TRAVELYTICS

Ayden Badyal, Alex Chen, Carter Jones, Daniel Shi

https://github.com/CMPT-276-SUMMER-2025/final-project-1-sky

Milestone 1

Finalized 2 APIs

The two main API's we will be using for the project are GeoDB cities and Weather APi.

GeoDB Cities

The GeoDB Cities API provides detailed information about cities, including population, timezone, elevation, and more. It also features an autocomplete search for cities and GPS-based data retrieval. This API will power the destination information module for our website. When a user searches and then selects a city, the website will display detailed contextual data such as the city's timezone, elevation, population, and GPS location. This helps users plan better by understanding the size, time difference, and terrain of the destination.

Feature 1: City Population Viewer

- Description: Display the current population of selected cities.
- User Benefits: Helps users gauge how busy or developed a city is, aiding in choosing between urban vs. less crowded travel experiences.
- Changes since proposal: No changes have been made.

Feature 2: Timezone Display

- Description: Show the city's current timezone.
- User Benefits: Crucial for scheduling flights and meetings, especially for business travellers, or planning activities in a different timezone.
- Changes since the proposal: No changes have been made.

Feature 3: City Elevation Information

- Description: Display the city's elevation.
- User Benefits: Important for travellers sensitive to altitude. Also helps anticipate possible weather effects due to elevation.
- Changes since the proposal: No changes have been made.

Weather API

WeatherAPI delivers real-time and forecasted weather data, air quality reports, and historical weather records. WeatherAPI will power our website's weather dashboard. It provides a 3-day forecast, past weather information, and air quality data—all essential for helping users plan trips based on environmental conditions.

Feature 4: 3-Day Weather Forecast

- Description: Show a 3-day weather forecast for temperature, rain chance, and other conditions.
- User Benefit: Allows the user to plan their travel based on the condition and be able to pack according to the weather.
- Changes since the proposal: No changes have been made.

Feature 5: Historical Weather Overview

- Description: Display weather from the past 7 days.
- User Benefit: Helps users understand typical recent weather trends in the area, especially useful for seasonal travel decisions.
- Changes since proposal: No changes have been made.

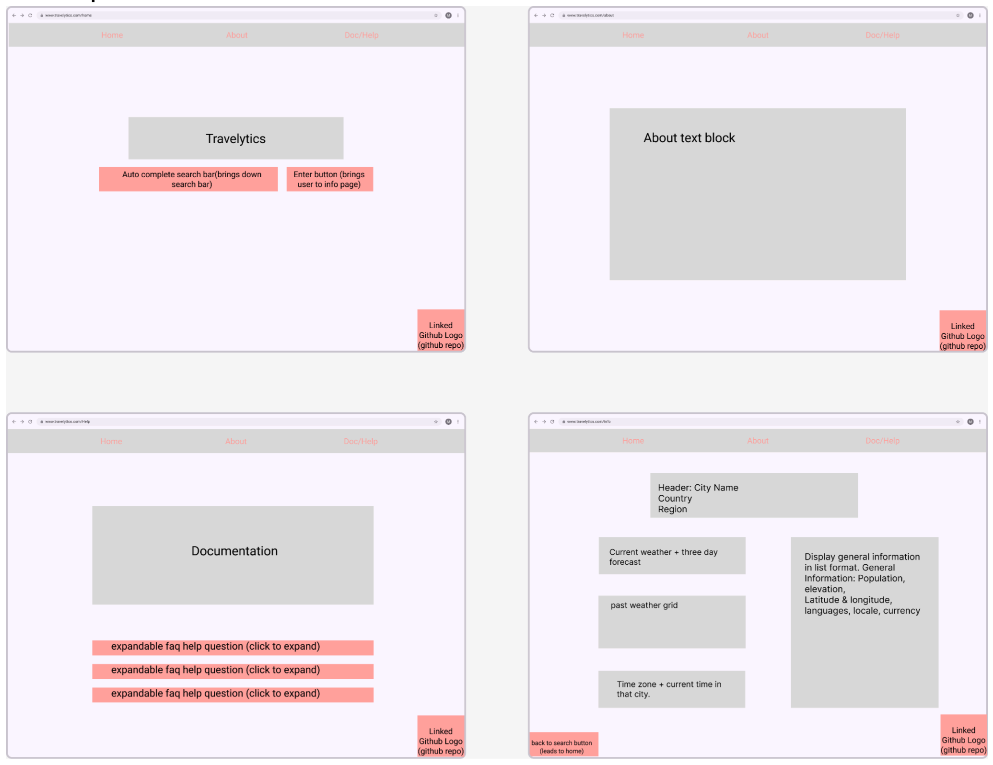
Feature 6: Air Quality Viewer

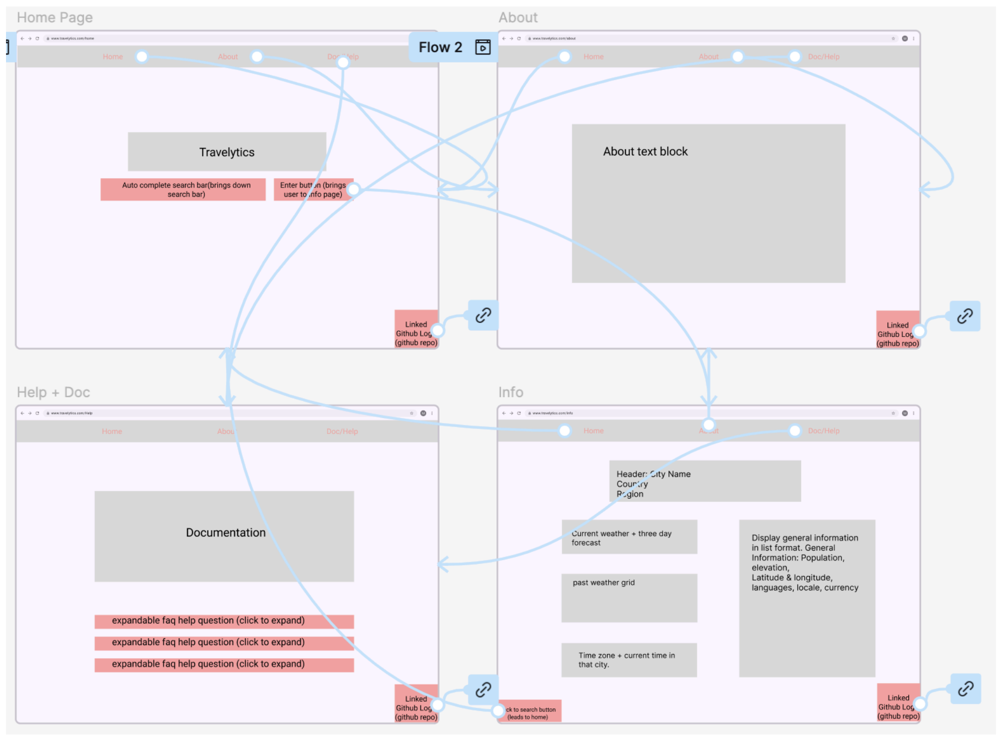
- Description: Real-time display of air quality for each city.
- User Benefit: Essential for travellers with lung concerns or general health considerations.
- Changes since the proposal: No changes have been made.

