



THE ROLLING CAPSTONES

AI TRIP CREATOR



TABLE OF CONTENTS

Introduction.....	5
Class Diagram.....	5
User Stories / User Characteristics.....	7
User Stories.....	7
Flight Aggregation.....	7
Login.....	7
User preferences.....	8
Generate Itinerary.....	8
User Preferences.....	9
Functional requirements.....	11
Use cases.....	11
1. User registration and login.....	11
2. Search for flights.....	12
3. Generate itinerary.....	13
4. Update user preferences.....	14
Requirements.....	15
1. User Management.....	15
2. Travel Planning.....	15
3. Itinerary Management.....	16
4. Notifications and Alerts.....	17
Subsystems.....	18
1. User Interface Subsystem.....	18
2. Flight Data Aggregation Subsystem.....	18
3. Accommodation Data Scraping Subsystem.....	18
4. Itinerary Generation Subsystem.....	19
5. Data Management Subsystem.....	19
6. Backend Infrastructure Subsystem.....	19
Architectural Requirements.....	20

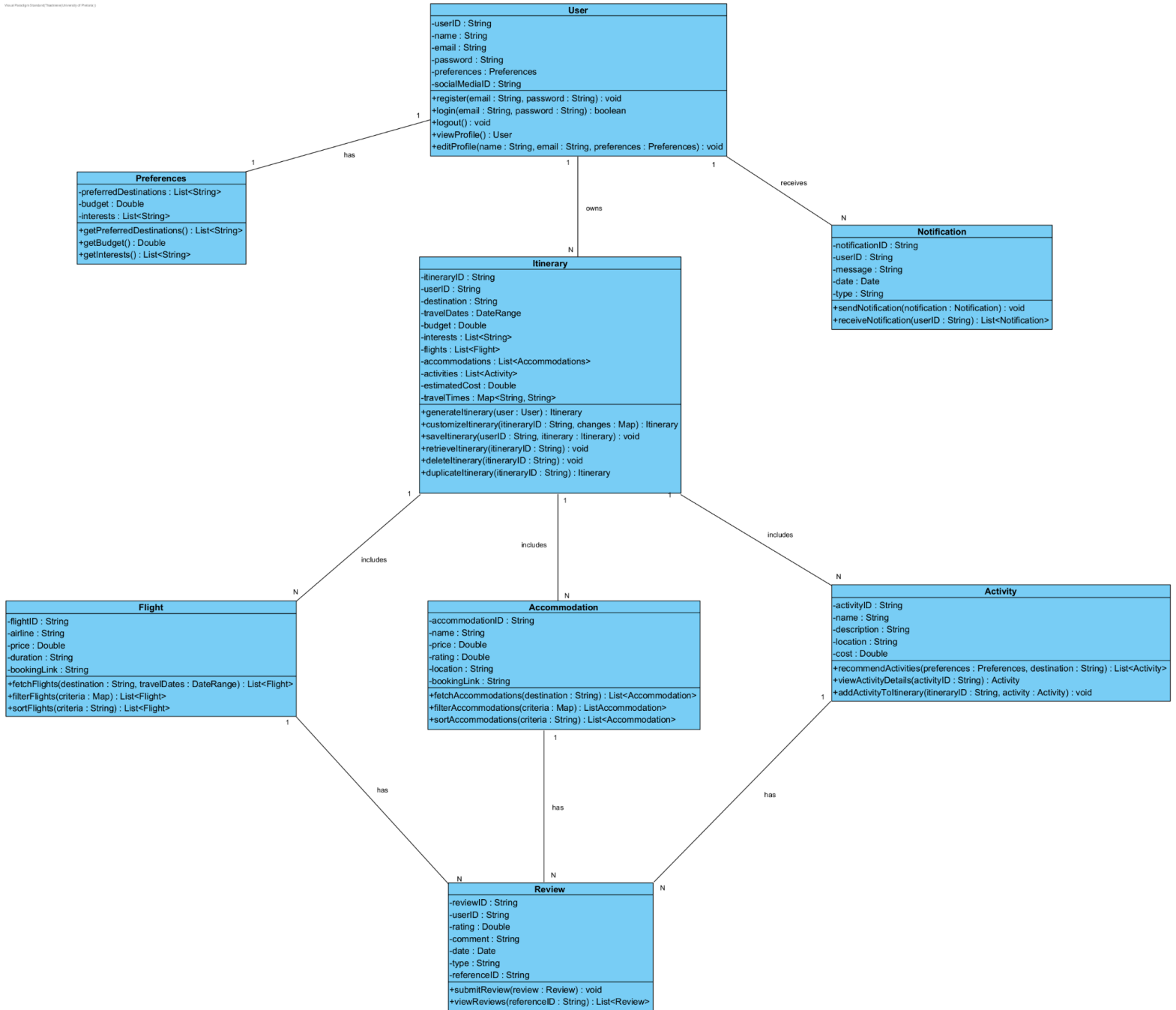
Quality Requirements.....	20
Performance.....	20
Scalability.....	20
Security.....	20
Usability.....	21
Reliability.....	21
Maintainability.....	21
Compatibility.....	21
Architectural Patterns.....	23
1. Microservices Architecture.....	23
1.1 Description:.....	23
1.2 Benefits:.....	23
2. Serverless Architecture.....	24
2.1 Description.....	24
2.2 Benefits:.....	24
2.3 Components:.....	24
3. Model-View-Controller (MVC) Pattern.....	25
3.1 Description:.....	25
3.2 Benefits:.....	25
3.3 Components:.....	25
Design Patterns.....	26
1. Singleton Pattern.....	26
2. Factory Pattern.....	27
3. Strategy Pattern.....	28
4. Observer Pattern.....	29
5. Composite Pattern.....	30
Constraints.....	31
API Limitations.....	31
Data Quality and Coverage.....	31
Integration and Compatibility.....	32
Legal and Licensing Constraints.....	32
Functionality.....	32
Support and Reliability.....	33

