**“Climate Change Data Visualization Web Application”**

CS551A: Fundamentals of Software Project Management

Project Proposal

7 March 2025


Team 4:

Sarah Julius

Zizhou Dou

Zhouming Liu

Minli Lu

Selu ‘Ita

# Table of Contents

**Table of Contents** ……1
**List of Figures**…..…….…….2
1. Project Vision……..……3-5
1.1 Mission……..……3
1.2 Goals……...…3
1.3 About the Team…….…3
1.4 What We Offer…….…….3-4
1.5 Main Roles……..……...…4
1.6.1 Project Constraints……..…..…….4
1.6.2 Our Approach to Constraints…….……5
2. Software Development Process..……6-8
2.1 Chosen Approach……...…...……6
2.2 Why Agile? ……..…...…6
2.3 XP/Scrum…….…..…6-7
2.4 Product Backlog / User End……..……7-8
2.5 Application Six Week Plan……8
3. Project Success Criteria……...…9
4. Technical Details……..…10-12
4.1 Product Overview…….10
4.2 Technology Selection……10-11
4.3 System Architecture…….11
4.4 Data Flow…….11
4.5 Possible Technical Challenges…….11-12
5. Key Use Case Models……13-16
5.1 Overview of Use Cases……13
5.2 User Case Modelling……13-14
5.3 Description of Key Use Cases……..14-15
5.4 Component Interaction…….15-16
5.5 Summary……..16
6. Candidate System Architecture..……17-20
6.1 System……..17
6.2 Frontend Layer…….17-18
6.3 Backend Layer……..18
6.4 Data Layer……18
6.5 Data Processing Layer……..18
6.6 Cloud Infrastructure……..19
6.7 Component Interaction Workflow……19-20
7. Initial Risk Assessment..……21-22
7.1 Potential Risks and Mitigation Strategies……21
7.2 Hosting Considerations……21-22
7.3 Validating Payer/Public Group Interest(s)……22
7.4 Learning New Technologies…….……22
7.5 Application Testing Strategies……..……22
8. Product Sprint Plan.……24-28
8.1 Sprint Breakdown…….24-26
8.2 Daily Stand-Up…….26-27
**APPENDIX**……28-29

## List of Figures

Figure 1: User Interface……4

Figure 2: Server-Side of App……8

Figure 3: Six Week Plan……8

Figure 4: Case Model Example (Australia)...……13

Figure 5: Data Graph Example…….……15

Figure 6: Architecture Outline…….……..20

Figure 7: Six Week Sprint Plan…….……23

Figure 8: Week 1 Sprint Plan...……..24

Figure 9: Week 2 Sprint Plan….……....24

Figure 10: Week 3 Sprint Plan….……..25

Figure 11: Week 4 Sprint Plan….……..25

Figure 12: Week 5 Sprint Plan….……..26

Figure 13: Week 6 Sprint Plan….……..26

# 1. Project Vision

## 1.1 Mission:

The goal of this project is to create an application that visualizes climate data in an intuitive way, helping users understand global climate change trends by country.

## 1.2 Goals:

The system is designed to be user friendly for all demographics inclusive of age, primary language, and dis/ability.

## 1.3 About the Team:

We are a collection of 5 individuals, currently pursuing degrees in IT. Our backgrounds vary widely between different degrees, interests, and careers. Eagerly looking forward to this project, we are prepared to use what we are currently learning at university, as well as the knowledge from our previous backgrounds, to complete this project in the prescribed six weeks.

## 1.4 What We Offer:

This information has been collected by the World Health Organization (WHO), giving promising accuracy to the listed data. We strive to provide both clear and interactive tools, such as numbers, graphs, and heat maps for effective data representation. Not only will the application be effective and accurate but display trends over time in an interesting way for the user. The platform will allow users to access data without login or account creation, providing ease for the user as well as a quicker turn-around for the software engineers. Users can search specific datasets, filter by year or region, and compare different climate indicators for each country. User interface will function as shown below:

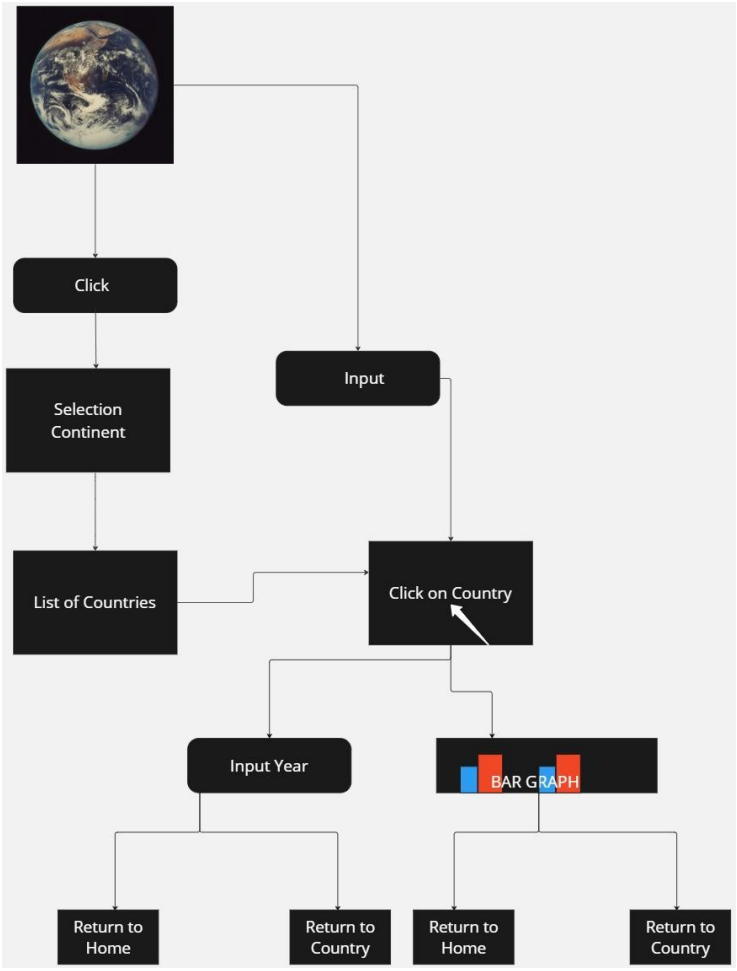

Figure 1: User Interface

1.5 Main Roles:
Project Leader/Developer – Selu
Software Architect/Developer – Zizhou
UX Design/Developer – Zhouming
Software Developer/Developer – Minli
Quality Assurance Tester/Developer – Sarah

1.6.1 Project Constraints:

a. Data Risk – The system will allow the user to search and view the represented data, as previously stated, without login or account creation. Although this does provide easier access, it also raises a potential data security risk.
b. Time – The client requires the application to take a total of six weeks to complete.
c. Data Sets – The data provided to the engineers, needs to be updated from the years 2018 to present.
d. Variant Data – WHO started recording this data every five years until 2010. The information would require both mathematical algorithms to find mean, mode, and median of the annual data prior to 2010 or representative data of only the sets that were taken.
e. Upkeep – With the interactive application, there will be more upkeep required to fix bugs or update the application as and when needed.