Mid-Fidelity Prototype

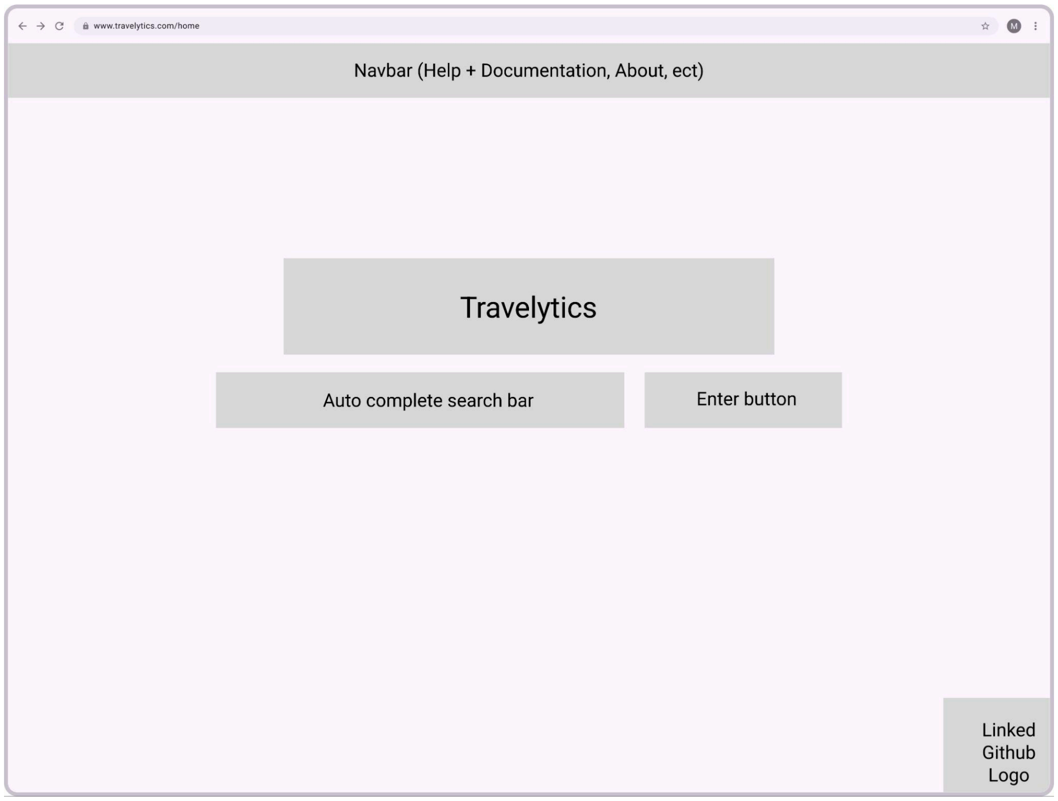
Interactive figma link:
https://www.figma.com/design/KKKtptn4ccFuYjjYODwL4R/M0?node-id=0-1&t=xyI7S9nTLNhK18ZZ-1