Strategies to Mitigate Constraints.....	33
Combination of Services.....	33
Fallback Mechanisms.....	33
Incremental Development.....	33
Service Contracts.....	34

Introduction

Our vision for the AI Trip Creator is to develop a user-friendly web application that revolutionizes how individuals plan their trips. By integrating an intelligent recommendation engine and offering a seamless user experience, the AI Trip Creator empowers users to efficiently and effectively create their dream itineraries.

Class Diagram



User Stories / User Characteristics

User Stories

Flight Aggregation

- As a user I would like to be able to view possible flights from a source destination to an arrival destination with various details like departure and arrival time.
- As a user I would like to be able to filter possible flights to be able to see flights that fit my needs for example price, time.

Login

- As a user I would like to be able to register an account to the web application so that I may use the program.
 - As a user I would I want to login into my account so that I may use the web application
 - As a user I want to be able to recover my password in case I have forgotten my account details
 - As a user I want to be able to save my holiday preferences so that in the future the web application is custom to me
-

User preferences

- As a user I want to be able to update my destination preferences so that data is fitted to my new destination
- As a user I want to be able to update my budget preferences so that data is fitted to my new budget
- As a user I want to be able to update the style in which I want to travel, so that data is fitted to it.
- As a user I would like to be able to save the changes made in my preferences.

Generate Itinerary

- As a user I would like to input my preferences so that I have a fitted itinerary.
 - As a user I would like to have an itinerary generated to my preferences
 - As a user I would like to view the itinerary generated for me
 - As a user I would like to be able to save the itinerary generated for me.
-

User Preferences

Preferred Destinations: As a user, I should be able to specify a list of destinations that I prefer or am interested in visiting.

Budget: As a user, I should be able to specify the maximum amount I am willing to spend on my trip.

Interests: As a user, I should be able to list interests or activities that I enjoy and would like to include in my travel itinerary.

Travel Dates: As a user, I should be able to specify my preferred dates or date ranges for travelling.

Accommodation Type: As a user, I should be able to choose my preferred type of accommodation.

Preferred Airlines: As a user, I should be able to list my preferred airlines for flights.

Travel Companions: As a user, I should be able to provide information about the number of travellers or specific travel companions.

Meal Preferences: As a user, I should be able to specify dietary restrictions or meal preferences.

Preferred Transportation: As a user, I should be able to choose my preferred mode of transportation for local travel within the destination.

Language Preferences: As a user, I should be able to specify my preferred language for communication and services.

Activity Intensity: As a user, I should be able to choose the preferred level of activity or physical intensity.