1.6.2 Our Approach to Constraints

a. Protected Data – The main issue with the data security risk is that hackers can more easily More specifically, users will not be able to adjust existing data or add data.
b. Timeline – To provide an interactive, but clear application, we have separated the tasks using Agile, via XP, scrums, and sprints. Further, the initial data in the project will be streamlined to fit the time frame. The initial application takes priority as more detailed data can be added later.
c. Data Sets – The missing data will need to be updated by the end of the first day. Using AI learning tools, the team will update the data sets. Project Leader will check data points for mistakes to ensure accuracy. Although updating the data will take time, this exercise will provide an opportunity for the team to familiarize themselves with the data sets more closely.
d. Representing Data – As an application responsible for informing the public of this information in a clear manner, it is imperative that the application only show the data we have. This is why the application will initially represent numbers rather than a graph to give a more accurate portrayal of the data. If the user types in a year in which WHO did not record any data, the page will display an error message. If the user would like to see trends over time, we will have a graph displaying the years the data was taken. Representing the data with these graphs will more easily reflect the existing data while maintaining the ability to see trends over time. Please refer to Figure 5 on page 15 for an example of the graph the user will see.
e. Upkeep – Testing and upkeep will be led by our Lead Tester. Due to the program being an interactive application, the need to constantly monitor for bugs is necessary. Since the application will be a finished product at this stage, we will use our internal programmers to monitor the application.

## 2. Software Development Process

### 2.1 Chosen Approach

The project will implement the Agile Model throughout the six-week period. Specifically, we will use a combination of Scrum, XP, and sprints.

### 2.2 Why Agile?

Agile development allows for rapid adjustments based on user feedback, ensuring a high-quality final product. Given the time limit, Agile ensures flexibility and faster delivery of a functional version. Since data visualization may require frequent UI/UX refinements, Agile development allows continuous testing and enhancements. Agile provides a list of values and principles we would like to implement as a group. We will heavily rely on communication over processes as we have a diverse group with different primary languages. Agile will help us set and coordinate individual goals, team goals, and client goals. Since we are a small team, we will be making decisions after group discussions. When obstacles arise, we will take note of low-priority ones and talk about them in face-to-face daily stand-ups. Any high-priority obstacles, we will direct immediately to the team.

### 2.3 XP/Scrum

For our development process, we will do a hybrid version of XP and Scrum. Due to the team's various leadership and conflict-resolution styles, the team works best with roles being equal, no hierarchy, and shared ownership of the product. Further, the entire team will take responsibility for all aspects of the project. XP also allows for daily communications, sprint planning, sprint reviews, sprint retrospectives, and backlog refinement. These will all be necessary for the success of the application.

The basic practices and principles of XP are listed below, with additional notes decided by the team to implement them:

a. Incremental Planning
   - sprints
   - daily scrums
   - sprint retrospectives

b. Small Releases
   - focus on main requirements
   - reviewed in scrums
   - start and end of sprints

c. Simple Design
   - prioritise basic design
   - focus on main requirements
   - reference to backlog list

d. Test First
   - test batches
   - product tester will combine batches and fix bugs
   - bugs discussed in daily meetups/sprint retrospective depending on priority levels

e. Refactoring
   - discuss high-priority obstacles in daily check-ins
   - lower priority obstacles in weekly sprint retrospectives
   - decide next steps, intended outcome, time-frame

f. Pair Programming

- implemented throughout planning phase
- exercise as needed during weekly meet-up outside of the course

g. Continuous Integration
   - improvements discussed in scrums and sprints
   - plan and set goals for improvements
   - test in batches and combine batches

h. On-Site Customer
   - discuss weekly with customer on progress
   - use samples of people to interact with app
   - plan adjustments as needed

Further details are laid out below for the Scrums. A scrum requires specific roles to be taken. The list and break-down of roles is below:

- Product Owner - Sarah
- Scrum Master - Selu
- Development Team - Everyone

Scrums will go as follows:

a. Sprints consist of 1-week intervals. More on this in section 8 below.
b. Scrums within the sprints will consist of face-to-face meetings led by the Scrum Master three times a week (twice for practicals and once for voluntary meetups) where we have a touchpoint for 5 minutes. This touchpoint will review what the team has accomplished and what is the goal for the day. This will also be the time for us to decide adjustments or discuss obstacles faced. The remainder of the face-to-face meetings will be individuals working on respective parts of the project together. The rest of the week will be touch-points via phone to inform each other what individuals are working on and what is next. This will create better transparency on the application and higher accountability on divided responsibilities.
c. Artefacts in the scrum will be recalled refocussing the group on the main goals and priorities for end result and weekly team goal. Shown below are the main priorities:
   - Each scrum will reference the product backlog (see section 2.5 for an outline) to determine if the product backlog is being prioritised in that week's sprint.
   - Measure the incremental deployable section of the product. Review if that section is "done" or if some adjustments or more time is necessary for that section of the product.

2.4 Product Backlog / User-End

The client has provided a list of countries and some percentile data regarding the pollution levels. The client has requested that the application include this data and provide a user-friendly experience. Due to the restricted deadline, the application's server-side will be the main priority. The server-side in large includes:

a. countries the client has listed
b. connecting countries to their respective data
c. data listable depending on year input
d. history of data shown in bar graph
   - x-axis showing the year
   - y-axis showing ascending numbers
   - two colours representing population and pollution

Below is a visual representation of the server-side of our application:

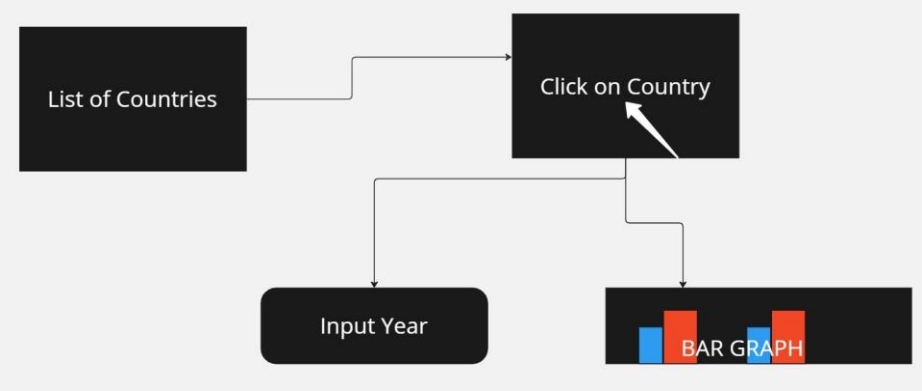


Figure 2: Server-Side of App

## 2.5 Application Six Week Plan

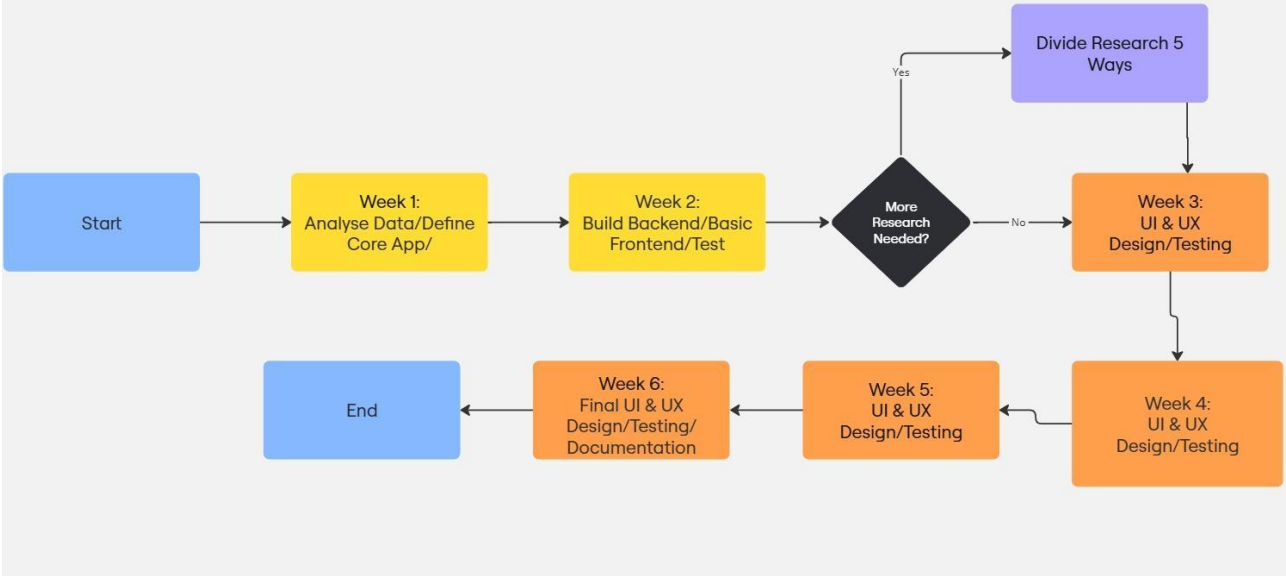


Figure 3: Six Week Plan

## 3. Project Success Criteria

The success of this project will be measured using the following criteria with explanations below:

a. System Performance:
   - Review system responsiveness.
   - Review load data efficiency.

b. Client Satisfaction:
   - Client provides feedback on the progress and final product.
     - Product Owner will act as client and testing will be done with sample groups.

c. User group testing (Diversity in Age, Demographic, First Language, etc.):
   - Testing implemented in the real world with feedback available afterward.
   - Main criteria for feedback will be as follows:
     - Clarity of Data Visualization: Can users easily interpret the charts and graphs?
     - Measuring Accessibility: What are the requirements for users?
     - User Satisfaction: What is their satisfaction with the load time?

d. User Engagement after App Launch:
   - Measure application activity
   - Record website traffic
   - If low, discuss advertisement possibilities

# 4. Technical Details

This section describes the technical architecture of the application, the technologies involved, how the product is conceived, and how it works. It focuses on product requirements, system architecture, and technology options, and analyses the potential limitations of these technologies.

## 4.1 Product Overview

The goal of this project is to develop a Web application based on data sets of pollution index changes in different years, mainly for data visualization, exploration and comparison of different data sets. The main features of the application include:

a. Data Presentation:
   - interactive visualization based on climate data (presented to the user as a line chart).

b. Data Filtering and Comparison:
   - allows users to select specific time ranges, geographic areas, variables for comparison.

c. Responsive Design:
   - suitable for desktop and mobile terminals.

d. High Performance and Scalability:
   - Use database systems to adapt to future data volume growth.

Non-functional requirements:

a. Security:
   - The user does not need a log-in to access the information.

b. Performance:
   - The system needs to support high concurrency to ensure real-time query and data visualization.

c. Scalability:
   - Support for future expansion of data sets.

## 4.2 Technology Selection

| **Category** | **Technology** | **Reason** | |
|---|---|---|---|
| Front-end | React + Next.js | Provides server-side rendering (SSR), SEO friendly, dynamic page rendering | |
| Front-end UI | Tailwind CSS/Ant Design | Modern UI component library, supports rapid development | |
| Back-end | Node.js (Express) | Lightweight, high concurrency support, sharing JavaScript code with the front end | |
| Database | PostgreSQL/MongoDB | For structured data, PostgreSQL can be used; for unstructured data, MongoDB can be used | |
| Data Visualization | D3.js/Chart.js/Recharts | Provides high-quality charting components that support visual analysis | |

| | | |
|---|---|---|
| Storage | AWS S3 / Firebase Storage | Stores large-scale data, such as CSV and JSON files |
| Deployment | Vercel/Netlify (front-end) / AWS EC2 / Railway (back-end) | Vercel/Netlify provides automated deployment and AWS EC2 supports a scalable back-end |
| API | RESTful API/GraphQL | RESTful API is used to ensure compatibility. GraphQL can be used to optimize queries |
| Logs and Monitoring | Sentry/Prometheus | Monitors system performance and error records |
| Data Processing | Pandas (Python)/Apache Spark | Suitable for large-scale data cleaning and processing |

Table 1: Technology Selection

4.3 System Architecture

Using the front-end separation architecture (SPA + API), the basic process is as follows:

a. Users visit the front-end page (React + Next.js) to obtain the pollution index data of the corresponding region through API.
b. Backend (Node.js + Express) provides API access to pollution index data (database storage JSON/CSV format).
c. The database (PostgreSQL/MongoDB) stores structured or semi-structured pollution index data.
d. Data visualization components (D3.js/Recharts) render interactive data analysis.
e. Log Monitoring System (Sentry) Records user behaviour and error logs to optimize system stability.

4.4 Data Flow

a. Data source: Data sets on changes in PM2.5 by country between 1991 and 2020 are stored in a database (PostgreSQL or MongoDB).
b. Data processing:
    - Data preprocessing using Python (Pandas/NumPy) or Apache Spark.
    - Generate structured data tables and store them in the database.
c. API layer:
    - Provide RESTful API, support by time, geographical region, variable query.
    - Scalable GraphQL in the future to optimize data query efficiency.
d. Front-end visualization:
    - React gets API data and draws interactive charts via D3.js/Recharts.
    - Support timeline adjustment, map interaction, data comparison and other functions.

4.5 Possible Technical Challenges

| **Challenges** | **Possible Impacts** | **Solutions** |
|---|---|---|
| Huge data set | Query and rendering may be slow | Using paging query, GraphQL to obtain data on demand |

| | | |
|---|---|---|
| Real-time requirement | Fast response to query | Data cache (Redis) and optimized index (PostgreSQL) |
| Front-end performance | Data rendering may cause a page to be stuck | React Virtualized |
| Scalability | New data may be added in the future | Adopt microservices architecture and split API |

Table 2: Possible Technical Challenges

## 5. Key Use Case Models

This section describes the key features of system acceptance that drive design decisions and determine how the built components will work together. We will use case diagrams and detailed use case descriptions to demonstrate the core interactions of the system.