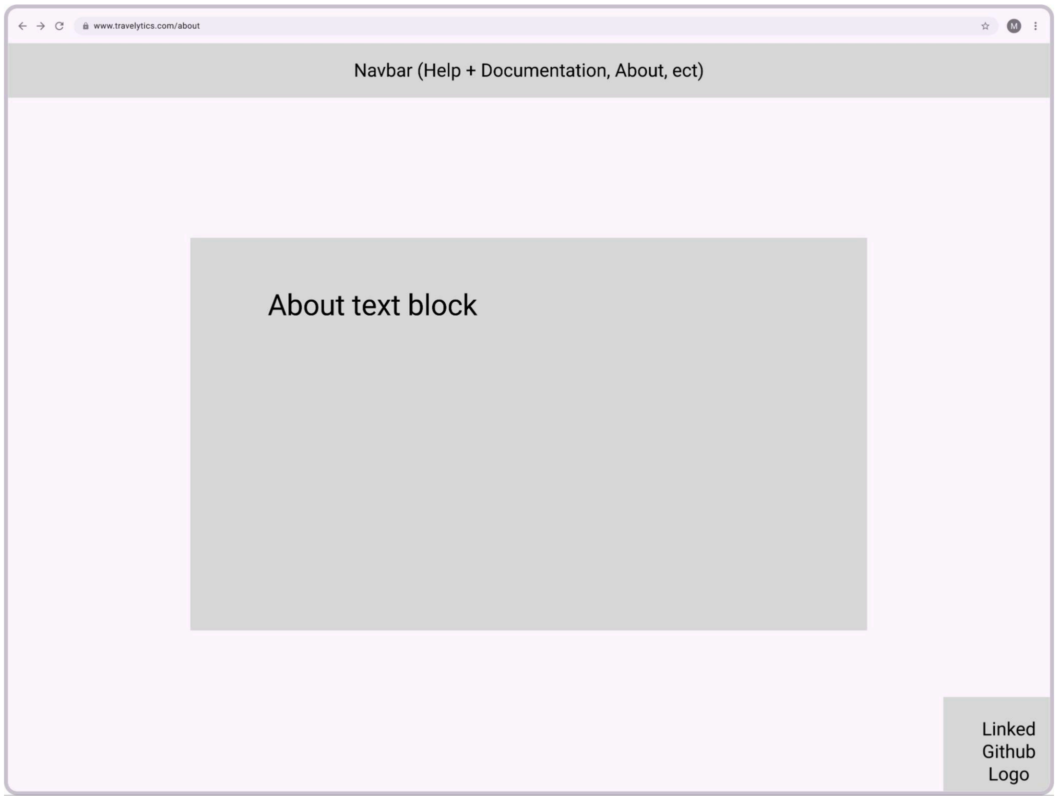
Red represents the interactive buttons

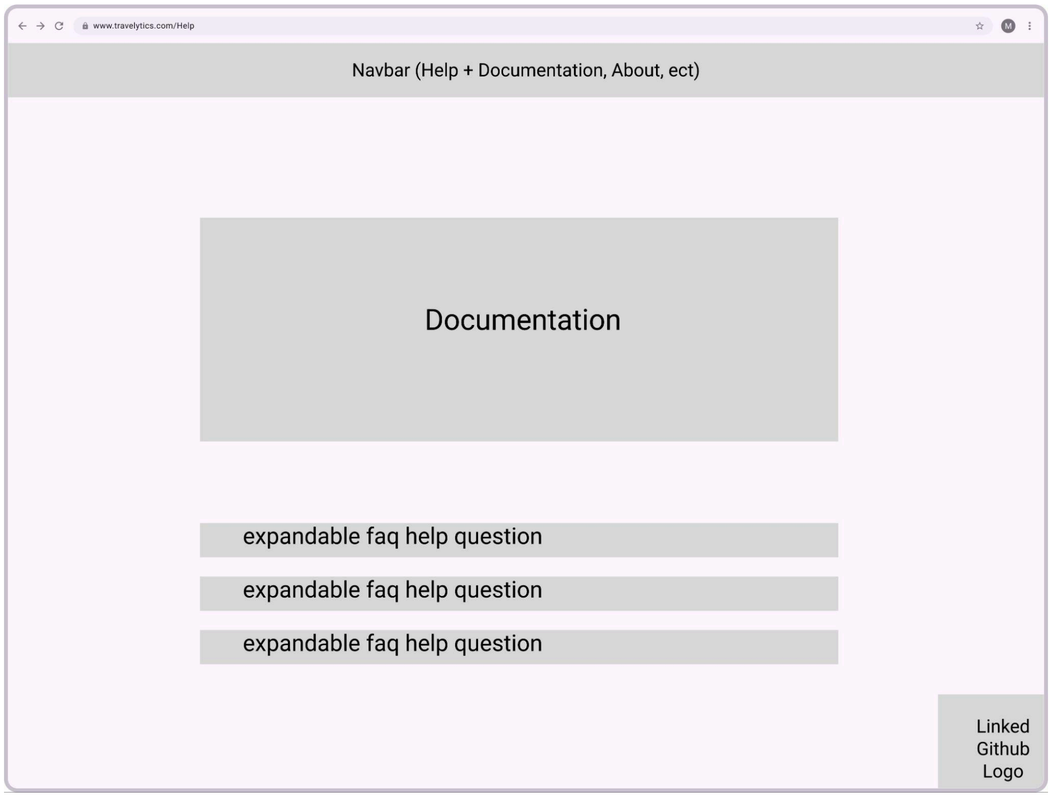

Home
About
Doc/Help
Travelytics
Auto complete search bar(brings down search bar)
Enter button (brings user to info page)
Linked Github Logo (github repo)
About text block
Documentation
expandable faq help question (click to expand)
expandable faq help question (click to expand)
expandable faq help question (click to expand)
Header: City Name Country Region
Current weather + three day forecast
past weather grid
Time zone + current time in that city.
Display general information in list format. General Information: Population, elevation, Latitude & longitude, languages, locale, currency
back to search button (leads to home)

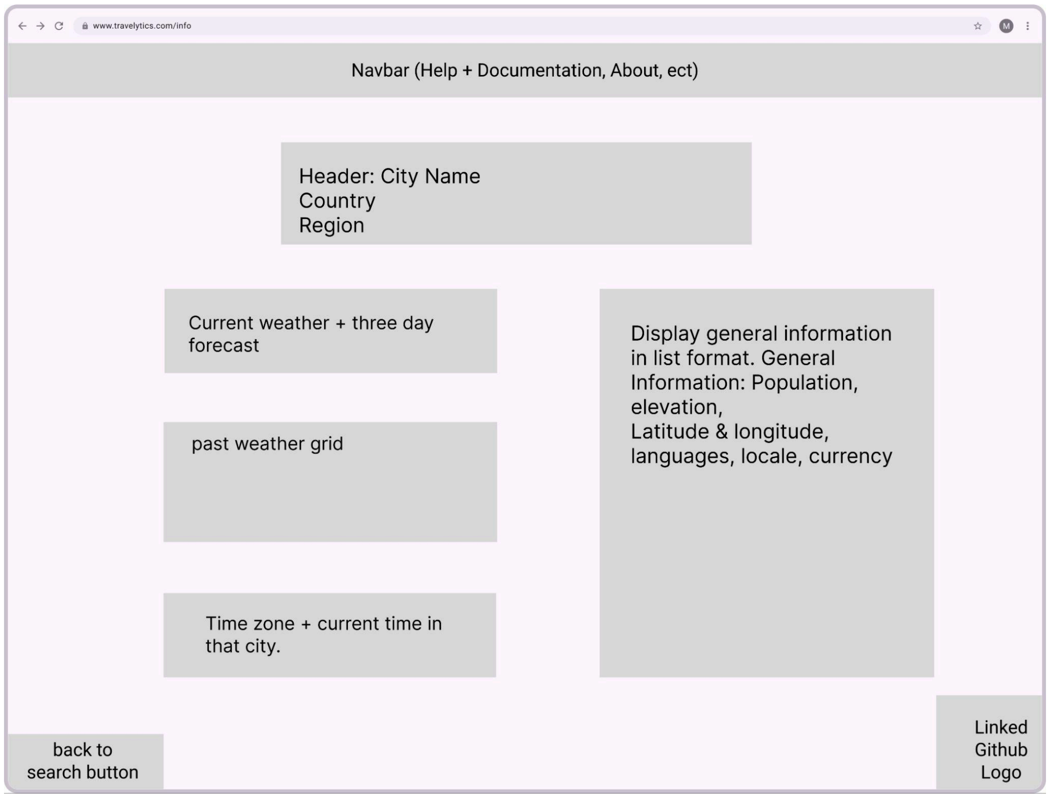

Home Page
About
Flow 2
Home
About
Doc/Help
Travelytics
Auto complete search bar(brings down search bar)
Enter button (brings user to info page)
Linked Github Logo (github repo)
About text block
Help + Doc
Info
Documentation
expandable faq help question (click to expand)
expandable faq help question (click to expand)
expandable faq help question (click to expand)
Header: City Name Country Region
Current weather + three day forecast
past weather grid
Time zone + current time in that city.
Display general information in list format. General Information: Population, elevation, Latitude & longitude, languages, locale, currency
Back to search button (leads to home)