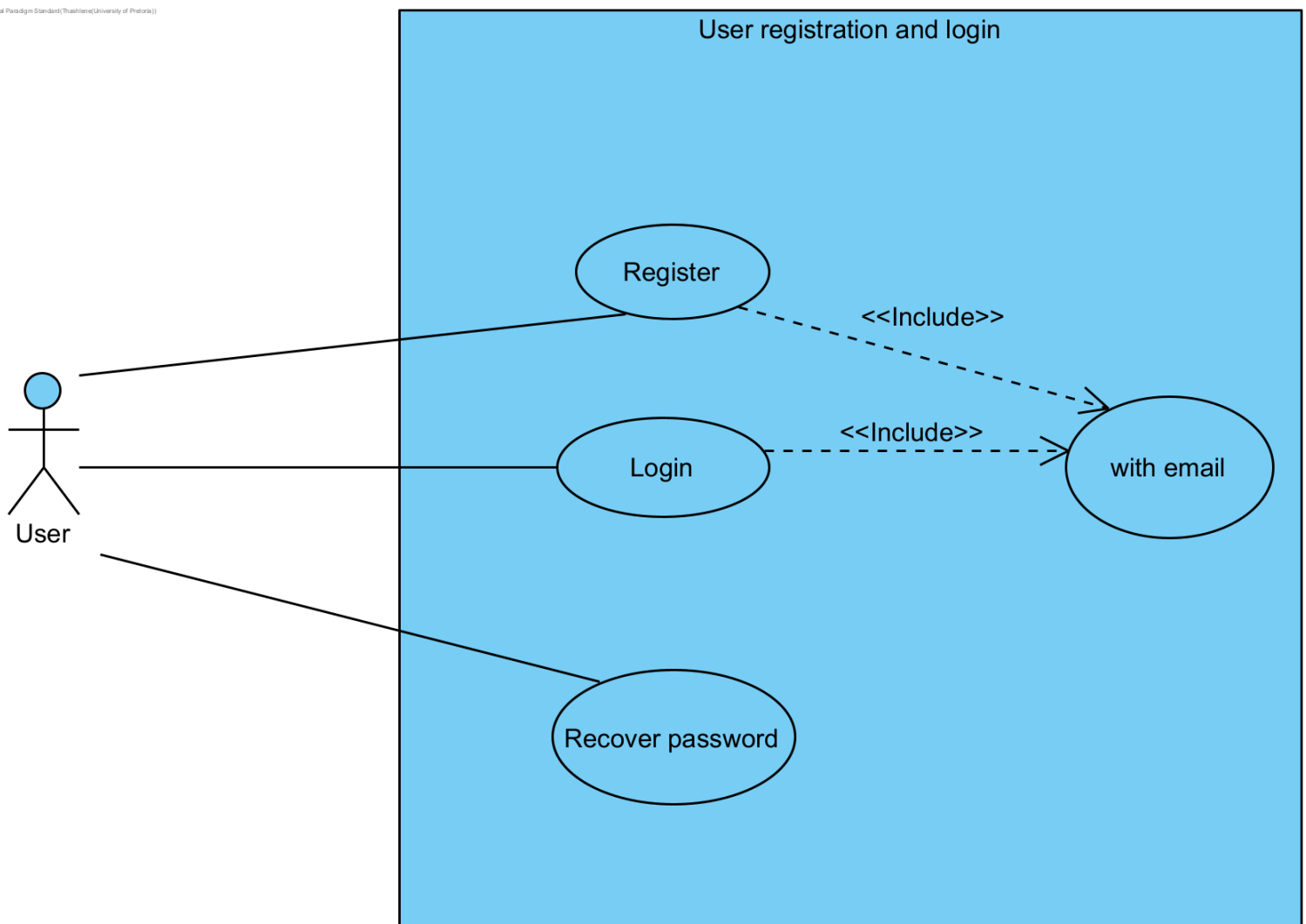
Cultural Interests: As a user, I should be able to specify specific cultural interests that I want to explore.

