //Unit of temperature
          measurement
   ```

   Listing 5.5 type checks the raw ambient air temperature data to ensure the temperature value is of type *double* and the unit of measurement of the ambient air temperature is of type *string*.

2. **Air Quality Event Store Database Schema**

   The schema of the air quality event store database will need to be modified. A temperature entity will have to be created in the database, the data model is shown in Figure 5.8, and a temperature data attribute will have to be added to the air quality event store Datomic schema as shown in Listing 5.6.

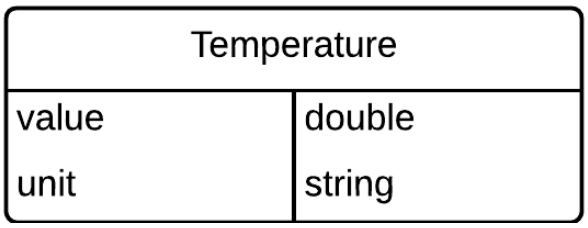


Figure 5.8: Temperature data model.

Listing 5.6: Temperature attributes in Air Quality Data database schema.

```
1            {:db/id #db/id[:db.part/db]
2             :db/ident :sensor/temperature
3             :db/valueType :db.type/ref
4             :db/cardinality :db.cardinality/one
5             :db/doc "Reference to temperature entity"
6             :db.install/_attribute :db.part/db}
7
8            {:db/id #db/id[:db.part/db]
9             :db/ident :temperature/unit
10            :db/valueType :db.type/string
11            :db/cardinality :db.cardinality/one
12            :db/doc "The unit of measurement of the temperature value"
13            :db.install/_attribute :db.part/db}
14
15           {:db/id #db/id[:db.part/db]
16            :db/ident :temperature/value
17            :db/valueType :db.type/double
18            :db/cardinality :db.cardinality/one
19            :db/doc "The measured value of the temperature"
20            :db.install/_attribute :db.part/db}
```

The data models would then be associated as illustrated in Figure 5.9.

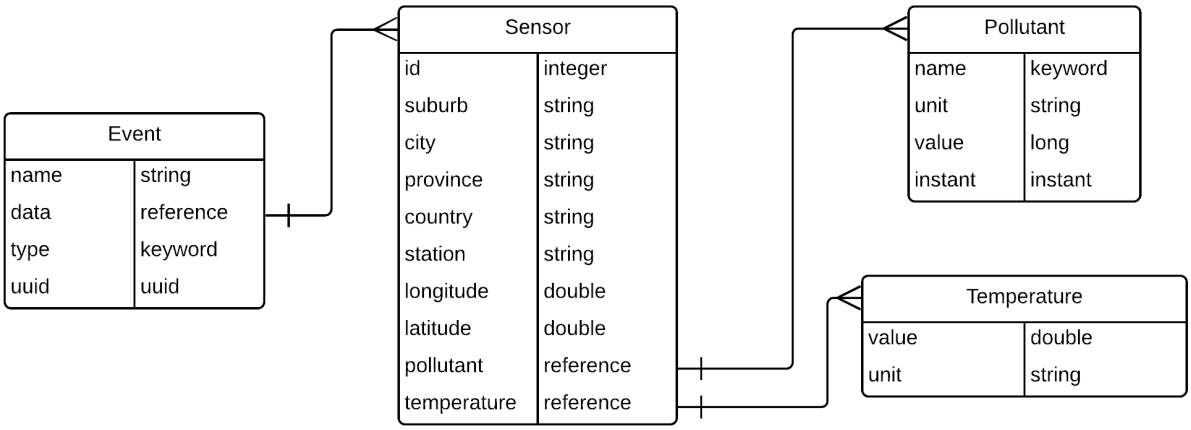


Figure 5.9: Air Quality Event Store database data model associations with the addition of the new temperature data model.

3. **Batch Processing**

   The event batch processing code will have to be modified to include the new temperature data attribute when transacting the air quality data to the event store as shown in Listing 5.7 lines 18-19.

Listing 5.7: Example of air quality data transacted to the air quality event store.

```
1   {:db/id (d/tempid :airQualityData)
2    :event/name "test"
3    :sensor/id       1
4    :sensor/suburb   "Athlone"
5    :sensor/city     "Cape Town"
6    :sensor/province "Western Cape"
7    :sensor/station  "AQM"
8    :sensor/latitude -33.96526
9    :sensor/longitude 18.50179479999997
10   :sensor/country  "South Africa"
11   :pollutant/name  :name/NO2
12   :pollutant/value 10
13   :pollutant/unit  "ug/m3"
14   :pollutant/instant #inst "2017-11-21T14:27:52.195-00:00"
15   :event/type :type/data
16   :event/uuid (d/squuid)
17   :temperature/value 24.5
18   :temperature/unit "Celsius"}
```

4. **AQI Queries and Calculations**

The AQI queries and calculations will have to be modified accordingly to include the temperature data in the calculation of AQI results for each suburb.

5. **User Interface**

   The HTML and JavaScript of the application web page may have to be modified if displaying temperature data in the browser is necessary.

Chapter 6

Acceptance Testing and Analysis of Results

This chapter provides details about the testing tools used, how each acceptance test, outlined previously in section 3.4, was conducted and the analysis and interpretation of the results of each test. The acceptance testing strategy is outline in Table 6.1. It comprises of the tests, previously listed in Table 1.1 in the terms of reference, that were performed on the application and the type of each test. The results of each test are discussed in the following sections. All the tests were performed using a laptop computer with the specifications previously given section 4.1.

Table 6.1: Acceptance tests performed on application.

Test Number	Type of Testing	Requirement Tested	Description
1	Robustness	Forms part of R3: the implementation of EDA. R7 robustness testing.	Testing the fault tolerance of the Kafka multi-broker cluster for robustness.
2	Robustness	Forms part of R3: the data validation of data received from the data generator/API. R7 robustness testing.	Testing that the Clojure Spec data validation works as expected for robustness and reliability.
3	Performance	R7 performance testing.	Profiling high level functions in the read model and eliminating potential bottlenecks that will affect the user experience.
4	Performance & Robustness	R7 performance and robustness testing.	Testing for memory leaks within the application for robustness.
5	Functionality	R7 functionality testing of R4.	Ensuring the real-time AQI updates displayed on the Google map in the user interface functions as expected.
6	Functionality	R7 functionality testing of R6.	Ensuring the air quality data CSV file download feature in the user interface works as expected.
7	Performance	R7 performance testing.	Testing the page response time of the application's web page for performance analysis.
8	Performance	R7 performance testing.	Load and stress testing the application for performance analysis.
9	User Experience	User experience testing R5.	Testing the user interface with volunteers using the application and filling out a survey rating their user experience.

Testing Tools

In this project a range of different software tools and libraries were leveraged to perform thorough testing of different aspects of the application code that was developed. These tools and libraries are listed in Table 6.2 with an explanation, in the second column, showing what aspect of the software was tested using these resources.

Table 6.2: Software used for testing

Software	**Usage**
Chrome Developer Tools	Page load time testing.
Gatling-2.2.5	Load and stress testing of application.
JDK (Java SE Development Kit) 1.8.0_131	Java Mission Control and Flight Recorder used for memory leak detection.
Ptaoussanis/tufte software library	Application profiling for bottleneck detection.

Chrome Developer tools are a set of web debugging and authoring tools built into Google Chrome. It gives web developers access to the browser internals and their web application. Gatling is an open-source load and performance testing tool for web applications. The Java SE Development Kit (JDK) is a collection of programming tools. It includes a JVM and other resources (e.g. tools for developing, debugging and monitoring Java applications) to complete the development of an application. As mentioned in section 2.3.3 previously, Clojure is a JVM hosted LISP language, therefore the JDK tools may also be used to assist with the development of Clojure web applications. The Ptaoussanis/tufte software library is designed for basic profiling and performance monitoring for Clojure or ClojureScript.

6.1 Acceptance Test 1: Fault Tolerance of Kafka Cluster

The fault tolerance of the Kafka cluster was tested to ensure that in the event that the leader node of a partition in the cluster failed, one of the replicas from the other two brokers would become the new leader and the normal operation of the application would not be affected.

6.1.1 Experimental Setup

The application was hosted locally in the development environment. While the application and Kafka cluster is in operation, the command in Listing 6.1 is run in the terminal to describe the Kafka *airQuality* topic. The topic description includes the replication factor of the topic, the topic partition count, the number of brokers running in the cluster and which broker in the cluster is the leader.

Listing 6.1: Describe Kafka topic terminal command.

```
1  $ bin/kafka-topics.sh --describe --zookeeper localhost:2800 --topic
       airQuality
```

The first line of the output in Figures 6.1 and 6.3 provide a summary of all three partitions, discussed in section 5.4, and each of the following lines provide information about each partition. The *leader* is the node 'responsible for the reads and writes of the partition. *Replicas* are the list of nodes that replicate the log for a partition,' irrespective of whether they are the leader, or currently alive. *Isr* indicates 'the subset of the *replicas* node list that is currently alive and caught-up to the leader' i.e. it gives the list of "in-sync" replicas.[38]

Once it was established from the topic description that all three Kafka brokers in the cluster are alive and running. The Java process ID of the lead broker of partition 0 (broker 0) in the Kafka cluster was identified and killed, and the description of the topic was examined thereafter by executing the command in Listing 6.1 once more.

6.1.2 Expected Results

The description of the Kafka topic *airQuality* after the termination of partition 0's lead broker should indicate that the cluster is now composed of two brokers which are still alive, the new leader broker for partition 0 is broker 1, and that the cluster is able to carry out operations as normal in the event of a broker failure.

6.1.3 Results of Testing

Figure 6.1 below describes the Kafka topic. It topic has been divided into three partitions which has been replicated by a factor of three across the three brokers in the Kafka cluster. Broker 0 is the leader of partition 0 in the cluster as illustrated in Figure 6.2. To test the fault tolerance of the cluster the Java process of broker 0 is killed, and the Kafka topic description is examined thereafter in Figure 6.3. Figure 6.3 shows that broker 1 is now the leader node of both partition 0 and partition 1 in the cluster, and that there are two brokers in the cluster that are still alive (as shown by the *isr* of each partition in the Figure 6.3 topic description and illustrated by Figure 6.4). The aforementioned observations indicate that the Kafka cluster is protected against fault tolerance.

```
→  air_quality_app  git:(master) ✗ ~/kafka/bin/kafka-topics.sh --describe --zookeeper local
host:2800 --topic airQuality
Topic:airQuality        PartitionCount:3        ReplicationFactor:3     Configs:
        Topic: airQuality       Partition: 0    Leader: 0       Replicas: 0,1,2 Isr: 0,1,2
        Topic: airQuality       Partition: 1    Leader: 1       Replicas: 1,2,0 Isr: 1,2,0
        Topic: airQuality       Partition: 2    Leader: 2       Replicas: 2,0,1 Isr: 2,0,1
```

Figure 6.1: Description of Kafka topic airQuality which has been divided into three partitions across a multi-node cluster consisting of three brokers. Broker 0 is leader for partition 0, broker 1 is the leader for partition 1 and broker 2 is the leader for partition 2 in the cluster. All three brokers in the cluster are alive and caught-up with the leader as indicated by the *isr* in the topic description.

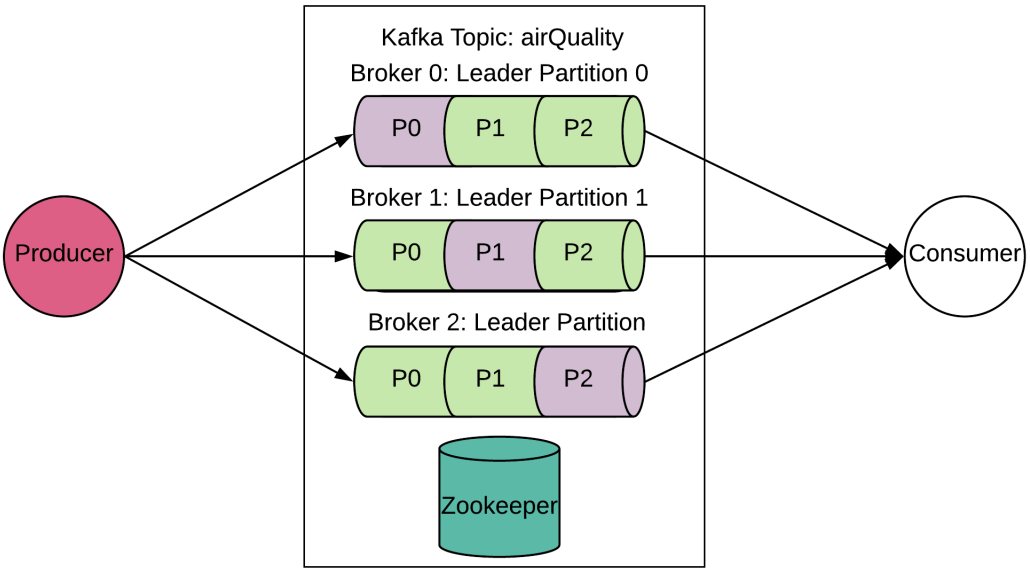


Figure 6.2: Illustration of Kafka topic airQuality which has been divided into three partitions across a multi-node cluster consisting of three brokers. Broker 0 is leader for partition 0 (P0), broker 1 is the leader for partition 1 (P1) and broker 2 is the leader for partition 2 (P2) in the cluster. All three brokers in the cluster are alive and caught-up with the leader.