### 5.1 Overview of Use Cases

The goal of this system is to provide a web application for visualising climate change data. Core functionality includes:

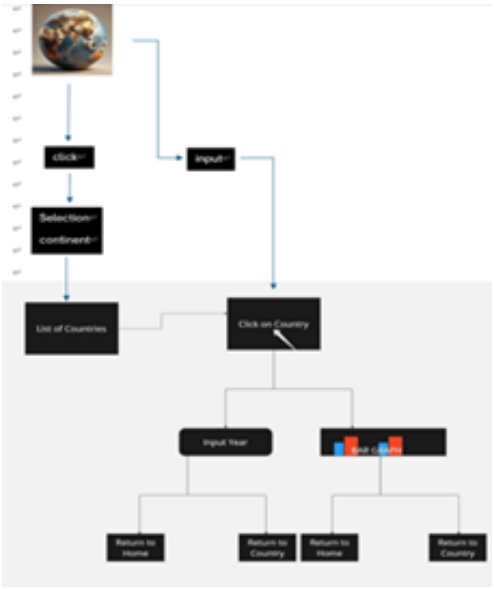


Figure 1: User Interface

a. Data Visualisation: Users can view graphs of climate data (e.g. temperature changes, $CO_2$ emissions).

b. Data Filtering: Users can filter data by time and region.

c. Data Comparison: allows users to select multiple data sets for comparison.

d. Download Data (optional): Users can export CSV files for further analysis.

### 5.2 User Case Modelling

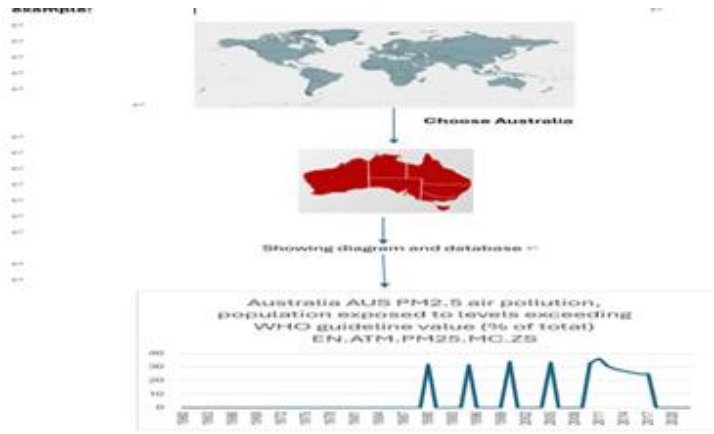


Figure 4: Case Model Example (Australia)

The lead role of the case modelling is as follows：

a. Guest Users:
   - no authentication is required, and they are able to view, filter and compare data.

a. System:
   - Provides data query, visualisation and interaction capabilities.

Key interactions are listed below:

a. Selection: The user can select a country to filter.
b. Click: the user clicks, and the system loads the appropriate data.
c. Example: When the user selects Australia, the system displays relevant climate data

for the region.

d. Showing Diagram and Database: The system provides visual diagrams and database information to support users in understanding data changes.

Selectable countries/areas are as follows:

a. Europe: France, UK
b. Asia: China, Japan, Korea
c. Americas: North America, Brazil
d. Africa: Uganda, Yemen, South Africa
e. Oceania: Australia, New Zealand

## 5.3 Description of Key Use Cases

**Use Case 1: Browsing Data**

| Description | Details |
|---|---|
| Use Case Name | Browse Data |
| Participants | Guest Users |
| Trigger Condition | User enters app home page |
| Main Process | 1. User accesses the website<br>2. The system displays the default data visualisation<br>3. User can interact with the charts (zoom, view details) |
| Exceptions | 1. Database connection failed -> error message displayed |

Table 3: Use Case 1: Browsing Data

**Use Case 2: Filtering Data**

| Description | Details |
|---|---|
| Use Case Name | Filter Data |
| Participants | Guest Users |
| Trigger Condition | User wants to view data for specific time period or region |
| Main Process | 1. User clicks on filter option<br>2. Selects time range, region, etc.<br>3. Submits the filter request<br>4. System returns to eligible data |
| Exceptions | 1. No data for the selected range -> 'No result' message displayed |

Table 4: Use Case 2: Filtering Data

**Use Case 3: Data Comparison**

| Description | Details |
|---|---|
| Use Case Name | Data Comparison |
| Participants | Guest Users |
| Trigger Condition | User wants to compare different data sets |

| | |
|---|---|
| Main Process | 1. User clicks on 'Data Comparison' function<br>2. Selects multiple datasets<br>3. Submits a request<br>4. System generates comparison charts |
| Exceptions | 1. User selects too many datasets -> Limit and prompt. |

Table 5: Use Case 3: Data Comparison

**Use Case 4: Data Download (Optional)**

| Description | Details |
|---|---|
| Use Case Name | Data Download |
| Participants | Guest Users |
| Trigger Condition | User wants to export data for external analysis |
| Main Process | 1. User clicks the 'Download' button<br>2. Selects the export format (CSV)<br>3. The system generates a download link |
| Exceptions | 1. Data generation failure -> error message displayed |

Table 6: Use Case 4: Data Download (Optional)

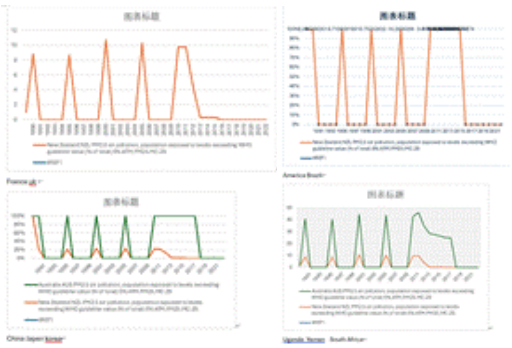

Figure 5: Data Graph Examples

5.4 Component Interaction

a. Front-end components (React + Next.js):
   - User-entered filters
   - Request back-end API for data
   - Rendering Data Visualisations via Recharts / D3.js

b. Backend components (Node.js + Express + PostgreSQL):
   - Processing of data query requests
   - Provide the API endpoint `/api/climate-data`
   - Managing database storage and index optimisation

c. Data flow and interaction:
   - User Request Data:
     - front-end calls to the API
   - Back-End Query Database:
     - optimising queries and improving performance
   - Data Return and Rendering:

- front-end using Recharts/D3.js to generate visualisation results

5.5 Summary

This application uses React + Next.js (front-end), Node.js + Express (back-end), PostgreSQL/MongoDB (database), combined with D3.js/Recharts (data visualization) and AWS/Firebase (cloud service). Realize interactive regional pollution index data analysis. The system architecture is based on RESTful apis to ensure high performance, scalability, and support for future functionality expansion. Part IV - Technical details of the application. This section describes the technical architecture of the application, the technologies involved, how the product is conceived, and how it works. It focuses on product requirements, system architecture, and technology options, and analyses the potential limitations of these technologies.