Functional requirements

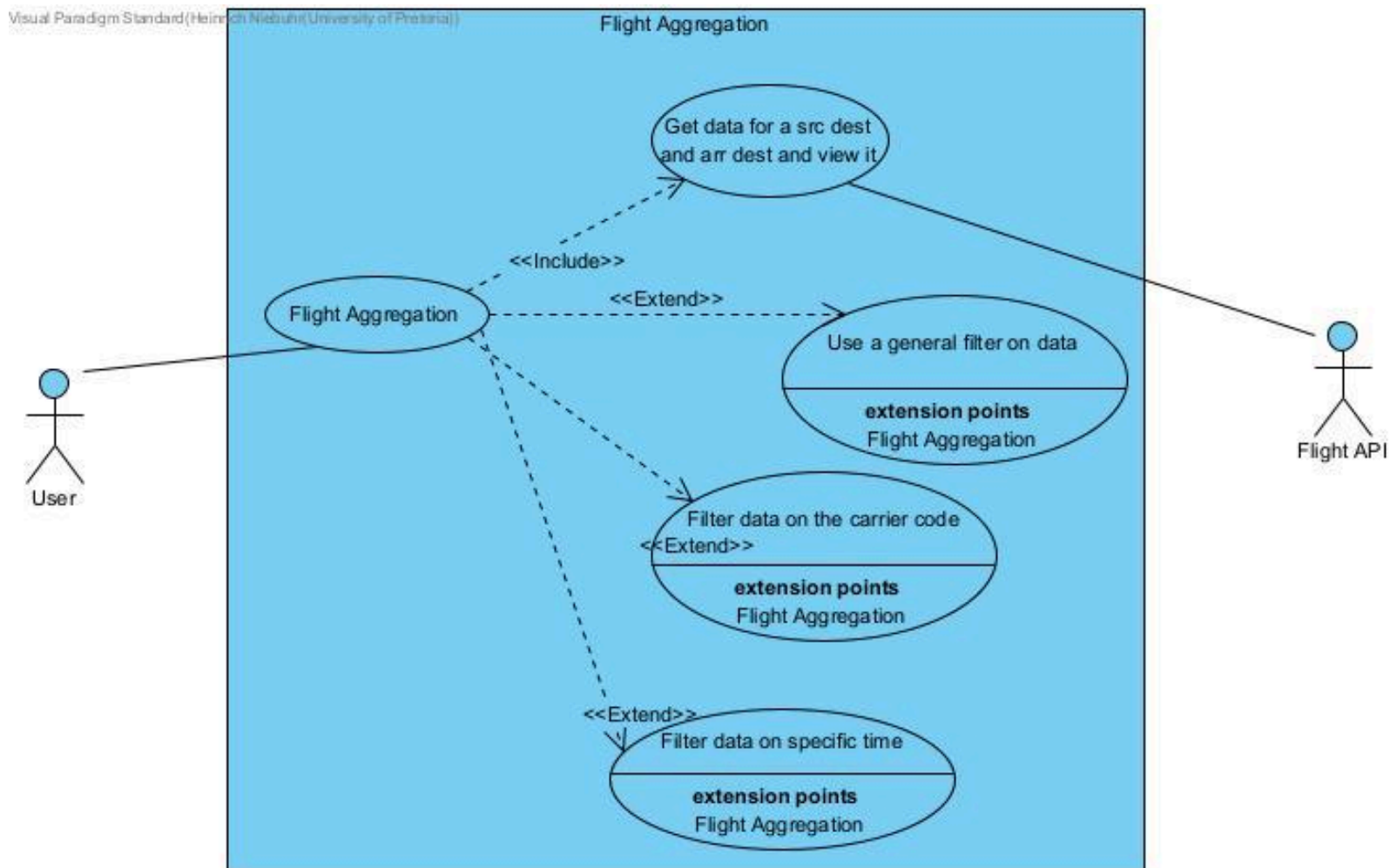
Use cases

1. User registration and login

