Developing a Web-Based Microservice Application for Visualising and Analysing Climate Change

A dissertation submitted in partial fulfilment of
the requirements for the degree of
COMPUTER SCIENCE WITH A YEAR IN INDUSTRY (BSC)
in
The Queen`s University of Belfast

by
Mr Aidan McKendry
{25/04/2023}

SCHOOL OF ELECTRONICS, ELECTRICAL ENGINEERING and COMPUTER SCIENCE

CSC3002- COMPUTER SCIENCE PROJECT

Dissertation Cover Sheet

A signed and completed cover sheet must accompany the submission of the Software Engineering dissertation submitted for assessment.

Work submitted without a cover sheet will NOT be marked.

Student Name:	**Aidan McKendry**
Student Number:	**40234153**
Project Title:	**Developing a web-based microservice application for visualising and analysing climate change**
Supervisor:	**Dr Zheng Li**

Declaration of Academic Integrity

Before submitting your dissertation, please check that the submission:

1. Has a full bibliography attached laid out according to the guidelines specified in the Student Project Handbook
2. Contains full acknowledgement of all secondary sources used (paper-based and electronic)
3. Does not exceed the specified page limit
4. Is clearly presented and proof-read
5. Is submitted on, or before, the specified or agreed due date. Late submissions will only be accepted in exceptional circumstances or where a deferment has been granted in advance.

By submitting your dissertation, you declare that you have completed the tutorial on plagiarism at http://www.qub.ac.uk/cite2write/introduction5.html and are aware that it is an academic offence to plagiarise. You declare that the submission is your own original work. No part of it has been submitted for any other assignment and you have acknowledged all written and electronic sources used.

6. If selected as an exemplar, I agree to allow my dissertation to be used as a sample for future students. (Please delete this if you do not agree.)

Signature: Aidan McKendry Date: {25/04/2023}

Acknowledgements

I would like to take this opportunity both personally and professionally to express my extreme gratitude to, Dr Zheng Li for his guidance, patience, and support on supervising me whilst undertaking this dissertation, Developing a Web-based Microservice Application for Visualising and Analysing Climate Change.

I would also like to acknowledge and extend my appreciation to all the lecturers and support staff within the Electronics, Electrical Engineering and Computer Science faculty, for their teaching, coaching and help developing my knowledge and skills. I am indebted to you all, on reaching this final stage of completing my BSc Hons Degree of Computer Science in Queens University, Belfast. I would like to give a special mention to the Student Wellbeing Team on their fabulous support they provide for struggling students.

Lastly, I am forever thankful of my loving family who have supported me massively during my education. Especially, my eldest brother who has inspired me greatly in undertaking this career pathway.

Abstract

The move to an available web-based product is an appealing solution to many types of software. With GIS processing large datasets is required for many of the functions in displaying and analysing data. This is a problematic issue when using a centralised database and can lead to performance issues. This project aims to use a microservice based architecture to mitigate these problems while incorporating multiple database technologies and programming languages.

Contents

1.0 Introduction and Problem Area

1.1 Background

A Geographic Information System is a software tool and has a range of uses in both private and public sector organisations such as development planning, resource management and is critical used in geographic scientific research. GIS systems have been defined by National Geographic as: “a computer system for capturing, storing, checking, and displaying data related to positions on Earth’s surface.” [2] GIS systems consist of a range functionality enabling users to manage, analyse and visualise data sets according to the users’ needs. For example, managing and visualising the flow and location of all rivers and bodies of water in the UK. Without this technology it would be labour intensive for users to compare data across different locations and time periods, making it difficult to gain the same data-driven insights without extensive manual work.

The versatility possible with this technology makes it ideal for a wide range of industries in the private and public sector including but not limited to agriculture, development and urban planning, environmental agencies, and retail. [21] GIS systems are used extensively in academic studies as a powerful resource in academic research with its ability to display data for scientific investigations; while ArcGIS is the main software used by professionals. [22]
The following academic papers utilise ArcGIS for visualising and analysing geospatial data. [8], [9], [10] There are many papers which use this technology for geographic research, however, this is not an exhaustive list.

The web-based GIS product that will be developed in this project will be designed to implement a microservice-based architecture and the main associated patterns and concepts. In this architecture style small individual components form services, utilising API calls between each other to make up a greater overall software system. These independent components are designed to only deal with a certain aspect of the system, which in theory should help with organisation of the product/ system as everything has its designed space.

1.2 Problem description

Traditionally GIS systems are built as desktop applications, commonly packaged as large monolithic style software products. These applications have significant hardware requirements for RAM in running the product and hard disk space for storing large amounts of data locally. This has initial financial cost implications to set up and establish these platforms operationally for organisations and businesses. Any organisation that wants to use this technology need a significant investment in hardware and managing a large, centralised database.

UDC Digital Utility Solutions which specialise in GIS-based software have quoted start up fees for using GIS as being as high as $500,000 to $800,000. [3] If insufficient investment is put into the hardware components, desktop GIS applications will have much longer response times and reduce its effectiveness in practice. If the centralised database is overloaded with data and not sufficiently maintained this will also result in a much slower application. Therefore, requiring training/ dedicated database maintenance teams in companies that are required to use these applications. This lack of accessibility has led to desktop-based GIS systems becoming outdated and caused a spike in demand for web and cloud GIS solutions. In monolithic web-based architecture it is common to have one large, centralised database for storing all data for use in the application [23]. With a GIS product that stores a vast amount of data this can lead to performance issues while storing and retrieving data from a single database server [24].

By implementing microservice architecture concepts will mitigate the hardware requirements by providing an available web-based application with improved performance. This presents an opportunity to make GIS technology more accessible for entities that previously wouldn't have been able to access this technology due to cost constraints. In implementing this architecture style many of the microservice design principles will be followed such as, single responsibility and API driven design. [25] While the main concept to be explored through this project is using multiple read only databases following the decentralized data management principle. This database-per-service strategy will offer the benefits of much faster response times when the system is used at a large scale along with much simpler data management. [26]

2.0 Related work

There are many existing academic papers which research areas around this problem domain. Some technical background has been provided as context, along with some notable areas identified from previous works that inspired this study and are explored further below.

2.1 Technical background

Container technology

Virtualisation is the process of creating a virtual version of a software resource as a level of abstraction, which has been key in the development of cloud computing. Traditionally this was achieved by using virtual machines emulating a physical computer. Docker container technology was launched in 2013 and allows for virtualisation at an OS level, which is a lighter weight alternative to virtual machines. This permits an agile software development through creation of servers or microservices and does not require a hypervisor to communicate with. Therefore, it is versatile in enabling applications to be run either on desktop or the cloud.

REST API

Representational State Transfer is a software architecture introduced by Computer Scientist Roy Fielding [27]. This architectural style allows programs hosted on servers or containers to communicate with each other. This concept was derived from the four basic operations for persistent storage in a relational database management system; Create, Read, Update and Delete (CRUD) operations. Furthermore, REST defines a set of protocols an API must follow. These rules were later implemented by RESTful APIs in the form of HTTP requests. This type of API is very common in cloud computing today.

2.2 Development of a GIS app

The development of a GIS system can become quite complex when working with geospatial data and several domain models for storing datasets. 'Developing GIS Applications with Objects: A Design Patterns Approach' uses the Object-Oriented Programming (OOP) concept along with a single database for storing data layer. [28] The design incorporates precise UML class diagrams that cover the full range of classes used in modelling geographic locations.

2.3 Microservice architecture

‘The transition from monolithic architecture to microservice architecture’ [6] documents a case study in switching to a microservice architecture for a finance product. It adheres to many of the microservice patterns such as a decentralised database following the database per service concept.

Scaling of microservices can be performed in three dimensions, X axis (horizontal duplication), Y axis (functional decomposition) and Z axis (data partitioning). The relationship between the three scaling dimensions of microservice architecture and performance are deliberated in ‘Towards Quantifiable Boundaries for Elastic Horizontal Scaling of Microservices’. [18] The number of replicas of a given microservice improves the overall performance. This is subsequently optimised when a combination of X and Z axis scaling are utilised. This is attained in the paper by experimenting on stateless and stateful microservices by using Kubernetes to scale MongoDB instances across these two dimensions in such a way that limits costs and reduces the makespan. This is defined as reference. (“Time in seconds which a combination needs to finish an experiment”).

2.4 Microservice architecture GIS system

Cloud based GIS systems are increasing in popularity mainly due to the benefits of better availability and efficiency for the user. There are a growing number of research papers around this topic, implementing a variety of microservice architecture concepts. Mete and Yomralioglu (2020) created a cloud-based GIS platform which implements a serverless architecture. [19] This research conducted an experiment with Amazons Aurora Serverless database and compared the average response times against a relational database. The results found that, response times were 2.72 times faster when using serverless technology.

3.0 System Requirements and Specification

3.1 Solution description

The aim of this project is to develop a web-based GIS system which benefits from microservice architecture concepts for a responsive system. This GIS system will be used in the final demonstration for visualising and analysing temperature change patterns in the USA. In conjunction with, wildfire severity change data and weather station locations from which the temperature records dataset was retrieved in each state. The temperature records dataset was acquired from the US Government National Weather Service. It holds a vast number of records for each state, ranging from temperature records commenced in 1900 to present day. The main reasons for selecting America for this demonstration is the availability and the abundance of raw data. Furthermore, the scale and size of America allows evaluation of the temperature change over a broader range of factors such as the distance from the equator or distance from the coast from which the data was acquired from.

Each layer of the GIS system will implement an MVC architecture approach along with the Single Database Pattern. MVC architecture refers to the splitting of a service into three distinct components which serve a single purpose: Model (application model for holding a schema representing the dataset), View (GUI for displaying to the user) and Controller (manipulates data for propagating to view component). [11] The single database pattern concept then defines the concept that each service will be the master of its own database. [12] This concept allows each microservice to be completely independent of one another as they do not share a centralised database.

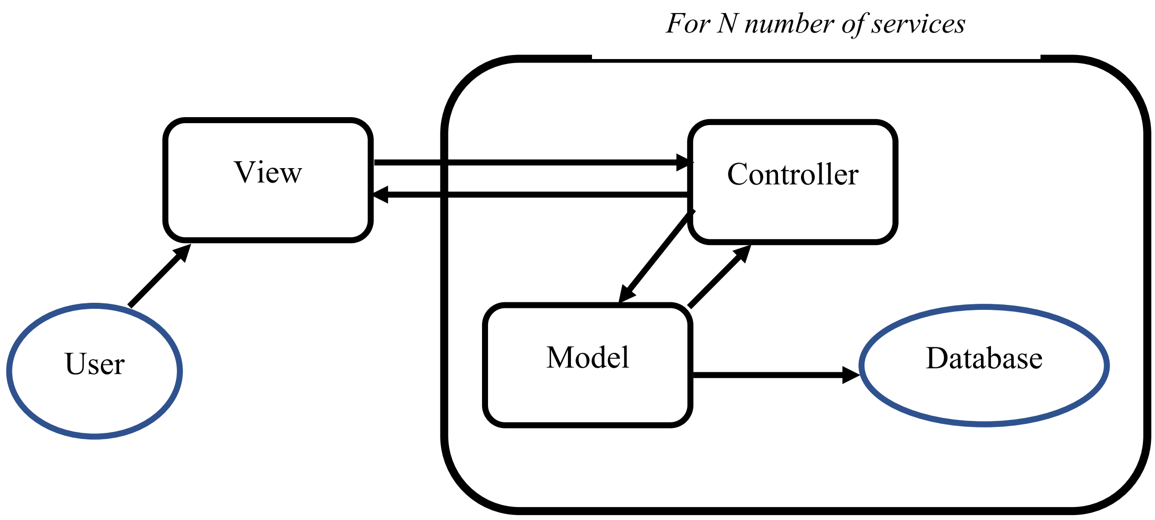


Fig. 1: MVC architecture

Fig. 1, shows how the view is unaware of the model component. This is an example, of the low degree of coupling but a high degree of cohesion which are cornerstone to a satisfactory microservice system [3]. The low degree of coupling is evident since each module only has one dependency. Thus, giving a simple flow for requests to sequentially progress through each module of that microservice. Once the request reaches the database level, the data saved in the database can be extracted through the model module. This is then propagated up through the levels again, where some manipulation of data can be performed at the controller layer (dependent on what the purpose of the service is) and then finally the response is sent to the view layer where the data will be displayed to the user that initiated the request.

3.2 Functional and non-functional system requirements

The MoSCoW template was used for prioritising both the functional requirements. [4]

Must have:

- A base-map using the Google Maps API.
- Data held in each service will be used to populate and dynamically style individual states for the map.
- Users will be able to interact with the map (zoom in, zoom out, drag cursor to move around the map) while maintaining data visualisation positions using dynamic clustering.
- Front end should add selection on year of recorded data being displayed, either playing through a range of years or just setting the year they would like to visualise the data for.
- Multiple database technologies implemented for different microservices.

Should have:

- Multiple layers for viewing datasets using different GIS data visualisation techniques and can be active simultaneously.
- Several GIS functions to aid with analysis, data management and distance.
- Users will be able to select which year of data they wish to have visualised on the map.
- Multiple programming languages used for different microservices.

Could have:

- Access for any device connected to internet, i.e., smartphone/ tablet access to application.

- Deployment via Kubernetes with auto scaling of services.

Won't have:

- User can not add their own datasets from the app as this will need to be done during creating the database image.
- Database write access so datasets will not be editable by users.

Non-functional requirements:

- Follow Microservice architecture concepts (single responsibility principle, high cohesion and low coupling).
- User will not have any hardware requirements.
- Fast response times for all GIS operations.
- Polyglot persistence and scaling of services through data partitioning and functional decomposition.
- Potential to easily scale services further by horizontal duplication in future works.

3.3 Advantages of solution

A microservice based web application utilising container technology eliminates the hardware requirements for users by hosting the database and application within a docker container which store the database and application remotely. Taking away from this financial cost barrier of hardware should make GIS systems a more efficient option to many organisations as hosting of the application is achieved remotely. Therefore, the only requirement for the users' machine is to display the view component, along with the base map. The financial cost will drop significantly for end users, with the subsequent benefit of providing this software via any machine that has internet access.

The use of Y and Z axis scaling of these services should also result in better performance. [13] Z axis scaling will be used to perform data partitioning on the temperature change records. Y-axis scaling will be implemented by splitting the microservices by functionality; meaning there will be a service which holds just the weather station locations and another service which holds the agricultural data and their respective controllers. This will allow the ability to scale up areas of the system such as; backend functions for visualising data during calls with large datasets.

This scalability in all dimensions is only made possible by utilising microservice architecture and would not be possible in a monolithic architecture system.

One of the core principles of microservice architecture, is the single responsibility principle. This means a service should deliver a single function as a requirement without relying on other services heavily. This contains the individual business logic for this piece of functionality in its own service. Keeping the business logic exclusively to its own service, should increase the simplicity of workflows through the system and therefore making it easier to understand.

3.4 Technology stack

Development environment

Docker will be used to create the database and application containers. While docker-compose, will be utilised to handle the orchestration of each service. Since these containers will all use a Linux operating system (Ubuntu 18.04 or 20.04), it will be helpful to have a dedicated Linux environment by using Oracle VirtualBox. Atom will also be accessed for programming IDE as it subsequently supports a vast number of programming languages.

