TRAVELYTICS

Ayden Badyal, Alex Chen, Carter Jones, Daniel Shi

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

https://youtu.be/gSQIN4Iacw0

https://travelytics276.vercel.app/

Milestone 2

Analysis of the project’s success in meeting the user needs and solving the problem

Our project, Travelytics, successfully addressed user needs by providing an intuitive, helpful, and visually clear platform for discovering weather and city information. To gather insightful feedback regarding our application, we opted to use Google Forms on the day of the peer testing session. This feedback was helpful and revealed several strengths as well as some possible future improvements.

Starting with what worked well, 5 out of 6 users rated our app as “good” in terms of the ability to navigate easily. This showed us that the interface of our application was very intuitive to use. All users who tested our application also found the city search feature straightforward to use. Moreover, 5 users found our weather data to be “mostly accurate,” and all city information was “very clear and informative,” coupled with a smooth, clean, and simple UI, which confirmed that the design choices we made had a positive impact on user experience.

From our feedback form, we also curated a list of what could be improved based on the feedback from our application testers. Multiple users mentioned that our dropdown menu was confusing or not clearly labelled. One suggestion was to indicate that it only supports Canadian cities at the time being. In addition, users noted that there wasn’t much to do beyond basic city searches and weather lookup.

Thus, this brings us to our opportunities for growth and possible future upgrades/implementations. Some features users desired were the following: ability to compare multiple cities side-by-side, save favorite cities, show travel-inspiring content like local attractions, add a map for better context, display alerts, or recommend activities based on city and weather.

Most users rated their experience between an 8 and 9 out of 10, which we believe means that we have achieved our goal of helping users quickly find accurate weather and city info in a clean, intuitive interface. Putting our application into a real-life test confirmed that our app has a strong, usable design. Moving forward, expanding features and improving UI feedback will make our app even more powerful and user-friendly.

Analysis of the project’s SDLC model and how it was used in the project

The chosen model, as mentioned in our M1 report, was Kanban along with Scrum. Throughout the project, our team maintained strong communication through our group chat, and whenever each of us started an issue on GitHub, aside from assigning it to ourselves via GitHub, we would also tell the group chat that we were working on a certain issue. The Kanban on our GitHub projects was not used as much as predicted. Rather, we kept track of our issues through our group chat with updates at each development step for the issue. We still held weekly meetings either virtually or in person to update each other on the work we did. We found that texting in our group chat was more efficient than having online calls, so we combined this with our in-person meetings either before or after our Wednesday classes. Our group found that keeping things over text allowed for more efficient tracking of what we discussed in our text meetings.

Detailed description of the features implemented for each API

Our project is powered by GeoNames and OpenWeatherMap API, which we originally had GeoDB cities, but switched due to limitations. GeoNames API powers all of the city-related data on our website. It provides key information such as the country's current population, timezone, elevation, and real-time local time. We also used it to help implement a city search bar with autocomplete functionality. The current population, however, isn't up to date, so we are communicating with the development team of the API and trying to get it updated. We were able to get moderation powers for the GeoNames API database; all cities featured by Travelytics have been updated on the web database, but changes have not been reflected in API responses. For the timezone, we originally had it show it showed the time when the user entered the city, but however added a live timer which updates so that the time is more consistent. For the rest of the features we added for GeoNames, we didn't have any changes from our original plan.

The OpenWeatherMap API provides an extensive report of all weather-related data. After switching from WeatherAPI, we expanded our weather features thanks to the broader data set and more consistent data offered by OpenWeatherMap API. Our first feature was our Current weather conditions, which displays the city's current temperature, humidity, wind speed, pressure, and general weather conditions using icons. The change we made was expanding the weather data presentation by showing more information, which was seen for every feature. We had originally planned to use a 3-day weather forecast, but updated it to a 5-day weather forecast, which was provided by the newer API. This feature shows the high, low, and average temperature of the following days while also showing the rain chance and wind speed. Originally, it only showed the current temperature, but the value was off by a bit, so we fixed that and added the high and lows in our forecast. Another feature we added was the 7-day historical weather overview, which showed the past weather over the last 7 days, so users can get a trend of the weather pattern. We didn't make any changes to this feature, although we enhanced some of its components to match the other features. Lastly, we have our air quality index view, which provides real-time air quality, which is in the Canadian AQHI standard for better relevance. Originally, we planned it as a simple air quality display, but we added a custom AQHI conversion and visual scaling to help users get a better idea.

Detailed description and overview of the CI/CD pipeline and how it was used in the project

For our project, we implemented a CI/CD pipeline using GitHub Actions to ensure reliability and consistency throughout development. Our pipeline is triggered whenever code is pushed to the main branch or a pull request is made targeting it. The first workflow, named CI/CD, runs on a Windows environment. It checks out the code, sets up Node.js, installs all dependencies using npm ci, and builds the project using npm run build. This ensures that any new changes do not break the build and that the application compiles correctly before merging.