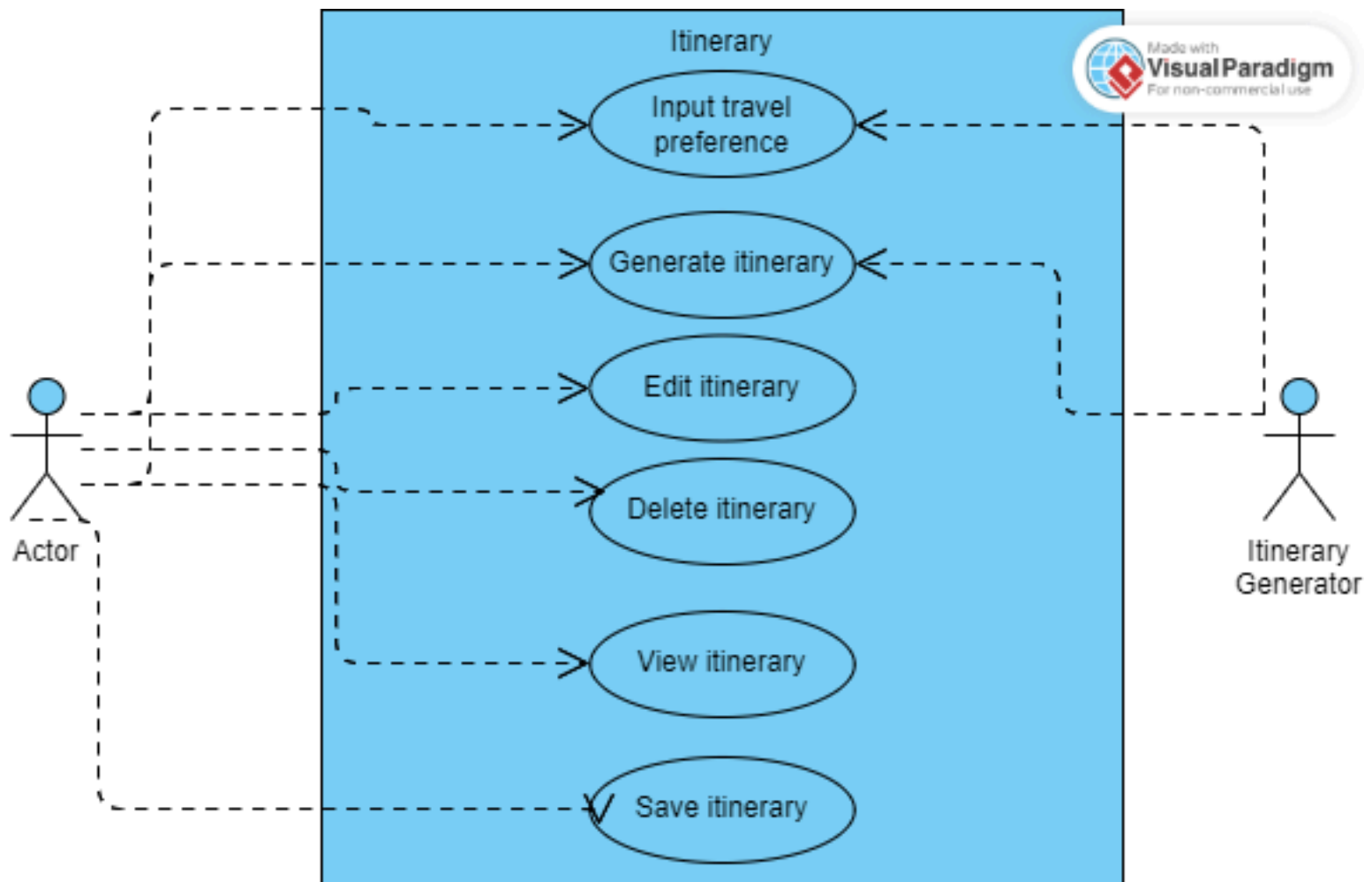
Visual Paradigm Standard (Thushilene University of Pretoria)