# 6. Candidate System Architecture

This application uses a three-tier microservices architecture. It separates data storage from processing, preventing performance bottlenecks on a single server. This improves response speed. The decoupling of the frontend and backend allows independent development and deployment. Teams can work more efficiently and expand features flexibly in the future.

Load balancing and cloud deployment ensure stable system functionality, even if some components fail. This enhances fault tolerance. The modular structure of microservices allows independent updates and maintenance. This reduces disruptions and improves both development and operations efficiency.

## 6.1 System

The system remains stable under high concurrent access. It also supports future functional expansion and optimization. These advantages make it well-suited for data-intensive climate change visualization platforms.

The system has five core layers, and each layer plays a distinct role:

a. Frontend Layer: This layer manages the user interface. It handles data presentation and user interactions.
b. Backend Layer: This layer processes business logic and manages APIs. It connects the frontend to the database.
c. Data Layer: This layer stores structured and unstructured data. It includes climate change datasets.
d. Data Processing Layer: This layer cleans, transforms, and analyses data. It also predicts trends.
e. Cloud Infrastructure: This layer provides computing resources. It ensures high availability through load balancing and container management.

## 6.2 Frontend Layer

The frontend is built using modern web technologies to provide an efficient, intuitive, and responsive user experience. It communicates with backend APIs, dynamically loads data, and presents climate change trends through data visualization.

a. Key Technologies:
    - Frontend Frameworks:
        - React.js / Vue.js for building
        - Single Page Applications (SPA)
        - component-based development and efficient rendering
    - Data Visualization Libraries:
        - D3.js for advanced data processing
        - SVG rendering
    - Chart.js for simple yet effective visualizations such as:
        - bar charts
        - line graphs
        - pie charts.
    - UI Component Libraries:
        - Tailwind CSS for flexible UI styling.
        - Material UI, based on Google's Material Design, offering prebuilt components.

b. Key Features
   - Data Filtering:
     - Users can filter data by country, year, and climate indicators (e.g., $CO_2$ emissions, temperature, air pollution index).
   - Interactive Visualizations:
     - Bar charts for $CO_2$ emissions and temperature variations.
     - Line graphs for tracking trends over time.
     - Heatmaps for visualizing global pollution distribution.
     - Users can interact with charts (hover details, time range adjustments, and downloadable reports in PNG/PDF).
   - Guest Access:
     - Users can access data without registration while maintaining secure API requests with rate limiting.

6.3 Backend Layer

The backend ensures efficient data processing and API communication, employing the following technologies:

a. API Server: Node.js + Express.js for building RESTful APIs with high concurrency handling.
b. Data Processing: Python (Flask / FastAPI) for data cleansing, transformation, analysis, and anomaly detection.
c. Database Management: PostgreSQL for structured data with complex queries.
d. Data Caching: Redis to store frequently accessed data, reducing database load and improving response times.

6.4 Data Layer

This layer is responsible for storing, managing, and retrieving climate data efficiently. The hybrid database architecture supports both structured and unstructured data:

a. PostgreSQL:
   - Ideal for relational data like country-specific $CO_2$ emissions and temperature records.
b. Data Indexing & Optimization:
   - B-Tree indexing in PostgreSQL for fast query execution.
   - Partitioned storage to optimize large historical datasets.
   - Archived historical data is periodically stored in AWS S3 / Google Cloud Storage to reduce database load.

6.5 Data Processing Layer

This layer ensures data accuracy, consistency, and visualization readiness by performing the following tasks:

a. Data Cleansing: Handling missing values with mean/median imputation and normalizing data formats.
b. Anomaly Detection: Identifying and filtering extreme outliers to maintain data reliability.
c. Trend Prediction (Future Feature): Applying Machine Learning using tools likes TensorFlow / Scikit-Learn for forecasting climate trends.
d. Big Data Processing: Apache Spark / Dask to enhance large-scale data processing efficiency.

6.6 Cloud Infrastructure

The system leverages cloud-native solutions for deployment and scalability:

a. Hosting: AWS, Google Cloud, or Vercel.
b. Containerization: Docker + Kubernetes for scalable deployment and fault tolerance.
c. Backend Deployment:
    - Serverless architecture (AWS Lambda / Google Cloud Functions).
    - Kubernetes-based load balancing and auto-scaling.
d. Database Hosting: PostgreSQL for managed cloud database services.

6.7 Component Interaction Workflow

a. User Interaction:
    - The user accesses the frontend to filter, explore, and visualize climate data.
    - Interactive controls allow adjustments based on time, location, and data type.
b. Frontend Requests:
    - The frontend, built with React.js or Vue.js, sends API requests using AJAX or Fetch.
    - These requests retrieve datasets and trigger background updates if needed.
c. Backend Processing:
    - The backend first checks Redis for cached data. If available, the data is returned immediately to reduce latency.
    - If no cached data exists, the backend queries PostgreSQL for structured datasets.
    - If the requested data is missing, the backend triggers data processing. This includes AI-driven predictions, data cleansing, and transformation using Python. These steps enhance data accuracy and usability.
d. Data Processing:
    - The data processing layer analyses trends, detects anomalies, and refines raw data.
    - The processed data is stored in Redis to speed up future requests.
    - It is also indexed in PostgreSQL for better accessibility.
e. Visualization & Analysis:
    - The frontend receives the processed data and renders interactive charts and maps using D3.js or Chart.js.
    - The UI updates dynamically, allowing users to explore real-time insights.
        - Users can also export data, compare historical trends, and view AI-g