Databases

A range of database software's will be implemented for holding the datasets of the GIS system to show independence of each individual microservice. MongoDB is a non-relational database and will be chosen as the database for each of the temperature change services. MariaDB and MySQL, are both relational databases and will store the weather station locations and wildfire severity change datasets.

Programming languages

The shell scripts will be written in the Unix-native Bash CLI language. The server-side applications for communicating with the database containers will be programmed in NodeJS and Python. NodeJS, will be chosen for the application communicating with MongoDB since it uses JSON (JavaScript Object Notation), this will make interpreting the datasets more seamless. A combination of JavaScript and HTML will be used to develop the frontend of the GIS application.

3.5 GIS functions

The system will implement three different types of GIS functions that are commonly used for analysing and displaying data. The user should be able to interact to create and combine a choropleth and graduated symbol maps, for analysing temperature change patterns across states.

Choropleth maps

A choropleth map is useful for analyzing statistical data relative to location by colouring an area of a country, state or county based on a given dataset. The example Fig.2 uses this type of map to present datasets on ageing populations and growing support on healthcare in Europe. This is done with an intuitive colour scheme where the locations are shaded darker where the average age is greater. [29]

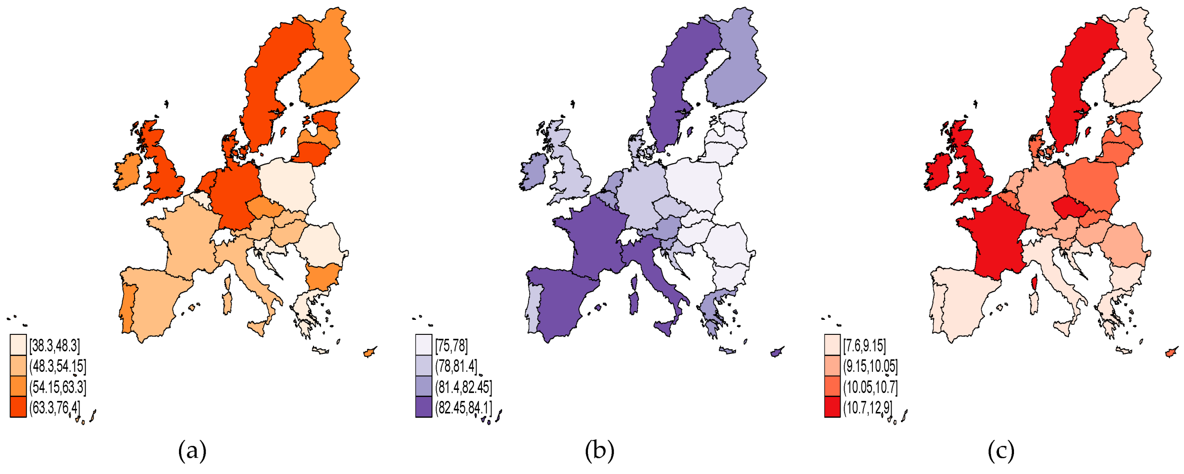


Fig. 2 – Choropleth map example

Graduated symbol maps

A graduated symbol map is useful for displaying quantitative data by having the size of a given symbol relate to units of measurement. The example Fig.3 shows energy consumption in metric tonnes for each country in Europe. [30]

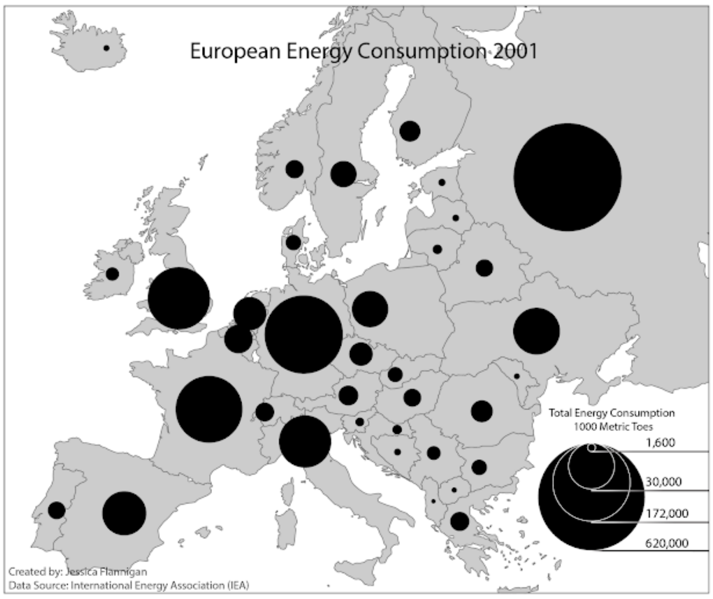

European Energy Consumption 2001
Total Energy Consumption
1000 Metric Toes
1,600
30,000
172,000
620,000
Created by: Jessica Flannigan
Data Source: International Energy Association (IEA)

Fig. 3 – Graduated symbol map example

4.0 Design

4.1 Design approach

As mentioned in the previous section, the powerful MVC architecture will be used as the approach to the system architecture. The view component will be shared between each service, while the models and controllers will vary depending on which microservice they are implementing. In addition to the scalability and understandability benefits that this architecture provides, each microservice should be lightweight which will allow data to be transferred and visualised quickly for the user.

To provide further detail on the description of the system architecture, Krutchens' 4+1 view model of architecture has been used as a blueprint. [14] This breaks down the system architecture into five concurrent views, logical, process, development, physical and scenario. The GIS system will not implement any concurrent processes; therefore, the process view has been omitted. Further the system has not been deployed to a cloud environment opting to be hosted locally instead so the development/ deployment view has been removed.

4.2 Logical view

A UML diagram Class diagram was designed to show how each service will communicate between the front end and internally with their own classes.

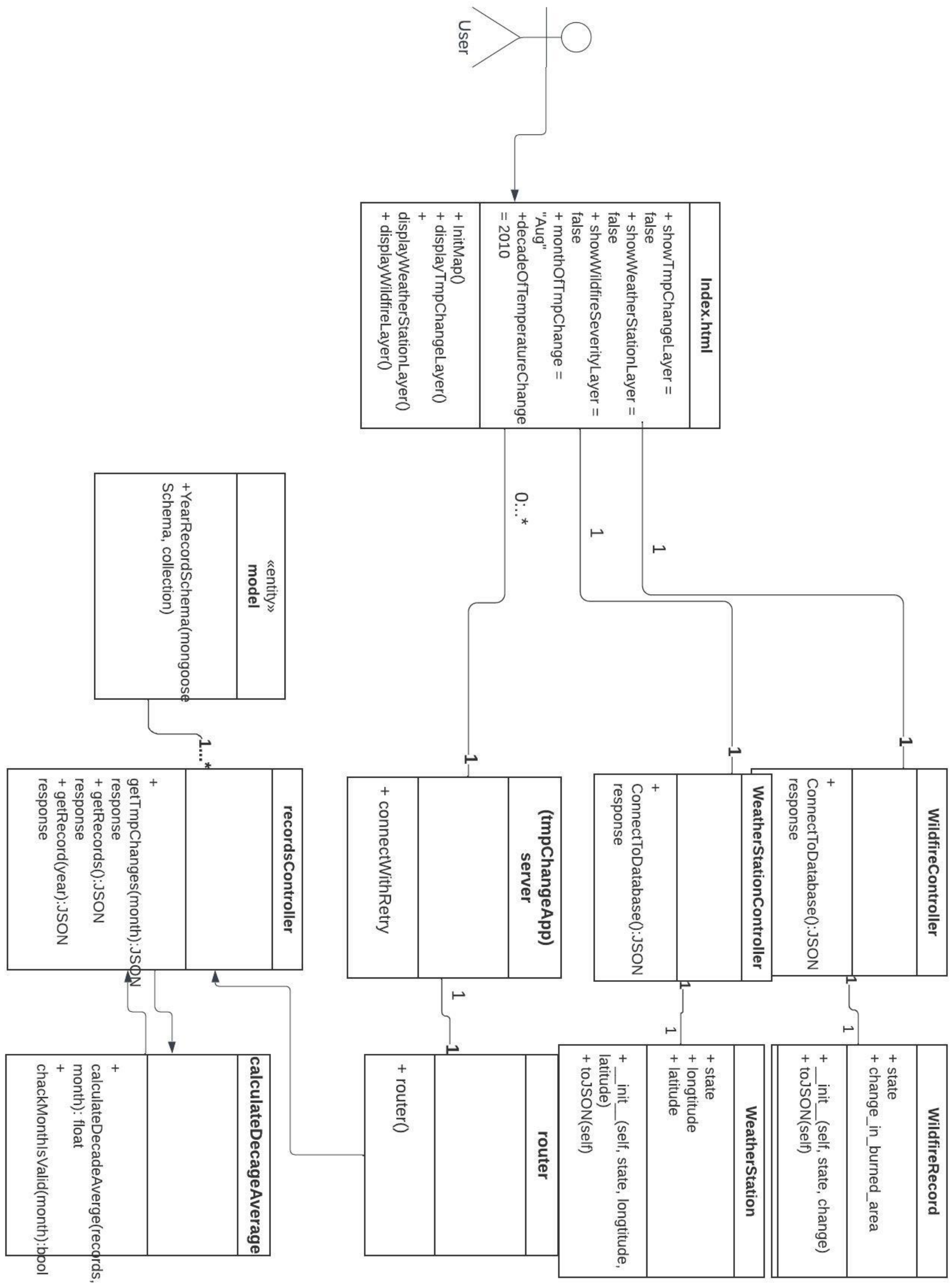

User
Index.html
+ showTmpChangeLayer = false
+ showWeatherStationLayer = false
+ showWildfireSeverityLayer = false
+ monthOfTmpChange = "Aug"
+decadeOfTemperatureChange = 2010
+ InitMap()
+ displayTmpChangeLayer()
+ displayWeatherStationLayer()
+ displayWildfireLayer()
0...*
1
1
«entity» model
+YearRecordSchema(mongoose Schema, collection)
1...*
recordsController
+ getTmpChanges(month):JSON response
+ getRecords():JSON response
+ getRecord(year):JSON response
(tmpChangeApp) server
+ connectWithRetry
WeatherStationController
+ ConnectToDatabase():JSON response
WildfireController
+ ConnectToDatabase():JSON response
calculateDecageAverage
+ calculateDecadeAverge(records, month): float
+ chackMonthIsValid(month):bool
router
+ router()
WeatherStation
+ state
+ longtitude
+ latiude
+ __init__(self, state, longitude, latitude)
+ toJSON(self)
WildfireRecord
+ state
+ change_in_burned_area
+ __init__(self, state, change)
+ toJSON(self)
1
1
1
1
1
1
1
1

Fig. 4 – UML Class Diagram

4.3 Development view

This view is useful for programmers and Software management. I have split up the following package diagram to be separated distinctly into the View, Controller and Model layers to visualise which part of the system resides where in the MVC model.

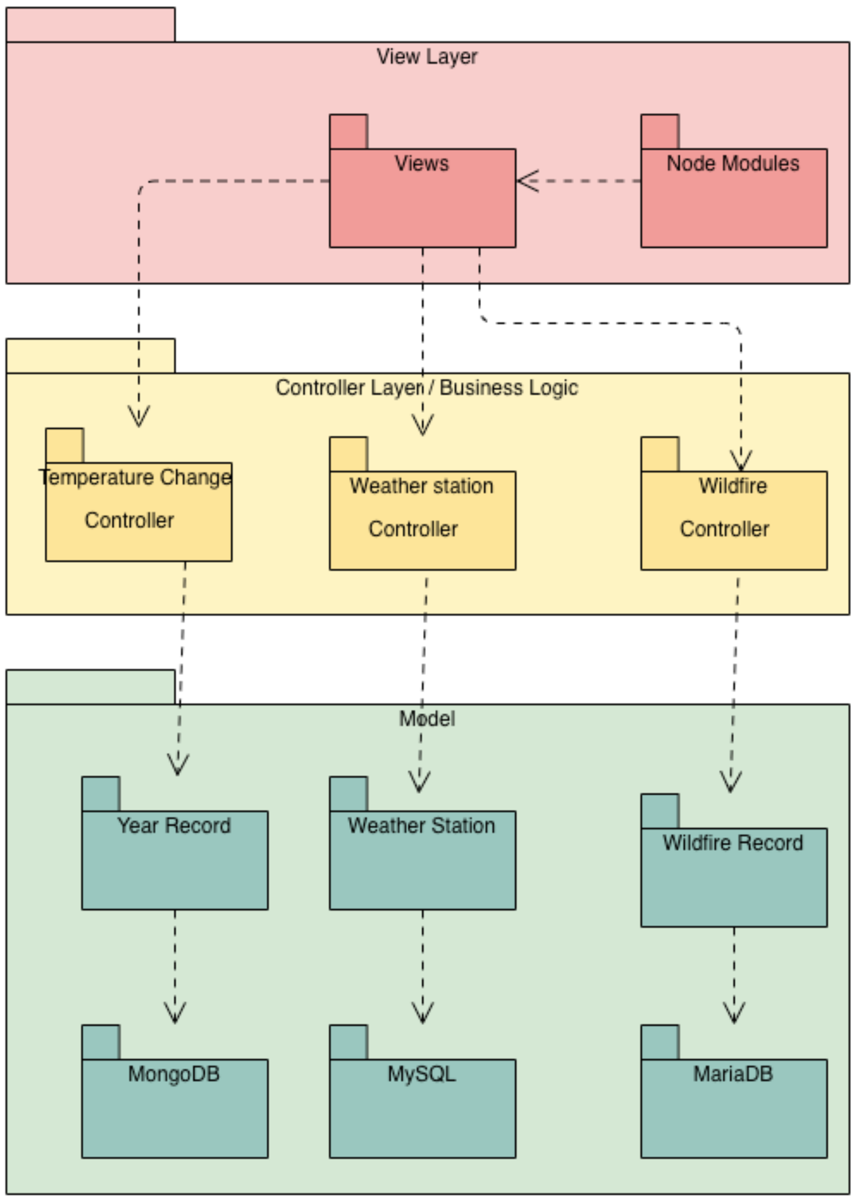


Fig. 5 – Package Diagram

4.5 Scenario view

Examples of all use cases for a user of the GIS system have been designed through sequence diagrams for displaying each layer. Each layer should contact its specific service for retrieving the data needed in the view component.