2. Search for flights

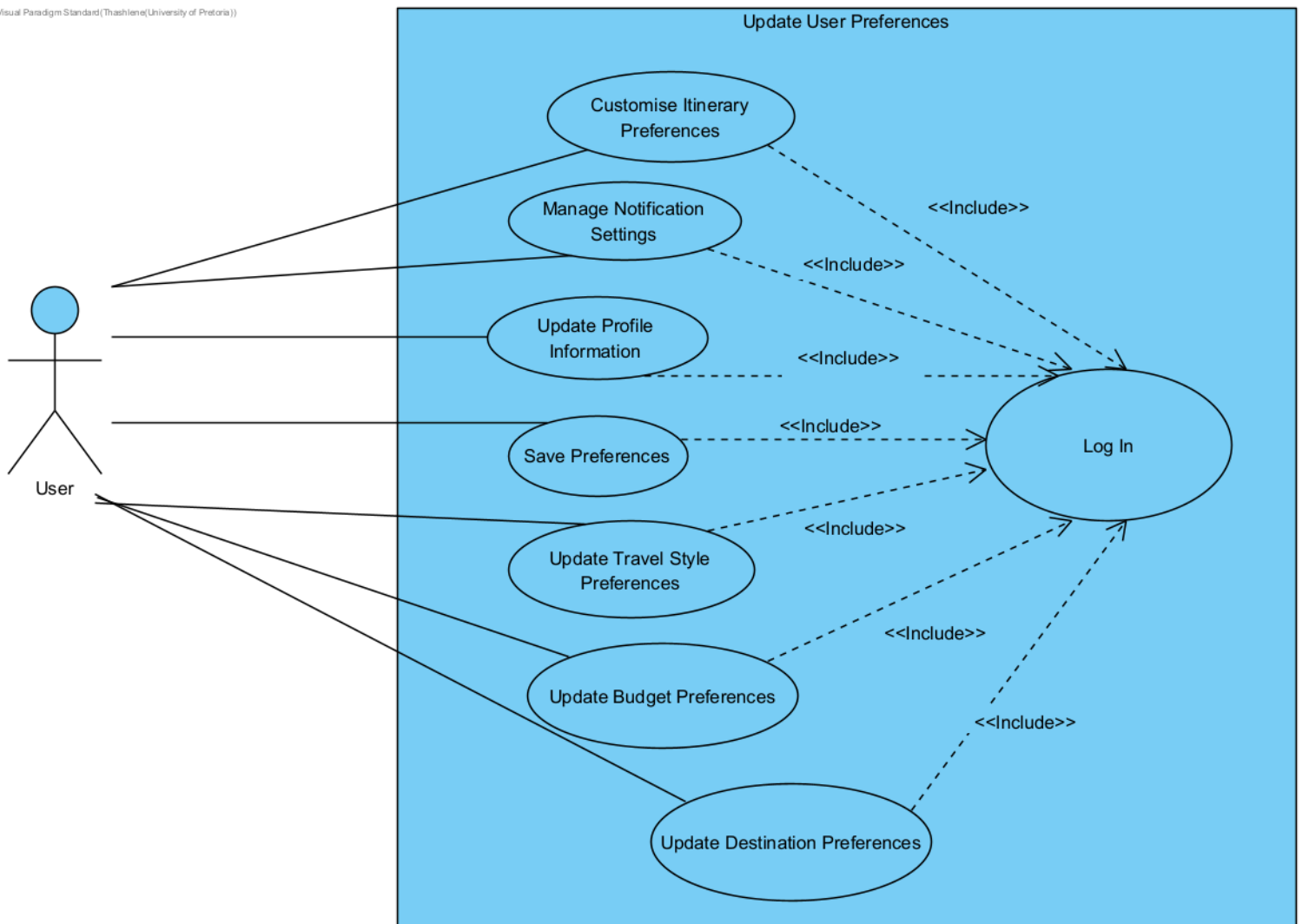


3. Generate itinerary



4. Update user preferences

Visual Paradigm Standard(Thashlene(University of Pretoria))



Requirements

1. User Management

1.1. User Registration and Authentication

- 1.1.1. Users must be able to create accounts using email, social media login (e.g., Google, Facebook), and password.
- 1.1.2. Users must be able to log in and log out securely.
- 1.1.3. Password reset functionality must be provided.

1.2. User Profile Management

- 1.2.1. Users must be able to view and edit their profile information (name, email, preferences).
- 1.2.2. Users must be able to manage their travel preferences (e.g., preferred destinations, budget, interests).

2. Travel Planning

2.1. Itinerary Creation

- 2.1.1. Users must be able to input travel details (destination, travel dates, budget, interests).
- 2.1.2. The system must generate personalised travel itineraries based on user input.
- 2.1.3. The system must allow users to customise and modify generated itineraries.

2.2. Flight Data Aggregation

- 2.2.1. The system must fetch and display flight options based on user input.
 - 2.2.2. Users must be able to filter and sort flight options by price, duration, and airline.
-

2.2.3. The system must provide booking links to selected flight options.

2.3. Accommodation Data Scraping

2.3.1. The system must fetch and display accommodation options at the chosen destination.

2.3.2. Users must be able to filter and sort accommodation options by price, rating, and location.

2.3.3. The system must provide booking links to selected accommodation options.

2.4. Activity Recommendations

2.4.1. The system must suggest activities and points of interest based on user preferences and destination.

2.4.2. Users must be able to view details and add activities to their itineraries.

2.5. Cost Optimization

2.5.1. The system must provide suggestions for reducing travel costs (e.g., alternative travel dates, budget accommodations).

2.5.2. The system must display estimated costs for the entire itinerary.

3. Itinerary Management

3.1. Itinerary Saving and Retrieval

3.1.1. Users must be able to save generated itineraries to their profile.

3.1.2. Users must be able to view, edit, and delete saved itineraries.

3.1.3. The system must allow users to duplicate saved itineraries for future trips.

3.2. Estimated Travel Times

3.2.1. The system must display estimated travel times between different points in the itinerary (e.g., from hotel to airport).