To automate test execution and maintain code quality, we configured a GitHub Actions workflow that triggers on every push and pull request to the main branch. The workflow checks out the repository, installs dependencies, and runs the full test suite via the npm script ‘npm run test’, which executes Jest tests. This setup ensures that no code is merged into the branch without passing all unit and integration tests, enforcing a quality gate and preventing deployed issues.

Together, these CI/CD workflows helped us catch issues early through continuous testing and guaranteed that only stable, build-passing code was deployed. While our current setup does not include automated deployment or monitoring, the workflows lay a strong foundation for integrating those features in the future.

Description of the project’s testing strategy and how it was implemented

Our project's testing strategy is centered on ensuring code correctness, reliability, and maintainability through comprehensive automated testing. The strategy involves unit and integration testing supported by a CI pipeline that runs tests automatically on every code change, using Jest and GitHub Actions.

1. Unit Testing with Jest

We used Jest as our primary testing framework due to its mocking capabilities, ease of use, and assertion library. Unit tests in our project focus on verifying individual functions and modules in isolation, independent from external dependencies.

For example, in openweatherService.test.ts, we test the fetchOpenWeatherData function from openweatherService.ts. This function interacts with multiple OpenWeather API endpoints to retrieve current weather, forecast, historical data, and air quality information. We mock external API calls using Jest’s mocking mechanisms:

- We mock the global fetch function to simulate the geocoding API response, returning fixed latitude and longitude coordinates.
- We use jest.spyOn(Promise, ‘allSettled’) to mock the parallel calls made for the current weather, forecast, historical data, and air quality. This mock returns predefined data structures shaped exactly like the real API responses.

These mocks enable the test to verify that the function correctly processes data without making actual network requests. Assertions cover key data properties for current weather, 5-day forecast data, and air quality index. This validates the functions' data parsing and transformation logic. We also include tests that check for proper error handling, such as throwing an error when neither city nor latitude/longitude is provided.

2. Integration testing of Geonames API and OpenWeatherMap API.

Unit tests validate our service functions in isolation, and our integration tests confirm that the system behaves as expected when interacting with the live APIs. These tests are located in the tests/integration folder and focus on verifying the correctness of API communication, data handling, and transformation logic when working with real network calls.

OpenWeatherMap Integration Testing

This test calls fetchOpenWeatherData with real parameters (such as Vancouver) and a valid API key. This does not mock the API but fetches real data from OpenWeatherMap endpoints for geocoding, current weather, 5-day forecast, historical data, and air quality. The assertions check that:

- Coordinates are valid floating-point numbers.
- Temperature and Humidity are numbers from fetch.

- Forecast_5day is a non-empty array where each entry contains valid data, temperature range, weather description, humidity, rain chance, and wind.
- AQHI is a number, and air quality is a string.

This ensures that if the API is returning empty data points or null data, we will know it is an API issue, over a logic issue.

GeoNames API Integration Testing

This test verifies the GeoNames service integration that retrieves city information based on user search queries. It calls the fetchGeoNamesData function with a query string and checks that:

- A non-empty array of city results is returned
- Each result contains expected properties like name, countryName, lat, and lng

This validates that the service handles real data correctly and that any changes in the GeoNames response structure will be detected.