Temperature change layer

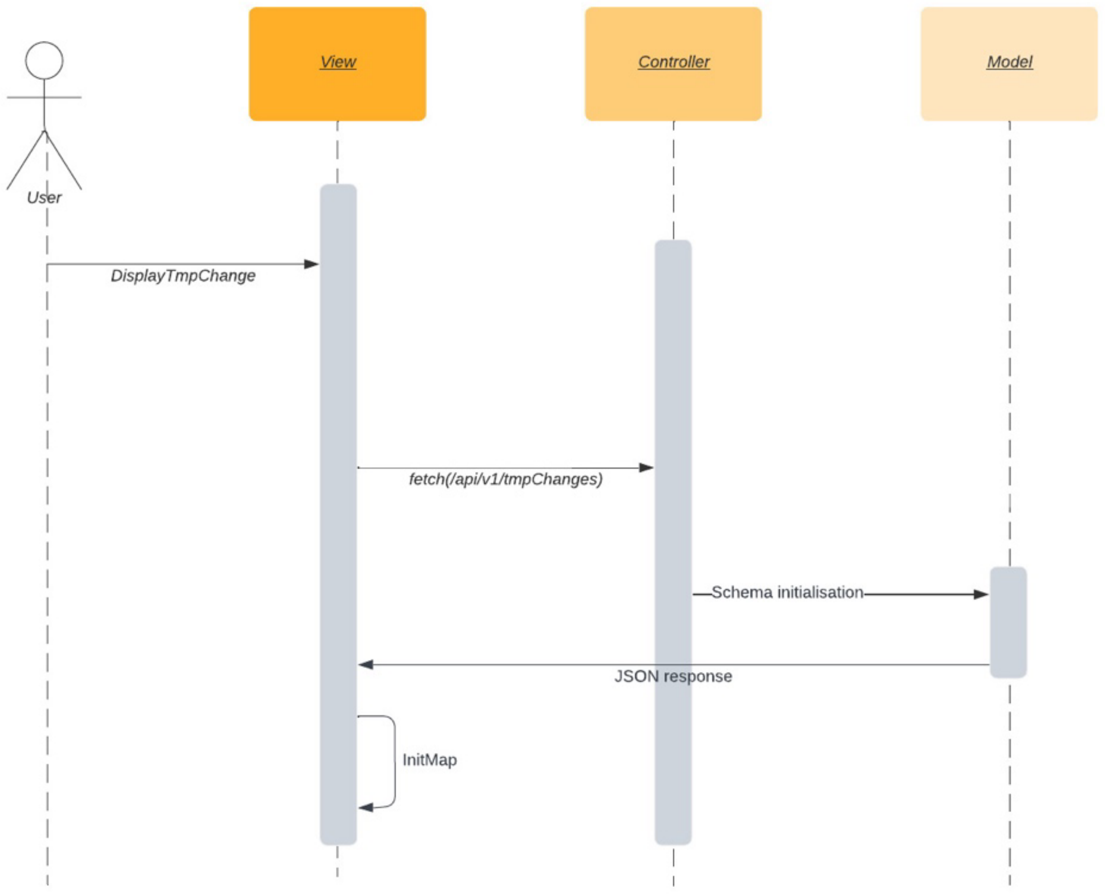


Fig. 6– Temperature Change layer sequence diagram

Wildfire layer

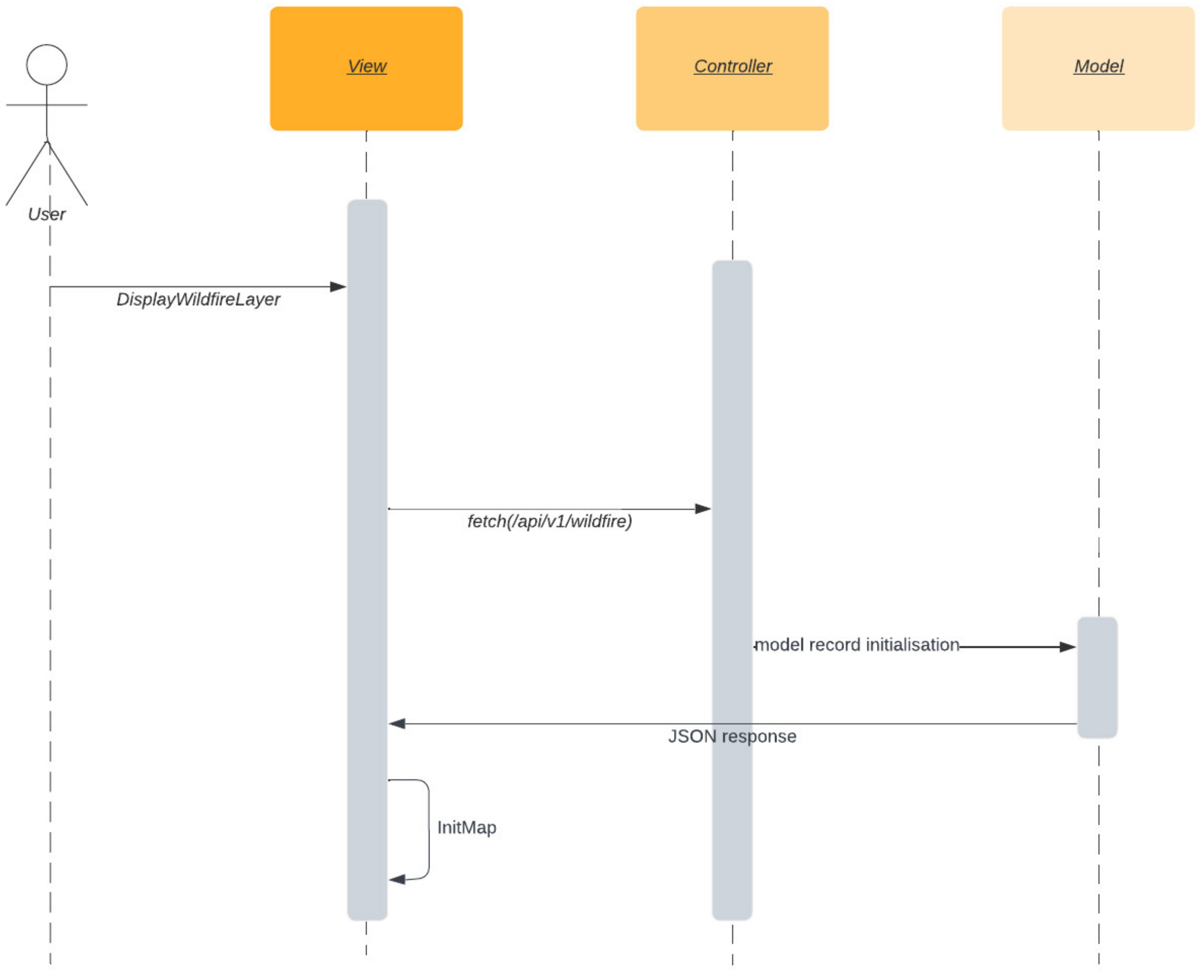


Fig. 7– Wildfire layer sequence diagram

Wildfire layer

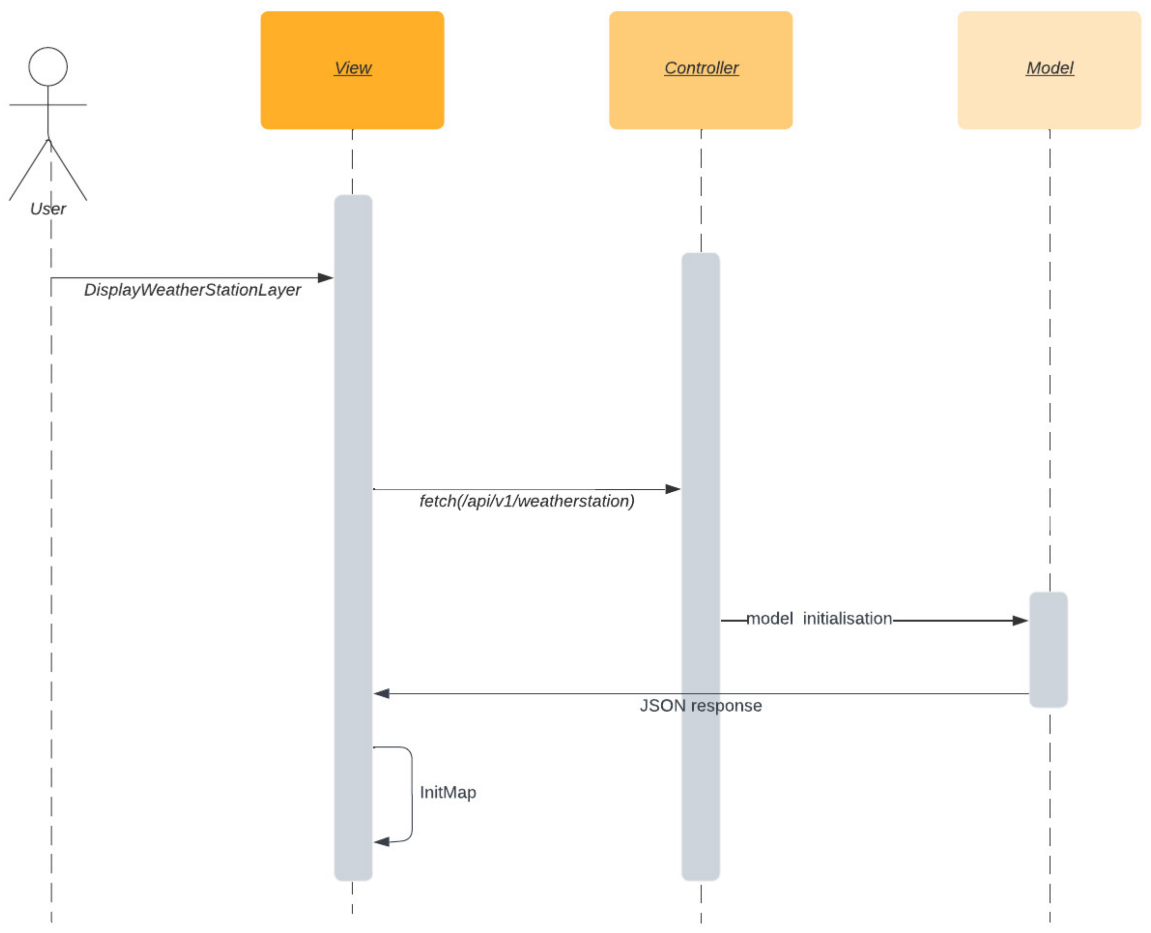


Fig. 8– Temperature Change layer sequence diagram

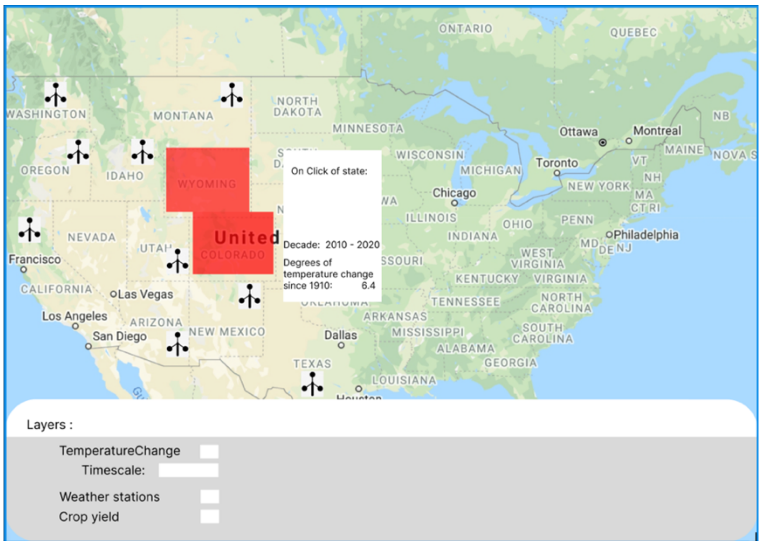
4.6 Detailed design

Front end design

The front-end of the GIS app is comprised of two components: the base map and the options for visualising data in layers on top of the map.

For the base map the Google maps API will be selected as it is lightweight and therefore, it is less likely to have issues with memory usage. It also has some very useful additional features for visualising the datasets. A Google Maps API Key will be generated, this will be used to authorise the connections for features still in beta such as data driven styling. [16] Data-driven styling implements a data visualisation layer on top of the base map by simply calling the FeatureLayer interface from a html file. This can be specified to divide a country, on multiple different boundaries to varying levels of locality. For the demonstration's purposes, the constant feature type of administrative area level 1 will be used to divide up the USA into a choropleth map, with customizable fill colour for each state. [17]

The Google Maps platform also allows for detailed map customization which could be useful for designing a base map that will not clash with the data layers. It is quite simple to implement in the front-end application, a code snippet of the API is provided in Table 1. Fig. 4 shows a mock-up using Figma to illustrate what the front end will look like in the final system.



On Click of state:

Decade: 2010 - 2020

Degrees of temperature change since 1910: 6.4

Layers :

TemperatureChange

Timescale:

Weather stations

Crop yield

Fig. 9 – Front-end mock-up

index.html

```
…
 <script type="text/javascript"
     async defer
          src="https://maps.googleapis.com/maps/api/js? key={APIKEY}&v=beta&&callback=initMap">
   </script>
…
```

Table 1: Index.html code snippet

Database design

The GIS system will feature the concept of polyglot persistence; meaning multiple different data storage technologies will be used to hold our datasets. [7] Each service can use a different database technology or programming language demonstrating how each microservice shall be implemented with complete independence of any other services.

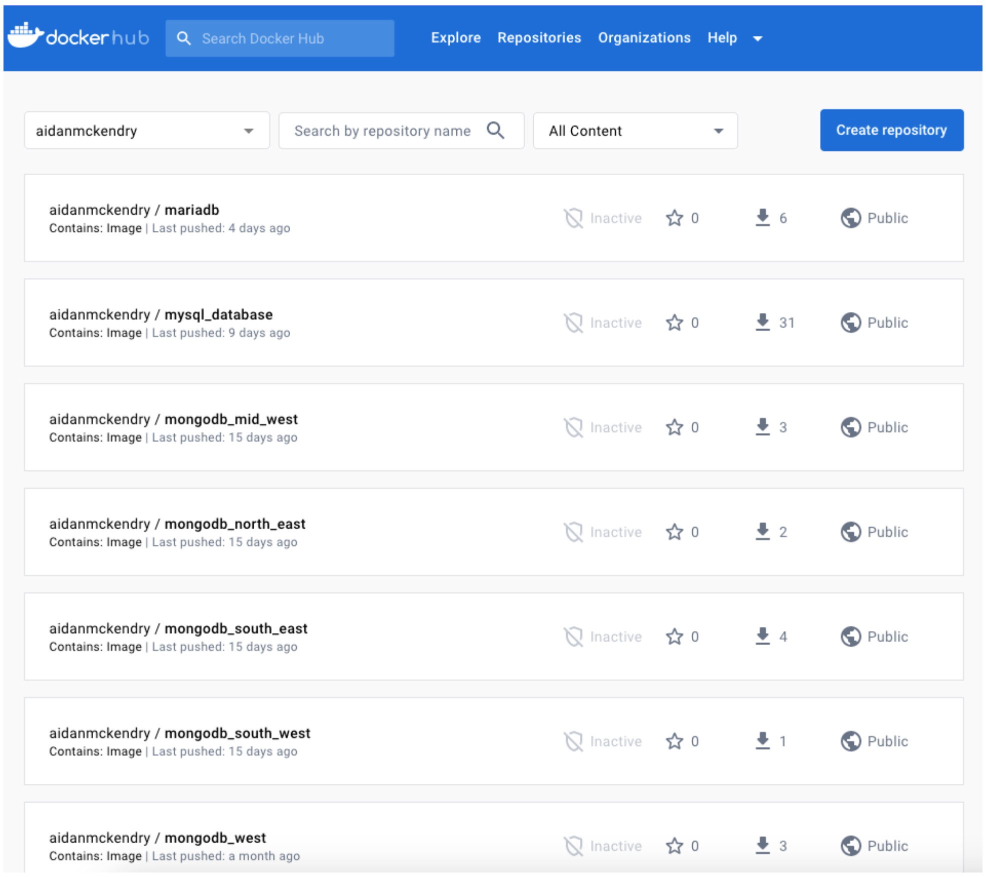
As mentioned in the previous section, the microservices will be scaled on both the Y and Z axis through data partitioning and functional decomposition. Each microservice will be comprised of a database and application for container. Data partitioning will be applied to the temperature change services allowing this aspect of the system to be subdivided by region. There should therefore be a total of five services: Northeast, Southeast, Southwest, Midwest, and West working in tandem to achieve this functionality with each regional database holding temperature records since 1900. The view component will be able to communicate with these databases through API calls and combine responses together to finally be displayed for the user. For the functional decomposition aspect, there will be two additional distinct services: one for displaying all weather station locations where this data was retrieved and another for displaying the states which have has a significant increase in wildfires over the past 20 years.