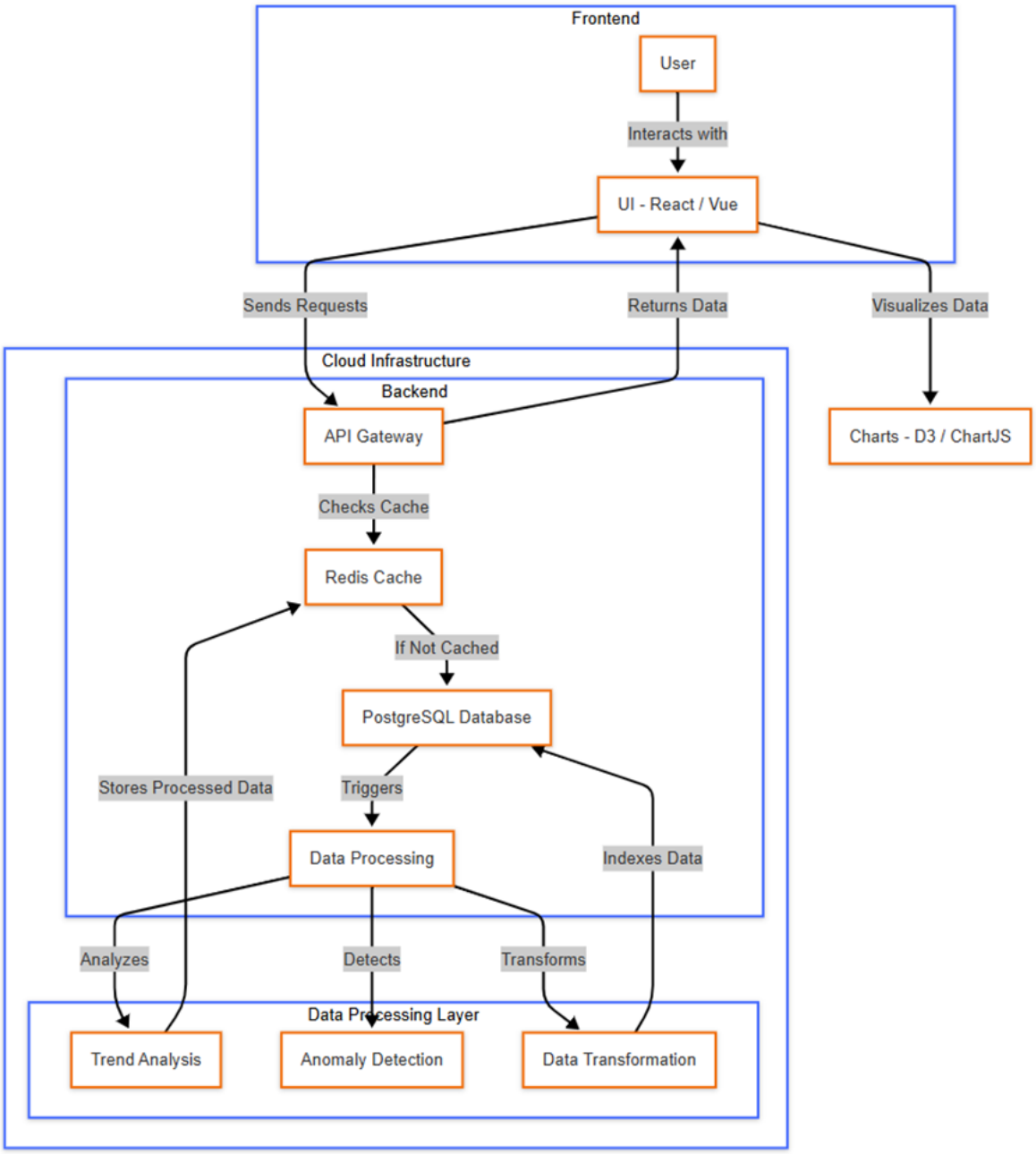


Figure 6: Architecture Outline

## 7. Initial Risk Assessment

### 7.1 Potential Risks and Mitigation Strategies

a. Large Dataset Issues: Some climate datasets may be too large, resulting in increased load times and potential performance bottlenecks.
   - Mitigation Strategy: Implement pagination, lazy loading, and data preprocessing to handle large datasets efficiently. Leverage database indexing and caching mechanisms to optimize data retrieval.

b. Unequal Workload Distribution and Poor Communication: Team members may unintentionally take on uneven workloads or lack clarity in expectations, leading to inefficiencies.
   - Mitigation Strategy: Clearly define roles within the Scrum framework (e.g., Product Owner, Scrum Master, Developers) and ensure regular, open discussions about progress and challenges during daily stand-ups. Foster a collaborative team environment to address concerns early.

c. Time Constraints: The six-week deadline for project completion is tight, leaving little room for delays.
   - Mitigation Strategy: Prioritize developing a Minimum Viable Product (MVP) in the initial phases, focusing on core functionality. After the MVP is completed, allocate the remaining time for enhancements and additional features.

d. Technical Challenges: The team may encounter technical issues with unfamiliar frameworks, tools, or integrations.
   - Mitigation Strategy: Dedicate time for learning new technologies during the first sprint. Leverage online tutorials, documentation, and community support for faster troubleshooting.

e. Stakeholder Misalignment: The project's goals or features might not align with stakeholder expectations.
   - Mitigation Strategy: Engage stakeholders regularly to review progress, gather feedback, and adjust priorities as needed. Use prototypes and iterative releases to ensure alignment.

f. Testing and Quality Assurance: Ensuring the application functions as intended across different devices and scenarios can be time intensive.
   quality
   - Mitigation Strategy: Conduct testing iteratively in each sprint, focusing on high-priority components first. Utilize automated testing tools to reduce manual effort.

### 7.2 Hosting Considerations

To ensure the application is accessible for users during development and after launch, hosting will rely on virtual cloud servers. Key considerations include:

a. Virtual Cloud Servers: A virtual server runs on an underlying physical computer and is made up of an operating system plus a set of software packages that provide the server functionality required. A virtual server is a stand-alone system that can run on any hardware in the cloud. These servers, also known as virtual machines (VMs), emulate physical hardware using virtualization software. VMs operate independently of the underlying hardware, providing flexibility and scalability.

b. Platform independence: VMs provide the same functionality as physical servers without hardware dependencies.
   - Resource scalability: Additional resources (CPU, memory, storage) can be

allocated dynamically as the application grows.
- Implementation: A hypervisor will manage the virtualization process, ensuring seamless deployment and management of server instances.7.3 Validating Audience Need

7.3 Validating Payer/Public Group Interest(s)

If the users of the application differ from the group funding it, validation strategies for the payer group include:

a. Identify Payer Group Motivations: Understand the payer group's goals, such as corporate social responsibility, brand recognition, or data-driven decision-making.
b. Demonstrate Value: Highlight how the application aligns with their objectives, such as providing actionable insights, increasing public awareness, or supporting educational initiatives.
c. Develop a Business Case: Present clear metrics for success, such as user engagement, data utilization, and the societal impact of the application.

To validate that there is an audience for the application:

a. Define Target Audience: Identify key demographics such as age, occupation, and interests. This project aims to serve climate researchers, policymakers, educators, and the general public interested in climate data.
b. Engage with Potential Users: Conduct surveys, interviews, or focus groups to gather feedback on the application's features and usability. Use this data to refine the application's value proposition.
c. Analyse competitors: Study existing climate visualization tools to identify gaps and opportunities for differentiation.

7.4 Learning New Technologies

Assuming the team needs to learn a new language or framework (e.g., React or Node)

a. Allocate Learning Time: Dedicate a portion of the first sprint to training sessions and hands-on practice.
b. Utilize Resources: leverage online tutorials, documentation, and community forums for guidance.
c. Collaborate: Encourage team members to share knowledge and assist each other in overcoming technical challenges.

7.5 Application Testing Strategies

To ensure application reliability and usability:

a. Unit Testing: Test individual components to verify their functionality in isolation.
b. Integration Testing: Validate that different components work together seamlessly.
c. User Testing: Involve end-users in testing to gather feedback on usability and performance.
d. Performance Testing: Assess the application's speed, scalability, and stability under various conditions.
e. Automated Testing: Implement automated scripts for repetitive tests to save time and improve accuracy.