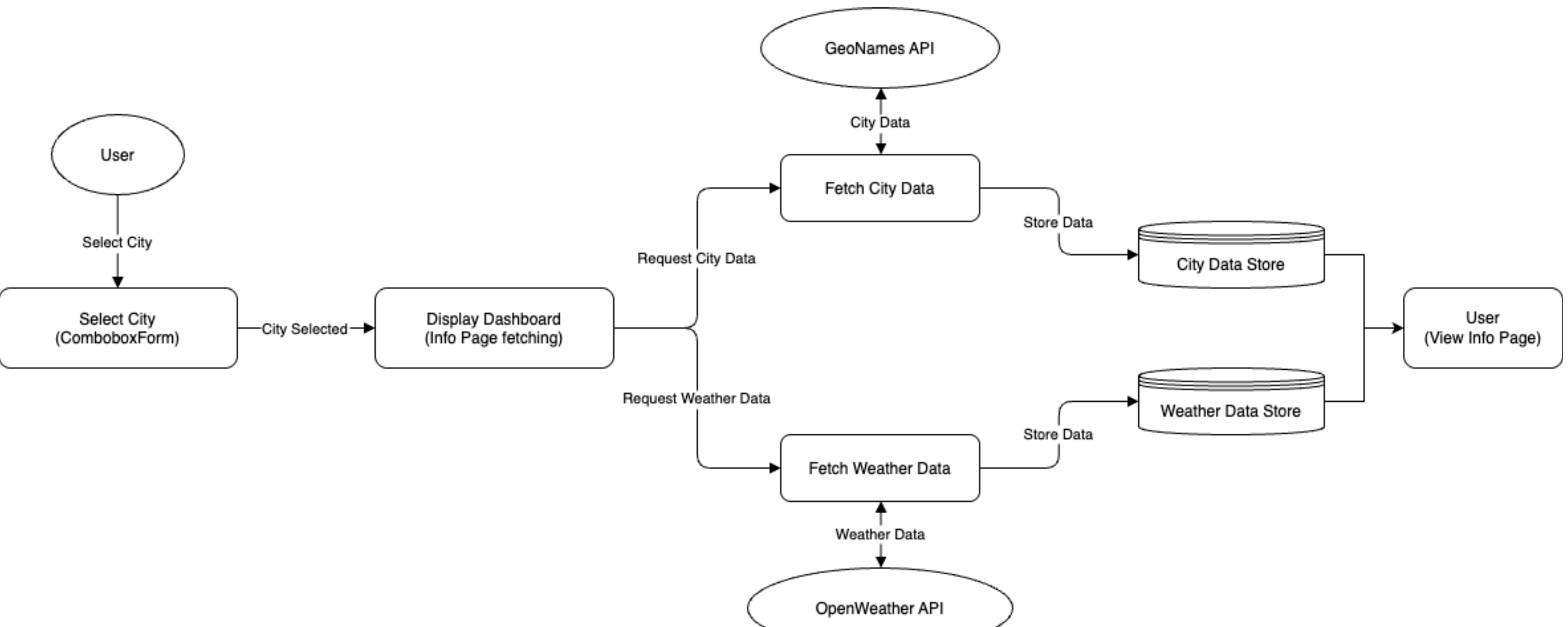
It is important that we have these integration tests because while the unit tests confirm internal logic, integration tests ensure API endpoints are reachable, authentication goes through, and response parsing is correct.

3. CI Pipeline

To automate test execution and maintain code quality, we configured a GitHub Actions workflow that triggers on every push and pull request to the main branch. The workflow checks out the repository, installs dependencies, and runs the full test suite via the npm script ‘npm run test’, which executes jest tests. This setup ensures that no code is merged into the branch without passing all unit and integration tests, enforcing a quality gate and preventing deployed issues.

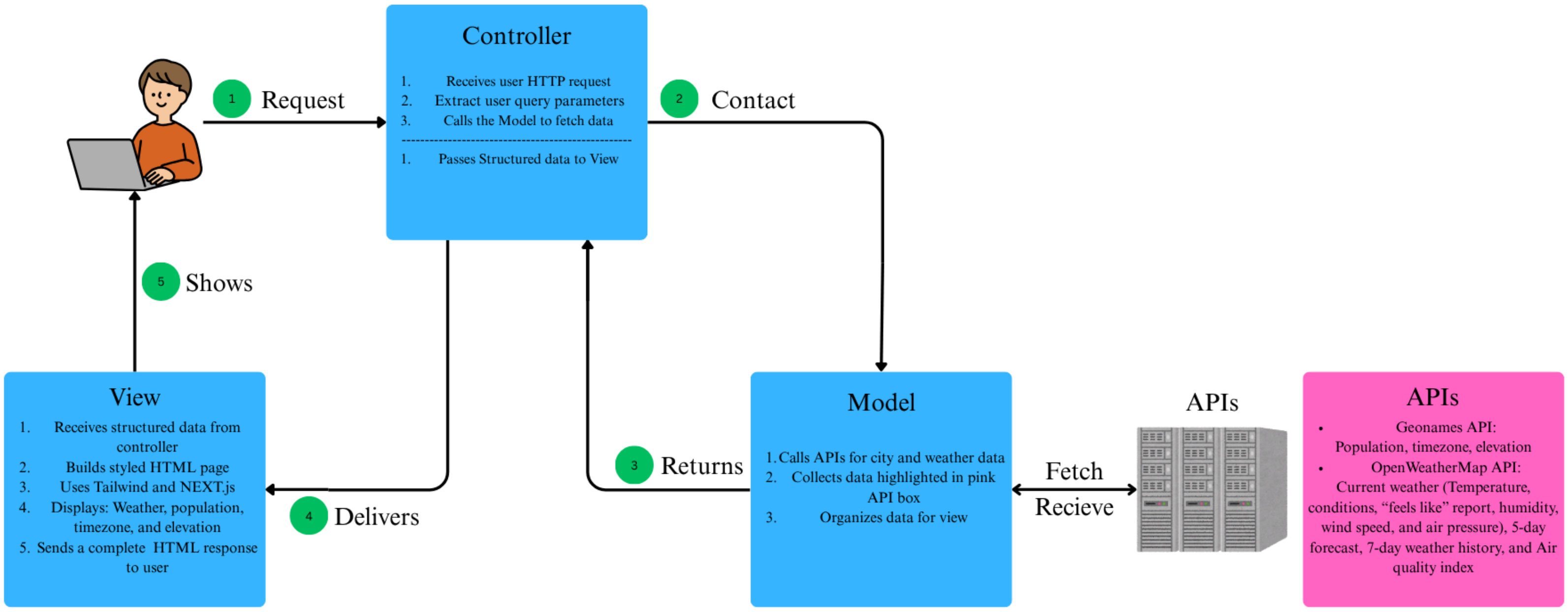
Detailed diagram of the project’s architecture