The database software should be pre-installed into a container image before it is started without the need for user interaction. This process is also known as silent installation [15]. Since this project is solely focusing on visualising a pre-defined dataset, each database should be read only. These datasets can also be imported directly into the database during image creation. This creates a straightforward experience for any users of the GIS system since it can be started quickly without having to set up any configuration.

5.0 Implementation

5.1 Database container creation

While setting up the database containers, a virtual Linux environment was set up through Oracle VirtualBox. On creation of the Docker container, the database software should be automatically installed and loaded with our datasets without user interaction. The container and database should be made available for connections from other applications hosted either locally or in containers of their own. Once each database containers scripts for installation and configuration were finalised, the final image was pushed to my personal dockerhub repository as shown in Fig.5. [32] This helped manage the images and has in-built version controlling so the docker-compose files for each of the services will always be pulling the latest image to run off.



Repository	Details	Status	Stars	Pulls	Visibility
aidanmckendry / mariadb	Contains: Image | Last pushed: 4 days ago	Inactive	0	6	Public
aidanmckendry / mysql_database	Contains: Image | Last pushed: 9 days ago	Inactive	0	31	Public
aidanmckendry / mongodb_mid_west	Contains: Image | Last pushed: 15 days ago	Inactive	0	3	Public
aidanmckendry / mongodb_north_east	Contains: Image | Last pushed: 15 days ago	Inactive	0	2	Public
aidanmckendry / mongodb_south_east	Contains: Image | Last pushed: 15 days ago	Inactive	0	4	Public
aidanmckendry / mongodb_south_west	Contains: Image | Last pushed: 15 days ago	Inactive	0	1	Public
aidanmckendry / mongodb_west	Contains: Image | Last pushed: a month ago	Inactive	0	3	Public

Fig. 10 – Dockerhub images

5.1.1 MongoDB

MongoDB is a popular NoSQL document-oriented database software which saves data in collections of JSON objects. For the creation of this pre-loaded database image 4 files are needed:

Dockerfile

This text file is a series of instructions that allows Docker to build an image. The following file is an example which implements the database for the western region temperature change records dataset.

Dockerfile

```
FROM ubuntu:20.04
RUN apt-get update -y && apt-get install -y gnupg
RUN apt-get update -y && apt-get install -y ca-certificates
RUN apt-get update -y && apt-get install -y wget
RUN apt-get update -y && apt-get install -y systemctl

COPY ./install_mongodb.sh /app/install_mongodb.sh
COPY ./load_data.sh /app/load_data.sh
COPY ./select_data.sh /app/select_data.sh
COPY ./TmpRecordsWest.csv /app/TmpRecordsWest.csv
WORKDIR /app
RUN chmod a+x ./install_mongodb.sh ./load_data.sh ./install_mongodb.sh ./select_data.sh
RUN ./install_mongodb.sh
RUN rm -f ./install_mongodb.sh
RUN ./load_data.sh
RUN mkdir -p /data/db
CMD ./select_data.sh

EXPOSE 27017
```

Table 2: Dockerfile

Silent installation

The unattended installation of MongoDB is shown below. The version is specified at 4.4.15 to maintain consistency in the version the container uses. The sed command on the last line was crucial to allowing connections from other apps hosted inside another container, it enables unrestricted binding from any IP address for connecting to the database container.

install_mongodb.sh

```
#!/bin/bash
apt-get update
apt-get install ca-certificates
```

```
apt-get install gnupg
apt-get install wget

wget -qO - https://www.mongodb.org/static/pgp/server-4.4.asc | apt-key add -
echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/4.4 multiverse" | tee /etc/apt/sources.list.d/mongodb-org-4.4.list
apt-get update
apt-get install -y mongodb-org=4.4.15 mongodb-org-server=4.4.15 mongodb-org-shell=4.4.15 mongodb-org-mongos=4.4.15 mongodb-org-tools=4.4.15

mongod --version
systemctl stop mongod
sed -i '/bindIp*/c \  bindIpAll: true' /../etc/mongod.conf
```

Table 3: Install_mongodb.sh

Loading datasets

The example below imports the dataset ‘TmpRecordsWest’ from a csv file into the database with the mongoimport command. This dataset should be unique to each database container we have as each service will implement either a different functionality of the GIS system or another region featuring a different partition of the temperature records.

load_data.sh

```
#!/bin/bash

systemctl start mongod
systemctl status mongod
mongoimport --type=csv --headerline --file=TmpRecordsWest.csv
systemctl stop mongod
```

Table 4: load_data.sh

Internal start of the service

This script is executed when the container is started with the CMD command. This starts the service and the database logs holding information on current operations within the database service or any errors that occur, as well as keeping the container running.

select_data.sh

```
#!/bin/bash

systemctl start mongod
tail -F /../var/log/mongodb/mongod.log
```

Table 5: select_data.sh

5.1.2 MariaDB

MariaDB is a popular open-source relational database management system that is highly compatible with MySQL. It was decided to use this as the data storage for the wildfire datasets.

Dockerfile

The Dockerfile for installing MariaDB is similar to the MongoDB container, it runs the required installation and set-up shell scripts in order. The only dependency for this container was to update the apt repository in the virtual linux environment for downloading systemctl commands for managing the service.

Dockerfile

```
FROM ubuntu:20.04

RUN apt-get update -y && apt-get install -y systemctl

COPY . /home/diicc/mydockerbuild
WORKDIR /home/diicc/mydockerbuild
RUN apt-get update -y
RUN apt-get install --assume-yes apt-utils
RUN chmod a+x ./install_mariadb.sh ./load_data.sh ./select_data.sh
RUN ./install_mariadb.sh
RUN ./load_data.sh
RUN rm -f ./install_mariadb.sh
CMD ./select_data.sh
```

Table 6: Dockerfile (MariaDB)

Silent installation

The unattended installation shell file for MariaDB is provided [A.9]. The correct version and corresponding release signing key are installed firstly then the service can be started. The high compatibility with MySQL allows for using the secure_our_mysql.sh file for securing the database with some basic security set up.

Loading datasets

When loading in the wildfire dataset, we first must create the database, user and configure their access rights. Then the MySQL_script is run creating a table with two fields to match the dataset format. Once this is completed the service is stopped.

load_data.sh

```
service mariadb start

rootpsw='1q2w3e4r'
user='mariadbuser'
pass='#1A2b%3C4d5E!'
table='table'

mysql -uroot -p$rootpsw <<MYSQL_SCRIPT
CREATE DATABASE $user;
CREATE USER'$user'@'localhost'IDENTIFIED BY'$pass';
CREATE USER'$user'@'%'IDENTIFIED BY'$pass';
GRANT ALL PRIVILEGES ON *.* TO'$table'@'localhost'WITH GRANT OPTION;
GRANT ALL PRIVILEGES ON *.* TO'$table'@'%'WITH GRANT OPTION;
FLUSH PRIVILEGES;
MYSQL_SCRIPT

echo "MySQL user created."
echo "Username:   $user"
echo "Password:   $pass"

# BELOW COLUMN NAMES AND DATA TYPES FOR DATASET
mysql <<MYSQL_SCRIPT
USE $user;
CREATE TABLE $table (STATE VARCHAR(30), CHANGE_IN_BURNED_AREA
VARCHAR(15));
LOAD DATA LOCAL INFILE'/home/diicc/mydockerbuild/wildfires.txt'INTO TABLE
$table CHARACTER SET utf8 FIELDS TERMINATED BY'\t'LINES TERMINATED
BY'\n';
MYSQL_SCRIPT

service mariadb stop
```

Table 7: load_data.sh (MariaDB)

Internal start of the service

select_data.sh

```
service mariadb start

tail -F /var/log/mysql/error.log
```

Table 8: select_data.sh (MariaDB)

5.1.3 MySQL

The files required the silent installation of MySQL were provided in full by Dr. Zheng Li. This relational database management software has been implemented to demonstrate the loose coupling between the different database containers and their services. MySQL has been implemented to host the weather station locations dataset.

Dockerfile

This database container runs on Ubuntu version 18.04 and runs the required scripts in the order specified in the dockerfile below.

Dockerfile

```
FROM ubuntu:18.04

COPY . /home/diicc/mydockerbuild
WORKDIR /home/diicc/mydockerbuild

RUN apt-get update -y
RUN apt-get install --assume-yes apt-utils
RUN chmod a+x ./install_mysql.sh ./load_data.sh ./select_data.sh
RUN ./install_mysql.sh
RUN ./load_data.sh
RUN rm -f ./install_mysql.sh

CMD ./select_data.sh
```

Table 9: Dockerfile (MySQL)

Silent installation

The installation of MySQL was fairly similar to that of MariaDB and follows many of the same steps such as securing the database through generating the secure_our_mysql.sh [A.10].

Loading datasets

In loading the dataset into the table we again specify the fields that will be needed for storing the weather station location dataset with the state, longitude and latitude.

load_data.sh

```
service mysql start

rootpsw='1q2w3e4r'
usertest='mydb'
```

```
passtest='#1A2b%3C4d5E!'
tabletest='mytab'

mysql -uroot -p$rootpsw <<MYSQL_SCRIPT
CREATE DATABASE $usertest;
CREATE USER'$usertest'@'localhost'IDENTIFIED BY'$passtest';
CREATE USER'$usertest'@'%'IDENTIFIED BY'$passtest';
GRANT ALL PRIVILEGES ON *.* TO'$usertest'@'localhost'WITH GRANT OPTION;
GRANT ALL PRIVILEGES ON *.* TO'$usertest'@'%'WITH GRANT OPTION;
FLUSH PRIVILEGES;
MYSQL_SCRIPT

echo "MySQL user created."
echo "Username:   $usertest"
echo "Password:   $passtest"

# BELOW COLUMN NAMES AND DATA TYPES FOR DATASET
mysql -u$usertest -p$passtest -D$usertest <<MYSQL_SCRIPT
CREATE TABLE $tabletest (STATE VARCHAR(30), LATITUDE VARCHAR(10),
LONGTITUDE VARCHAR(10));
LOAD DATA LOCAL INFILE'/home/diicc/mydockerbuild/WeatherStations.txt'INTO
TABLE $tabletest CHARACTER SET utf8 FIELDS TERMINATED BY'\t'LINES
TERMINATED BY'\n';
MYSQL_SCRIPT

service mysql stop
```

Table 10: load_data.sh (MySQL)

Internal start of the service

select_data.sh

```
service mysql start


tail -F /var/log/mysql/error.log
```

Table 11: select_data.sh (MySQL)

5.2 Two container applications

5.2.1 App for the MongoDB database containers

Node JS is a JavaScript framework, being used here to develop a server-side application that implements the Model and Controller components of the MVC system architecture, for any service which uses MongoDB as a database. Since MongoDB saves our dataset in JavaScript Object Notation (JSON), it is helpful to use a JavaScript based development language.

The express framework has been used as the middleware for this web application. As seen in the server file, we define the app as an express application and link it to the routes file where we define all the services endpoints. [A.1] [A.2] The server file is also responsible for connecting the application to the database. The Mongoose library is imported from the node modules folder, then the Mongoose connect method is used to establish the database connection within a promise retry until the connection is successful.

Model implementation

The Model component is only responsible for managing the data that we get from the database. [A.3] To achieve this, the model package was created to keep it distinct from the other files. This model class then only has one dependency to the Mongoose library for creating a schema in JSON format of the records we expect to get from the database. Finally, we set the collection that this data should be retrieved from inside MongoDB. This allows us to make simple queries to the database and save the responses as a JSON object for performing further operations.

Controller implementation

Within the controller package the getTmpChanges endpoint is used for returning the temperature change data to the view component. A database query for the oldest average temperature records and the selected decade for temperature change are used to calculate the change in temperature over the specified time period. Error handling is implemented in 404 not found http status code response if the query parameters are invalid or a 500-response code if there are any unexpected errors. [A.4]

Docker compose

Docker compose is an orchestration tool for connecting each database service and their respective server applications. This tool requires a yaml file which defines and links the database and application containers together into one single multi container application. [A.11] For ease of use I decided to align the start of all of the temperature change regions through this one file.

5.2.2 App for the MySQL and MariaDB database containers

For the application containers that communicate with the MySQL and MariaDB database containers, a Python Flask web app was created. Python was used to demonstrate how each application is independent and can be written in its own language due to the loose coupling between services.

Model implementation

The model for this class defines a record object which matches the dataset fields. This object is initialised during the database query within the controller. There is also a method which converts the object into a JSON string used to send the final response. An example of how this was done in the wildfire service is shown in [A.8].

Controller implementation

The controller for the python applications is responsible for connecting to the database and performing the query for retrieving the data. An example is listed in [A.9] of how the controller performs a query simply for all data present in the table.

5.3 Front-end microservice

Implements the view component of the MVC system architecture. This component is shared between all the server-side applications. [A.10] This front-end component is comprised of a html file with an embedded JavaScript script which holds the logic for communicating with the backend services through a fetch command to each of the microservices endpoints. It uses Boolean values to handle what layers should be shown on top of the base map which uses the Google maps API as mentioned in the system design. For the user interaction with these layers I decided to put all options inside a collapsible button so it can be hidden. [33]

6.0 Testing

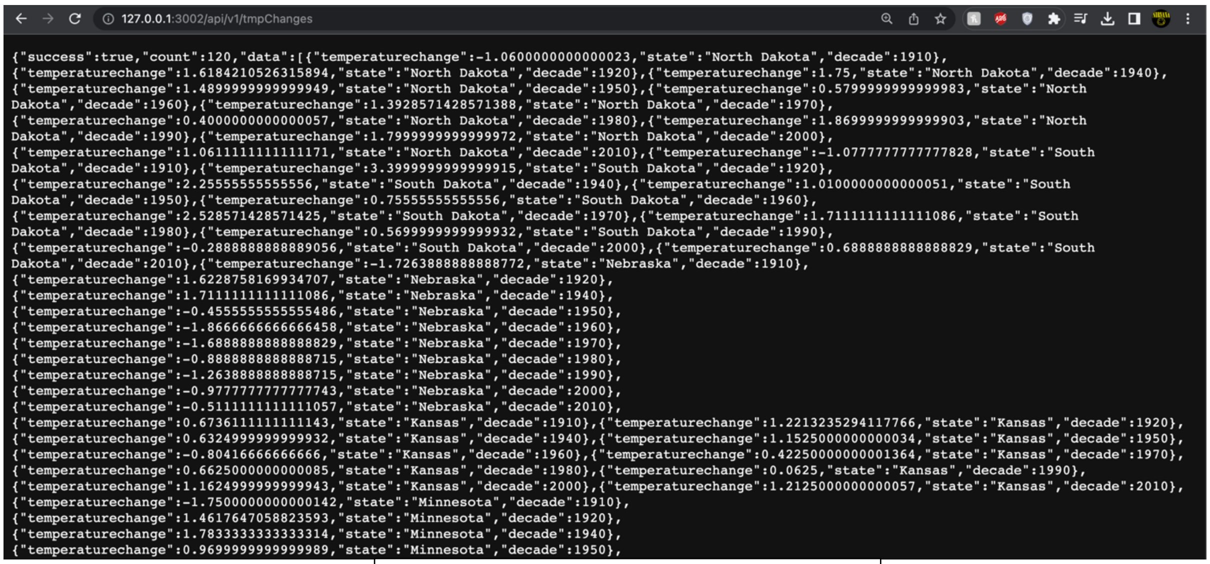
6.1 Manual tests

Manual testing was the main testing method used for the system during development. This allowed for executing white-box, black-box, and integration test cases.