# 8. Product Sprint Plan

To keep the project on track, the sprints will reference the following timeline:

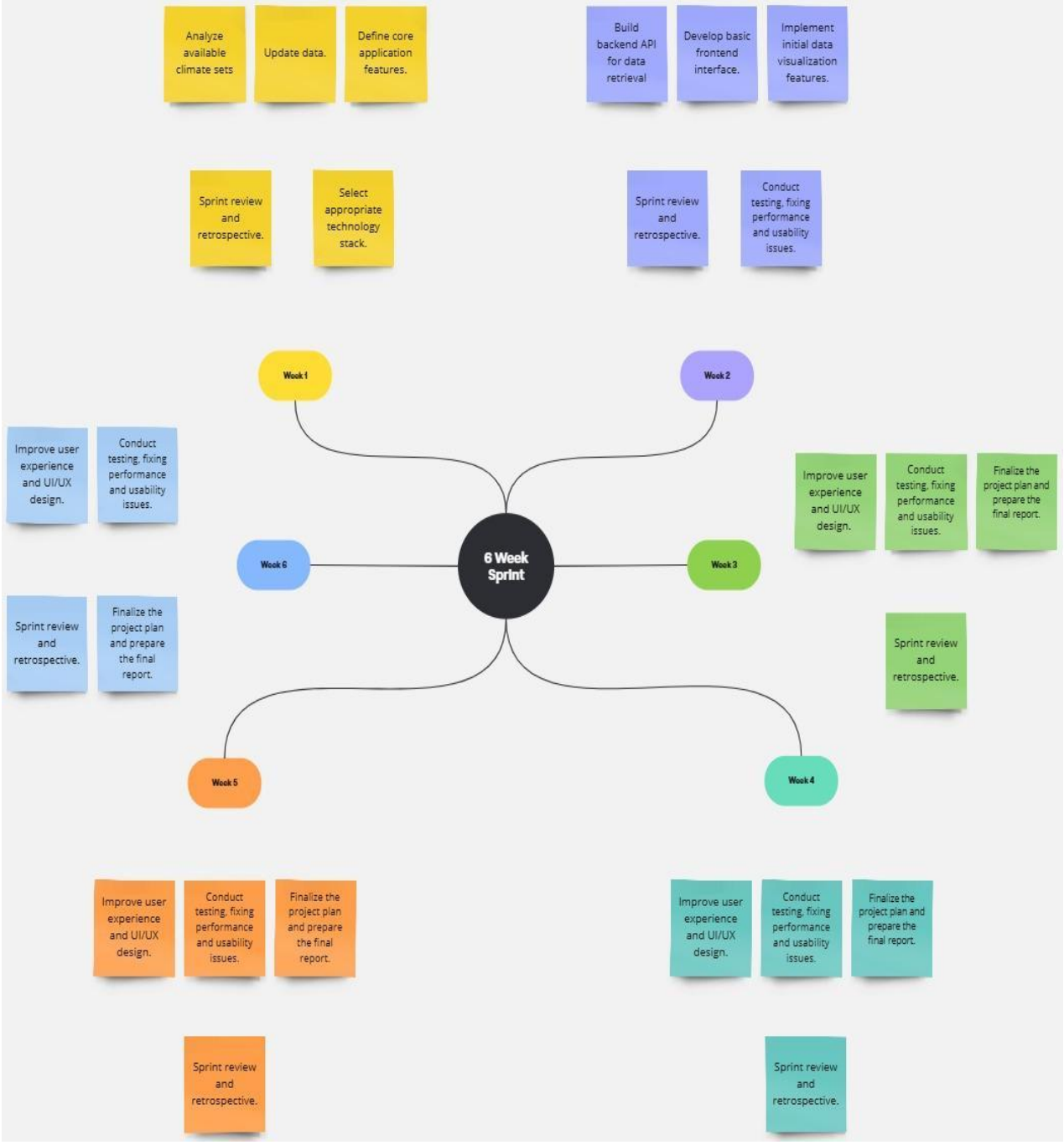


Figure 7: Six Week Sprint Plan

## 8.1 Sprint Breakdown

### Sprint 1 (Week 1): Research and Planning

- Analyse available climate datasets.
- Update data.
- Define core application features.
- Select appropriate technology stack (React, Node.js, etc.).
- Sprint review and retrospective.

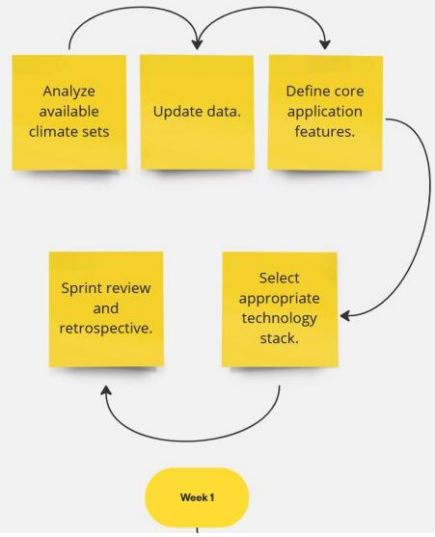


Figure 8: Week 1 Sprint Plan

a. Sprint 2 (Week 2): Initial Development
   - Build the backend API for data retrieval.
   - Develop the basic frontend interface.
   - Implement initial data visualization features.
   - Sprint review and retrospective.

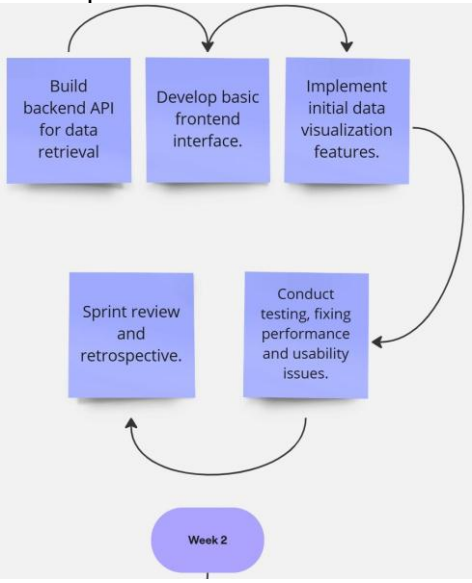


Figure 9: Week 2 Sprint Plan

b. Sprint 3 (Week 3): Optimization and Testing
   - Improve user experience and UI/UX design.
   - Conduct testing, fixing performance and usability issues.
   - Finalize the project plan and prepare the final report.
   - Sprint review and retrospective.