```
→  air_quality_app  git:(master) ✗ ~/kafka/bin/kafka-topics.sh --describe --zookeeper local
host:2800 --topic airQuality
Topic:airQuality        PartitionCount:3        ReplicationFactor:3     Configs:
        Topic: airQuality       Partition: 0    Leader: 1       Replicas: 0,1,2 Isr: 1,2
        Topic: airQuality       Partition: 1    Leader: 1       Replicas: 1,2,0 Isr: 1,2
        Topic: airQuality       Partition: 2    Leader: 2       Replicas: 2,0,1 Isr: 2,1
```

Figure 6.3: Description of Kafka topic airQuality which has been divided into three partitions across a multi-node cluster consisting of three brokers. Broker 1 is the leader for both partition 0 and partition 1, and broker 2 is the leader for partition 2. Broker 0 is no longer alive because it was manually terminated, whilst brokers 1 and 2 are still alive and caught-up with the leader as indicated by the *isr* in the topic description.

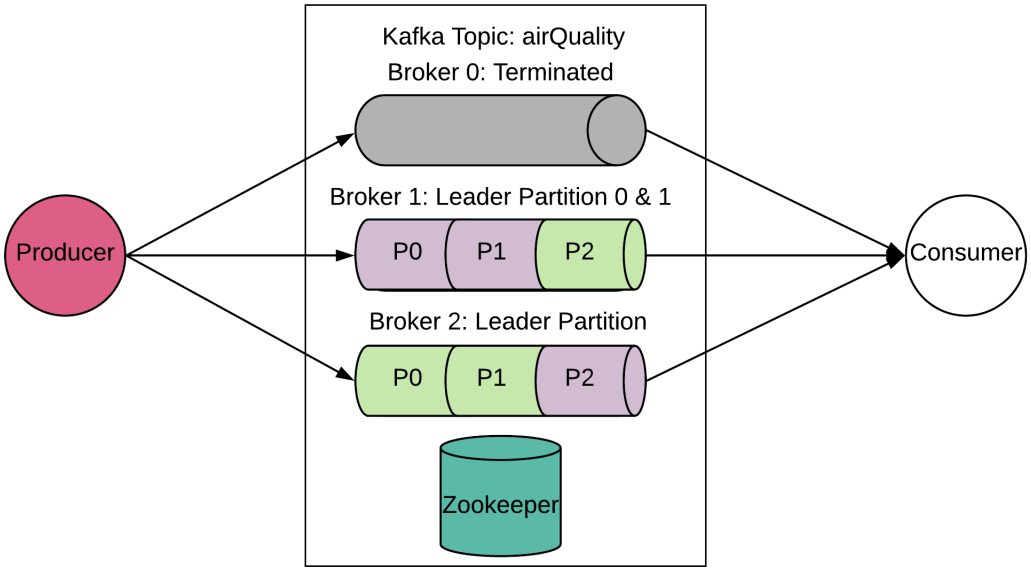


Figure 6.4: Illustration of Kafka topic airQuality which has been divided into three partitions across a multi-node cluster consisting of three brokers. Broker 1 is the leader for both partition 0 (P0) and partition 1 (P1), and broker 2 is the leader for partition 2 (P2) in the cluster. Broker 0 is no longer alive after it was manually terminated, whilst brokers 1 and 2 are still alive and caught-up with the leader.

6.2 Acceptance Test 2: Clojure Spec Data Validation

This test aimed to verify that the Clojure Spec data validation performs as expected for the validation of the data received from the data generator. It serves to test that the data types of the air quality data received is checked for validity, and that data attributes with data types that do not match the database schema are detected and logged to assist with debugging at a later stage.

6.2.1 Experimental Setup

The Clojure Spec validation code was run on datasets containing valid air quality data (Listing 6.2) and then the experiment was repeated with datasets containing invalid air quality data (Listing 6.3 line 9). Listing 6.3 is just one variation of the invalid datasets used to test the Clojure Spec validation process, datasets containing incorrect data types for each air quality data attribute were used during testing to ensure the validation process worked as intended for all the data attributes.

6.2.2 Expected Results

If all the air quality data attributes passed and was valid, *"SUCCESS - Data validation passed"* should be logged in the console. If not all the data attributes were valid and some of the data did not pass the Clojure Spec codes, then *"FAILURE - Data validation failed: Clojure Spec error message"* should be logged in the console as previously discussed in section 5.3.

6.2.3 Results of Testing

Figure 6.5 shows the resultant data map of valid data after Clojure Spec validation (for the same data in Listing 6.2) and with the success message in the console. Figure 6.6 shows the resultant data map of invalid data (for the same data in Listing 6.3). The failure message and the Clojure Spec error message is logged in the console. The Clojure Spec data validation performs as expected.

Listing 6.2: Example of valid dataset of air quality data used during Clojure Spec data validation testing.

```
1   {:sensorID 1,
2    :areaName "Athlone",
3    :stationName "AQM",
4    :cityName "Cape Town",
5    :provinceName "Western Cape",
6    :countryName "South Africa",
7    :latitude -33.96526,
8    :longitude 18.50179479999997,
9    :pollutantName :name/NO2,
10   :unit "ug/m3",
11   :value 10,
12   :time inst "2017-11-20T16:47:56.082-00:00" }
```

Listing 6.3: Example of invalid dataset of air quality data used during Clojure Spec data validation testing.

```
1   {:sensorID 1,
2    :areaName "Athlone",
3    :stationName "AQM",
4    :cityName "Cape Town",
5    :provinceName "Western Cape",
```

```
    :countryName "South Africa",
    :latitude -33.96526,
    :longitude 18.50179479999997,
    :pollutantName "NO2", //Invalid datatype string, Clojure Spec predicate
       accepts only keyword for this attribute
    :unit "ug/m3",
    :value 10,
    :time inst "2017-11-20T16:47:56.082-00:00" }
```

```
air-quality-app.core $> {:unit "ug/m3", :value 10, :time inst
   "2017-11-20T16:47:56.082-00:00", :longitude 18.50179479999997,
   :countryName "South Africa", :areaName "Athlone", :sensorID 1,
   :stationName "AQM", :cityName "Cape Town", :latitude -33.96526,
   :provinceName "Western Cape", :pollutantName :name/NO2}
air-quality-app.core $>
   17-11-21 14:22:26 subha-Aspire-5750G SUCCESS: Data validation passed
```

Figure 6.5: Map of valid data after Clojure Spec validation with success message in console log.