While developing the scripts for setting up the database containers, the easiest way to test if a script did the intended set up correctly was to create the docker container by running each script in order. If there were any errors in commands within the script, this would be returned in the stack trace once the error occurred. During this stage of development, it was advised to keep detailed documentation of each command within the script. The documentation kept a record of the purpose of each command which was crucial to have as a reference once inevitably bugs appeared. This practice is very common within DevOps. [31] All troubleshooting attempts were recorded in the documentation as a reference of what hasn't worked as intended when trying to find a potential solution online. Researching the problems extensively online and eliminating these bugs was necessary in performing the set-up of the databases correctly to be instantly available with no user interaction on starting the container.

Integration testing was performed manually to ensure that each application container and database container were able to connect and communicate successfully. This was done by executing the docker compose file which started the multi container applications in one command from the terminal. The documentation was helpful during the integration testing of the MongoDB since there was difficulty successfully connecting the two containers. One of the bugs that occurred during development was identified using the documentation around the configuration of the MongoDB database since would not allow connections from other remote applications. To solve this the sed command for setting the bind IP to any port (0.0.0.0) on creation of the database image.

Black-box verification testing was performed by loading each services endpoint through the localhost loopback interface, returning the JSON data that was to be received by the view component in the MVC architecture. An example of this is shown in fig.6 below. Logging requests in the console was also a useful debugging tool as this was possible to track the progress of user requests through each component of the MVC architecture.



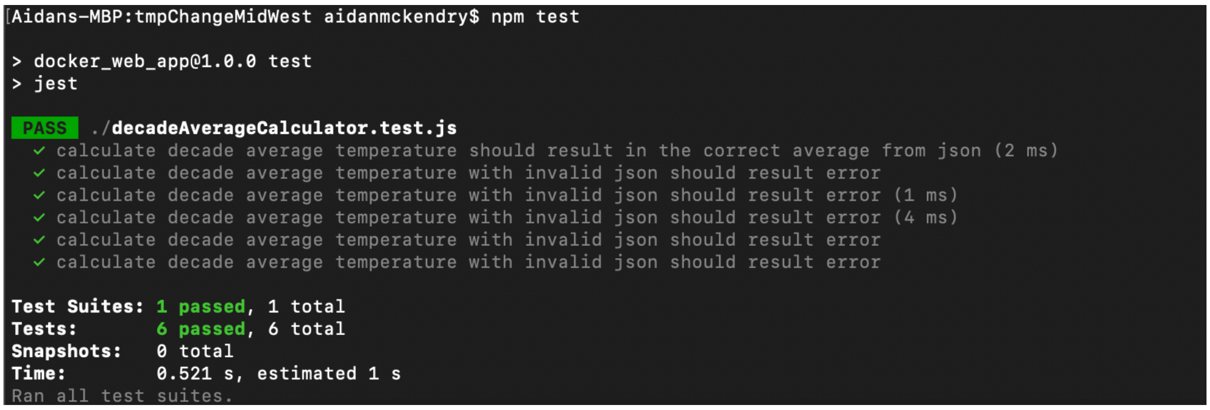
```
{"success":true,"count":120,"data":[{"temperaturechange":-1.0600000000000023,"state":"North Dakota","decade":1910},
{"temperaturechange":1.6184210526315894,"state":"North Dakota","decade":1920},{"temperaturechange":1.75,"state":"North Dakota","decade":1940},
{"temperaturechange":1.4899999999999949,"state":"North Dakota","decade":1950},{"temperaturechange":0.5799999999999983,"state":"North
Dakota","decade":1960},{"temperaturechange":1.3928571428571388,"state":"North Dakota","decade":1970},
{"temperaturechange":0.4000000000000057,"state":"North Dakota","decade":1980},{"temperaturechange":1.8699999999999903,"state":"North
Dakota","decade":1990},{"temperaturechange":1.7999999999999972,"state":"North Dakota","decade":2000},
{"temperaturechange":1.0611111111111171,"state":"North Dakota","decade":2010},{"temperaturechange":-1.0777777777777828,"state":"South
Dakota","decade":1910},{"temperaturechange":3.3999999999999915,"state":"South Dakota","decade":1920},
{"temperaturechange":2.25555555555556,"state":"South Dakota","decade":1940},{"temperaturechange":1.0100000000000051,"state":"South
Dakota","decade":1950},{"temperaturechange":0.75555555555556,"state":"South Dakota","decade":1960},
{"temperaturechange":2.528571428571425,"state":"South Dakota","decade":1970},{"temperaturechange":1.7111111111111086,"state":"South
Dakota","decade":1980},{"temperaturechange":0.5699999999999932,"state":"South Dakota","decade":1990},
{"temperaturechange":-0.2888888888889056,"state":"South Dakota","decade":2000},{"temperaturechange":0.6888888888888829,"state":"South
Dakota","decade":2010},{"temperaturechange":-1.7263888888888772,"state":"Nebraska","decade":1910},
{"temperaturechange":1.6228758169934707,"state":"Nebraska","decade":1920},
{"temperaturechange":1.7111111111111086,"state":"Nebraska","decade":1940},
{"temperaturechange":-0.4555555555555486,"state":"Nebraska","decade":1950},
{"temperaturechange":-1.8666666666666458,"state":"Nebraska","decade":1960},
{"temperaturechange":-1.6888888888888829,"state":"Nebraska","decade":1970},
{"temperaturechange":-0.8888888888888715,"state":"Nebraska","decade":1980},
{"temperaturechange":-1.2638888888888715,"state":"Nebraska","decade":1990},
{"temperaturechange":-0.9777777777777743,"state":"Nebraska","decade":2000},
{"temperaturechange":-0.5111111111111057,"state":"Nebraska","decade":2010},
{"temperaturechange":0.6736111111111143,"state":"Kansas","decade":1910},{"temperaturechange":1.2213235294117766,"state":"Kansas","decade":1920},
{"temperaturechange":0.6324999999999932,"state":"Kansas","decade":1940},{"temperaturechange":1.1525000000000034,"state":"Kansas","decade":1950},
{"temperaturechange":-0.80416666666666,"state":"Kansas","decade":1960},{"temperaturechange":0.42250000000001364,"state":"Kansas","decade":1970},
{"temperaturechange":0.6625000000000085,"state":"Kansas","decade":1980},{"temperaturechange":0.0625,"state":"Kansas","decade":1990},
{"temperaturechange":1.1624999999999943,"state":"Kansas","decade":2000},{"temperaturechange":1.2125000000000057,"state":"Kansas","decade":2010},
{"temperaturechange":-1.7500000000000142,"state":"Minnesota","decade":1910},
{"temperaturechange":1.4617647058823593,"state":"Minnesota","decade":1920},
{"temperaturechange":1.7833333333333314,"state":"Minnesota","decade":1940},
{"temperaturechange":0.9699999999999989,"state":"Minnesota","decade":1950},
```

Fig. 11 – Endpoint response

6.2 Functional tests

Unit tests were used to test the two methods for calculating the average temperature in the controller component of the temperature change services. Input partitioning was used to define a test suite with 6 test cases which tests cover all of the methods in the calculate decade average class. This tests error handling for invalid or null values being passed into the month and JSON response parameters.

The test class was created using jest, a popular testing framework for JavaScript. This was installed into the node modules which could then be ran from the command line using npm test. This code was used as verification for each regional service. An example of this for the mid-west region is shown in Fig 5.



```
[Aidans-MBP:tmpChangeMidWest aidanmckendry$ npm test

> docker_web_app@1.0.0 test
> jest

 PASS  ./decadeAverageCalculator.test.js
  ✓ calculate decade average temperature should result in the correct average from json (2 ms)
  ✓ calculate decade average temperature with invalid json should result error
  ✓ calculate decade average temperature with invalid json should result error (1 ms)
  ✓ calculate decade average temperature with invalid json should result error (4 ms)
  ✓ calculate decade average temperature with invalid json should result error
  ✓ calculate decade average temperature with invalid json should result error

Test Suites: 1 passed, 1 total
Tests:       6 passed, 6 total
Snapshots:   0 total
Time:        0.521 s, estimated 1 s
Ran all test suites.
```

Fig. 12 – Test suite output

6.3 User acceptance tests

User acceptance testing was carried out last by testing the final system through the front-end app. Each possible user journey was covered from the user sequence diagrams such as turning on all layers at once to ensure that the view can display all data layers at once. This tested each microservice in processing the fetch requests to the JSON response endpoints.

7.0 System Evaluation

7.1 Strengths and successes

Throughout this project a lot was learned in producing a web application while following my research around the best practices for a microservice architecture system. I believe that the final system being delivered achieves all the functional requirements listed in the 'must have' and 'should have' sections above. The final application presents a dynamic map allowing customization on multiple aspects of dynamically displaying the temperature change records while maintaining dynamic clustering.

It was valuable to get experience with many of the technologies used professionally in developing web-based applications. Docker was used throughout for both hosting the containers and orchestrating the start of multi container applications consistently with docker compose.

Learning how to split up a centralized database into the database per service pattern in MVC architecture was particularly interesting. Following the architecture design in coordinating queries and concatenating the responses to be interpreted by the shared view component was another part I enjoyed, along with working with both SQL and No-SQL database technologies. The database technologies used handled queries and connections differently which was interesting and was a chance to explore developing a web app in Python and NodeJS.

I found it very rewarding getting the database containers installed and configured fully automatically, especially MongoDB. The ability to read from three different database technologies simultaneously is also a big strength of the final system which fulfils the microservice pattern of a read-only database per service.

7.2 Shortcomings and failures

Loading the map through the google maps API call was the biggest bottleneck in the final user acceptance tests. Google maps as the base map was not the most efficient solution, it is not as lightweight as hoped and another map software in place of this would have improved loading times of the final system. Due to this the functional requirement of a play button through all decades of recorded temperatures was not achievable and was instead added as a query to set the year for viewing which could still be dynamically changed through a HTML dropdown list.

The final system would have also benefited from being deployed to a cloud environment. This would have made GIS app available to any user and meant that the application can only be ran when hosting the docker container locally on the user's laptop or desktop. This has drawbacks in performance of the system because the requirement of running the containers locally means there is still hardware requirements on the users' system.

7.3 Performance and reliability

Speed of API response times is fast for the multi container applications; however, Google maps loading is the longest part of the performance taking roughly 3-4 seconds to initialise the map along with the fill colours for each state. Activating each layer does not result in long loading times so the functional requirement of short loading times has been fulfilled.

7.4 Future enhancements

Implementing automatic horizontal scaling of microservices through Kubernetes or another orchestration tool could be implemented to achieve load balancing. In addition to increasing efficiency and optimising performance, load balancing would provide additional fault tolerance, reduce single point of failure, and increase availability. This would be very beneficial to a GIS app for processing large amounts of data.

8.0 References

[1] Project git repo. [Online]. Available:
https://gitlab.eeecs.qub.ac.uk/40234153/3002_project

[2] National Geographic, "Geographic Information System", n.d. [Online]. Available:
https://education.nationalgeographic.org/resource/geographic-information-system-gis

[3] UDC, "How much does a GIS system cost?", 30th April 2021. [Online]. Available:
https://www.udcus.com/blog/2021/04/30/how-much-does-gis-system-cost

[4] Fenton and Bieman, 2014, From Monolithic Systems to Microservices, Davide Taibi. [Online]. Available:
https://pdfs.semanticscholar.org/6a46/bb3280e6f4ad87c215a2d953d8c031f828a0.pdf

[5] Stanislava Simonova, "Requirements Gathering for Specialized Information Systems in Public Administration", 2021 International Conference on Information and Digital Technologies (IDT). [Online]. Available:
https://ieeexplore.ieee.org/abstract/document/9497618

[6] Eivind-Solberg, "The transition from monolithic architecture to microservice architecture."[Online]. Available:
https://www.duo.uio.no/bitstream/handle/10852/95663/1/Master-thesis--Eivind-Solberg.pdf

[7] Noa Roy-Hubara, "Selecting databases for Polyglot Persistence applications." [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169023X21000744

[8] Johnston, K., Ver Hoef, J.M., Krivoruchko, K. and Lucas, N., 2001. *Using ArcGIS geostatistical analyst* (Vol. 380). Redlands: Esri.

[9] Childs, C., 2004. Interpolating surfaces in ArcGIS spatial analyst. Arc User, July-September, 3235(569), pp.32-35.

[10] Wong, W.S.D. and Lee, J., 2005. *Statistical analysis of geographic information with ArcView GIS and ArcGIS*. Wiley.

[11] Deacon, J., 2009. Model-view-controller (mvc) architecture. [Online]. [Citado em: 10 de março de 2006.] Available: http://www. jdl. co. uk/briefings/MVC. pdf, 28.

[12] Ryan, Michael., 2021. Microservice design patterns: single database per service. [Online]. Available: https://kenzanmedia.medium.com/microservice-design-patterns-single-database-per-service-bfa36728e862

[13] Richardson, Chris. 2023. The scale cube. [Online]. Available: https://microservices.io/articles/scalecube.html

[14] P. B. Kruchten, 1995. "The 4+1 View Model of architecture," in *IEEE Software*, vol. 12, no. 6, pp. 42-50, Nov. 1995, doi: 10.1109/52.469759.