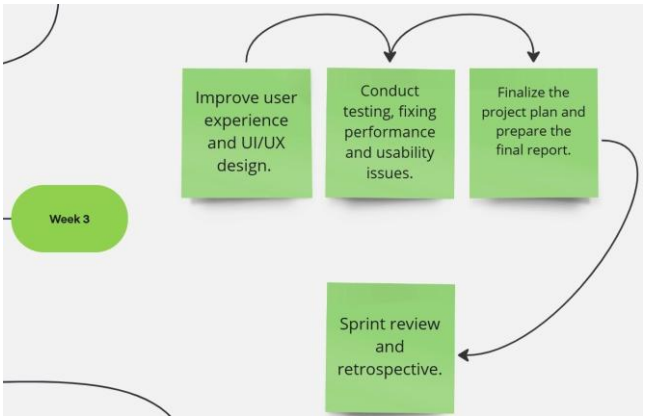


Figure 10: Week 3 Sprint Plan

c. Sprint 4 (Week 4): Optimization and Testing
   - Improve user experience and UI/UX design.
   - Conduct testing, fixing performance and usability issues.
   - Finalize the project plan and prepare the final report.
   - Sprint review and retrospective

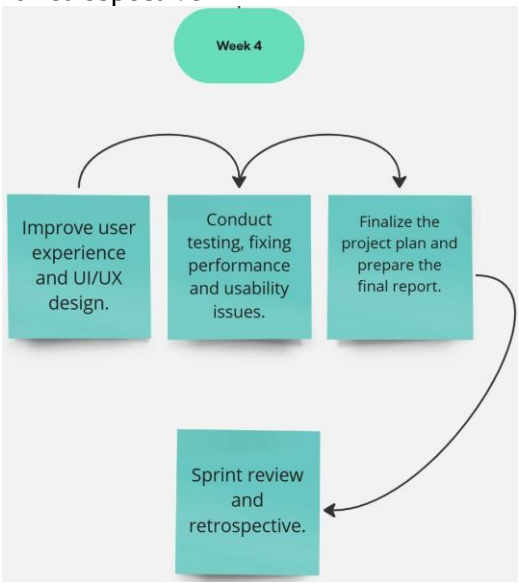


Figure 11: Week 4 Sprint Plan

d. Sprint 5 (Week 5): Optimization and Testing
   - Improve user experience and UI/UX design.
   - Conduct testing, fixing performance and usability issues.
   - Finalize the project plan and prepare the final report.
   - Sprint review and retrospective.

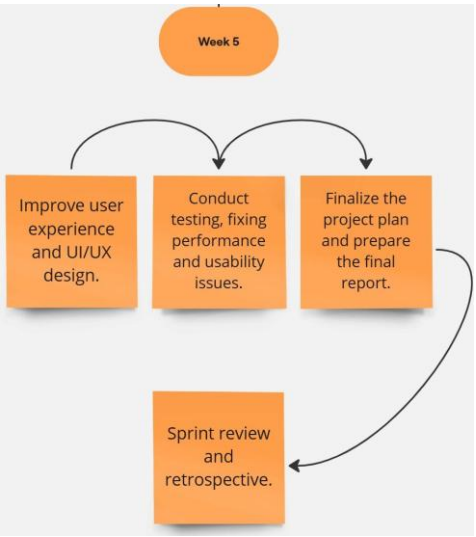

Figure 12: Week 5 Sprint Plan

e. Sprint 6 (Week 6): Optimization and Testing
   - Improve user experience and UI/UX design.
   - Conduct testing, fixing performance and usability issues.
   - Finalize the project plan and prepare the final report.
   - Sprint review and retrospective.

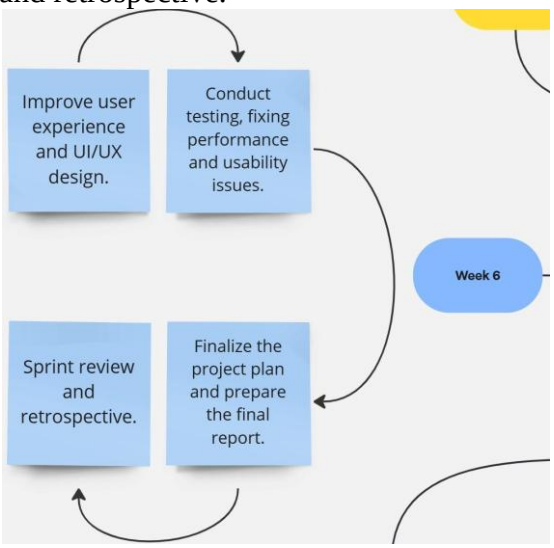

Figure 13: Week 6 Sprint Plan

8.2 Daily Stand-Up:

The listed questions below are to be referenced by the team in our daily communication. These will be implemented by each individual separately but will make sure the group is working on their respective jobs with the focus on the end goals.

Here are the questions:

a. Questions:
- What has everyone done?
- Any obstacles?
- On track to finish this sprint?
  - If not, take note to check in again and possibly move to next sprint
- What needs to still be done?
- What's everyone going to do today?

- What’s the feedback from the people you’ve shown the app to?
- Is what we are working on directly answering the product backlog?

## Appendices

1. Project Vision……..……3-5
1.1 Mission……..……3
1.2 Goals……..…3
1.3 About the Team……..…3
1.4 What We Offer…….…….3-4
Figure 1: User Interface……..4
1.5 Main Roles……..……..…4
1.6.1 Project Constraints……..…..…..…4
1.6.2 Our Approach to Constraints…….……5
2. Software Development Process..……6-8
2.1 Chosen Approach……...…...……6
2.2 Why Agile? ……..…..…6
2.3 XP/Scrum…….…..…6-7
2.4 Product Backlog / User End……..……7-8
Figure 2: Server-Side of App……..8
2.5 Application Six Week Plan……8
Figure 3: Six Week Plan……..8
3. Project Success Criteria……...…9
4. Technical Details……..…10-12
4.1 Product Overview……10
4.2 Technology Selection……10-11
Table 1: Technology Selection……10-11
4.3 System Architecture……11
4.4 Data Flow……11
4.5 Possible Technical Challenges…….11-12
Table 2: Possible Technical Challenges…….11-12
5. Key Use Case Models……13-16
5.1 Overview of Use Cases……13
Figure 1: User Interface…….13
5.2 User Case Modelling……13-14
Figure 4: Case Model Example (Australia).....13
5.3 Description of Key Use Cases……..14-15
Table 3: Use Case 1: Browsing Data……..14
Table 4: Use Case 2: Filtering Data……..…..14
Table 5: Use Case 3: Data Comparison……..…..…..14-15
Table 6: Use Case 4: Data Download (Optional).....15
Figure 5: Data Graph Examples…….15
5.4 Component Interaction…….15-16
5.5 Summary……..16
6. Candidate System Architecture..……17-20
6.1 System……..17
6.2 Frontend Layer…….17-18
6.3 Backend Layer……..18
6.4 Data Layer……18
6.5 Data Processing Layer……..18
6.6 Cloud Infrastructure……..19
6.7 Component Interaction Workflow……19-20
Figure 6: Architecture Outline……20
7. Initial Risk Assessment..……21-22

7.1 Potential Risks and Mitigation Strategies……21
7.2 Hosting Considerations……21-22
7.3 Validating Payer/Public Group Interest(s)……22
7.4 Learning New Technologies……..……22
7.5 Application Testing Strategies……..……22
8. Product Sprint Plan.……23-27
Figure 7: Six Week Sprint Plan……...23
8.1 Sprint Breakdown……24-26
Figure 8: Week 1 Sprint Plan……..24
Figure 9: Week 2 Sprint Plan……...24
Figure 10: Week 3 Sprint Plan……25
Figure 11: Week 4 Sprint Plan……25
Figure 12: Week 5 Sprint Plan……26
Figure 13: Week 6 Sprint Plan……26
8.2 Daily Stand-Up…….26-27