M0 resubmission Wireframe for comparison


www.travelytics.com/home
Navbar (Help + Documentation, About, ect)
Travelytics
Auto complete search bar
Enter button
Linked Github Logo


www.travelytics.com/about
Navbar (Help + Documentation, About, ect)
About text block
Linked Github Logo


www.travelytics.com/Help
Navbar (Help + Documentation, About, ect)
Documentation
expandable faq help question
expandable faq help question
expandable faq help question
Linked Github Logo


www.travelytics.com/info
Navbar (Help + Documentation, About, ect)
Header: City Name Country Region
Current weather + three day forecast
past weather grid
Time zone + current time in that city.
Display general information in list format. General Information: Population, elevation, Latitude & longitude, languages, locale, currency
back to search button
Linked Github Logo

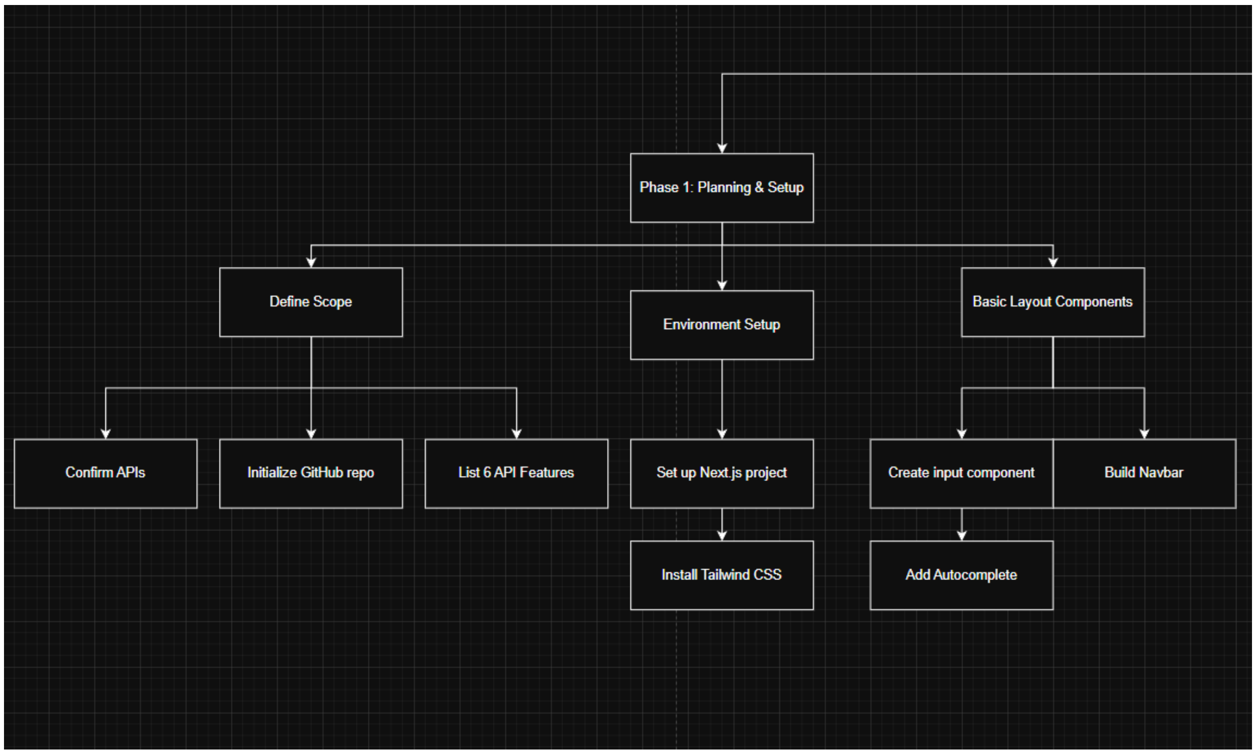
Chosen SDLC Model

The SDLC model we wish to use is agile, more specifically kanban with scrum. All of us within the group have had extremely positive experiences with using kanban in our mini-projects. We have added Scrum to encourage collaboration among our group members. The agile model will help us prioritize what is important,- such as flexibility, interactions, and working code.

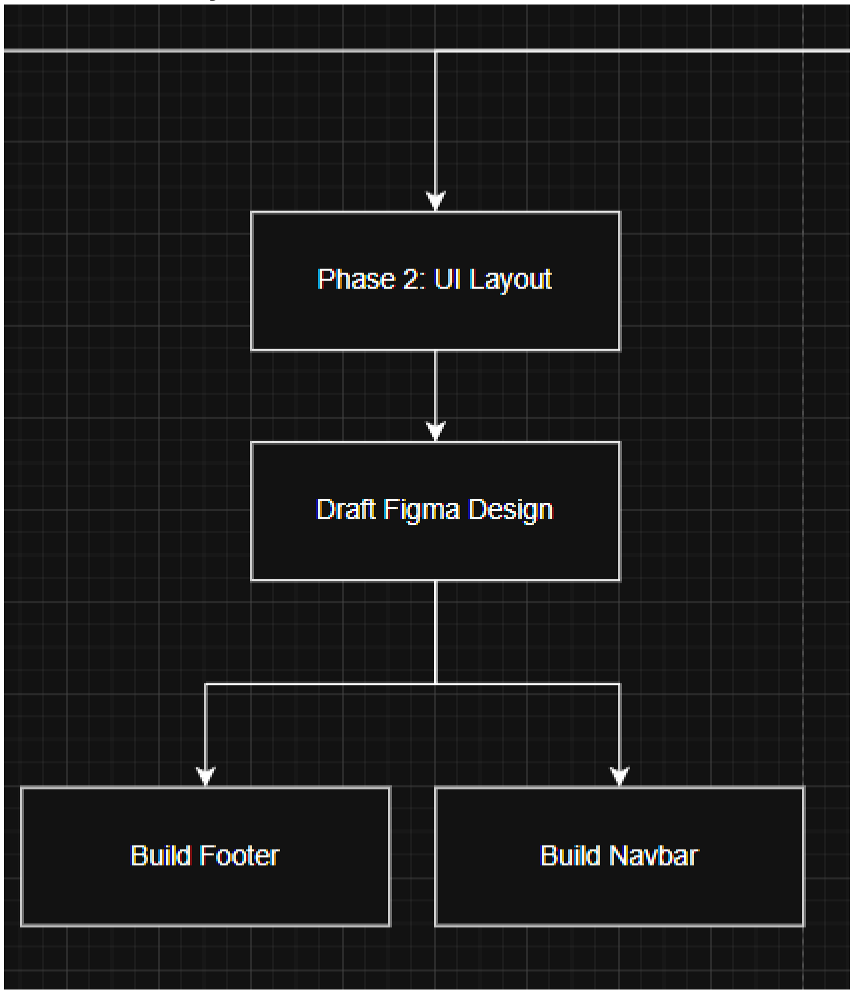
Work Breakdown Structure

NOTE: The Work Breakdown structure is too large to fit in this document, so it has been split in parts. Linked below is the full WBS layout:
https://drive.google.com/file/d/181Uxrq4mRzKv-9ZJqx8LLSUVnjHv7x4o/view?usp=sharing