[15] Umar Manzoor, Samia Nefti "Silent Unattended Installation Package Manager--SUIPM". [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5172599

[16] "Data-driven styling (beta)." Google Maps Platform. [Online]. Available: https://developers.google.com/maps/documentation/javascript/reference/data-driven-styling

[17] C. A. Brewer and L. Pickle. "Evaluation of Methods for Classifying Epidemiological Data on Choropleth Maps in Series". [Online]. Available: https://www.tandfonline.com/doi/epdf/10.1111/1467-8306.00310?needAccess=true

[18] Manuel Ramírez López, Josef Spillner, "Towards Quantifiable Boundaries for Elastic Horizontal Scaling of Microservices." [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/3147234.3148111

[19] Mete, M.O. and Yomralioglu, T., 2021. Implementation of serverless cloud GIS platform for land valuation. *International Journal of Digital Earth*, *14*(7), pp.836-850. [Online]. Available: https://www.tandfonline.com/doi/full/10.1080/17538947.2021.1889056

[20] Slocum, Z. and Tang, W., 2020. Integration of Web GIS with high-performance computing: a container-based cloud computing approach. *High Performance Computing for Geospatial Applications*, pp.135-157. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-47998-5_8#Sec5

[21] Anna Williams "WHAT ARE THE DIFFERENT INDUSTRIES THAT USE GIS?", [Online]. Available: https://mgiss.co.uk/what-are-the-different-industries-that-use-gis/

[22] – Enlyft "Companies using ArcGIS", [Online]. Available: https://enlyft.com/tech/products/arcgis

[23] Xing Lin, Yi Zhang, Yu Liu and Yong Gao, "Spatial data integrity ensuring mechanism in SDBMS," Proceedings. 2005 IEEE International Geoscience and Remote Sensing Symposium, 2005. IGARSS '05., Seoul, 2005, pp. 4 pp.-, doi: 10.1109/IGARSS.2005.1526260. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1526260

[24] Chaowei Yang, Huayi Wu, Qunying Huang, Zhenlong Li, Jing Li, Wenwen Li, Lizhi Miao & Min Sun, "Spatial data integrity ensuring mechanism in SDBMS "[Online]. Available:https://www.researchgate.net/publication/267388803_WebGIS_performance_issues_and_solutions

[25] Soma, "10 Microservices Design Principles and Best Practices for Experienced Developers" [Online]. Available: https://medium.com/javarevisited/10-microservices-design-principles-every-developer-should-know-44f2f69e960f

[26] Mehmet Ozkaya, "The Database-per-Service Pattern" [Online]. Available: https://medium.com/design-microservices-architecture-with-patterns/the-database-per-service-pattern-9d511b882425

[27] - Roy Fielding, 2000, Architectural Styles andthe Design of Network-based Software Architectures. University of California, Irvine. Available at: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[28] – Gordillo, S.E., Balaguer, F., Mostaccio, C.A. and Das Neves, F. “Developing GIS Applications with Objects: A Design Patterns Approach,” [Online]. Available: http://sedici.unlp.edu.ar/bitstream/handle/10915/131855/Developing_GIS_Applications_with_Objects.pdf-PDFA.pdf?sequence=1&isAllowed=y

[29] - M. Cristea, G. G. Noja, P. Stefea, and A. L. Sala, “The Impact of Population Aging and Public Health Support on EU Labor Markets,” *International Journal of Environmental Research and Public Health*, vol. 17, no. 4, p. 1439, Feb. 2020, doi: 10.3390/ijerph17041439.

[30] – Jessica Flannigan, Graduated symbol map. [Online].

Available: http://jessicaflannigan.blogspot.com/p/european-energy-consumption-2001.html

[31] – Will Kelly, “A DevOps guide to documentation” [Online]. Available: https://opensource.com/article/21/3/devops-documentation dev ops documentation

[32] – Pablo Ezequiel, “Creating a Docker image with MongoDB” [Online]. Available: https://pablo-ezequiel.medium.com/creating-a-docker-image-with-mongodb-4c8aa3f828f2

[33] – W3Schools. “ HTML Collapsible” , [Online].
Available: **https://www.w3schools.com/howto/howto_js_collapsible.asp**

9.0 Appendices

9.1 Source code examples

Appendix 1

server.js

```
const express = require("express");
const mongoose = require("mongoose");
const dotenv = require('dotenv');
const path = require('path');
const cors = require('cors');
const Router = require("./routes");

dotenv.config({path: './config/config.env'});
const app = express();
const PORT = process.env.PORT || 3000;

// Body parser
app.use(express.json());
// Enabling CORS
app.use(cors());

const options = {
  autoIndex: false,
  reconnectTries: 10,
  reconnectInterval: 500,
  poolSize: 10,
  bufferMaxEntries: 0
}

// can a .env file be created to use (process.env.NODE_ENV) to connect to mongodb either
on local address or container address.. ??
//   https://stackoverflow.com/questions/21987311/check-is-nodejs-connection-come-from-
localhost - check this
const connectWithRetry = () => {
  console.log('MongoDB connection with retry')
  // for running app locally - connect to "mongodb://127.0.0.1:37017/test"
  // for connecting app when running on container - 'mongodb://mongodb:27017/test'
  mongoose.connect('mongodb://127.0.0.1:37017/test').then(()=>{
    console.log('Connected successfully')
    mongoose.connection.db.listCollections().toArray(function (err, names) {
        console.log(names);
        module.exports.Collection = names; // exports all collection names (states) stored in
this database
    });
  }).catch(err=>{
    console.error.bind(console, "Connection Error: ");
    console.log('MongoDB connection unsuccessful, retrying in 5 seconds.')
    setTimeout(connectWithRetry, 5000)
  })
}
connectWithRetry()


app.use(Router);

app.listen(PORT, () =>
  console.log(`Server is running in ${process.env.NODE_ENV} mode at port ${PORT}`)
);
```

Table A.1: server.js

router.js

```
const express = require("express");
const { getRecords } = require('./controllers/recordsController');
const { getRecord } = require('./controllers/recordsController');
const { getTmpChanges } = require('./controllers/recordsController');
var router = express.Router();

router.route('/api/v1/records').get( getRecords );
router.route('/api/v1/record').get( getRecord );
router.route('/api/v1/tmpChanges').get( getTmpChanges );

//  below section to be moved to gateway app
router.get('/', function(req, res) {
  res.sendFile(__dirname + '/views/index.html');
})

// exports all endpoints in this file
module.exports = router;
```

Table A.2: router.js

Appendix 3

model.js

```
const mongoose = require("mongoose");

const YearRecordSchema = new mongoose.Schema({
  State: {
    type: String,
    required: true,
    },
  Year: {
    type: Number,
    required: true,
    default: 1900,
    },
  Jan: {
    type: Number,
    required: false,
  },
  Feb: {
    type: Number,
    required: false,
  },
  Mar: {
    type: Number,
    required: false,
  },
  Apr: {
    type: Number,
    required: false,
  },
  May: {
    type: Number,
    required: false,
  },
  Jun: {
    type: Number,
    required: false,
  },
  Jul: {
    type: Number,
    required: false,
  },
  Aug: {
    type: Number,
    required: false,
  },
  Sep: {
    type: Number,
    required: false,
  },
  Nov: {
    type: Number,
    required: false,
  },
  Dec: {
    type: Number,
    required: false,
  },
  Annual: {
    type: Number,
    required: true,
  }
},
{collection: 'TmpRecordsWest'})

module.exports.TmpRecords = mongoose.model("TmpRecordsWest",
YearRecordSchema);
```

Table A.3: model.js

Appendix 4

recordsController.js

```
const mongoose = require('mongoose');
const calculateDecadeAverage = require('./calculateDecadeAverage');
var { TmpRecords } = require('../models/models');
var { Collection } = require('../server');
var _ = require("underscore");

// @description GET single records
// @route GET /api/v1/records/:year || GET /api/v1/record?year=value
// @access Public
exports.getRecord = async (req, res, next) => {
  let { year } = req.query;
  let query = {};
  if (year) query.year = year

  try {
    const tmprecords = await TmpRecords.find({Year: { $eq: year}});
    return res.status(200).json({
      success: true,
      count: tmprecords.length,
      data: tmprecords
    });
  } catch (error) {
    console.error(error);
    res.status(500).json({
      error: 'Server error occurred for year: ' + year
    });
  }
};
// @description GET tmp changes for all states by decade decades
// @route GET /api/v1/tmpChanges
// @access Public
exports.getTmpChanges = async (req, res, next) => {
  let { month } = req.query;
  console.log("month provided as parameter: ", month)
  if (!calculateDecadeAverage.checkMonthIsValid(month)) {
    return res.status(400).json({
      success: false,
      error: "Invalid month given as parameter"
    });
  }

  var stateList = ["Pennsylvania", "New York", "Vermont", "Maine", "New Hampshire",
"Massachusetts", "Rhode Island", "New Jersey"];
  var query = {};

  var decadesToCalculate = [1910, 1920, 1940, 1950, 1960, 1970, 1980, 1990 , 2000, 2010];
  var endOfDecadesToCalculate = [1920, 1940, 1950, 1960, 1970, 1980, 1990 , 2000, 2010,
2020];
  var jsonResponseData = [];
  try {
    // loop for calculating each states temperature change by decade
    for (state in stateList) {
      // add parameter for date range in calculations ... instead of just decade 5 year & 20 year
date span options
      // will need check on oldest records returned, if response is empty, search for next date
range
      const oldestStateRecords = await TmpRecords.find({ Year : { $gte :  1900, $lt : 1910},
State: [stateList[state]], [month]:{$type: 1}});
      console.log("records being passed to calculate method", oldestStateRecords);
      var                                oldestAverage                                =
calculateDecadeAverage.calculateDecadeAverage(oldestStateRecords, month);

      for (decade in decadesToCalculate) {
        query[month] = {$type: 1}; // filters out missing records - not of type double
        query["State"] = stateList[state]; // filters query by correct to correct state only
        query["Year"]   =   {   $gte   :   decadesToCalculate[decade],   $lt   :
endOfDecadesToCalculate[decade]}; // filters query by decade range
        var decadeTmpRecords = await TmpRecords.find(query);

        decadeAverageTmp                                                        =
calculateDecadeAverage.calculateDecadeAverage(decadeTmpRecords, month);
        tmpChange = decadeAverageTmp - oldestAverage;

        let tmpChangeToJson = {'temperaturechange': tmpChange, 'state': stateList[state],
'decade': decadesToCalculate[decade]};
        // console.log(tmpChangeToJson)
        jsonResponseData = jsonResponseData.concat(tmpChangeToJson);
        // console.log(jsonResponseData);
      }
    }
    //const jsonContent = JSON.stringify(jsonResponseData);
    return res.status(200).json({
      success: true,
      count: jsonResponseData.length,
      data: jsonResponseData
    });
  } catch (error) {
    console.error(error);
    res.status(500).json({
      error: 'Server error occurred'
    });
  }
};

// @description GET all records
// @route GET /api/v1/records
// @access Public
exports.getRecords = async (req, res, next) => {

  try {
    const tmprecords = await TmpRecords.find({});
    return res.status(200).json({
      success: true,
      count: tmprecords.length,
      data: tmprecords
    });
  } catch (error) {
    console.error(error);
    res.status(500).json({
      error: 'Server error occurred'
    });
  }
};
```

Table A.4: recordsController.js

Appendix 5

calculateDecadeAverage.js

```
const express = require("express");
const { getRecords } = require('./controllers/recordsController');
const { getRecord } = require('./controllers/recordsController');
const { getTmpChanges } = require('./controllers/recordsController');
var router = express.Router();

router.route('/api/v1/records').get( getRecords );
router.route('/api/v1/record').get( getRecord );
router.route('/api/v1/tmpChanges').get( getTmpChanges );


// exports all endpoints in this file
module.exports = router;
```

Table A.5: calculateDecadeAverage.js

Appendix 6

Dockerfile

```
FROM node:16

WORKDIR /usr/src/app
COPY package*.json ./
RUN npm install
COPY . .

EXPOSE 3000
CMD [ "node", "server.js" ]
```

Table A.6: Dockerfile

Appendix 7

WildfireControllerApp.py

```
import mariadb
from model.WildfireRecord import WildfireRecord
import logging
from flask import Flask, jsonify, Response
import json
n = 2

app = Flask(__name__)
@app.route('/')

def connect_to_database():
    while n >= 0:
        connection = mariadb.connect(
            host='mariadb',
            port=3306,
            user='mydb',
            password='#1A2b%3C4d5E!',
            database='mydb'
        )
        cursor = connection.cursor()

        # db query
        cursor.execute("SELECT * FROM mytab")
        rowHeaders = [x[0] for x in cursor.description]
        allResults = cursor.fetchall()

        data = []
        for result in allResults:
            app.logger.info('%s: result', result)
            change_in_burned_area = result[1].replace('\r', '')
            record = WildfireRecord(result[0], change_in_burned_area)
            jsonifiedRecord = record.toJSON()
            data.append(jsonifiedRecord)

        connection.close()
        # send json response
        r =  {
            "success": True,
            "count": len(data), # should return correct count of json objects, weather station
records ...
            "data": data
        }
        reply = json.dumps(r)
        response = Response(response=reply, status=200, mimetype='appplication/json')
        response.headers["Content-Type"]="application/json"
        response.headers["Access-Control-Allow-Origin"]="*"
        return response



if __name__ =='__main__':
    app.run(debug=True, host='0.0.0.0',port=3007)
```

Table A.7: WildfireControllerApp.py

Appendix 8

wildfireRecord.py

```
import json
class WildfireRecord:
    def __init__(self, state, change_in_burned_area):
        self.state = state
        self.change_in_burned_area = change_in_burned_area
    def toJSON(self):
        return f"""{{"state": { f"'{self.state}'"}, "change_in_burned_area": {self.change_in_burned_area}}}"""
```

Table A.8: wildfireRecord.py

Appendix 9

install_mysql.sh

```
export DEBIAN_FRONTEND=noninteractive

MYSQL_ROOT_PASSWORD='1q2w3e4r'

echo debconf mysql-server/root_password password $MYSQL_ROOT_PASSWORD | debconf-set-selections
echo debconf mysql-server/root_password_again password $MYSQL_ROOT_PASSWORD | debconf-set-selections

apt-get -qq install mysql-server > /dev/null # Install MySQL quietly
sed -i '/bind-address*/c\bind-address=0.0.0.0' /etc/mysql/mysql.conf.d/mysqld.cnf # setting the bind ip to allow remote connections


service mysql start

apt-get -qq install expect > /dev/null

tee ~/secure_our_mysql.sh > /dev/null << EOF
spawn $(which mysql_secure_installation)
expect "Enter password for user root:"
send "$MYSQL_ROOT_PASSWORD\r"
expect "Press y|Y for Yes, any other key for No:"
send "y\r"
expect "Please enter 0 = LOW, 1 = MEDIUM and 2 = STRONG:"
send "2\r"
expect "Change the password for root ? ((Press y|Y for Yes, any other key for No) :"
send "n\r"
expect "Remove anonymous users? (Press y|Y for Yes, any other key for No) :"
send "y\r"
expect "Disallow root login remotely? (Press y|Y for Yes, any other key for No) :"
send "y\r"
expect "Remove test database and access to it? (Press y|Y for Yes, any other key for No) :"
send "y\r"
expect "Reload privilege tables now? (Press y|Y for Yes, any other key for No) :"
send "y\r"
expect "Reload privilege tables now? (Press y|Y for Yes, any other key for No) :"
send "y\r"
EOF

expect ~/secure_our_mysql.sh

rm -v ~/secure_our_mysql.sh # Remove the generated Expect script
```

```
echo "MySQL setup completed. Insecure defaults are gone. Please remove this scriptmanually when you are done with it (or at least remove the MySQL root passwordthat you put inside it."

sed -i's/127.0.0.1/0.0.0.0/g'/etc/mysql/mysql.conf.d/mysqld.cnf
sed -i'/max_allowed_packet*/c\max_allowed_packet=1073741824'/etc/mysql/mysql.conf.d/mysqld.cnf
sed -i'/key_buffer_size*/c\key_buffer_size=100M'/etc/mysql/mysql.conf.d/mysqld.cnf
sed -i'/max_connections*/c\max_connections=400'/etc/mysql/mysql.conf.d/mysqld.cnf
sed -i'/\[mysqld\]/a\# Skip reverse DNS lookup\nskip-name-resolve'/etc/mysql/mysql.conf.d/mysqld.cnf

service mysql stop
```