Data Flow Diagram:



Users first submit a city through our auto-complete search bar, which is routed to the info page, where the application fetching begins. From there, it fetches data from both API's which are separately stored in local state variables. Then these variables are redisplayed in a user friendly UI on the info dashboard.

MVC Model:



Controller
1. Receives user HTTP request
2. Extract user query parameters
3. Calls the Model to fetch data
1. 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, and elevation
5. Sends a complete HTML response to user
4 Delivers
3 Returns
Model
1. Calls APIs for city and weather data
2. Collects data highlighted in pink API box
3. Organizes data for view
Fetch
Recieve
APIs
APIs
• Geonames API: Population, timezone, elevation
• OpenWeatherMap API: Current weather (Temperature, conditions, “feels like” report, humidity, wind speed, and air pressure), 5-day forecast, 7-day weather history, and Air quality index

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.

List of known bugs and issues with the project and their severity (table format)

Bug	Description	Steps to Reproduce	Github Issue Link
"Select City" dropdown selects top option on Enter key	When the user clicks into the "Select City" dropdown and presses the Enter key without manually choosing a city, the app automatically selects the top city on the list. This is not the intended behavior and can lead to incorrect city data being shown.	1. Click into the "Select City" dropdown. 2. Without typing or using the arrow keys, press the Enter key. 3. Observe that the top city is selected by default.	https://github.com/CMPT-276-SUMMER-2025/final-project-1-sky/issues/81

Description of the project's future work and potential improvements

For future work, we plan to expand Travelytics beyond its current 50 Canadian cities to include global support. This allows travelers to quickly get information for cities around the world. We also plan to add a compare feature, so users can look at two or more cities side-by-side. This would make it easier to compare weather, population, and other factors when deciding where to travel. Another improvement is a favourites feature that saves cities even after a page refresh. This gives users quick access to the cities they search for most often. We also want to enhance the city information we provide. This could include details like natural disaster seasons, local events, and seasonal highlights to help people discover reasons to visit our beautiful country. In addition, we're considering adding a map feature so users can visualize the city's location while researching weather, and activity recommendations based on the weather and location, making trip planning even more useful and personalized. Overall, these features aim to make Travelytics more powerful, helpful, and engaging for travelers planning their next adventure.

Lessons learned and project takeaways

One big challenge we faced was using the GeoDB Cities API for our city autocomplete feature. At first glance, it seemed like a good choice, but we soon found out it only gave us 10 cities per request into the JSON format that it returned. That made it hard to give users enough options when they typed in a city name. We tried utilizing pages and tried to get 20 to 50 cities, but it didn't work as expected since we were encountering the "Too Many Requests" API call error. After testing things out, we decided to switch to the GeoNames API, which gave us more results and worked better for what we needed since we were able to implement an array of cities we desired, and the GeoNames API would only be called upon search, not by initial loading of the website. This showed us how important it is to leave time in the project for changes when things don't go as planned.

We also learned a lot about working together as a team. Before this project, not everyone on our team was comfortable using Git and GitHub. By the end, we had all practiced branching, merging, resolving conflicts, and writing clear commit messages, which are useful skills for future group work and internships. We split up tasks based on what each person was good at, checked in with each other regularly, and used GitHub to keep everything organized. Pull requests helped us review each other's code before adding it in, and GitHub Issues made it easy to keep track of what still needed to be done. This helped us stay on schedule and avoid confusion. In the end, we learned more than just how to use APIs and GitHub. Additionally, during our app testing, we recognized the importance of considering the user's perspective. We had to adjust things like the layout, loading speed, and dropdown options to make the app easier and more enjoyable to use. Overall, from this project, we learned how to work together, solve problems as a group, and adapt as things shift quickly when issues arise or when we hit roadblocks.