3.2.2. The system must provide optimized routes for the travel itinerary.

3.2.3. Social Sharing

3.2.4. Users must be able to share their itineraries via social media or email.

3.2.5. The system must generate shareable links for itineraries.

3.3. User Reviews and Ratings

3.3.1. Users must be able to view user reviews and ratings for flights, accommodations, and activities.

3.3.2. Users must be able to submit their reviews and ratings for experiences included in their itineraries.

4. Notifications and Alerts

4.1. Notifications

4.1.1. Users must receive notifications about itinerary updates, travel alerts, and booking confirmations.

4.1.2. The system must support push notifications, email alerts, and in-app notifications.

Subsystems

A project of this scale requires us to break it down into manageable segments, or subsystems. Segmenting the project also means that the subsystems have been identified, which is the case. What follows is a list of the subsystems that comprise the Web Application we have been tasked with developing.

1. User Interface Subsystem

This subsystem encompasses the basic functionality of the Web Application. Users should be able to interact with a user-friendly interface to provide information about their planned trip(s). This information includes (but is not limited to):

- Preferred Destination
- The length of the trip
- Modes of transportation

2. Flight Data Aggregation Subsystem

This subsystem will use Google Flight API to extract flight data, aggregate it, and based on the user's input, the appropriate data will be displayed.

3. Accommodation Data Scraping Subsystem

Web scraping for suitable accommodation based on the user's preferences and data aggregation to provide a list of accommodation options for the user.

4. Itinerary Generation Subsystem

Personalized itineraries can be generated using an AI-powered system while ChatGPT can provide an experience similar to communicating with a trip advisor.

5. Data Management Subsystem

Firestore, MongoDB, or PostgreSQL are options for storage of user preferences such as travel methods and itineraries.

6. Backend Infrastructure Subsystem

Service communication by means of Kafka or RabbitMQ. This will enable effective communication between the various parts of the web application.

Architectural Requirements

Quality Requirements

Performance

The system should perform well to maintain user engagement and interaction. The maximum response time for any user request should be as low as possible, ensuring a smooth and satisfactory user experience. Flight and accommodation data aggregation and scraping processes should be complete within a reasonably low time frame as well. The system must be capable of handling many concurrent users without performance degradation.

Scalability

The system should scale according to demand, ensuring it can accommodate a growing user base. This includes handling a significant increase in user base per year and an increase in data volume per year without performance issues.

Security

User data must be protected using industry-standard encryption algorithms. All data must be encrypted in transit using HTTPS and at rest. The system should implement OAuth 2.0 for secure authentication and role-based access control for authorisation. Vulnerability assessments should be conducted bi-annually, with critical vulnerabilities resolved within 30 days.

Usability

The system should be easy to navigate with an intuitive user interface. The application should comply with WCAG 2.1 AA standards to ensure accessibility. Additionally, user support should be available through chatbots and email, with a response time of 24 hours for user inquiries.

Reliability

The system should have an uptime of 99.9% to ensure high availability. The error rate for user transactions and requests should be less than 0.1%. Data should be backed up every 24 hours, with a recovery time objective (RTO) of 2 hours and a recovery point objective (RPO) of 1 hour.

Maintainability

The codebase must adhere to clean coding principles and be well-documented to facilitate future maintenance and updates by development teams. The code should maintain a maximum cyclomatic complexity of 10 for any function/method. Comprehensive documentation, including API documentation, user manuals, and developer guides, must be provided. The system should achieve at least 80% code coverage with automated unit and integration tests.

Compatibility

The system should work across a variety of current browsers and devices to provide a consistent user experience. This ensures users can access the system regardless of the browser or device being used.

Architectural Patterns

1. Microservices Architecture

1.1 Description:

Decompose the application into loosely coupled, independently deployable services. Each service corresponds to a specific business functionality.

1.2 Benefits:

Scalability: Individual services can be scaled independently.

Flexibility: Easier to update and deploy individual components without affecting the entire system.

Resilience: Failure in one service does not bring down the entire system.

Components: User Interface Service: Handles user inputs and displays data.

Flight Data Service: Integrates with flight APIs to fetch data.

Accommodation Service: Scrapes and aggregates accommodation data.

Itinerary Service: Generates and manages travel itineraries.