Table A.9: install_mysql.sh

Appendix 10

install_mariadb.sh

```
#!/bin/bash
export DEBIAN_FRONTEND=noninteractive
MYSQL_ROOT_PASSWORD='1q2w3e4r'

 apt-get update
 apt-get install software-properties-common -y
 apt-key adv --fetch-keys 'https://mariadb.org/mariadb_release_signing_key.asc'
 add-apt-repository 'deb [arch=amd64,arm64,ppc64el]
https://mirror.netcologne.de/mariadb/repo/10.6/ubuntu bionic main'

# Install MariaDB
 apt-get update
 apt-get install mariadb-server -y

 echo "Installation complete"

service mariadb status
service mariadb start
service mariadb enable
service mariadb status

apt-get -qq install expect > /dev/null

tee ~/secure_our_mysql.sh > /dev/null << EOF
spawn $(which mysql_secure_installation)
expect "Enter password for user root:"
send "$MYSQL_ROOT_PASSWORD\r"
expect "Press y|Y for Yes, any other key for No:"
send "y\r"
expect "Please enter 0 = LOW, 1 = MEDIUM and 2 = STRONG:"
send "2\r"
expect "Change the password for root ? ((Press y|Y for Yes, any other key for No) :"
send "n\r"
expect "Remove anonymous users? (Press y|Y for Yes, any other key for No) :"
send "y\r"
expect "Disallow root login remotely? (Press y|Y for Yes, any other key for No) :"
send "y\r"
expect "Remove test database and access to it? (Press y|Y for Yes, any other key for No)
:"
send "y\r"
expect "Reload privilege tables now? (Press y|Y for Yes, any other key for No) :"
send "y\r"
expect "Reload privilege tables now? (Press y|Y for Yes, any other key for No) :"
send "y\r"
EOF

expect ~/secure_our_mysql.sh
```

```
rm -v ~/secure_our_mysql.sh # Remove the generated Expect scrip

sed -i '/bind-address*/c\bind-address=0.0.0.0' /etc/mysql/mariadb.conf.d/50-server.cnf #
setting the bind ip to allow remote connections
sed -i '/#log_error*/c\log_error = /var/log/mysql/error.log' /etc/mysql/mariadb.conf.d/50-
server.cnf # setting the bind ip to allow remote connections

service mariadb stop
```

Table A.10: Install_mariadb.sh

Appendix 11

docker-compose.yml

```
# docker-compose.yml
version: "3.3"
services:
  tmp-change-west-app:
    build: ./tmpChangeWest
    hostname: tmp-change-west-app
    container_name: tmp_change_app_west
    depends_on:
      - mongodb-west
    links:
      - mongodb-west
    ports:
      - 3000:3000
    environment:
      WAIT_HOSTS: mongodb:37017
    volumes:
      - .:/tmpChangeWest/usr/src/app
    links:
      - mongodb-west
  mongodb-west:
    hostname: mongodb-west
    image: aidanmckendry/mongodb_west:1.0
    container_name: mongo_west
    ports:
      - 37017:27017
    volumes:
      - data:/data/db
  tmp-change-south-west-app:
    build: ./tmpChangeSouthWest
    hostname: tmp-change-south-west-app
    container_name: tmp_change_app_south_west
    depends_on:
      - mongodb-south-west
    links:
      - mongodb-south-west
    ports:
      - 3001:3001
    environment:
      WAIT_HOSTS: mongodb:37018
    volumes:
      - .:/tmpChangeSouthWest/usr/src/app
    links:
      - mongodb-south-west
  mongodb-south-west:
    hostname: mongodb-south-west
    image: aidanmckendry/mongodb_south_west:1.0
    container_name: mongo_south_west
    ports:
      - 37018:27017
    volumes:
      - data:/data/db
  tmp-change-mid-west-app:
    build: ./tmpChangeMidWest
    hostname: tmp-change-mid-west-app
    container_name: tmp_change_app_mid_west
    depends_on:
      - mongodb-mid-west
    links:
      - mongodb-mid-west
    ports:
      - 3002:3002
    environment:
      WAIT_HOSTS: mongodb:37019
# may need to change this voklume because it is nested within mongoose_tutorial directory
    volumes:
      - .:/tmpChangeMidWest/usr/src/app
    links:
      - mongodb-mid-west
  mongodb-mid-west:
    hostname: mongodb-mid-west
    image: aidanmckendry/mongodb_mid_west:1.2
    container_name: mongo_mid_west
    ports:
      - 37019:27017
    volumes:
      - data:/data/db
  tmp-change-south-east-app:
    build: ./tmpChangeSouthEast
    hostname: tmp-change-south-east-app
    container_name: tmp_change_app_south_east
    depends_on:
      - mongodb-south-east
    links:
      - mongodb-south-east
    ports:
      - 3003:3003
    environment:
      WAIT_HOSTS: mongodb:37020
# may need to change this voklume because it is nested within mongoose_tutorial directory
    volumes:
      - .:/tmpChangeSouthEast/usr/src/app
    links:
      - mongodb-south-east
  mongodb-south-east:
    hostname: mongodb-south-east
    image: aidanmckendry/mongodb_south_east:1.0
    container_name: mongo_south_east
    ports:
      - 37020:27017
    volumes:
      - data:/data/db
  tmp-change-north-east-app:
    build: ./tmpChangeNorthEast
    hostname: tmp-change-north-east-app
    container_name: tmp_change_app_north_east
    depends_on:
      - mongodb-north-east
    links:
      - mongodb-north-east
    ports:
      - 3004:3004
    environment:
      WAIT_HOSTS: mongodb:37021
# may need to change this voklume because it is nested within mongoose_tutorial directory
    volumes:
      - .:/tmpChangeNorthEast/usr/src/app
    links:
      - mongodb-north-east
  mongodb-north-east:
    hostname: mongodb-north-east
    image: aidanmckendry/mongodb_north_east:1.1
    container_name: mongo_north_east
    ports:
      - 37021:27017
    volumes:
      - data:/data/db

volumes:
  data:
```

Table A.11: docker-compose.yml (Temperature change services)

WildfireRecord.py

```
# docker-compose.yml
version: "3.3"
services:
  app:
    build: ./wildfireApp
    hostname: app
    container_name: wildfire_app
    depends_on:
      - mariadb
    links:
      - mariadb
    ports:
      - 3007:3007
    environment:
      WAIT_HOSTS: mysql:3307
    volumes:
      - .:/wildfireApp/app
    links:
      - mariadb
  mariadb:
    hostname: mariadb
    image: aidanmckendry/mariadb:1.0
    environment:
      MYSQL_DATABASE: 'mydb'
      # So you don't have to use root, but you can if you like
      MYSQL_USER: 'mydb'
      # You can use whatever password you like
      MYSQL_PASSWORD: '#1A2b%3C4d5E!'
      # Password for root access
      MYSQL_ROOT_PASSWORD: '1q2w3e4r'
    # not found in dockerhub...
    # dockerhub image = aidanmckendry/mysql_database
    container_name: mariadb
    ports:
      - 3307:3306
    volumes:
      - data:/data/db

volumes:
  data:
```

Table A.12: docker-compose.yml (wildfire app)

Appendix 13

index.html

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta
      charset="utf-8"
      name="viewport"
      content="width=device-width,
      initial-scale=1,
      shrink-to-fit=no" />
    <style type="text/css">
      html { height: 100% }
      body { height: 100%; margin: 0; padding: 0 }
      #map { height: 80% }
      .collapsible {
        background-color: #777;
        color: white;
        cursor: pointer;
        padding: 18px;
        width: 100%;
        border: none;
        text-align: left;
        outline: none;
        font-size: 15px;
      }

      .active, .collapsible:hover {
        background-color: #555;
      }

      .content {
        padding: 0 18px;
        display: none;
        overflow: hidden;
        background-color: #f1f1f1;
      }
    </style>
    <title>Base Map</title>

  </head>
  <div class="w3-sidebar w3-bar-block w3-collapse w3-card" style="width:100%;"
id="mySidebar">
    <button class="w3-bar-item w3-button w3-hide-large"
      onclick="w3_close()">Close &times;
    </button>
    <h3 class="card-subheading">Layers:</h3>
    <button type="button" class="collapsible">Layer 1</button>
    <div class="content">
    <input type="checkbox" onclick="displayTmpChangeLayer()" id="layer1"
name="layer1" value="Bike">
    <label for="layer1"> Temperature change layer (1900 - selected)</label><br>
    <label for="language">Select a decade for temperature change:</label>
    <form>
     <select name="End Decade" id="decade_of_tmp_change" onchange =
"updateDecadeTmpChangeLayer()">
       <option value="1910">1910</option>
       <option value="1920">1920</option>
       <option value="1930">1930</option>
       <option value="1940">1940</option>
       <option value="1950">1950</option>
       <option value="1960">1960</option>
       <option value="1970">1970</option>
       <option value="1980">1980</option>
       <option value="1990">1990</option>
       <option value="2000">2000</option>
       <option value="2010" selected>2010</option>
     </select>
    </form>
    <label for="language">Select a month for temperature change:</label>
    <form>
     <select name="Month for average" id="month_of_tmp_change" onchange =
"updateDecadeTmpChangeLayer()">
       <option value="Jan">Jan</option>
       <option value="Feb">Feb</option>
       <option value="Mar">Mar</option>
       <option value="Apr">Apr</option>
       <option value="May">May</option>
       <option value="Jun">Jun</option>
       <option value="Jul">Jul</option>
       <option value="Aug" selected>Aug</option>
       <option value="Sep">Sep</option>
       <option value="Oct">Oct</option>
       <option value="Nov">Nov</option>
       <option value="Dec">Dec</option>
     </select>
    </form>
    <img src="TmpChangeKey.jpg" alt="Layer key:"
    width="200"
    height="70"/>
   </div>
   <button type="button" class="collapsible">Layer 2</button>
   <div class="content">
    <input type="checkbox" onclick="displayWeatherStationLayer()" id="layer2"
name="layer2">
    <label for="layer2"> Weather station locations layer </label><br>
   </div>
   <button type="button" class="collapsible">Layer 3</button>
   <div class="content">
```

```
    <input type="checkbox" onclick="displayWildfireLayer()" id="layer3" name="layer3">
    <label for="layer3"> Change in wildfire burned acreage layer (1984-2001 and 2002-
2020)</label><br>
    <img src="WildfireKey.png"
      width="200"
      height="80"/>
   </div>
 </div>

 <div class="w3-main" style="width:100%;">

   <div class="w3-teal">
    <button class="w3-button w3-teal w3-xlarge" onclick="w3_open()">&#9776;</button>
   </div>
 </div>
 <script type="text/javascript"
   async                                                                                              defer
src="https://maps.googleapis.com/maps/api/js?key=AIzaSyAZ5nlcAPtz1Jftib7lnAQVRd1
CZYR3riE&v=beta&&callback=initMap">
 </script>
 <script type="text/javascript">

   // Booleans for layer display
  let showTmpChangeLayer = false;
  let showWeatherStationLayer = false;
  let showWildfireSeverityLayer = false;
  let monthOfTmpChange = "Aug";
  let decadeOfTemperatureChange = 2010;

 // Key value pair array for storing tmp change values against place IDs
   let states = {
    // West
    "ChIJ-bDD5__lhVQRuvNfbGh4QpQ": 0, // Washington
    "ChIJ6Znkhaj_WFMRWIf3FQUwa9A": 0, // Idaho
    "ChIJVWqfm3xuk1QRdrgLettlTH0": 0, // oregan
    "ChIJPV4oX_65j4ARVW8IJ6IJUYs": 0, // California
    "ChIJcbTe-KEKmYARs5X8qooDR88": 0, // Nevada
    "ChIJ04p7LZwrQVMRGGwqz1jWcfU": 0, // Montana
    "ChIJaS7hSDTiXocRLzh90nkisCY": 0, // Wyoming
    "ChIJG8CuwJzfAFQRNduKqSde27w": 0, // Alaska
    "ChIJzfkTj8drTIcRP0bXbKVK370": 0, // Utah
    "ChIJt1YYm3QUQIcR_6eQSTGDVMc": 0, // Colorado
    // South West
    "ChIJaxhMy-sIK4cRcc3Bf7EnOUI": 0, // Arizona
    "ChIJqVKY50NQGIcRup41Yxpuv0Y": 0, // New Mexico
    "ChIJSTKCCzZwQIYRPN4IGI8c6xY": 0, // Texas
    "ChIJnU-ssRE5rIcRSOoKQDPPHF0": 0, // Oklahoma
    // Mid West
    "ChIJY-nYVxKD11IRyc9egzmahA0": 0, // North Dakota
    "ChIJpTjphS1DfYcRt6SGMSnW8Ac": 0, // South Dakota
    "ChIJ7fwMtciNk4cRxArzDwyQJ6E": 0, // Nebraska
    "ChIJawF8cXEXo4cRXwk-S6m0wmg": 0, // Kansas
    "ChIJmwt4YJpbWE0RD6L-EJvJogI": 0, // Minnesota
    "ChIJGWD48W9e7ocR2VnHV0pj78Y": 0, // Iowa
    "ChIJfeMiSNXmwIcRcr1mBFnEW7U": 0, // Missouri
    "ChIJr-OEkw_0qFIR1kmG-LjV1fI": 0, // Wisconsin
    "ChIJGSZubzgtC4gRVlkRZFCCFX8": 0, // Illinois
    "ChIJEQTKxz2qTE0Rs8liellI3Zc": 0, // Michigan
    "ChIJHRv42bxQa4gRcuwyy84vEH4": 0, // Indiana
    "ChIJwY5NtXrpNogRFtmfnDlkzeU": 0, // Ohio
    // South East
    "ChIJYSc_dD-e0ocR0NLf_z5pBaQ": 0, // Arkansas
    "ChIJZYIRslSkIIYRA0flgTL3Vck": 0, // Louisiana
    "ChIJGdRK5OQyKIYR2qbc6X8XDWI": 0, // Mississippi
    "ChIJyVMZi0xzQogR_N_MxU5vH3c": 0, // Kentucky
    "ChIJA8-XniNLYYgRVpGBpcEgPgM": 0, // Tennessee
    "ChIJdf5LHzR_hogR6czIUzU0VV4": 0, // Alabama
    "ChIJRQnL1KVUSogRQzrN3mjHALs": 0, // West Virginia
    "ChIJ35Dx6etNtokRsfZVdmU3r_I": 0, // Maryland
    "ChIJzbK8vXDWTIgRlaZGt0lBTsA": 0, // Virginia
    "ChIJgRo4_MQfVIgRGa4i6fUwP60": 0, // North Carolina
    "ChIJ49ExeWml-IgRnhcF9TKh_7k": 0, // South Carolina
    "ChIJV4FfHcU28YgR5xBP7BC8hGY": 0, // Georgia
    "ChIJvypWkWV2wYgR0E7HW9MTLvc": 0, // Florida
    // North East
    "ChIJieUyHiaALYgRPbQiUEchRsI": 0, // Pennsylvania
    "ChIJqaUj8fBLzEwRZ5UY3sHGz90": 0, // New York
    "ChIJ_87aSGzctEwRtGtUNnSJTSY": 0, // Vermont
    "ChIJ1YpTHd4dsEwR0KggZ2_MedY": 0, // Maine
    "ChIJ66bAnUtEs0wR64CmJa8CyNc": 0, // New Hampshire
    "ChIJ_b9z6W1l44kRHA2DVTbQxkU": 0, // Massachusetts
    "ChIJn0AAnpX7wIkRjW0_-Ad70iw": 0 // New Jersey
   };