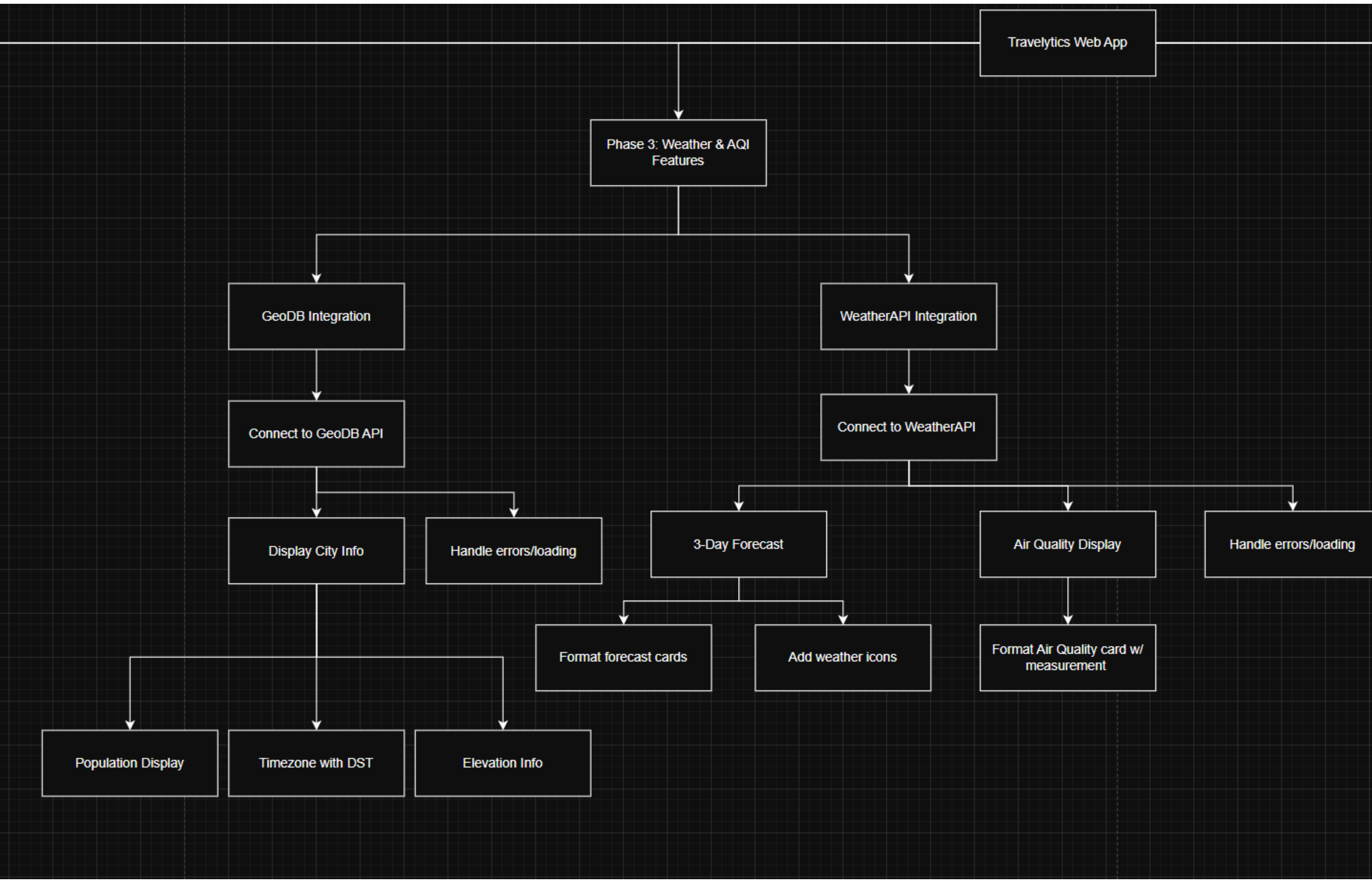
Phase 1: Planning & Setup



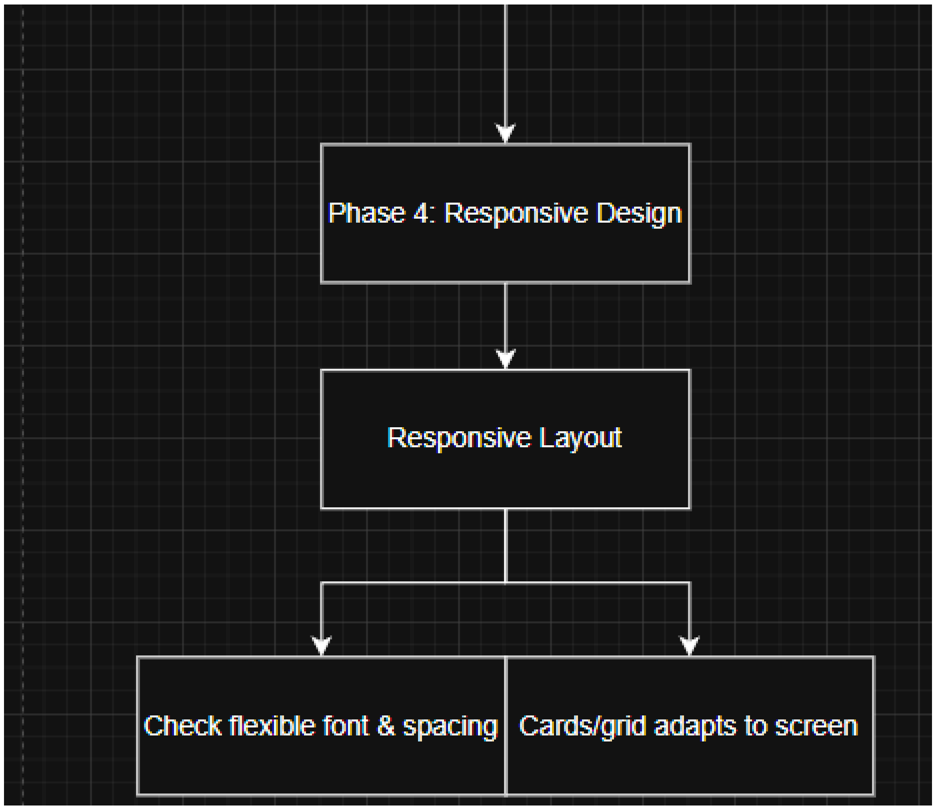
Phase 2: UI Layout



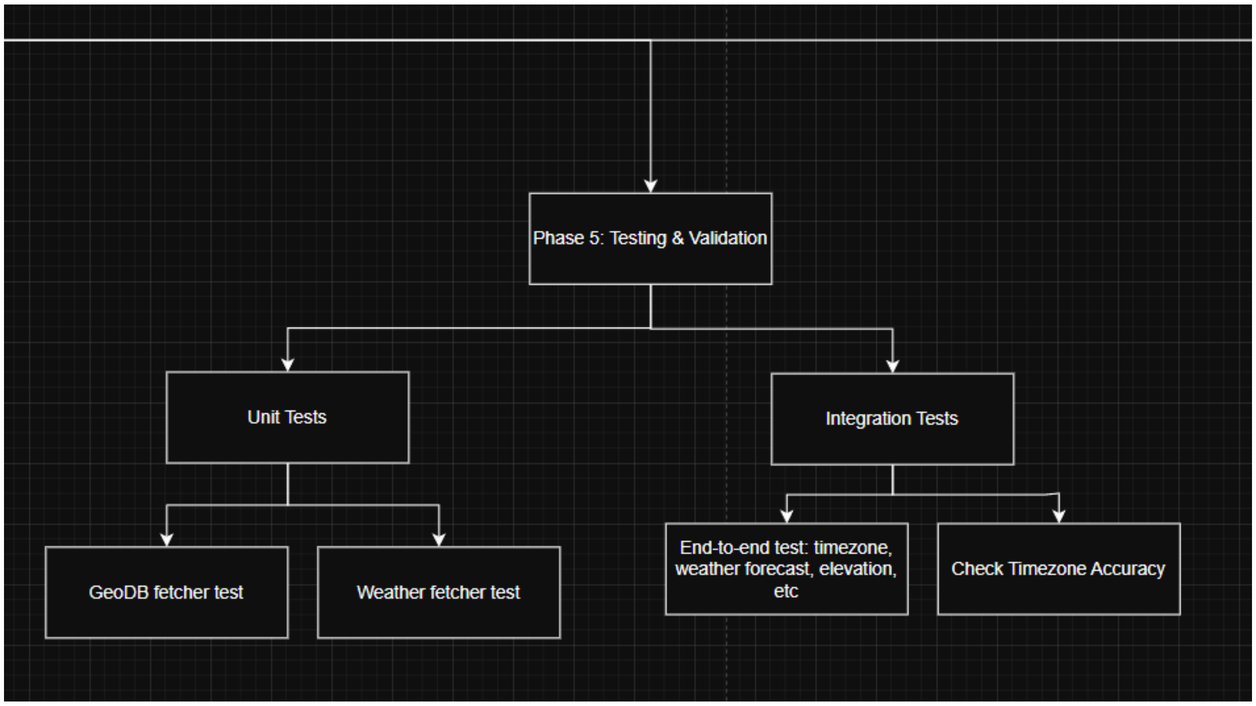