```
air-quality-app.core $> 17-09-09 15:28:26 subha-Aspire-5750G ERROR
   [data-generator.data-generator:86] - FAILURE - Data validation
   failed: In: [:data-generator.data-generator/pollutantName] val: "NO2"
   fails spec: :data-generator.data-generator/pollutantName at:
   [:data-generator.data-generator/pollutantName] predicate: keyword?
air-quality-app.core $> {:unit "ug/m3", :value 10, :time inst
   "2017-11-20T17:28:26.286-00:00", :longitude 18.50179479999997,
   :countryName "South Africa", :areaName "Athlone", :sensorID 1,
   :stationName "AQM", :cityName "Cape Town", :latitude -33.96526,
   :provinceName "Western Cape", :pollutantName "NO2"}
```

Figure 6.6: Logging of Clojure Spec data validation error message and map of data after validation

6.3 Acceptance Test 3: Bottleneck Identification

The Tufte library was used to provide a basic profile of the read model and identify bottlenecks in the code that would result in a poor user experience. More information on application profiling is provided in section 2.4.3 of the literature review.

6.3.1 Experimental Setup

The components of the application within the read model (blue box) in the figure below were profiled for possible bottlenecks.

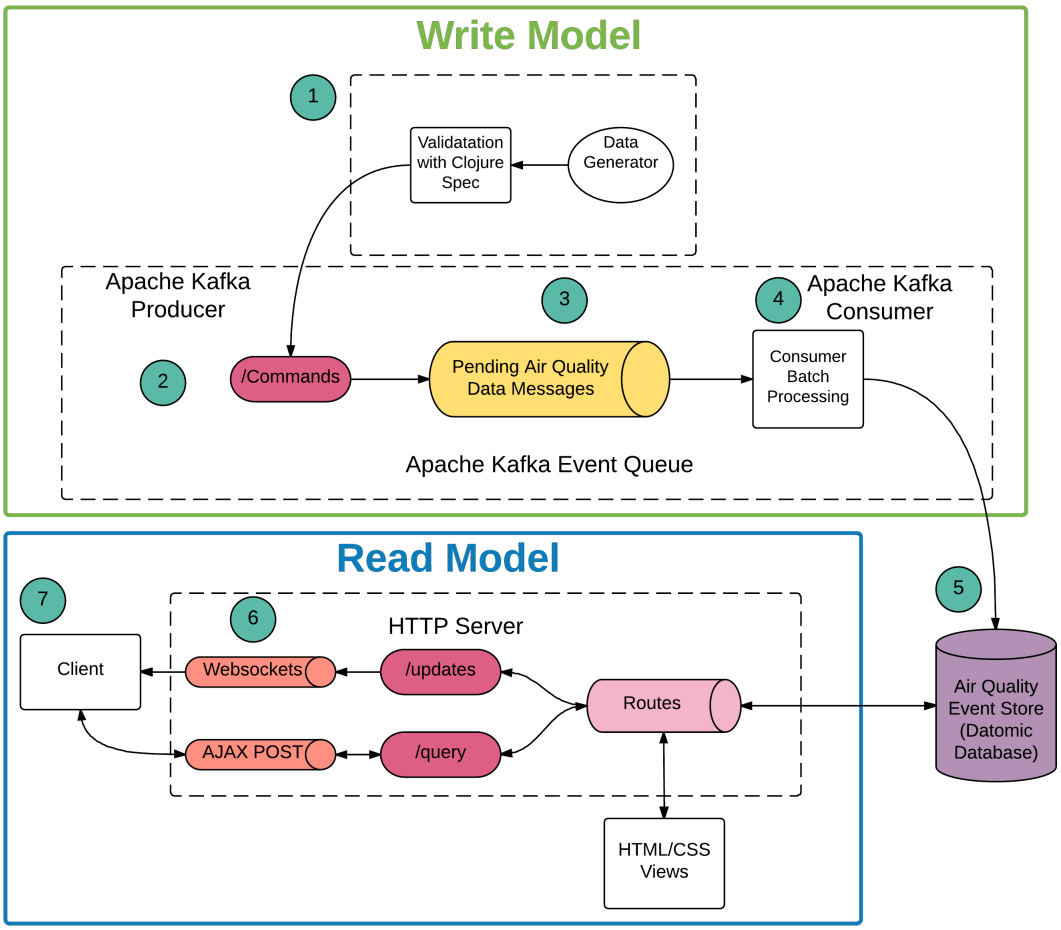


Figure 6.7: Read model tested for bottlenecks within the blue box.

6.3.2 Results of Testing

Initially, the application queries presented themselves as bottlenecks within the application. The application queries were consequently optimized, the database schema was adjusted and concurrency was introduced in the queries to eliminate the bottleneck. The optimized query times are shown in the table below:

Table 6.3: Query time after optimization.

	Approximate Time
Query for real-time updates for all suburbs and pollutants	292.81 ms
Query for CSV data	1.13 μs

The other higher level functions in the read model (the application route handlers) take less than 970 [ns] to execute and therefore the application can currently be assumed to be free of potential bottlenecks within the read model.

6.4 Acceptance Test 4: Memory Leak Detection

Java Mission Control Flight Recorder and Mbeans Server was used to test the application for the possibility of memory leaks. The concerns around memory leaks and application performance are discussed in section 2.4.4 of the literature review.

6.4.1 Experimental Setup

The application was hosted locally in the development environment whilst the JDK was running. Flight Recorder was used to monitor the application's performance over a period of time, while the Mbeans Server provided the current performance statistics.

6.4.2 Results of Testing

From Figure 6.8 and 6.9 it can be seen that the application does not have any memory leaks as the memory usage remains almost constant. It is important to note that the memory scale on the graph's y-axis goes up to the available physical memory of the

machine and therefore in some cases, such as this, the Java heap may only be present in a small section at the bottom of the graph. It is also important to examine the Flight Recorder GCs over the recorded time period, specifically the heap statistics. If the heap usage in the heap after GC statistics increases from the first to the last old GC, it can indicate the presence of a memory leak. The heap usage of first old GC, shown in Figure 6.10, is 50.62 MB and the heap usage of the last GC, shown in Figure 6.11, is 50.61 MB, therefore the heap usage of the application is not increasing and this serves to further emphasize that the application does not possess a memory leak.

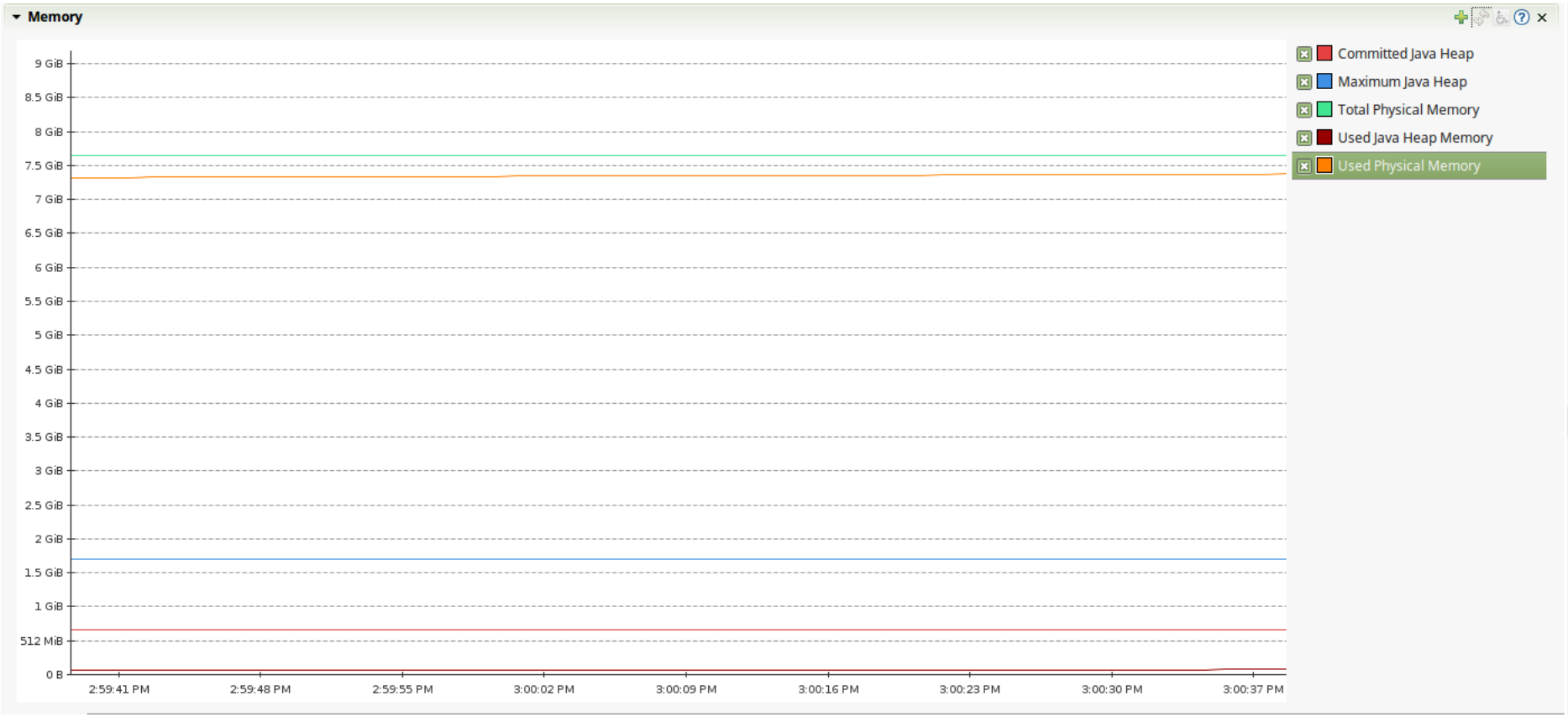


Figure 6.8: Java Mission Control Mbeans Server Memory Overview.

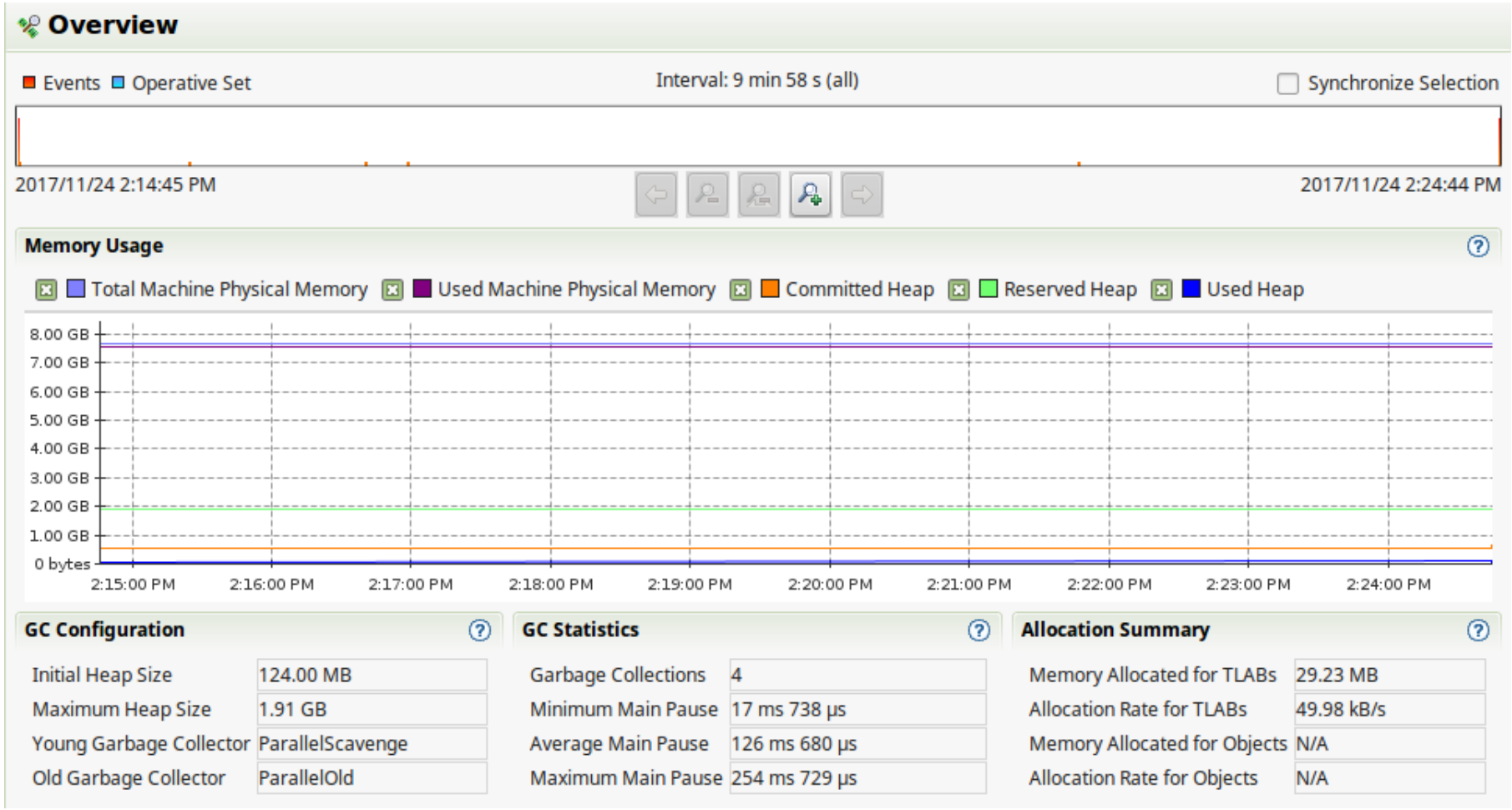


Figure 6.9: Java Mission Control Flight Recorder Memory Overview over a period of 10 minutes.

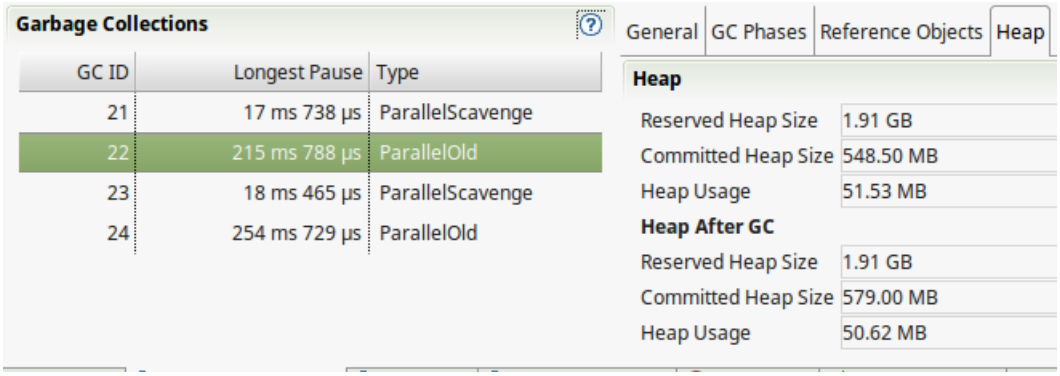


Figure 6.10: Heap statistics in the first old GC from the Flight Recorder recording over the period of 10 minutes.

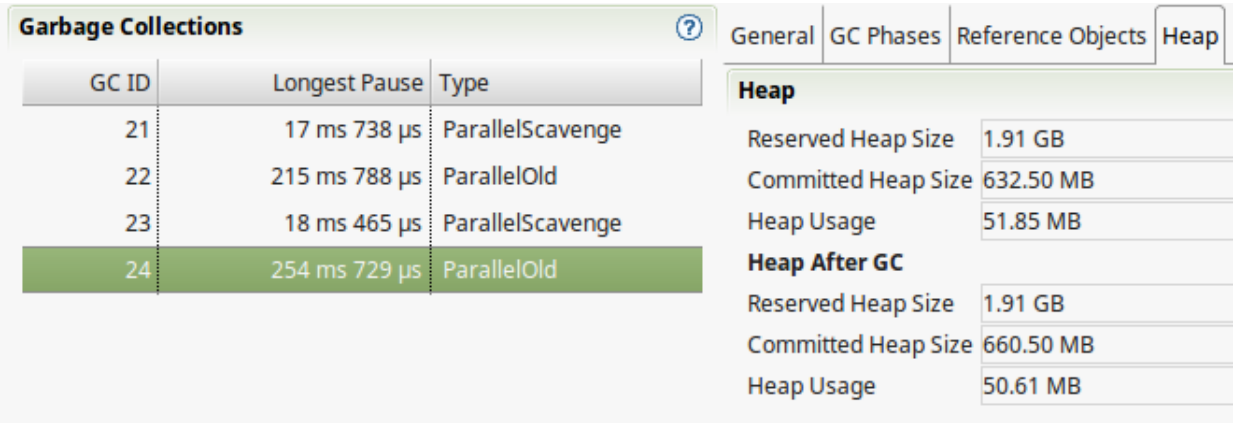


Figure 6.11: Heap statistics in the last old GC from the Flight Recorder recording over the period of 10 minutes.

6.5 Acceptance Test 5: Functionality

The functionality tests were conducted to verify the application features performed as was intended during development and that it meets the requirements specified. Each functionality test focuses on the application processing the browser request and returning the expected response. The result of each test is based on whether or not the application successfully or unsuccessfully executed the function.

6.5.1 Google Map AQI Real-Time Update

As previously mentioned in section 5.7.2, the application's WebSocket server is current set to push AQI updates to the client every 10 minutes. The Google map AQI update test focused on verifying the AQI data for each location was displayed correctly in the following steps:

1. All the sensor pin locations for the City of Cape Town are visible on the Google map.
2. When the user clicks on a map pin, the map zooms in on the pin location and an information window should open and display the AQI data for that area.

Results of Testing

It can be seen from the Figures 6.12 and 6.13 that when the web page loads, the Google map displays all the air quality monitoring sensor pin locations for the City of Cape Town. When the user clicks on a pin location the map zooms in on the pin and an information window opens displaying the real-time AQI data for the area. The application successfully displays real-time AQI updates on the Google map and the feature works as was intended.

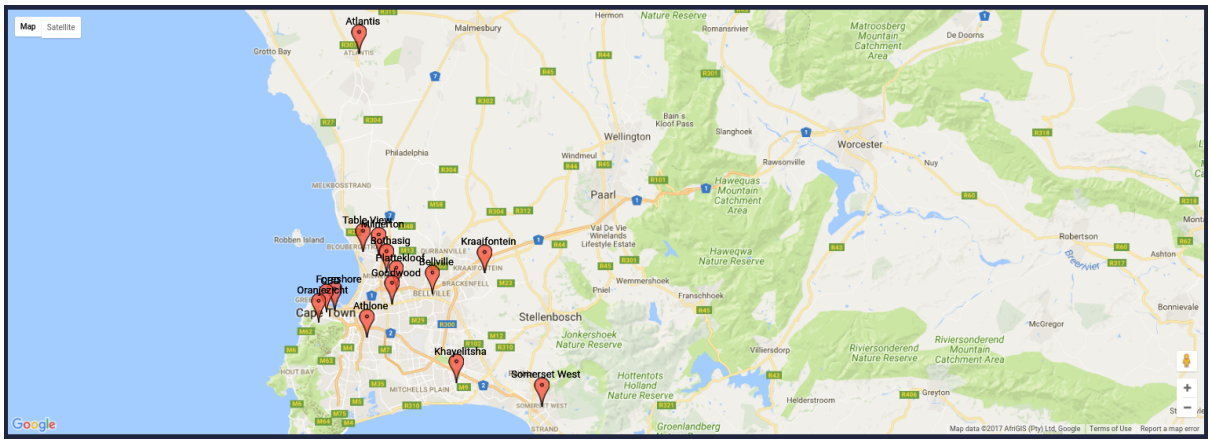


Figure 6.12: Google map with all the sensor pin locations for the City of Cape Town visible.

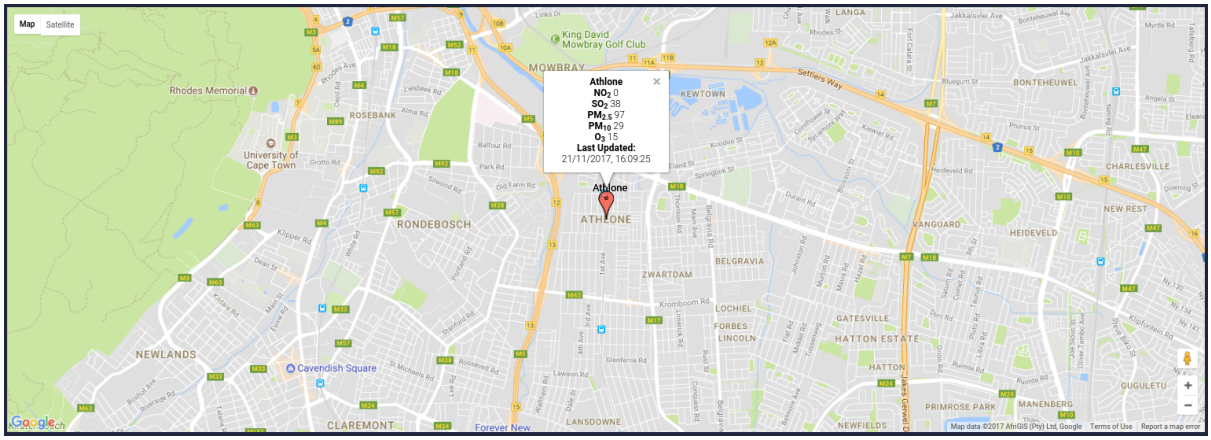


Figure 6.13: Google map information window displaying the AQI data for the area after user clicked on map pin.

6.5.2 AQI CSV Data File Download

The CSV file download test focused on verifying that the AJAX POST request was processed correctly by the browser and the correct response was returned with the correct air quality data in the following steps:

1. The user enters the required CSV download form information (suburb name, start date and end date).
2. Upon submission a request success alert should be displayed.

3. Upon clicking 'okay' on the success alert, a CSV file with the correct (based on user input) air quality data should be downloaded.

Results of Testing

It can be seen from the Figures 6.14 - 6.16 that the user is able to fill in the CSV download request form with ease. Upon submission a successful request alert message is displayed and when the user clicks 'okay,' a CSV file containing the air quality data based on the user input is downloaded by the browser. The application successfully performs the CSV file download of air quality data, based on the user input, as expected.

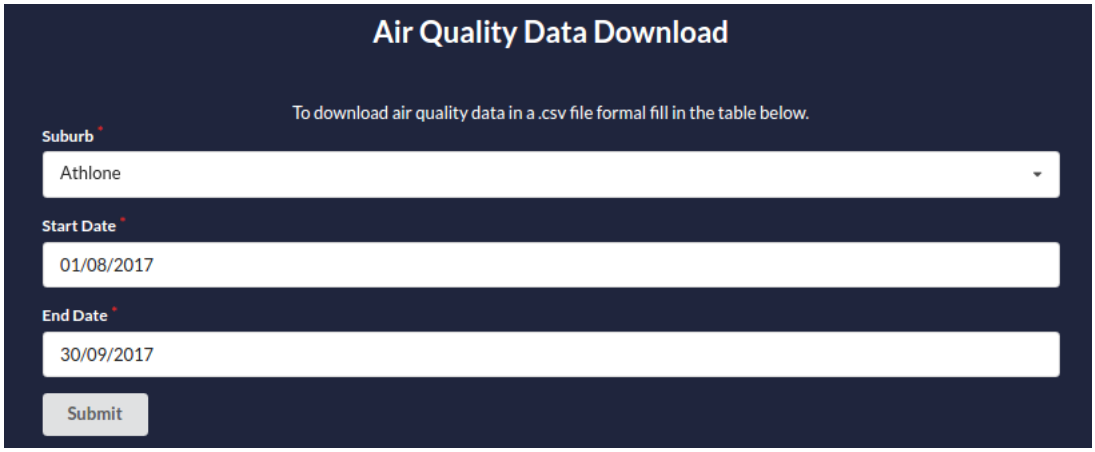


Figure 6.14: CSV download form with information from user.

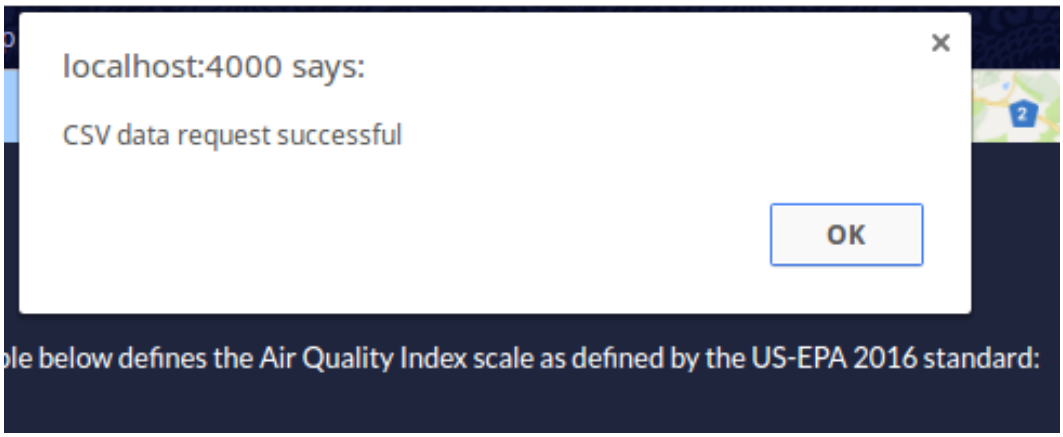


Figure 6.15: Successful request alert.