Appendix

Detailed Contributions:

M2 Tasks distributed:
Analysis of the project's success: Alex
Analysis of SDLC: Daniel
Description of features per API: Ayden
Overview of CI/CD: Ayden & Carter
Testing Strategy: Carter
Detailed diagram of the project's architecture: Daniel & Carter
Known bugs: Alex
Projects future work: Alex
Lessons Learned and Project Takeaways: Alex

Appendix: All teammates

M1 Tasks distributed:
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

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, 4:21 AM	Daniel Added next.js routing structure for multiple page layout.
June 27, 10:04 AM	Alex merges the following PR that Daniel completed, which was creating Next.js page routing. https://github.com/CMPT-276-SUMMER-2025/final-project-1-sky/pull/12
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
July 10, 10:11 AM	Ayden merges the following PR that Daniel completed, which is a hyperlinked Github icon footer. https://github.com/CMPT-276-SUMMER-2025/final-project-1-sky/pull/30
July 10, 10:21 AM	Alex merges the following PR that Daniel completed, which is the autocomplete search bar. https://github.com/CMPT-276-SUMMER-2025/final-project-1-sky/pull/31
July 16, 11 PM	Alex merged the following PR that Ayden completed, which is the removal of content from the homepage: https://github.com/CMPT-276-SUMMER-2025/final-project-1-sky/pull/32#event-18664848110
July 17, 12:11 PM	Daniel Merges the following PR that Alex completes, which is the GeoDB cities API setup: https://github.com/CMPT-276-SUMMER-2025/final-project-1-sky/pull/39
July 17, 12:11PM	Alex merges the following PR that Daniel Completed, which is the expandable FAQ card. https://github.com/CMPT-276-SUMMER-2025/final-project-1-sky/pull/36
July 17, 11:27 PM	Ayden merges the following PR that Alex completes, which is the autocomplete feature: https://github.com/CMPT-276-SUMMER-2025/final-project-1-sky/pull/40
July 18, 12:08	Ayden Merges the following PR that Alex

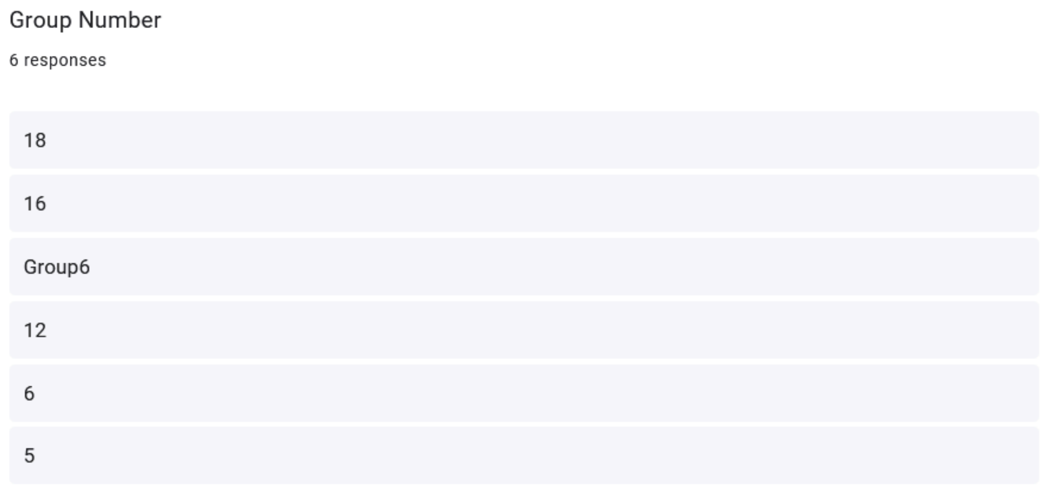
	makes the search component to be type safe: https://github.com/CMPT-276-SUMMER-2025/final-project-1-sky/pull/41
July 20, 9:30 PM	Alex merges the follow Pr that Ayden makes OpenWeatherMap api setup anf features: https://github.com/CMPT-276-SUMMER-2025/final-project-1-sky/pull/43
July 22, 10 PM	Alex suggests switching from GeoDB cities API to Geonames API due to issues
July 23, 8 PM	Ayden merges API 1 features that Alex implemented in the following PR: https://github.com/CMPT-276-SUMMER-2025/final-project-1-sky/pull/46
July 23, 11:50 AM	Carter adds a navbar to the landing page, Home, About and Docs pages have been added to navbar. Loose layout from M1 prototype added
July 24, 11 am	Ayden completes the check-in report API 1 feature, API 2 feature, Application interface, and CI/CD.
July 24, 6:21 PM	Alex merges the following PR that Daniel completes, which is styling components onto the landing page. https://github.com/CMPT-276-SUMMER-2025/final-project-1-sky/pull/51
July 24, 7:42 PM	Alex merges the following PR that Daniel Completes, which is making the application build ready for deployment. https://github.com/CMPT-276-SUMMER-2025/final-project-1-sky/pull/53
July 24, 3:35 PM	Carter adds testing with try and catch blocks for the GeoNames API
July 27, 12:30 PM	Alex merges the following PR that Ayden did, which was then passed down to Daniel to finish: https://github.com/CMPT-276-SUMMER-2025/final-project-1-sky/pull/48
July 27, 12:30 PM	Alex merges the following Pr that Ayden did, which was about completing the features for the second API: https://github.com/CMPT-276-SUMMER-2025