Phase 3: Weather Features



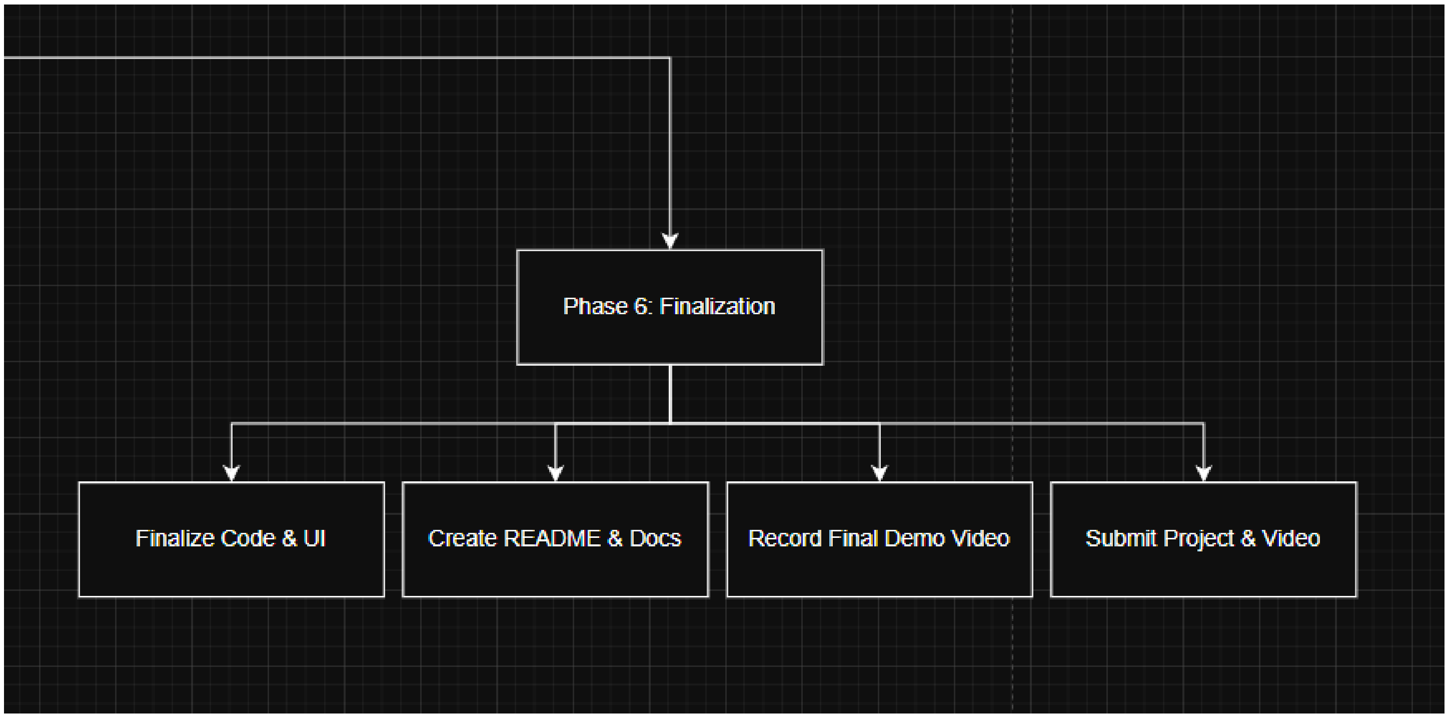
Phase 4: Responsive Design


Phase 4: Responsive Design
Responsive Layout
Check flexible font & spacing
Cards/grid adapts to screen