   // Key value pair array for storing place IDs against state names
   const placeIdForStates = {
    // West
    'Washington': "ChIJ-bDD5__lhVQRuvNfbGh4QpQ",
    'Idaho': "ChIJ6Znkhaj_WFMRWIf3FQUwa9A",
    'Oregan': "ChIJVWqfm3xuk1QRdrgLettlTH0",
    'California': "ChIJPV4oX_65j4ARVW8IJ6IJUYs",
    'Nevada': "ChIJcbTe-KEKmYARs5X8qooDR88",
    'Montana': "ChIJ04p7LZwrQVMRGGwqz1jWcfU",
    'Wyoming': "ChIJaS7hSDTiXocRLzh90nkisCY",
    'Alaska': "ChIJG8CuwJzfAFQRNduKqSde27w",
    'Utah': "ChIJzfkTj8drTIcRP0bXbKVK370",
    'Colorado': "ChIJt1YYm3QUQIcR_6eQSTGDVMc",
    // South West
    'Arizona': "ChIJaxhMy-sIK4cRcc3Bf7EnOUI",
    'New Mexico': "ChIJqVKY50NQGIcRup41Yxpuv0Y",
    'Texas': "ChIJSTKCCzZwQIYRPN4IGI8c6xY",
    'Oklahoma': "ChIJnU-ssRE5rIcRSOoKQDPPHF0",
    // Mid West
    "North Dakota": "ChIJY-nYVxKD11IRyc9egzmahA0",
    "South Dakota": "ChIJpTjphS1DfYcRt6SGMSnW8Ac",
    "Nebraska": "ChIJ7fwMtciNk4cRxArzDwyQJ6E",
    "Kansas": "ChIJawF8cXEXo4cRXwk-S6m0wmg",
    "Minnesota": "ChIJmwt4YJpbWE0RD6L-EJvJogI",
    "Iowa": "ChIJGWD48W9e7ocR2VnHV0pj78Y",
    "Missouri": "ChIJfeMiSNXmwIcRcr1mBFnEW7U",
    "Wisconsin": "ChIJr-OEkw_0qFIR1kmG-LjV1fI",
    "Illinois": "ChIJGSZubzgtC4gRVlkRZFCCFX8",
    "Michigan": "ChIJEQTKxz2qTE0Rs8liellI3Zc",
    "Indiana": "ChIJHRv42bxQa4gRcuwyy84vEH4",
    "Ohio": "ChIJwY5NtXrpNogRFtmfnDlkzeU",
    // South East
    "Arkansas": "ChIJYSc_dD-e0ocR0NLf_z5pBaQ",
    "Louisiana": "ChIJZYIRslSkIIYRA0flgTL3Vck",
    "Mississippi": "ChIJGdRK5OQyKIYR2qbc6X8XDWI",
    "Kentucky": "ChIJyVMZi0xzQogR_N_MxU5vH3c",
    "Tennessee": "ChIJA8-XniNLYYgRVpGBpcEgPgM",
    "Alabama": "ChIJdf5LHzR_hogR6czIUzU0VV4",
    "West Virginia": "ChIJRQnL1KVUSogRQzrN3mjHALs",
    "Maryland": "ChIJ35Dx6etNtokRsfZVdmU3r_I",
    "Virginia": "ChIJzbK8vXDWTIgRlaZGt0lBTsA",
    "North Carolina": "ChIJgRo4_MQfVIgRGa4i6fUwP60",
    "South Carolina": "ChIJ49ExeWml-IgRnhcF9TKh_7k",
    "Georgia": "ChIJV4FfHcU28YgR5xBP7BC8hGY",
    "Florida": "ChIJvypWkWV2wYgR0E7HW9MTLvc",
    // North East
    "Pennsylvania": "ChIJieUyHiaALYgRPbQiUEchRsI",
    "New York": "ChIJqaUj8fBLzEwRZ5UY3sHGz90",
    "Vermont": "ChIJ_87aSGzctEwRtGtUNnSJTSY",
    "Maine": "ChIJ1YpTHd4dsEwR0KggZ2_MedY",
    "New Hampshire": "ChIJ66bAnUtEs0wR64CmJa8CyNc",
    "Massachusetts": "ChIJ_b9z6W1l44kRHA2DVTbQxkU",
    "New Jersey": "ChIJn0AAnpX7wIkRjW0_-Ad70iw"
   };

   async function initMap() {
    let featureLayer;
    let map;
    var start = {
      lat: 36.7783,
      lng: -119.4179
    };
    map = new google.maps.Map(
      document.getElementById('map'), {
        zoom: 4,
        center: new google.maps.LatLng(start),
        mapId: "78a963aba4a764a6"
      });

    featureLayer                                                                 =
map.getFeatureLayer(google.maps.FeatureType.ADMINISTRATIVE_AREA_LEVEL_1);

    // Calling all tmp change services
    if (Boolean(showTmpChangeLayer)) {
      var                    tmpChangeWest                    =                    await
fetch(`http://localhost:3000/api/v1/tmpChanges/?month=${monthOfTmpChange}`).then(re
sponse => response.json());
      var                  tmpChangeSouthWest                   =                  await
fetch(`http://localhost:3001/api/v1/tmpChanges/?month=${monthOfTmpChange}`).then(re
sponse => response.json());
      var                   tmpChangeMidWest                    =                  await
fetch(`http://localhost:3002/api/v1/tmpChanges/?month=${monthOfTmpChange}`).then(re
sponse => response.json());
      var                  tmpChangeSouthEast                   =                  await
fetch(`http://localhost:3003/api/v1/tmpChanges/?month=${monthOfTmpChange}`).then(re
sponse => response.json());
      var                  tmpChangeNorthEast                   =                  await
fetch(`http://localhost:3004/api/v1/tmpChanges/?month=${monthOfTmpChange}`).then(re
sponse => response.json());

      // Updating the temperature change values for each state
      updateStatesFromResponse(tmpChangeWest)
      updateStatesFromResponse(tmpChangeSouthWest)
      updateStatesFromResponse(tmpChangeMidWest)
      updateStatesFromResponse(tmpChangeSouthEast)
      updateStatesFromResponse(tmpChangeNorthEast)

      console.log('states array after updating temperature change', states);

      featureLayer.style = (featureStyleFunctionOptions) => {
        const placeFeature = featureStyleFunctionOptions.feature;
        const temperaturechange = states[placeFeature.placeId];

        // fill colours are not being displayed on map correctly...
        let fillColor = getFillColour(temperaturechange);
        console.log('fill colour', fillColor);

        return {
          fillColor,
          fillOpacity: 0.8,
        };
      };
    }

    if (Boolean(showWeatherStationLayer)) {
      var weatherStations = [];
      var weatherStationsJsonEncoded = await fetch('http://localhost:3006/').then(response
=> response.json());
      for (let x = 0; x < weatherStationsJsonEncoded.data.length; x++) {
        weatherStations.push(JSON.parse(weatherStationsJsonEncoded.data[x]));
      }
      console.log("weather stations: ", weatherStations);

      updateMarkersForWeatherStations(weatherStations);
    }

    if (Boolean(showWildfireSeverityLayer)) {
      // scale marker depending on burned acreage value

      var wildfireRecords = [];
      var wildfireRecordsJsonEncoded = await fetch('http://localhost:3007/').then(response
=> response.json());
      for (let x = 0; x < wildfireRecordsJsonEncoded.data.length; x++) {
        wildfireRecords.push(JSON.parse(wildfireRecordsJsonEncoded.data[x]));
      }

      console.log("wildfire records: ", wildfireRecords);

      updateMarkersForWildfires(wildfireRecords);

      }

    function updateStatesFromResponse(response) {
    const data = response.data;
    console.log('decade of temperature change before filter: ',
decadeOfTemperatureChange);

    var filteredTemperatureChange = data.filter(d => d.decade ==
decadeOfTemperatureChange);
    console.log('filtered json response data', filteredTemperatureChange);

    console.log('states array before updating temperature change', states);
    // update states array for displaying temperature change
    for (let i = 0; i < filteredTemperatureChange.length; i++) {
        let state = filteredTemperatureChange[i].state;
        let placeId = placeIdForStates[state];
        let tempChange = filteredTemperatureChange[i].temperaturechange;
        states[placeId] = tempChange;
      }
    }

    function updateMarkersForWeatherStations(response) {
    const weatherStationIcon = {
        url: "https://pic.onlinewebfonts.com/svg/img_239346.png", // url of free weather
station icon
        scaledSize: new google.maps.Size(30, 30), // scaled size
```

```
        origin: new google.maps.Point(0,0), // origin
        anchor: new google.maps.Point(0, 0) // anchor
      };

      for (weatherStation in response) {
        console.log("latitude",    response[weatherStation].latitude,    "    longtitude",
response[weatherStation].longtitude)
        var marker = new google.maps.Marker({
          position:    {lat:    response[weatherStation].longtitude,    lng:
response[weatherStation].latitude},
          map: map,
          icon: weatherStationIcon
        });
      }
    }

    function makeGraduatedSymbolIcon(changeInBurnedArea) {
      let size =  30;
      if (changeInBurnedArea > 2) {
        size = 55
      }
      if (changeInBurnedArea > 3) {
        size = 90
      }
      var weatherStationIcon = {
          url: "https://img.icons8.com/ios/2x/filled-circle.png", // url of free circle icon
          scaledSize: new google.maps.Size(size, size), // scaled size
          origin: new google.maps.Point(0,0), // origin
          anchor: new google.maps.Point(0, 0) // anchor
      };
      return weatherStationIcon
    }

    function updateMarkersForWildfires(response) {
      for (record in response) {

        let state = response[record].state;
        // look up place id from state name
        let placeId = placeIdForStates[state];
        let changeInBurnedArea = response[record].change_in_burned_area;

        console.log("state: ", state, ", change_in_burned_area: ", changeInBurnedArea, ", for
loop iteration: ", record)

        if (response[record].change_in_burned_area > 0.5) {
          let geocoder = new google.maps.Geocoder();
          geocoder.geocode( { 'placeId': placeId}, function(results, status) {
          if (status == 'OK') {
            console.log("Creating graduated symbol for state: ", state, ", change in burned area
due to wild fires: ", changeInBurnedArea)
```

```
          var marker = new google.maps.Marker({
            map: map,
            position: results[0].geometry.location,
            icon:                              makeGraduatedSymbolIcon(changeInBurnedArea)//
response[record].change_in_burned_area
          });
        } else {
          console.log('Geocode was not successful for the following reason: ', status, " err",
geocode );
          }
        });
      }
    }
  }

  function getFillColour(temperaturechange){
    // five blue colour grades (-2 - -0.5)
    if (temperaturechange < -15.0) {
      return fillColor = "#002568";
    } else if (temperaturechange < -2.0) {
      return fillColor = "#0644B3";
    } else if (temperaturechange < -1.5) {
      return fillColor = "#566DEE";
    } else if (temperaturechange < -1.0) {
      return fillColor = "#54C5EF";
    } else if (temperaturechange < -0.5) {
      return fillColor = "#87F7F3";
    } else if (temperaturechange < 0) {
      return fillColor = "#BAFEFB";
      // four yellow colour grades (0 - 2)
    } else if (temperaturechange < 0.5) {
      return fillColor = "#FAFBAC";
    } else if (temperaturechange < 1.0) {
      return fillColor = "#FCF747";
    } else if (temperaturechange < 1.5) {
      return fillColor = "#FCCB14";
    } else if (temperaturechange < 2.0) {
      return fillColor = "#FA9E18";
      // three orange colour grades (2 - 3.5)
    } else if (temperaturechange < 2.5) {
      return fillColor = "#EF710D";
    } else if (temperaturechange < 3.0) {
      return fillColor = "#C45A07";
    } else if (temperaturechange < 3.5) {
      return fillColor = "#FA7048";
      // five red colour grades (3.5 +)
    } else if (temperaturechange < 4) {
      return fillColor = "#DA4723";
    } else if (temperaturechange < 4.5) {
      return fillColor = "#D80808";
    } else if (temperaturechange < 5.0) {
      return fillColor = "#AC0707";
    } else if (temperaturechange < 5.5) {
      return fillColor = "#930606";
    } else if (temperaturechange < 10) {
      return fillColor = "#630101";
    }
  }
}

//var decadeList = document.getElementById("decade_of_tmp_change");
//var decadeValue = decadeList.value;
//var decadeOfTemperatureChange = parseInt(decadeValue)
function updateDecadeTmpChangeLayer() {
  var decadeList = document.getElementById("decade_of_tmp_change");
  var decadeValue = decadeList.value;
  decadeOfTemperatureChange = parseInt(decadeValue)
  var monthList = document.getElementById("month_of_tmp_change");
  var monthValue = monthList.value;
  monthOfTmpChange = monthValue;
  // updateStatesFromResponse()
  initMap()
}

function displayTmpChangeLayer() {
  var checkbox = document.getElementById("layer1")
  if (checkbox.checked == true) {
    showTmpChangeLayer = true;
  } else {
    showTmpChangeLayer = false;
    // set the temperature changes back to zero
    for (state in states) {
      let placeId = placeIdForStates[state];
      states[placeId] = 0;
      }
  }
  initMap()
}

function displayWeatherStationLayer() {
  var checkbox = document.getElementById("layer2")
  if (checkbox.checked == true) {
    showWeatherStationLayer = true;
  } else {
    showWeatherStationLayer = false;
  }
  initMap()
}

function displayWildfireLayer() {
    var checkbox = document.getElementById("layer3")
    if (checkbox.checked == true) {
      showWildfireSeverityLayer = true;
    } else {
      showWildfireSeverityLayer = false;
    }
    initMap()
  }

  window.initMap = initMap;
  </script>

  <script>
    function w3_open() {
      document.getElementById("mySidebar").style.display = "block";
    }

    function w3_close() {
      document.getElementById("mySidebar").style.display = "none";
  }
  </script>
  <body onload="initMap()">
    <div id="map"></div>
  </body>
  <script>
  var coll = document.getElementsByClassName("collapsible");
  var i;

  for (i = 0; i < coll.length; i++) {
    coll[i].addEventListener("click", function() {
      this.classList.toggle("active");
      var content = this.nextElementSibling;
      if (content.style.display === "block") {
        content.style.display = "none";
      } else {
        content.style.display = "block";
      }
    });
  }
</script>
</html>
```

Table A.13: index.html