```
Suburb: Athlone,
,Timestamp,O3[ug/m3],NO2[ug/m3],SO2[ug/m3],PM10[ug/m3],PM2.5[ug/m3],
,2017-08-01T16:47:35Z,72,5,51,78,5,
,2017-08-01T17:06:42Z,77,23,55,90,7,
,2017-08-01T17:50:13Z,79,28,32,218,0,
,2017-08-01T18:00:13Z,49,20,49,108,17,
,2017-08-01T18:10:13Z,34,31,45,132,20,
,2017-08-07T14:24:25Z,2,14,94,85,8,
,2017-08-07T14:39:25Z,25,29,33,163,22,
,2017-08-07T15:20:42Z,20,21,19,216,19,
,2017-08-07T16:35:38Z,57,6,5,148,20,
,2017-08-07T16:45:38Z,66,34,88,1,15,
,2017-08-08T14:33:43Z,6,12,66,212,4,
,2017-08-08T14:43:43Z,16,32,71,10,2,
,2017-08-08T14:53:43Z,46,21,85,55,7,
,2017-08-08T14:58:43Z,84,17,69,218,19,
,2017-08-08T15:08:43Z,32,6,77,124,19,
,2017-08-08T15:24:20Z,47,14,45,212,6,
,2017-08-08T15:39:53Z,71,13,3,67,24,
,2017-08-08T16:04:53Z,98,10,6,70,22,
,2017-08-08T16:14:53Z,64,33,19,31,19,
,2017-08-08T16:35:18Z,50,28,96,197,8,
,2017-08-09T14:53:44Z,12,23,31,218,13,
,2017-08-09T15:08:44Z,28,23,33,125,3,
```

Figure 6.16: Sample data of air quality information contained in downloaded CSV file.

6.6 Acceptance Test 6: Page Response Time

This section discusses how the page response time testing was conducted and the results of the testing. Page response time testing is a measure of how long it takes for a web page to load, and is necessary to ensure the application is responsive and provides a good user experience.

6.6.1 Experimental Setup

The application was hosted locally in the development environment and the Chrome Developer Tools Timeline was used for response time testing. The page response time of the application using the Chrome Developer Tools without CPU throttling, on a high-end device (2x slower) and a low-end device (5x slower) was measured.

6.6.2 Results of Testing

The results of the page response time testing using Chrome Developer Tools are summarized in Table 6.4.

Table 6.4: Results of page response time testing.

	Time [ms]		
Device and CPU Throttling	**No CPU Throttling**	**High-end Device 2x Slower**	**Low-end Device 5x Slower**
Loading	129.3	298.3	357.3
Scripting	1180.6	2299.1	3668.6
Rendering	442.6	917.3	1261.7
Painting	21.6	50	52
Other	145.3	303.3	431.4
Idle	367.6	883.9	1026.7
Total [ms]	**2287**	**4752**	**6798**

It is clear from the page response time results in Table 6.4 that Scripting has a significant negative effect on the page response time. Upon further investigation, using the bottom-up and call-tree analysis provided by Chrome Developer Tools, it was discovered the JavaScript for Google map API has a significant negative impact on the performance and is responsible for around 50% of the total scripting time alone.

6.7 Acceptance Test 7: Load and Stress Testing

This section details how the load and stress testing of the application was performed and the results of the testing. The significance of load and stress testing is discussed in sections 2.4.5 and 2.4.6.

6.7.1 Experimental Setup

The application was hosted locally in the development environment whilst the Gatling load test tool was running. Gatling was used to create a load test simulation mimicking expected browser behaviour to use for the purposes of load and stress testing the application. The application performance was not tested independently of the front-end of the application. The performance of requesting the web page (shown as Home Page in Figure 6.17) of the application and a Google map AQI update was measured. Although the Google map AQI updates are not requests and are pushed from the WebSockets server

(SSE), load testing will give an indication of how many concurrent users the application can push updates to. The CSV file download request was not measured as it is an AJAX request and cannot be tested with the Gatling tool. There are other free software alternatives to Gatling that may be used to test the AJAX request performance but it is limited to 50 virtual concurrent users.

6.7.2 Results of Testing

The performance per request is shown in Figure 6.17 and 6.18 below.

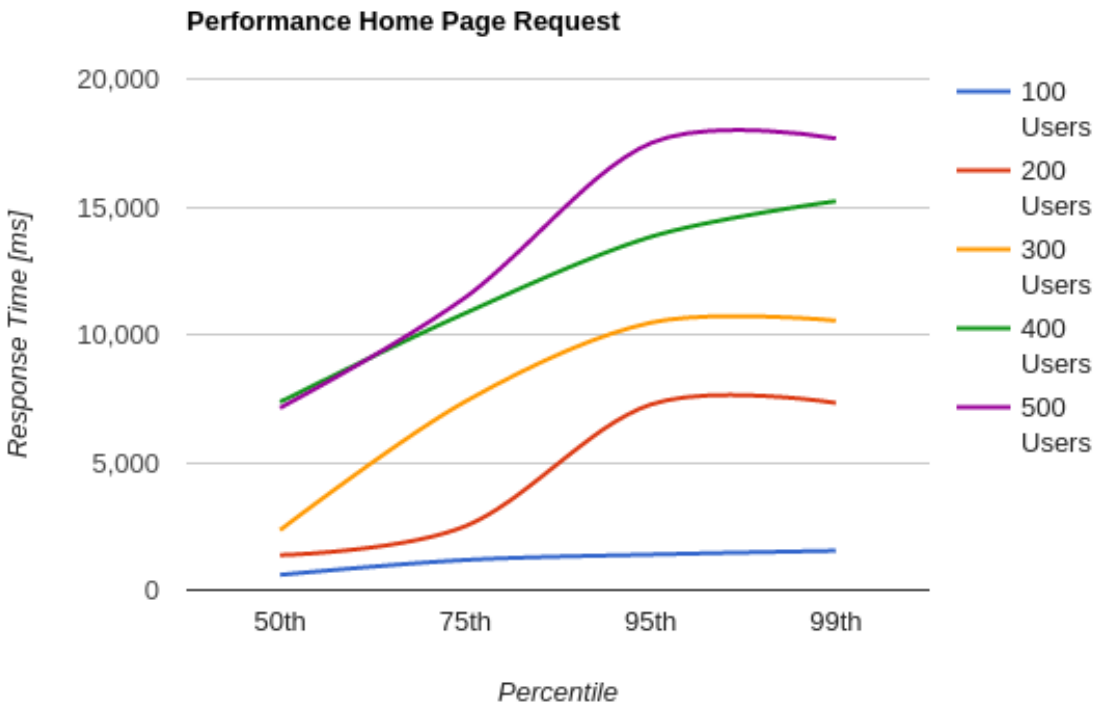


Figure 6.17: Line graph of Home Page request concurrent user performance.

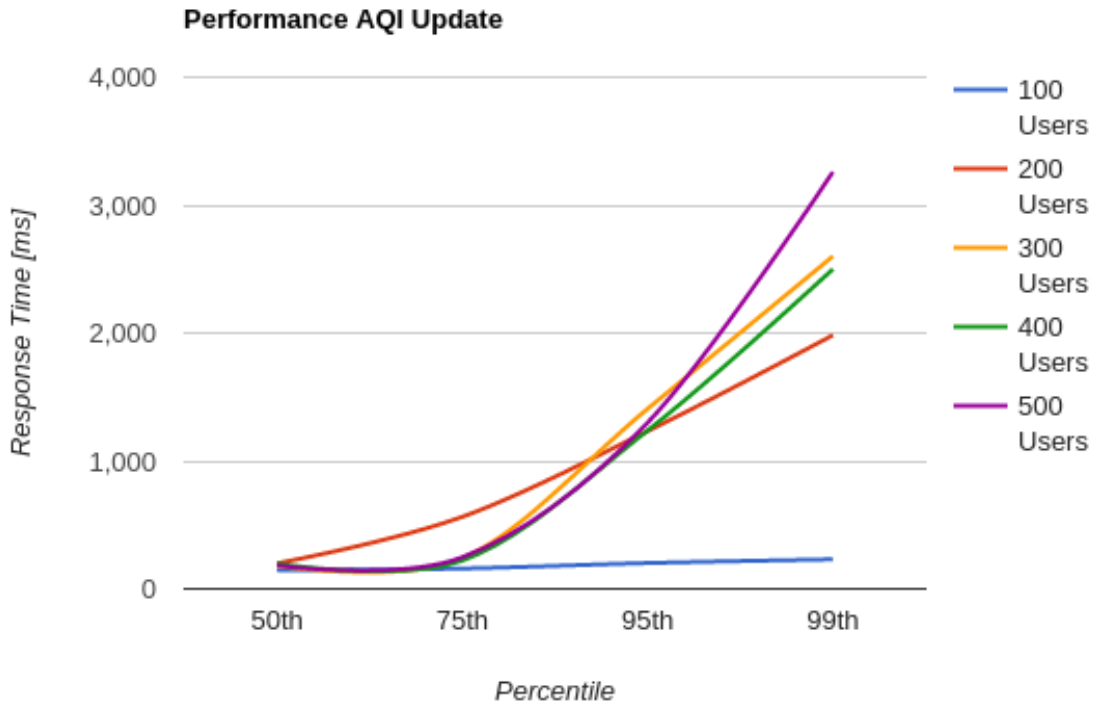


Figure 6.18: Line graph of AQI update request concurrent user performance.

Table 6.5: Number of concurrent users and percentage of total requests vs response time [ms]

Response Time [ms]	**Percentage of Requests**						
	Number of Concurrent Users						
	50	**100**	**200**	**300**	**400**	**450**	**500**
t <800	100%	88%	69%	62%	65%	63%	62%
800 <t <1200	0%	4%	7%	9%	5%	5%	6%
t >1200	0%	8%	24%	29%	30%	32%	30%
Failed	0%	0%	0%	0%	0%	0%	2%

From Figures 6.17-6.18 and Table 6.5 it can be seen that the response time positively correlates to the user load: as the number of concurrent users increases so does the response time and the performance of the application decreases. The results from the Gatling tool indicated that the HTTP Kit web server (with the default of four server threads) was able to process a mean of approximately 60 requests/s under the various concurrent user load conditions in Table 6.5. It can be seen that the application can

handle 200 users adequately (at mean of 45 requests/s), with 69% of request response times below 800 ms, and starts to time-out and fail to respond to requests at around 500 concurrent users. The page load time and slow scripting of the Google map API had a knock-on effect on the application's load and stress testing performance.

6.8 Acceptance Test 8: User Experience Testing

Monitored in-person usability testing was performed to test the user experience of the application.

6.8.1 Experimental Setup

The application was hosted locally in the development environment. Volunteers were invited to use the application, test its functionality and then fill out a survey rating their user experience. Six volunteers were used, all of whom where of a similar skills level, being postgraduate electrical engineering and computer science students. In the survey users were asked to score aspects related to user interface out of ten (ten being the highest achievable score and zero being the lowest). A copy of the survey and the ethical clearance form is available in Appendix C.

6.8.2 User Experience Survey

The user experience survey focused on the following aspects of the user interface:

1. Clarity with respect to ease of use;
2. Conciseness;
3. Familiarity with respect to the application being naturally and intuitively understood;
4. Responsiveness with regard to speed and the interface feedback;
5. Consistency with regard to the interface design;
6. Attractiveness;

7. Efficiency with respect to ability to achieve goals with ease;
8. Forgiveness with regard to the ability to correct user mistakes; and
9. Informativeness with respect to the air quality information supplied.

6.8.3 Results of Survey

Figure 6.19 shows each aspect of the user interface received an above average score from the users. There are a few aspects of the interface that are weaker than others, such as the clarity, familiarity and informativeness, but overall it may be considered a success. The general feedback from users did yield a few recommendations for improvement which will be discussed later in section 8.2.

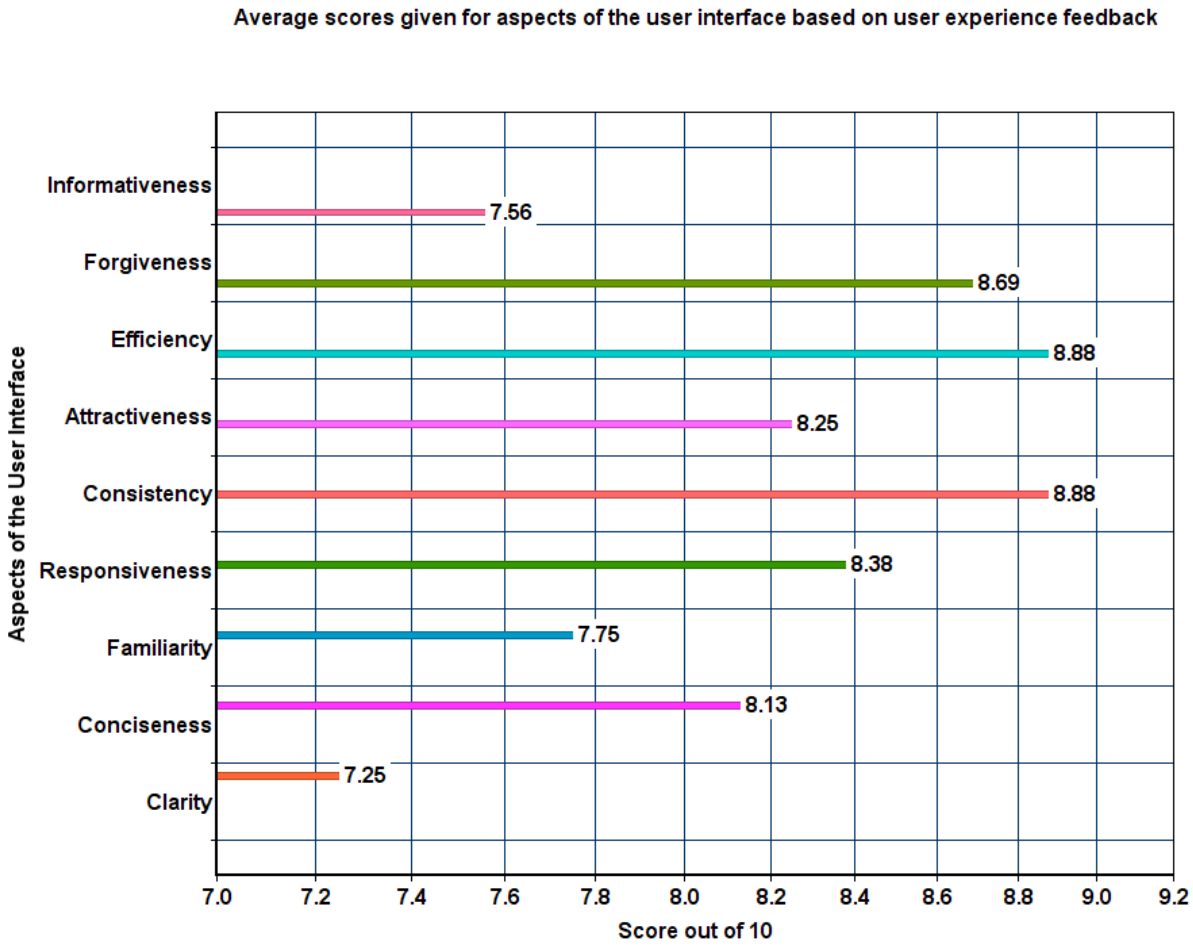


Figure 6.19: Horizontal bar graph of the average scores given by users for aspects of the user interface based on their user experience.

Chapter 7

Conclusions

The research, design and implementation has yielded a real-time air quality index web application prototype for the City of Cape Town that has been designed for scalability, extensibility and data analysis, thereby fulfilling the main objective of the research project. The operation of the system was achieved through the incorporation of the CQRS, Event Sourcing and MVC architectural patterns. The data generator used to simulate air quality data can be replaced easily with a real air quality data API as shown in section 4.3 and the dataset may be extended with ease as discussed in section 5.10. The application architecture itself serves as a basic Event Sourcing template for any application that requires scalability and is data-centric in nature. A summary of the application architectural main points is given in Figure 7.1.

The application's robustness, functionality, performance and the quality of the user interface in development were tested. The acceptance tests performed were outlined in the terms of reference. The Kafka cluster was proven to be fault tolerant and robust in the event of broker failures. The Clojure Spec data validation performed as expected and "invalid" air quality data is logged in the event of data validation failures to assist with the debugging process at a later stage. The real-time AQI updates and CSV air quality data file download features perform as intended. The application's read model has been optimized to be free of potential bottlenecks and as a whole the application does not possess any memory leaks.

The page response time results were affected by the slow scripting of the Google map API and this in turn impacted on the load and stress test results of the application. Nevertheless, the application can adequately accommodate 200 concurrent users with 69% of request response times below 800 ms (with the HTTP Kit web server processing

approximately 45 requests/s on average at 200 concurrent users). It only starts to fail to respond to requests at around 500 concurrent users. On average each aspect of the user interface received a score above 7 (out of 10) and therefore it can be considered an overall success. There is, however, still room for improvement on the weaker aspects of the user interface which received the lowest scores on average, such as the clarity and forgiveness of the user interface. Section 8.2 of the Recommendations includes suggestions on how the user interface may be improved to provide an improved overall performance and user experience. A brief overview of the application acceptance testing results is given in Figure 7.2.

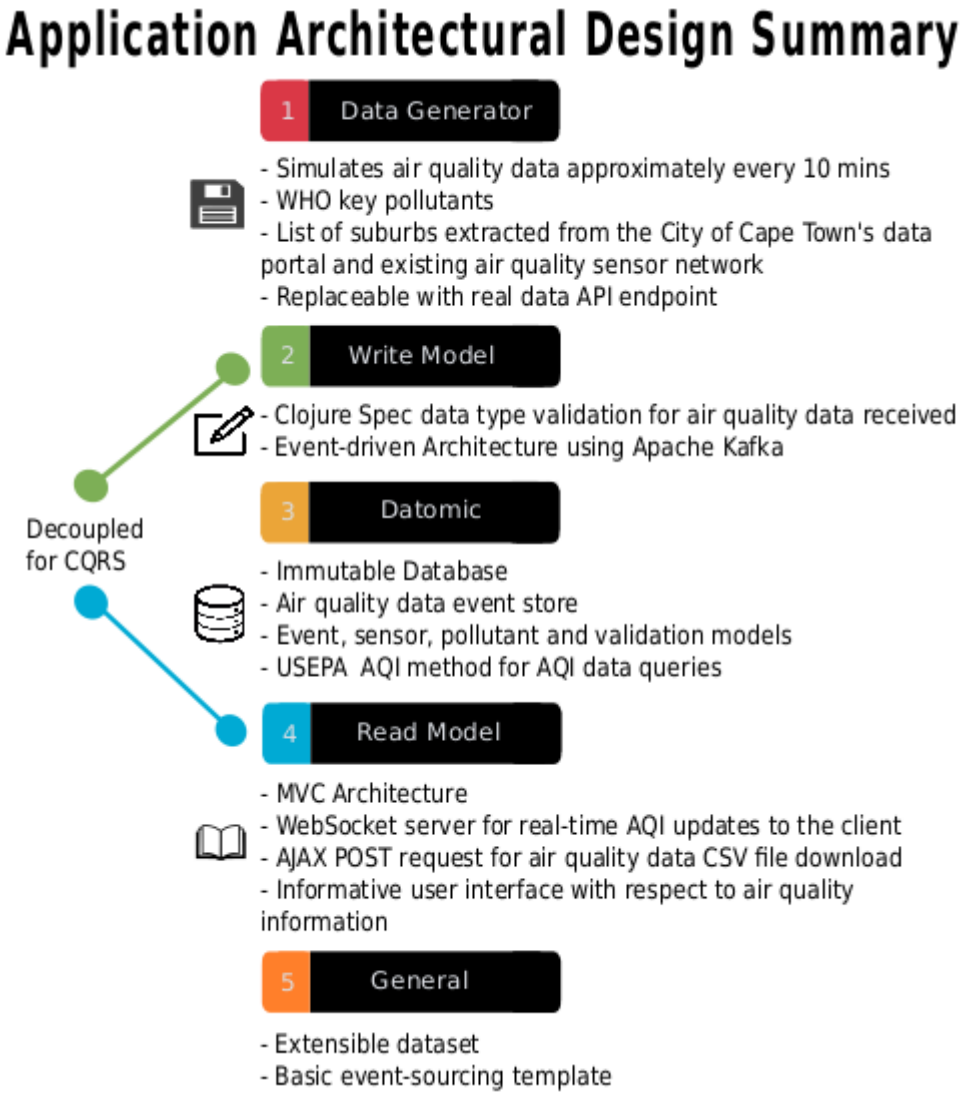


Figure 7.1: Summary of application's main architectural features.

Furthermore, the back-end of the application has been developed using a single programming language, Clojure, to the greatest extent possible because that makes the application easier to maintain by a small team or individual in this context. This does not mean that another language that is more well-suited to a particular aspect of the

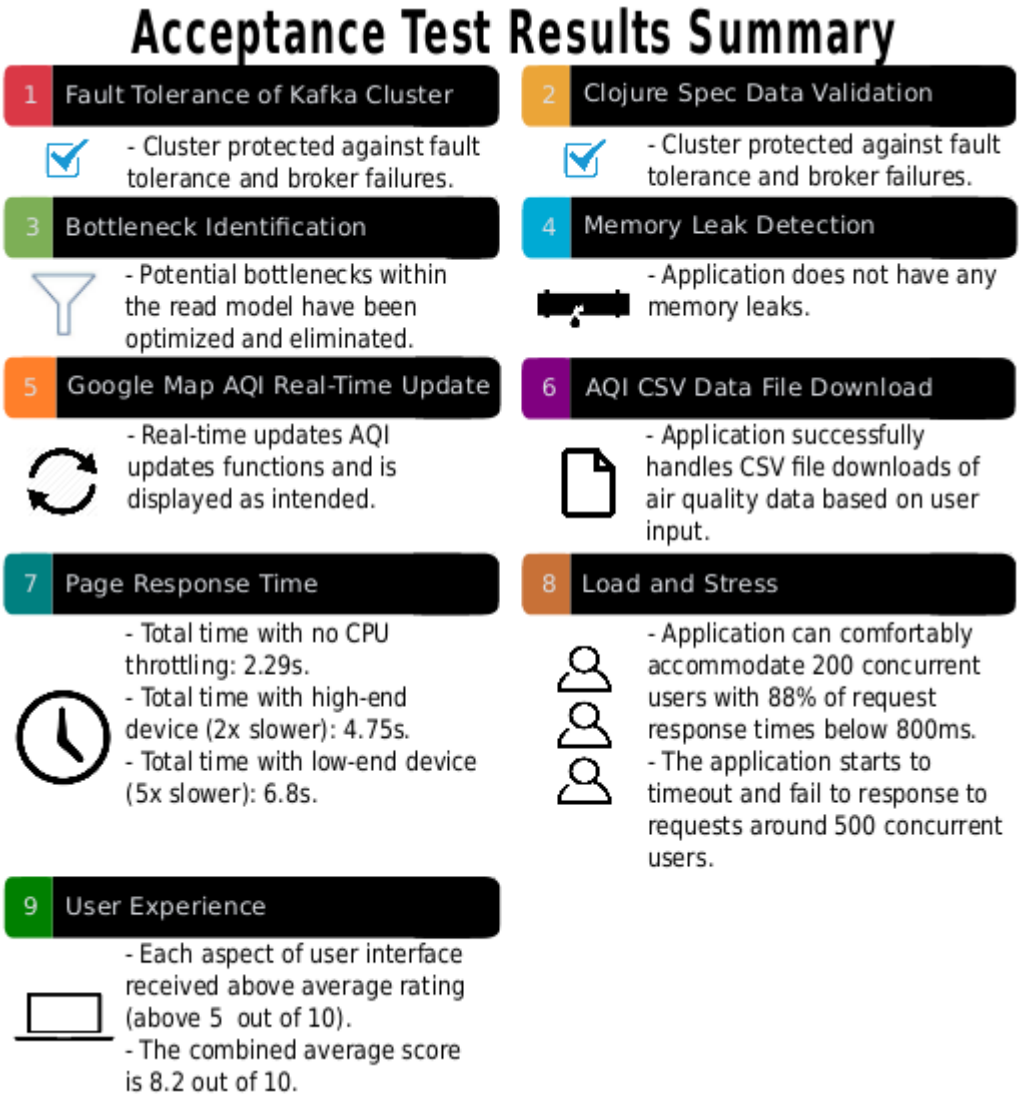


Figure 7.2: Summary of acceptance testing results.

project should not be used, or that this approach is better. Based on the context it will be easier for another individual to continue to improve this project if they have to learn as few programming languages as possible beforehand.

The application prototype has the capability to highlight areas within the city that indicate problematically high pollution levels, and increase public awareness with respect to the levels of air pollution within the city. Furthermore this application can be used as a tool for strategic urban planning. The application is designed with extensibility in mind, and this design approach lays the foundation for the development of predictive models and trend analysis, such as highlighting seasonal allergen trends which could provide services beyond that of real-time air quality data updates. For example this system can provide the basis around which an accurate pollution forecast for citizens can be structured.

In conclusion, the application is able to push real-time AQI updates to the client applications; the user interface was shown to provide a good user experience and is able to handle a large number of simultaneous requests; and the required features were shown to work as intended. All the aspects of the acceptance testing were completed successfully. Additionally, it has met the main objective, sub-objectives, requirements and functionality outlined in Chapter 1. Taking all the aforementioned points into consideration it is concluded that the project was a success.

Chapter 8

Recommendations for Future Work

This final chapter outlines the recommendations that have been made with regard to the further research and development of the project. It is separated into two main sections: architectural recommendations and user interface recommendations. The architectural recommendations offer recommendations with respect to the back-end of the application and how it may be improved, whereas the user interface recommendations section focuses on recommendations to improve the user interface in order to provide a better user experience and is based on user feedback obtained from the user experience testing discussed in section 6.8.

8.1 Architectural Recommendations

The recommendations in this section include suggested improvements to the back-end of the application. The selection of the recommended back-end improvements are based primarily around the extensibility, and providing better robustness and reliability to the system to help ensure there will be no logging of invalid recordings and no loss of data in the system. These refinements may infer some trade-offs, but these decisions and whether such trade-offs should be made are left as investigations for further research on this topic.

8.1.1 Data Validation

Microservice for Data Validation

A microservice may be created to validate the raw sensor data, or the data received from the City of Cape Town's data API, and transform it before it is sent to the air quality application as illustrated in Figure 8.1. This will allow for better fault isolation and split the application into two independent modular services that are independently deployable, scalable and responsible for highly defined tasks. Each service may then be written in different programming languages and be managed by different teams if necessary.

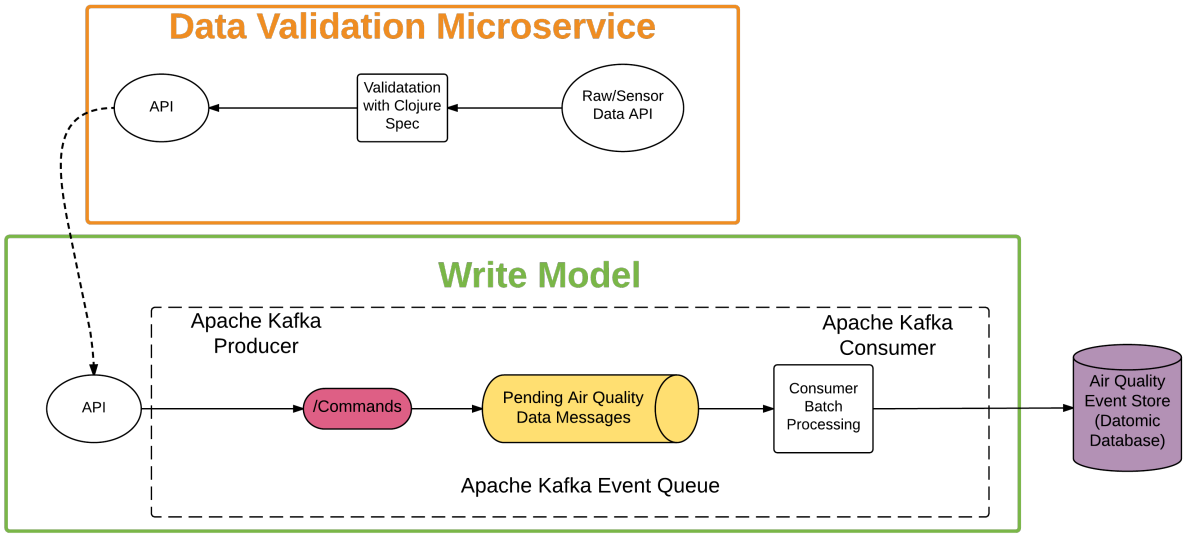


Figure 8.1: Illustration of data validation microservice.

Microservices are a version of Service Oriented Architecture (SOA). SOA is basically a collection of services that communicate with each other. This communication may be as simple as transferring data to each other, or as complicated as many services managing a particular activity. There is no specific definition of what a microservice is, but it's architectural style may be described 'an approach to developing a single application as a suite of small services, each running it's own process and communicating through lightweight mechanisms, usually a HTTP resource API. These services are independently deployable and there is a bare minimum of centralized management of these services.'[43]

Data Validation for Sensor Ranges

As Datomic is ACID compliant, erroneous sensor data of the incorrect data type will not be transacted to the database as previously mentioned in section 5.2 and any erroneous sensor data failing the Clojure Spec validation will be logged to assist with debugging at a later point. However, a further sensor data check needs to be implemented to ensure that the sensor readings received are between the specific sensor's accepted measurement ranges for greater reliability and accuracy.

8.1.2 Extending the Dataset for Data Science

As previously discussed in section 2.1.3, an investigation into which meteorological factors highly correlated to the pollution forecast by AQI specifically for the City of Cape Town would be advisable for more accurate air quality data analysis, and the development and implementation of a reliable ambient air pollution predictive model. This additional data may be received from the same API endpoint as the air quality data, or from a different API endpoint. If the data is received from the same API endpoint then only the following changes need to be made to the system: the various data attributes need to be added to the Datomic air quality event store schema and the batch processing code needs be adjusted to accommodate the new attributes as discussed in section 5.10.2. Alternatively if the data is received from a different API endpoint, the previously mentioned changes still need to be made and the same Kafka queue may be utilized, or the event data may be placed on a separate Kafka queue under a different Kafka topic name (e.g. Atmospheric data), as illustrated in Figure 8.2, for the separation of concerns and data sources.

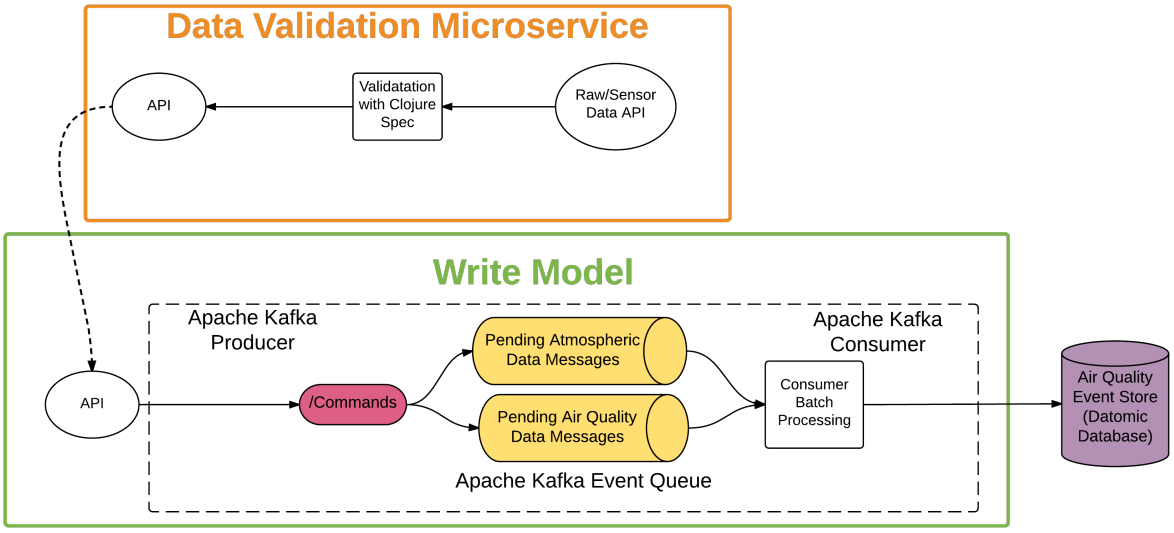


Figure 8.2: Illustration of possible write model design for extending the dataset to include atmospheric data from a different API endpoint.

8.1.3 Stream Processing to Built Aggregates

It is possible to extend the processing of events from the event stream or queue to further to build aggregates, such as caches for the latest AQI values and search indexes, as can been seen in Figure 8.3 taken from Martin Kleppman's Confluent blog post, *Making Sense of Stream Processing*.[18]

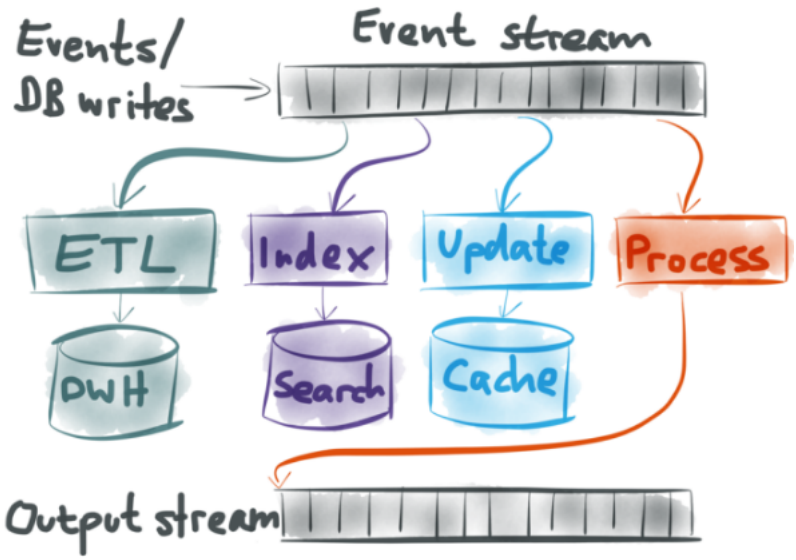


Figure 8.3: Stream Processing to build aggregates.

Martin Klepmann goes on to provide some possible suggestions on what more can be done once one has an event stream:

1. Update full-test search indexes, so users are able to search an up-to-date version of the data. It will be useful if a search feature is implemented on the front-end in the future to provide filtered views of the data.
2. Invalidate or refill caches so reads may be served from fast and up-to-date caches.
3. It is also possible to take an event stream, process it in some way (e.g. maybe join two event streams together) and create a new output stream, allowing the new output stream to act as the input stream to another system. This allows the clean development of complex applications. For example, as discussed in the previous section, an event stream containing meteorological data and another event stream containing air quality data may be combined into a single event stream which can be then be the input event stream to the air quality index application.

8.1.4 Datomic Transaction Retry for Reliability and Robustness

Datomic does not automatically retry transactions after a transactor failure or timeout, therefore it would be beneficial to develop a transaction retry strategy in the event that the transactor fails or experiences a timeout. This would improve the robustness, reliability and fault tolerance of the application and ensure all the necessary data is transacted to the database as expected.

8.1.5 Read Model Architecture

Investigate and implement ways to make the standard implementation of the MVC read model more performant. There are many ways to do this such as caching, load balancing and other client side considerations. Inspiration may also be drawn from the gaming industry and game design, as games possess highly performant user interfaces and some online games contain databases which are read from and written to.[44]

8.1.6 Web Server

The application currently makes use of an HTTP Kit Clojure web server which is considered as a basic development server with few performance tuning options. There are more performant Clojure web servers that one may consider using instead of HTTP Kit that allow for finer control and performance tuning, such as the Aleph, Nginx-Clojure and Immutant web servers.

8.2 User Interface Recommendations

Stopping the use of the Google map API is recommended to improve the performance of the user interface due to the API's long scripting time. The general feedback from users during the user interface testing indicated that they would prefer if a filter was used to display AQI updates in each area within the city, as opposed to displaying AQI updates on a Google map i.e. users would prefer to select options from a list and have the resultant AQI data be displayed on the web page. Users would also prefer seeing recommended AQI thresholds for each pollutant next to the real-time value as it is easier to compare the two values and does not require the user to remember the AQI table values after reading about them on the web page as illustrated in Figure 8.4. This ties in with point 6 (recognition rather than recall) of the user interface heuristics outlined in section 2.4.7.

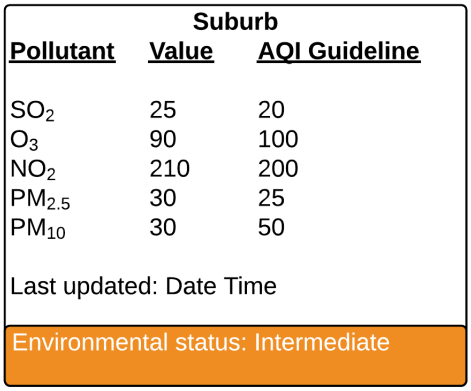


Figure 8.4: Illustration of displaying AQI pollutant level guidelines next to real-time AQI updates for each suburb for easier comparison on behalf of the user.

Furthermore, users would also prefer tabs to switch between a very basic view like that in Figure 8.4 to a more analytical view of statistics and graphs of pollution trends for

each suburb. Cross-platform optimization and testing is also advised to provide a better user experience on multiple devices and screen sizes.

Glossary

broker A node in the Kafka cluster. xii, 48, 79, 80, 99–103, 121

consumer Pulls messages from a Kafka topic. 26, 27, 48, 60, 73, 79, 87

Datalog A declarative database query language. 43, 62

declarative A style of composing the structure and components of a computer program that expresses the logic of computation without describing its control flow. 29, 62

deserialized The reverse process of serialized. 79

destructuring A way of extracting multiple values from objects and arrays. 59

dynamic programming language Execute at runtime many common programming behaviours that static programming languages execute during compilation. 31

encode The process of converting a sequence of characters into a specific format for transmission or storage . 85

enumerated an enumerated type is a data type consisting of a set of named values. 64, 66

extensibility A design principle in which the implementation takes future growth of the system into consideration. 6, 12, 54, 66, 121

functionality The range of operations, capabilities and usefulness of an application, system or device . 4, 8, 9, 11, 35, 43–45, 49, 53, 54, 86, 98, 112, 119, 121, 124

idempotent An operation that may be executed multiple times without changing the result after the initial execution of the operation. 87

immutable Unable to be changed. 3, 7, 22, 24, 25, 31, 33, 42, 46, 54

instrumentation Ability to monitor and measure the program's performance, diagnose errors and write trace information. 59

interoperability The ability of computer systems, or software, to exchange and make use of information. 33

multi-broker A Kafka cluster consisting of more than one node. xii, 9, 44, 48, 79, 98

mutable May be changed. 24, 29, 87

performance The effectiveness of a system, including throughput, availability and response time. xiv, 4, 6, 8–10, 23, 33, 35–37, 45, 46, 49, 64, 69, 79, 98, 99, 107, 116–119, 121, 122, 129

predicate An expression that is used to evaluate if something is true or false. 59, 75

producer Push messages to a Kafka topic. 27, 56, 59, 60, 73, 74, 78, 79, 87

programming paradigm Programming paradigms are a method of classifying programming languages based on their features. It is a style or way of programming. 29

pure functions Functions that always evaluates to the same result given the same argument value(s). It cannot depend on any hidden value or I/O. Evaluation of the result does not cause any semantically observable side effect or output. 29, 31

Push API enables the sending of push messages to a web application via a push service. 6, 10, 40, 46

reliability The ability of a computer system to perform it's intended tasks as expected without experiencing system failure. 9, 35, 98, 128

robustness Ability of a system to deal with errors during execution and input errors. 4, 6, 8–10, 35, 43–45, 98, 121, 128

runtime state As soon as a software program or application is executed, it is in runtime state. 73, 87

scalability The capability of a system to expand to meet the demands of an increase, or increasing, workload. 6, 12, 29, 33, 54, 121

serialized The process of transforming data structures into a format that may be transmitted or stored and reconstructed again at a later point. 79, 131

side effects Occur when a procedure changes a variable from outside its scope. 29

state Present condition of a system or entity. 2, 3, 22–25, 27–29, 32, 33, 38, 58, 60, 69, 87

synchronous A synchronous operation blocks a process until it is complete. 85

user interface The means by which the user and a computer system interact . xii, xiv, 4, 5, 9, 12, 13, 15, 28, 33, 37, 38, 43–45, 50, 54, 70, 72, 73, 88, 93, 98, 119–122, 125, 129

Acronyms

ACID - Atomic, Consistent, Isolated and Durable Set of database transaction properties to ensure validity in the event of errors, power failures etc. 63

AJAX - Asynchronous JavaScript and XML . 34, 44, 45, 57, 70, 88, 114, 118

API - Application Program Interface . xii, xv, 6, 7, 10, 22, 33, 35, 41, 43, 44, 47, 57–59, 63, 66, 68, 75, 87, 94, 117, 120, 122, 127, 128, 130

AQI - Air Quality Index An index for daily air quality reporting that provides an indication of the human health effects due to short-term exposure to each pollutant. xi, xiii–xv, xvii, 2–5, 9, 14, 16–18, 20, 21, 43, 44, 50, 66, 72, 74, 81–84, 87, 88, 91–94, 97, 113, 114, 117, 119, 122, 127, 128, 130, 131

CQRS - Command Query Responsibility Segregation A system architectural pattern that separates the system into two clear parts, one is used for writing (commands that update the system's state) and the other part is used for reading (querying for information without changing the system's state). xi, xvi, 1, 5, 7, 10, 11, 14, 15, 22–24, 43, 44, 47, 55, 56, 122

CRUD - Create, Read, Update and Delete . 3, 24

CSS - Cascading Style Sheet A language used to describe the presentation of web pages. 34, 45, 71, 89

CSV - Comma Separated Values . xiii, xiv, 7, 9, 50, 55, 57, 70, 72, 88, 93, 94, 108, 114–116, 118, 122

EDA - Event-Driven Architecture An architectural pattern that involves the production, consumption, detection of and reaction to events. 3–5, 7, 9, 10, 14, 22, 26, 27, 43–45, 49, 55, 56, 99

EDN - Extensible Data Notation A data format. 31, 63

GC - Garbage Collection A process to recycle memory used by a Java based application. xiv, 36, 109, 112

HTML - HyperText Markup Language . 34, 45, 71, 74, 89, 91, 97

HTTP - Hypertext Transfer Protocol . xvi, 23, 33, 34, 56, 57, 70, 86, 87, 119, 122, 127, 130

IMDB - Internet Movie Database . 34

JDK - Java SE Development Kit . 100, 108

JSON - JavaScript Object Notation . 70, 74, 86

JVM - Java Virtual Machine The Java platform runtime engine. It allows any program written in Java, or other languages, to be compiled into Java bytecode to run on any computer that has a native JVM. 31, 36, 62, 88, 100

LISP - List Programming . 5, 31–33, 74, 100

MVC - Model-View-Controller An architectural pattern that separates the domain logic from the user input logic and the presentation of information to the client. xi, 1, 7, 10, 11, 14, 15, 22, 28, 43–45, 47, 55, 57, 70, 122, 130

NO_2 - Nitrogen Dioxide . 17, 75, 82

O_3 - Ozone . 17, 75, 82

PM_{10} - Particulate Matter 10 . 1, 17, 75, 82–84

$PM_{2.5}$ - Particulate Matter 2.5 . 1, 17, 18, 75, 82

REST - Representational State Transfer . 22, 33, 34, 44, 57

SAAQIS - South African Air Quality Information System . 22, 41

SO_2 - Sulphur Dioxide . 17, 75, 82

SOA - Service Oriented Architecture An approach involving the development of an application as a suite of small services that run their own process and communicate via lightweight mechanisms, such as an HTTP resource API. 127

SSE - Server-Sent Events Technology that allows the browser to receive automatic updates from the server via HTTP requests. 23, 34, 117

URL - Uniform Resource Locator . 33, 34

USEPA - United States Environmental Protection Agency . xi, xvii, 5, 11, 16, 18, 19, 43, 44, 55, 72, 82–84, 92

WHO - World Health Organization . xvi, 1–3, 5, 7, 11, 15, 17, 43, 54, 67, 75, 82, 90

XML - Extensible Markup Language . 85

Bibliography

[1] *Pollution Index by Country 2016.* Numbeo, 2016, 2016-3-17. [Online]. Available: http://www.numbeo.com/pollution/rankings_by_country.jsp

[2] *Ambient air quality and health factsheet.* World Health Organization, 2016-3-17. [Online]. Available: http://www.who.int/mediacentre/factsheets/fs313/en/

[3] J. Yoder, R. Johnson, Q. D. Wilson, and M. Douglas, "Connecting business objects to relational databases," 09 1998.

[4] M. Fowler, *Event Sourcing*, 2017, 2017-3-20. [Online]. Available: https://martinfowler.com/eaaDev/EventSourcing.html

[5] B. Bishoi, A. Prakash, and V. Jain, "A comparative study of air quality index based on factor analysis and us-epa methods for an urban environment," *Aerosol and Air Quality Research*, vol. 9, no. 1, pp. 1–17, 2009. [Online]. Available: http://aerosol.ieexa.cas.cn/aaqrkw/kwlwqj/201207/W020120730541574921620.pdf

[6] D. Mintz, *Technical Assistance Document for the Reporting of Daily Air Quality – the Air Quality Index (AQI).* Research Triangle Park, North Carolina: U.S. EPA Office of Air Quality Planning and Standards, May 2016. [Online]. Available: https://www3.epa.gov/airnow/aqi-technical-assistance-document-may2016.pdf

[7] E. Cogliani, "Air pollution forecast in cities by an air pollution index highly correlated with meteorological variables," *Atmospheric Environment*, vol. 35, no. 16, pp. 2871–2877, 2001.

[8] L. Li, J. Qian, C.-Q. Ou, Y.-X. Zhou, C. Guo, and Y. Guo, "Spatial and temporal analysis of air pollution index and its timescale-dependent relationship with meteorological factors in guangzhou, china, 2011," *Environmental Pollution*, vol. 190, pp. 75–81, 2014.

[9] J. Vidal, "How are cities around the world tackling air pollution?" *The Guardian*, 2017. [Online]. Available: https://www.theguardian.com/environment/2016/may/17/how-are-cities-around-the-world-tackling-air-pollution

[10] *Why Event Sourcing.* Eventuate.io, 2017, 2017-03-27. [Online]. Available: http://eventuate.io/whyeventsourcing.html

[11] N. Narkhede, *Event sourcing, CQRS, stream processing and Apache Kafka: What's the connection?* Confluent, 2017, 2017-01-30. [Online]. Available: https://www.confluent.io/blog/event-sourcing-cqrs-stream-processing-apache-kafka-whats-connection/

[12] B. Calderwood, "From rest to cqrs - clojure/conj 2015," 2015. [Online]. Available: https://www.youtube.com/watch?v=qDNPQo9UmJA

[13] K. Garus, *Introduction to Event Sourcing and Command-Query Responsibility Segregation*, 2016, 2016-3-21. [Online]. Available: http://squirrel.pl/blog/2015/08/31/introduction-to-event-sourcing-and-command-query-responsibility-segregation/

[14] A. Morrison, *The rise of immutable data stores.* Price Waterhouse Coopers, 2017. [Online]. Available: http://usblogs.pwc.com/emerging-technology/the-rise-of-immutable-data-stores/

[15] *What is an event queue.* Techopedia.com, 2016, 2016-3-27. [Online]. Available: https://www.techopedia.com/definition/24963/event-queue

[16] *Reference 3: Introducing Event Sourcing.* Microsoft, 2016, 2016-03-22. [Online]. Available: https://msdn.microsoft.com/en-us/library/jj591559.aspx#sec6

[17] G. Hohpe, "Programming without a call stack - event-driven architectures," *Objekt Spektrum*, 2006.

[18] M. Klepmann, *Stream processing, Event sourcing, Reactive, CEP and making sense of it all.* Confluent, Inc., 2015, 2016-3-27. [Online]. Available: http://www.confluent.io/blog/making-sense-of-stream-processing/

[19] B. Morris, "Event stores and event sourcing: some practical disadvantages and problems," 2017. [Online]. Available: http://www.ben-morris.com/event-stores-and-event-sourcing-some-practical-disadvantages-and-problems/

[20] S. Prakash, A. Kumar, and R. Bhushan Mishra, "Mvc architecture driven design and agile implementation of a web-based software system," *IJSEA*, vol. 4, no. 6, pp. 13–28, 2013.

[21] E. Elliott, "Master the javascript interview: What is functional programming?" 2017. [Online]. Available: https://medium.com/javascript-scene/master-the-javascript-interview-what-is-functional-programming-7f218c68b3a0

[22] S. Wassel, *Functional vs. object-oriented programming (OOP).* lynda.com, 2017.

[23] *The Clojure Programming Language.* Clojure.org, 2016, 2016-3-19. [Online]. Available: https://clojure.org/

[24] A. Williams, *The New Stack Makers: Adrian Cockcroft on Sun, Netflix, Clojure, Go, Docker and More - The New Stack*, 2014. [Online]. Available: https://thenewstack.io/the-new-stack-makers-adrian-cockcroft-on-sun-netflix-clojure-go-docker-and-more/

[25] S. D. Halloway and A. Bedra, *Programming Clojure*, 2nd ed. Pragmatic Bookshelf, 2012.

[26] *What is REST.* RESTfulAPI.net, 2016-12-16. [Online]. Available: https://restfulapi.net/

[27] R. T. Fielding and R. N. Taylor, "Principled design of the modern web architecture," *ACM Trans. Internet Technol.*, vol. 2, no. 2, pp. 115–150, May 2002. [Online]. Available: http://doi.acm.org/10.1145/514183.514185

[28] *Real-time Apps with WebSockets & Server-sent Events.* SitePoint, 2017, 2017-01-30. [Online]. Available: https://www.sitepoint.com/real-time-apps-websockets-server-sent-events/

[29] *What is a front-end framework and why use one?* The Balance, 2017, 2017-04-30. [Online]. Available: https://www.thebalance.com/what-is-a-front-end-framework-and-why-use-one-2071948

[30] D. Winslow, *Why You Need a Local Testing Server (and How To Do It).* DWUser.com Education Center, 2016, 2016-03-27. [Online]. Available: http://www.dwuser.com/education/content/why-you-need-a-testing-server-and-how-to-do-it/

[31] D. Shen, Q. Luo, D. Poshyvanyk, and M. Grechanik, "Automating performance bottleneck detection using search-based application profiling," in *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, ser. ISSTA 2015. New York, NY, USA: ACM, 2015. [Online]. Available: http://doi.acm.org/10.1145/2771783.2771816

[32] *Memory Leak Testing - Why it is important and how is it done.* Intense Testing, 2017, 2017-06-30. [Online]. Available: https://intensetesting.wordpress.com/2014/03/28/memory-leak-testing-why-it-is-important-how-is-it-done/

[33] *What is load testing.* SearchSoftwareQuality, 2016, 2016-03-27. [Online]. Available: http://searchsoftwarequality.techtarget.com/definition/load-testing

[34] J. Nielsen, "Enhancing the explanatory power of usability heuristics," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser.

CHI '94. New York, NY, USA: ACM, 1994, pp. 152–158. [Online]. Available: http://doi.acm.org/10.1145/191666.191729

[35] J. Nielson, *10 Heuristics for User Interface Design.* Nngroup.com, 1995. [Online]. Available: https://www.nngroup.com/articles/ten-usability-heuristics/

[36] A. Miller, *Clojure Spec Guide Documentation.* Clojure.org, 2017, 2017-1-9. [Online]. Available: https://clojure.org/guides/spec

[37] *Clojure Spec About Documentation.* Clojure.org, 2017, 2017-1-9. [Online]. Available: https://clojure.org/about/spec

[38] *Apache Kafka Documentation.* Apache Software Foundation, 2017, 2017-1-9. [Online]. Available: https://kafka.apache.org/documentation/

[39] *Datomic-The fully transactional, cloud-ready, distributed database.* Datomic, 2016, 2016-3-27. [Online]. Available: http://www.datomic.com

[40] D. Higgenbotham, *Datomic for Five Year Olds.* Flyingmachinestudios.com, 2013, 2017-01-20. [Online]. Available: http://www.flyingmachinestudios.com/programming/datomic-for-five-year-olds/

[41] T. K. Boguski, *Environmental Science and Technology Briefs for Citizens Understanding Units of Measurement*, 2nd ed. Center for Hazardous Substance Research Kansas State University, 2006. [Online]. Available: https://cfpub.epa.gov/ncer_abstracts/index.cfm/fuseaction/display.files/fileID/14285

[42] S. Sierra, *Component.* GitHub Repository, June 2017. [Online]. Available: https://github.com/stuartsierra/component

[43] M. Fowler, *Microservices*, 2014, 2016-9-27. [Online]. Available: https://martinfowler.com/articles/microservices.html

[44] D. Johnson and J. Wiles, "Effective affective user interface design in games," *Ergonomics*, vol. 46, no. 13-14, pp. 1332–1345, 2003.

[45] P. Taoussanis, *Tufte.* GitHub Repository, June 2017. [Online]. Available: https://github.com/ptaoussanis/tufte

[46] J. Reeves, *Compojure.* GitHub Repository, May 2017. [Online]. Available: https://github.com/weavejester/compojure

[47] P. Taoussanis, *Timbre.* GitHub Repository, June 2017. [Online]. Available: https://github.com/ptaoussanis/timbre

[48] J. R. Mark McGranaghan, *Ring*. GitHub Repository, May 2017. [Online]. Available: https://github.com/ring-clojure/ring

[49] C. Emerick, *tools.nrepl*. GitHub Repository, June 2017. [Online]. Available: https://github.com/clojure/tools.nrepl

[50] S. C. Michael Klishin, *Clj-time*. GitHub Repository, September 2016. [Online]. Available: https://github.com/clj-time/clj-time

[51] L. Hinman, *Cheshire*. GitHub Repository, May 2017. [Online]. Available: https://github.com/dakrone/cheshire

Appendix A

Addenda

A.1 Data Produced by Data Generator

Example of air quality sensor data produced by the data generator is given in Listing A.1.

Listing A.1: Example of air quality sensor produced by the data generator.