Phase 5: Testing and Validation


Phase 5: Testing & Validation
Unit Tests
Integration Tests
GeoDB fetcher test
Weather fetcher test
End-to-end test: timezone, weather forecast, elevation, etc
Check Timezone Accuracy

Phase 6: Finalization



Project Schedule with Milestones and Deadline

Date	**Internal Tasks & Milestones**
Jun 27-Jul 2	Project Planning, work on milestone 1
Jul 3	Buffer Day to check over Milestone 1
July 4	Submit M1 report
Jul 4-5	Film & Edit Video
Jul 6	Buffer Day for Video
Jul 7	Submit Video
Jul 8-12	Navbar, Footer, GeoDB API integration (autocomplete, population, timezone) + location
Jul 12	Buffer Day for July 8 - 12 implementation period
Jul 13-17	WeatherAPI integration (3 day forecast), air

	quality, past weather implementation
Jul 19-24	UI polish, responsive design
Jul 25	Buffer Day to check over UI and responsiveness
Jul 26–Aug2	QA testing, validation, finalize core logic
Aug 3-4	Record and edit the final video
Aug 5	Buffer Day to check over the site and docs
Aug 6	Submit a Repo that contains finalized code
Aug 7	Buffer Day for Video recording/editing
Aug 8	Submit Final Video

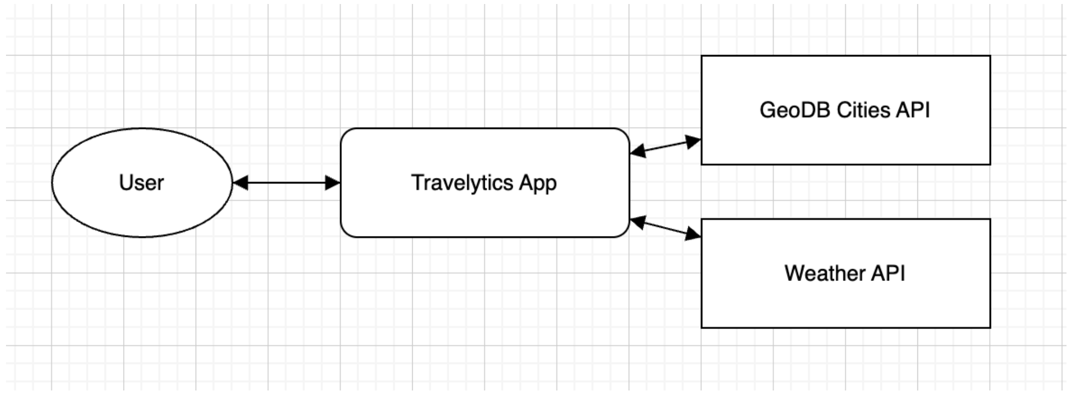
Risk Assessment

- Low-risk
    - Typos or inaccurate labels
        - Conduct peer reviews and use spell-check tools during development
    - Issues with the deployment of the website
        - Use staging environments to test deployments before production. Maintain deployment checklists and backup hosting solutions. Document the deployment process thoroughly
    - Git merge conflicts
        - Establish a clear branching strategy, encourage frequent commits, and conduct regular code reviews
    - Minor api changes
        - Monitor API changelogs, version APIs where possible, and use abstraction layers to isolate API calls for easier updates
    - Incomplete or unclear project documentation
        - Assign documentation as a parallel task during development, perform regular doc reviews and maintain a living documentation repository
- Medium-risk
    - Api limit hit when testing
        - Use backup api keys with higher rate limits when available
    - The application works on some browsers but fails on others
        - Test across multiple browsers and use the compatibility tools
    - Slow loading times due to excessive api requests
        - Reduce the number of requests and store (cache) results when possible
    - Team member is unavailable due to unforeseen events

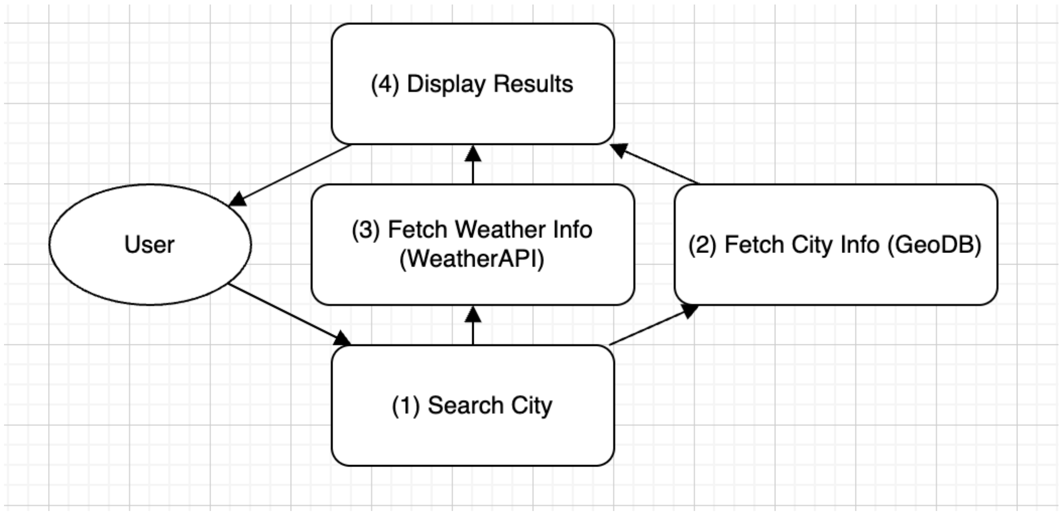
        - Make sure team members know each other's work and keep things well documented so someone else can step in.
    - Inconsistent data returned from api
        - Validate and clean data before using it, and add checks in your code to handle unexpected results.
- High-risk
    - One of our primary API's becomes unavailable
        - Use the backup API
    - The project scope is too big due to underestimating the workload
        - Focus on the most important features first and break the work into smaller, manageable parts
    - Unexpected technical challenges when integrating multiple APIs with the user interface
        - Test small parts early and write the code in a way that makes it easy to plug things together
    - A major bug is found near the end of the testing phase, with the possibility of not being able to fix it due to a lack of time
        - Test often throughout the project, not just at the end, so we can catch the issues early
    - Cite crashes under heavy use
        - Test how your website performs with lots of users and improve speed where needed