	/final-project-1-sky/pull/58
July 27, 3:30 PM	Alex merges the following PR that Ayden did, which was fixing a component in the homepage search bar: https://github.com/CMPT-276-SUMMER-2025/final-project-1-sky/pull/60
July 27, 5:13PM	Alex merges the following PR that Daniel did, which was updating the UI to include logo, favicon, gradient background and better text card usage. https://github.com/CMPT-276-SUMMER-2025/final-project-1-sky/pull/62
July 27, 10:20 PM	Daniel merges the following PR that Ayden completed, which was adding an animated background to the homepage: https://github.com/CMPT-276-SUMMER-2025/final-project-1-sky/pull/64
July 28, 11:30 PM	Carter merges the following PR that Ayden completed, which was updating the weather features to be more accurate: https://github.com/CMPT-276-SUMMER-2025/final-project-1-sky/pull/68
July 28, 3:11 PM	Ayden merges the following PR that Daniel completed, which was making the animated background global, rather than just homepage. https://github.com/CMPT-276-SUMMER-2025/final-project-1-sky/pull/66
July 28, 11:30 PM	Carter merges the following PR that Ayden completed, adding more detailed weather info to the forecast. https://github.com/CMPT-276-SUMMER-2025/final-project-1-sky/commit/95686711b3398b87f6622fe16e1824b5ba33b188
July 29, 1:27 PM July 29, 11:21 PM	Carter added integration testing for the OpenWeather API. Carter also added integration tests to CI
July 30, 10:50 AM	Alex merges the following PR that Daniel Completed, which was fixing Github icon not working. https://github.com/CMPT-276-SUMMER-2025/final-project-1-sky/pull/70

July 30, 12:59 PM	Carter merges the following PR that Daniel Completed, which was making text colouring more consistent on dashboard. https://github.com/CMPT-276-SUMMER-2025/final-project-1-sky/pull/72
Aug 1, 10:39 AM	Carter merges the following PR completed by Daniel, which updates the FAQ placeholder text with real questions. https://github.com/CMPT-276-SUMMER-2025/final-project-1-sky/pull/77
Aug 1	Ayden merges the following PR requested by Alex to resolve missing cities: https://github.com/CMPT-276-SUMMER-2025/final-project-1-sky/pull/79
August 4, 3 PM	Carter breaks up app logic into a service and route.ts files so that proper unit and integration testing can be achieved.
August 5, 7 PM	Carter adds Integration and Unit testing for API 1 and 2 with the new Jest architecture.
August 6, 2:28 PM	Carter updates MVC to reflect API changes
August 6, 5:47 PM	Carter makes integration and unit tests work on PRs on main. https://github.com/CMPT-276-SUMMER-2025/final-project-1-sky/commit/28e7af78121a3c25acd88b81cbd7344c73d9e6ea
August 6, 7:02 PM	Carter merges the updated route and testing code with proper comments. https://github.com/CMPT-276-SUMMER-2025/final-project-1-sky/commit/53bbefd597b540bc237a0b6e60ec98be6006c15d

Peer testing form:
https://docs.google.com/forms/d/e/1FAIpQLSc8WNRHKMy9gO6i8D8x-n4EBIrAB4V8NAYa9P7IU1uJ90--Cw/viewform?usp=header

Peer Testing Feedback received (Screenshots):



Group Number

6 responses

18

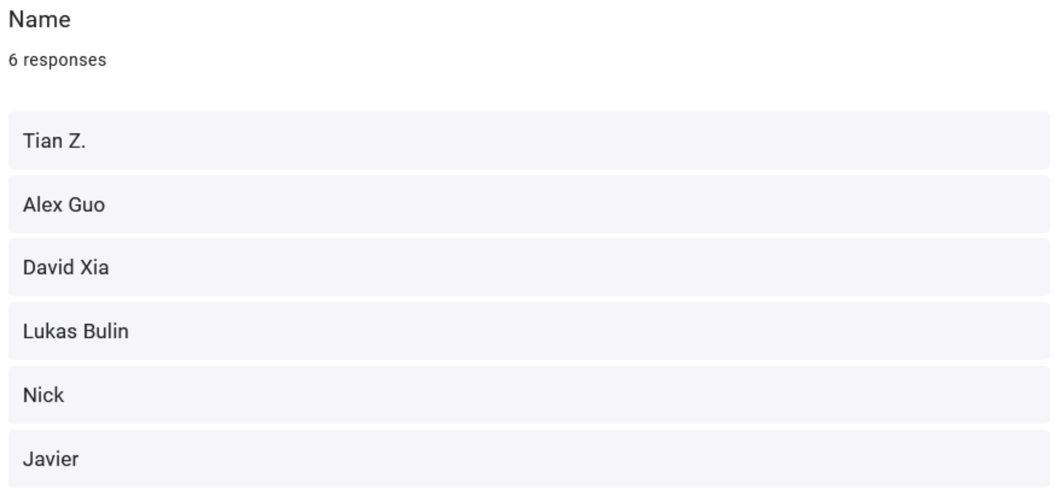
16

Group6

12

6

5



Name

6 responses

Tian Z.