```
[(1 "Athlone" 18.50179479999997 -33.96526 :name/NO2 30)
(1 "Athlone" 18.50179479999997 -33.96526 :name/SO2 57)
(1 "Athlone" 18.50179479999997 -33.96526 :name/PM2.5 17)
(1 "Athlone" 18.50179479999997 -33.96526 :name/PM10 115)
(1 "Athlone" 18.50179479999997 -33.96526 :name/O3 60)
(2 "Atlantis" 18.4869555 -33.5062662 :name/NO2 16)
(2 "Atlantis" 18.4869555 -33.5062662 :name/SO2 65)
(2 "Atlantis" 18.4869555 -33.5062662 :name/PM2.5 2)
(2 "Atlantis" 18.4869555 -33.5062662 :name/PM10 150)
(2 "Atlantis" 18.4869555 -33.5062662 :name/O3 90)
(3 "Bellville" 18.6294384 -33.8942695 :name/NO2 1)
(3 "Bellville" 18.6294384 -33.8942695 :name/SO2 88)
(3 "Bellville" 18.6294384 -33.8942695 :name/PM2.5 1)
(3 "Bellville" 18.6294384 -33.8942695 :name/PM10 7)
(3 "Bellville" 18.6294384 -33.8942695 :name/O3 12)
(4 "Bothasig" 18.5403799 -33.8596601 :name/NO2 22)
(4 "Bothasig" 18.5403799 -33.8596601 :name/SO2 72)
(4 "Bothasig" 18.5403799 -33.8596601 :name/PM2.5 18)
(4 "Bothasig" 18.5403799 -33.8596601 :name/PM10 36)
```