Data Flow Diagram (level 0 and 1)

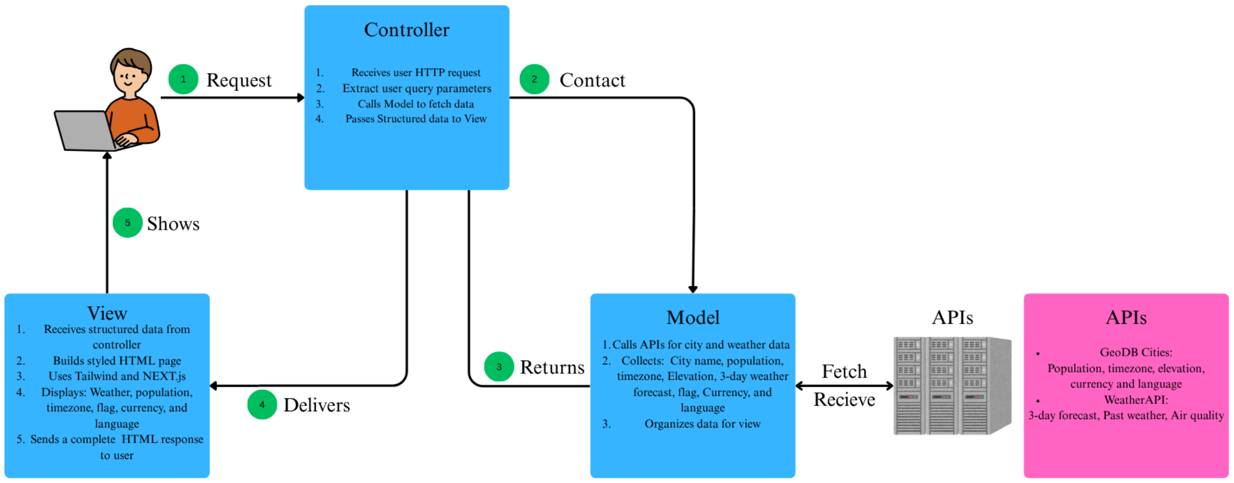
Level 0



Level 1


(4) Display Results
User
(3) Fetch Weather Info (WeatherAPI)
(2) Fetch City Info (GeoDB)
(1) Search City

MVC Model Diagram


Controller
1. Receives user HTTP request
2. Extract user query parameters
3. Calls Model to fetch data
4. Passes Structured data to View
1 Request
2 Contact
5 Shows
View
1. Receives structured data from controller
2. Builds styled HTML page
3. Uses Tailwind and NEXT.js
4. Displays: Weather, population, timezone, flag, currency, and language
5. Sends a complete HTML response to user
4 Delivers
3 Returns
Model
1. Calls APIs for city and weather data
2. Collects: City name, population, timezone, Elevation, 3-day weather forecast, flag, Currency, and language
3. Organizes data for view
Fetch
Recieve
APIs
APIs
• GeoDB Cities: Population, timezone, elevation, currency and language
• WeatherAPI: 3-day forecast, Past weather, Air quality

This diagram represents the high-level MVC architecture that Travelytics will follow. The user will begin by submitting a request through the web interface by searching for their desired city. The controller receives the request and interprets the user's input. It will then call the Model, which will deal with the data-fetching requests. The Model will fetch and receive info for our two chosen APIs contained in the pink box. Once the model has collected the needed data, it will process it and hand it back to the controller. The Controller will then send this data to the View, which formats the given data with HTML, CSS, and JavaScript to produce a response the user can understand and interact with. Then the process repeats for any further city search requests.

Appendix

Tasks:
M1 Tasks distributed among us:
Explanation of the finalized 2 APIs: Ayden
Mid-Fidelity prototype: Daniel
SDLC Explanation: Daniel
Work Breakdown Structure: Alex
Project Schedule: Alex
Risk Assessment: Ayden
DFD: Daniel
MVC: Carter
Appendix: Carter

M0 Task distributions:
Project overview: Alex
The problem: Daniel
Potential users: Ayden
Personas: Carter
Original APIs: Ayden
Revised APIs (in red): Carter
User stories: Daniel
Wireframe: Daniel
Front-end Tech stack: Alex

Project Proposal:
No direct distribution. An online meeting was held, and Alex completed the document based on the team's group decision for each part as we worked collaboratively.

Change Log:

Date of Change	What was Changed
June 22, 7:02 PM	Carter fixed M0 TA feedback: “ It would be better to have some backup APIs for your original APIs, if possible.” Made a clear separation of the main APIs and the backup APIs. Carter fixed M0 TA feedback: Fill out the paid API disclosure form. Made a

	disclaimer that we are only using the free version of the APIs, so no form needs to be filled out.
June 26, 3:15 PM	Alex created the M1 document, shared it, and added all tasks that needed to be completed Alex also added the project schedule
June 26, 5:38 PM	Daniel updates the wireframe in M0 to reflect TA feedback.
June 27,1:18 PM	Ayden added Finalized APIs Daniel added Chosen SDLC model and explanation Alex deleted and created new Project schedule Daniel linked buttons to make Figma interactive
June 27, 1:32 PM	Ayden updated Finalized APIs Alex added detail to the project schedule Daniel added a link to the DFD Figma
June 27, 8:47 PM	Carter added an MVC diagram from Canva + a description of DFD.
June 27, 9:32 PM	Ayden adds all project features + overview to the finalized APIs
June 30, 3:27	Ayden makes minor changes to feature lists Daniel updates the Mid-fidelity prototype to be interactive Alex adds WBS Ayden adds risk assessment. Daniel replaces the DFD link with screenshots
July 1, 5:17 PM	Alex reformats the WBS to be broken down into phases to be more readable
July 3, 11:49 PM	Daniel updated the interactive Figma link

	because the old one stopped working
July 4, 1:06	Ayden finalizes changes to his API breakdowns Carter finalizes MVC and adds an appendix to the document Daniel finalizes that his parts are complete Alex finalizes that his parts are complete
July 4, 1:30 PM	Carter finalizes the appendix task distribution and the change log