Alex Guo

David Xia

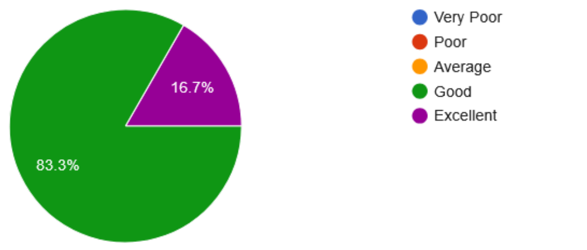
Lukas Bulin

Nick

Javier

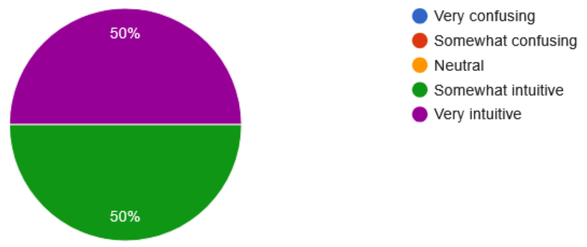
How intuitive was the app to navigate?

6 responses



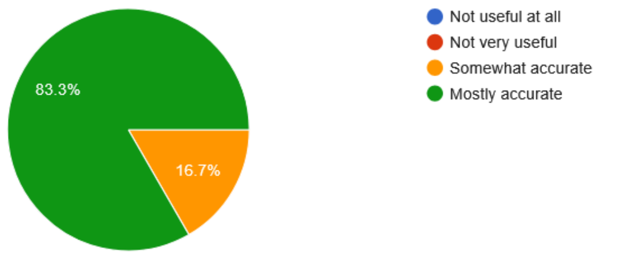
How useful did you find the city discovery functionality?

6 responses



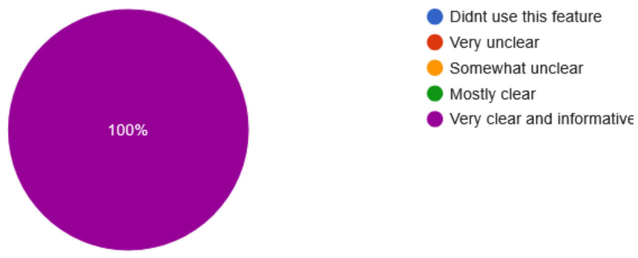
How accurate and helpful was the real-time weather data?

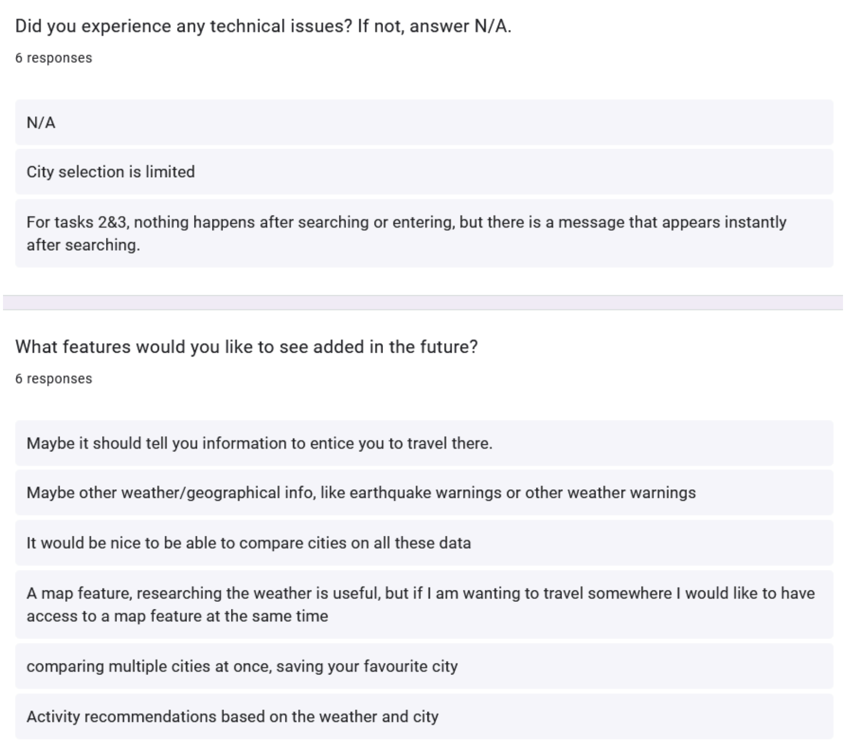
6 responses



How clear and informative was the city information?