```
    (4 "Bothasig" 18.5403799 -33.8596601 :name/O3 72)
    (5 "CBD" 18.423784 -33.9253946 :name/NO2 30)
    (5 "CBD" 18.423784 -33.9253946 :name/SO2 10)
    (5 "CBD" 18.423784 -33.9253946 :name/PM2.5 22)
    (5 "CBD" 18.423784 -33.9253946 :name/PM10 143)
    (5 "CBD" 18.423784 -33.9253946 :name/O3 82)
    (6 "Foreshore" 18.4409577 -33.9221814 :name/NO2 35)
    (6 "Foreshore" 18.4409577 -33.9221814 :name/SO2 94)
    (6 "Foreshore" 18.4409577 -33.9221814 :name/PM2.5 16)
    (6 "Foreshore" 18.4409577 -33.9221814 :name/PM10 96)
    (6 "Foreshore" 18.4409577 -33.9221814 :name/O3 16)
    (7 "Goodwood" 18.5522531 -33.9116724 :name/NO2 4)
    (7 "Goodwood" 18.5522531 -33.9116724 :name/SO2 36)
    (7 "Goodwood" 18.5522531 -33.9116724 :name/PM2.5 9)
    (7 "Goodwood" 18.5522531 -33.9116724 :name/PM10 21)
    (7 "Goodwood" 18.5522531 -33.9116724 :name/O3 32)
    (8 "Khayelitsha" 18.6769451 -34.037444 :name/NO2 39)
    (8 "Khayelitsha" 18.6769451 -34.037444 :name/SO2 33)
    (8 "Khayelitsha" 18.6769451 -34.037444 :name/PM2.5 23)
    (8 "Khayelitsha" 18.6769451 -34.037444 :name/PM10 95)
    (8 "Khayelitsha" 18.6769451 -34.037444 :name/O3 0)
    (9 "Oranjezicht" 18.4094969 -33.939516 :name/NO2 3)
    (9 "Oranjezicht" 18.4094969 -33.939516 :name/SO2 70)
    (9 "Oranjezicht" 18.4094969 -33.939516 :name/PM2.5 7)
    (9 "Oranjezicht" 18.4094969 -33.939516 :name/PM10 24)
    (9 "Oranjezicht" 18.4094969 -33.939516 :name/O3 94)
    (10 "Plattekloof" 18.5596741 -33.8876347 :name/NO2 9)
    (10 "Plattekloof" 18.5596741 -33.8876347 :name/SO2 58)
    (10 "Plattekloof" 18.5596741 -33.8876347 :name/PM2.5 4)
    (10 "Plattekloof" 18.5596741 -33.8876347 :name/PM10 171)
    (10 "Plattekloof" 18.5596741 -33.8876347 :name/O3 5)
    (11 "Milnerton" 18.525165 -33.83432 :name/NO2 21)
    (11 "Milnerton" 18.525165 -33.83432 :name/SO2 61)
    (11 "Milnerton" 18.525165 -33.83432 :name/PM2.5 21)
    (11 "Milnerton" 18.525165 -33.83432 :name/PM10 198)
    (11 "Milnerton" 18.525165 -33.83432 :name/O3 43)
    (12 "Somerset West" 18.8432656 -34.0756899 :name/NO2 3)
    (12 "Somerset West" 18.8432656 -34.0756899 :name/SO2 53)
    (12 "Somerset West" 18.8432656 -34.0756899 :name/PM2.5 20)
    (12 "Somerset West" 18.8432656 -34.0756899 :name/PM10 58)
    (12 "Somerset West" 18.8432656 -34.0756899 :name/O3 43)
```