6 responses





Did you experience any technical issues? If not, answer N/A.

6 responses

N/A

City selection is limited

For tasks 2&3, nothing happens after searching or entering, but there is a message that appears instantly after searching.

What features would you like to see added in the future?

6 responses

Maybe it should tell you information to entice you to travel there.

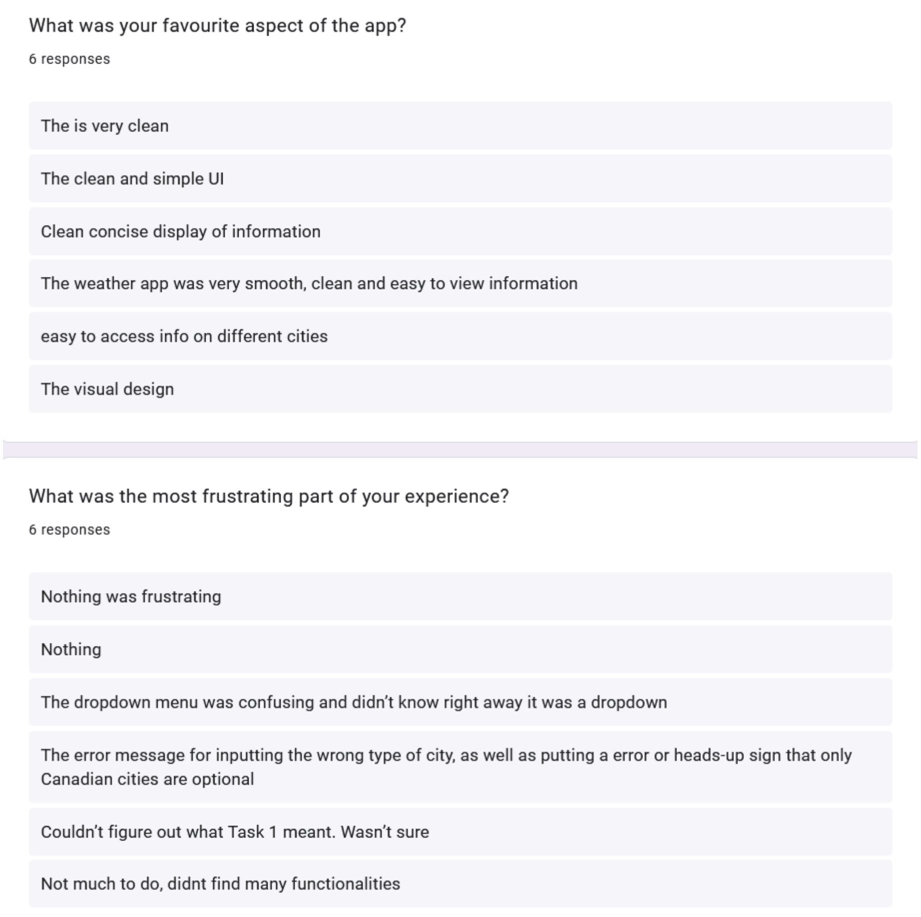
Maybe other weather/geographical info, like earthquake warnings or other weather warnings

It would be nice to be able to compare cities on all these data

A map feature, researching the weather is useful, but if I am wanting to travel somewhere I would like to have access to a map feature at the same time

comparing multiple cities at once, saving your favourite city

Activity recommendations based on the weather and city



What was your favourite aspect of the app?

6 responses

The is very clean

The clean and simple UI

Clean concise display of information

The weather app was very smooth, clean and easy to view information

easy to access info on different cities

The visual design

What was the most frustrating part of your experience?

6 responses

Nothing was frustrating

Nothing

The dropdown menu was confusing and didn’t know right away it was a dropdown

The error message for inputting the wrong type of city, as well as putting a error or heads-up sign that only Canadian cities are optional

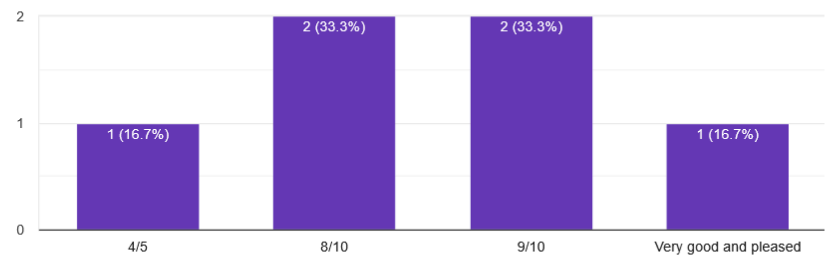
Couldn’t figure out what Task 1 meant. Wasn’t sure

Not much to do, didnt find many functionalities

How would you rate your overall experience with Travelytics?

6 responses

Copy chart



Please provide any additional thoughts that you would like to share in this text box below.

3 responses

Nothing

N/A

I hope the dropdown button had one direction