```
61    (13 "Table View" 18.4947987 -33.8275375 :name/NO2 38)
62    (13 "Table View" 18.4947987 -33.8275375 :name/SO2 27)
63    (13 "Table View" 18.4947987 -33.8275375 :name/PM2.5 15)
64    (13 "Table View" 18.4947987 -33.8275375 :name/PM10 7)
65    (13 "Table View" 18.4947987 -33.8275375 :name/O3 90)
66    (14 "Kraaifontein" 18.7318822 -33.8607445 :name/NO2 15)
67    (14 "Kraaifontein" 18.7318822 -33.8607445 :name/SO2 45)
68    (14 "Kraaifontein" 18.7318822 -33.8607445 :name/PM2.5 15)
69    (14 "Kraaifontein" 18.7318822 -33.8607445 :name/PM10 111)
70    (14 "Kraaifontein" 18.7318822 -33.8607445 :name/O3 24)]
```

The data generator produces a Clojure vector of lists. The data in each list is of the format (*sensor_ID suburb_name logitude latitude pollutant_name pollutant_value*). Each list in the vector is then passed to the *validate-sensor-data* function, discussed in section 5.3, through the process of recursion.

Appendix B

Addenda

B.1 Air Quality Event Store Database Schema

The data attributes of the air quality event store are shown in the Datomic database schema in listing B.1.

Listing B.1: Air quality event store database schema.

```
[;;Event attributes

{:db/id #db/id[:db.part/db]
 :db/ident :event/name
 :db/valueType :db.type/string
 :db/cardinality :db.cardinality/one
 :db/doc "The name of the event"
 :db.install/_attribute :db.part/db}

{:db/id #db/id[:db.part/db]
 :db/ident :event/data
 :db/valueType :db.type/ref
 :db/cardinality :db.cardinality/one
 :db/doc "The event data"
 :db.install/_attribute :db.part/db}

{:db/id #db/id[:db.part/db]
 :db/ident :event/type
 :db/valueType :db.type/ref
```

```
 :db/cardinality :db.cardinality/one
 :db/doc "The type of event"
 :db.install/_attribute :db.part/db}

{:db/id #db/id[:db.part/db]
 :db/ident :event/uuid
 :db/valueType :db.type/uuid
 :db/cardinality :db.cardinality/one
 :db/unique :db.unique/value
 :db/doc "The uuid of the event"
 :db.install/_attribute :db.part/db}

;;Event type enums
{:db/id #db/id[:db.part/db]
 :db/ident :type/data}

;;Sensor attributes
{:db/id #db/id[:db.part/db]
 :db/ident :data/sensor
 :db/valueType :db.type/ref
 :db/cardinality :db.cardinality/one
 :db/doc "reference between event data and sensor data"
 :db.install/_attribute :db.part/db}

{:db/id #db/id[:db.part/db]
 :db/ident :sensor/id
 :db/valueType :db.type/long
 :db/cardinality :db.cardinality/one
 :db/index true
 :db/doc "The sensor ID"
 :db.install/_attribute :db.part/db}

{:db/id #db/id[:db.part/db]
 :db/ident :sensor/suburb
 :db/valueType :db.type/string
 :db/cardinality :db.cardinality/one
 :db/index true
 :db/fulltext true
 :db/doc "The name of the suburb in which the sensor is located"
 :db.install/_attribute :db.part/db}

```

```
{:db/id #db/id[:db.part/db]
 :db/ident :sensor/city
 :db/valueType :db.type/string
 :db/cardinality :db.cardinality/one
 :db/doc "The name of the city in which the sensor is located"
 :db.install/_attribute :db.part/db}

{:db/id #db/id[:db.part/db]
 :db/ident :sensor/province
 :db/valueType :db.type/string
 :db/cardinality :db.cardinality/one
 :db/doc "The name of the province in which the sensor is located"
 :db.install/_attribute :db.part/db}

{:db/id #db/id[:db.part/db]
 :db/ident :sensor/country
 :db/valueType :db.type/string
 :db/cardinality :db.cardinality/one
 :db/doc "The name of the country in which the sensor is located"
 :db.install/_attribute :db.part/db}

{:db/id #db/id[:db.part/db]
 :db/ident :sensor/station
 :db/valueType :db.type/string
 :db/cardinality :db.cardinality/one
 :db/doc "The name of the station which the sensor is associated"
 :db.install/_attribute :db.part/db}

{:db/id #db/id[:db.part/db]
 :db/ident :sensor/longitude
 :db/valueType :db.type/double
 :db/cardinality :db.cardinality/one
 :db/doc "The longitudinal co-ordinates of the sensor location"
 :db.install/_attribute :db.part/db}

{:db/id #db/id[:db.part/db]
 :db/ident :sensor/latitude
 :db/valueType :db.type/double
 :db/cardinality :db.cardinality/one
 :db/doc "The latitudinal co-ordinates of the sensor location"
 :db.install/_attribute :db.part/db}
```

```
102
103 {:db/id #db/id[:db.part/db]
104  :db/ident :sensor/pollutant
105  :db/valueType :db.type/ref
106  :db/cardinality :db.cardinality/one
107  :db/doc "Reference to pollutant entity"
108  :db.install/_attribute :db.part/db}
109
110 {:db/id #db/id[:db.part/db]
111  :db/ident :sensor/validation
112  :db/valueType :db.type/ref
113  :db/cardinality :db.cardinality/one
114  :db/doc "Reference to Clojure Spec data validation entity"
115  :db.install/_attribute :db.part/db}
116
117 ;; Pollutant attributes
118
119 {:db/id #db/id[:db.part/db]
120  :db/ident :pollutant/name
121  :db/valueType :db.type/ref
122  :db/cardinality :db.cardinality/one
123  :db/doc "The name of the pollutant"
124  :db/index true
125  :db/fulltext true
126  :db.install/_attribute :db.part/db}
127
128 {:db/id #db/id[:db.part/db]
129  :db/ident :name/NO2}
130
131 {:db/id #db/id[:db.part/db]
132  :db/ident :name/SO2}
133
134 {:db/id #db/id[:db.part/db]
135  :db/ident :name/O3}
136
137 {:db/id #db/id[:db.part/db]
138  :db/ident :name/PM2.5}
139
140 {:db/id #db/id[:db.part/db]
141  :db/ident :name/PM10}
142
```

```
{:db/id #db/id[:db.part/db]
 :db/ident :pollutant/unit
 :db/valueType :db.type/string
 :db/cardinality :db.cardinality/one
 :db/doc "The unit of measure of the pollutant"
 :db.install/_attribute :db.part/db}

{:db/id #db/id[:db.part/db]
 :db/ident :pollutant/value
 :db/valueType :db.type/long
 :db/cardinality :db.cardinality/one
 :db/doc "The measured value of the pollutant"
 :db.install/_attribute :db.part/db}

{:db/id #db/id[:db.part/db]
 :db/ident :pollutant/instant
 :db/valueType :db.type/instant
 :db/cardinality :db.cardinality/one
 :db/index true
 :db/doc "The time instant at which the data was recorded"
 :db.install/_attribute :db.part/db}]
```

Appendix C

Addenda

C.1 User Experience Survey

Cross Platform Air Qulaity Index Application for the City of Cape Town
User Interface Survey

Please rate the following aspects of the application's user interface on a scale of 1 – 10. 1 being very bad and 10 being excellent.

1) **Clarity** - with respect to ease of use. ☐

2) **Conciseness** ☐

3) **Familiarity** -with respect to the application being naturally and intuitively understood. ☐

4) **Responsiveness** - with respect to the speed and interface feedback. ☐

5) **Consistency** - with respect to the interface design. ☐

6) **Attractiveness** ☐

7) **Efficiency** - with respect to the ability to achieve goals without a 'fuss." ☐

8) **Forgiveness** - with respect to the ability to correct user mistakes. ☐

9) **Informativeness** - the informativeness of the user interface with respect to air quality information. ☐

General Comments/Feedback

_____

_____

_____

_____

_____

_____

_____

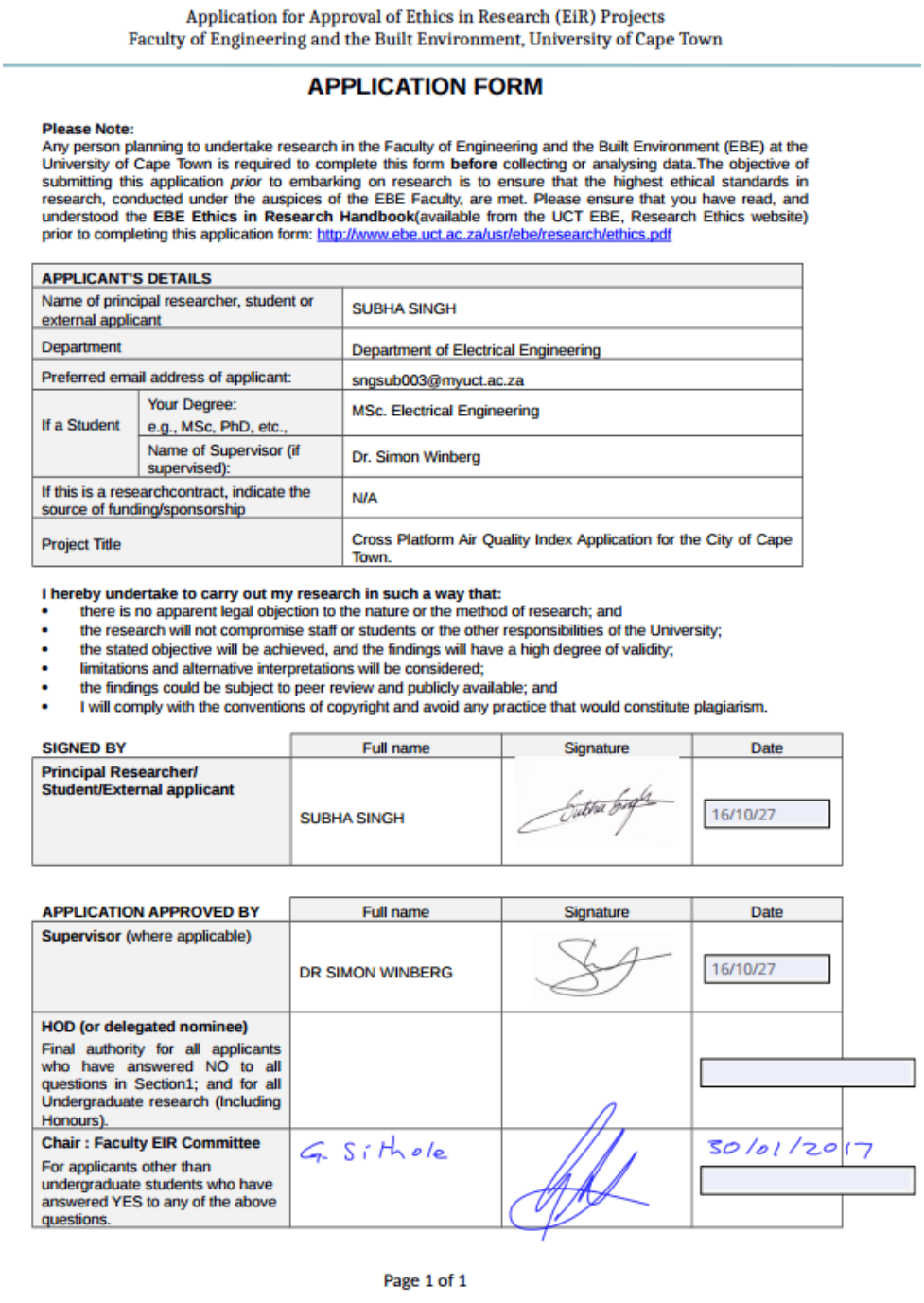
Figure C.1: User experience survey.

C.2 Ethics Forms

The ethics application was accepted with the following comments made by the reviewers:

This is a low risk project.

The signed ethics form may be seen in the figure below.



Application for Approval of Ethics in Research (EiR) Projects
Faculty of Engineering and the Built Environment, University of Cape Town

APPLICATION FORM

Please Note:
Any person planning to undertake research in the Faculty of Engineering and the Built Environment (EBE) at the University of Cape Town is required to complete this form **before** collecting or analysing data.The objective of submitting this application *prior* to embarking on research is to ensure that the highest ethical standards in research, conducted under the auspices of the EBE Faculty, are met. Please ensure that you have read, and understood the **EBE Ethics in Research Handbook**(available from the UCT EBE, Research Ethics website) prior to completing this application form: http://www.ebe.uct.ac.za/usr/ebe/research/ethics.pdf

APPLICANT'S DETAILS		
Name of principal researcher, student or external applicant		SUBHA SINGH
Department		Department of Electrical Engineering
Preferred email address of applicant:		sngsub003@myuct.ac.za
If a Student	Your Degree: e.g., MSc, PhD, etc.,	MSc. Electrical Engineering
	Name of Supervisor (if supervised):	Dr. Simon Winberg
If this is a researchcontract, indicate the source of funding/sponsorship		N/A
Project Title		Cross Platform Air Quality Index Application for the City of Cape Town.

I hereby undertake to carry out my research in such a way that:

- there is no apparent legal objection to the nature or the method of research; and
- the research will not compromise staff or students or the other responsibilities of the University;
- the stated objective will be achieved, and the findings will have a high degree of validity;
- limitations and alternative interpretations will be considered;
- the findings could be subject to peer review and publicly available; and
- I will comply with the conventions of copyright and avoid any practice that would constitute plagiarism.

SIGNED BY	Full name	Signature	Date
Principal Researcher/ Student/External applicant	SUBHA SINGH		16/10/27

APPLICATION APPROVED BY	Full name	Signature	Date
Supervisor (where applicable)	DR SIMON WINBERG		16/10/27
HOD (or delegated nominee) Final authority for all applicants who have answered NO to all questions in Section1; and for all Undergraduate research (Including Honours).			
Chair : Faculty EIR Committee For applicants other than undergraduate students who have answered YES to any of the above questions.	G. Sithole		30/01/2017

Page 1 of 1

Figure C.2: Signed ethics form.