User Management Service: Manages user authentication and profiles. Social Sharing Service: Handles sharing functionalities

2. Serverless Architecture

2.1 Description

A serverless architecture allows building and running of applications and services without thinking about servers. The cloud provider handles the infrastructure management tasks like scaling, patching, and maintaining servers, which will enable focus solely on code.

2.2 Benefits:

Cost-Efficiency: Pay only for the computation time consumed.

Automatic Scaling: Automatically scales to handle varying loads.

Simplified Operations: Reduced need for server management and maintenance.

2.3 Components:

Frontend (SPA/PWA): A single-page application (SPA) or progressive web app (PWA) to provide a responsive user interface.

API Gateway: Routes user requests to appropriate serverless functions

- **Role:** The API Gateway acts as a reverse proxy to accept all API calls, route them to the appropriate serverless functions, and return the results. It handles tasks such as authorization, throttling, and monitoring.
 - **Functionality:** It ensures secure and efficient communication between the frontend and backend by managing API endpoints and processing requests.
-

Lambda Functions: small, single-purpose functions that execute backend logic in response to events like HTTP requests from the API Gateway. They are stateless and designed to run for a short duration.(e.g., fetching flight data, generating itineraries).

Database (DynamoDB, Firestore): Store user data, preferences, and itineraries

3. Model-View-Controller (MVC) Pattern

3.1 Description:

Separate the application into three interconnected components: Model, View, and Controller.

3.2 Benefits:

Clear separation of concerns: Each component handles specific aspects of the application, making it easier to manage and develop.

Reusability: Components can be reused across different parts of the application.

Maintainability: Easier to update and maintain due to the separation.

3.3 Components:

Model: Manages the data and business logic (e.g., user preferences, itinerary data).

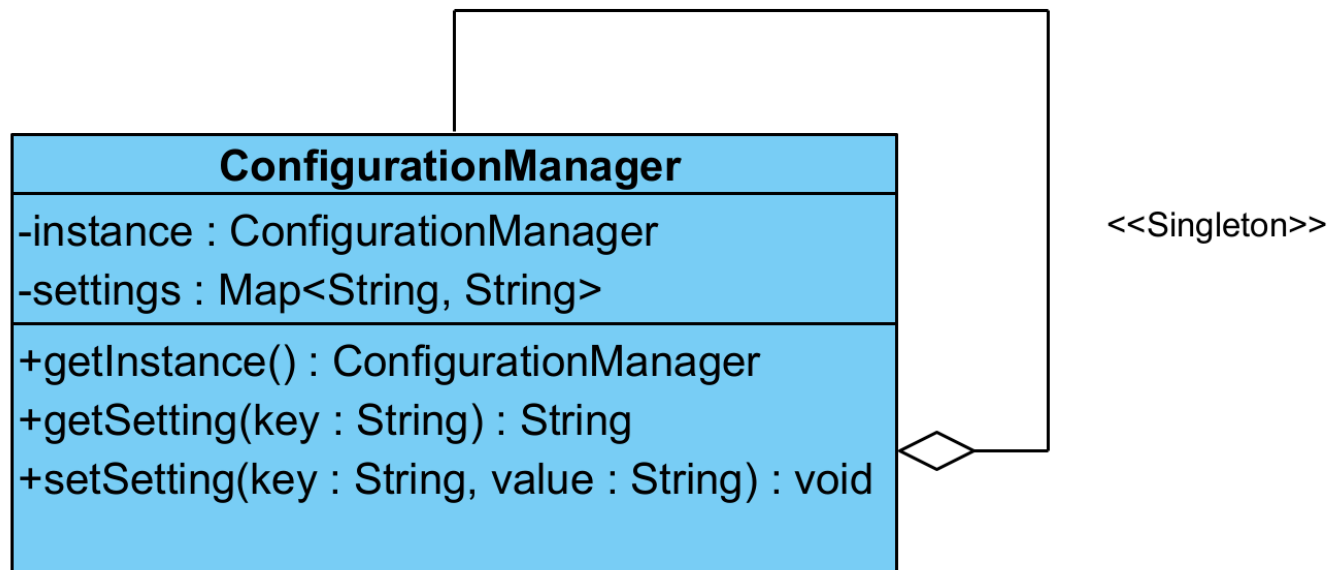
View: Displays data to the user and handles user interactions. **Controller:** Processes user input, interacts with the model, and updates the view.

Design Patterns

1. Singleton Pattern

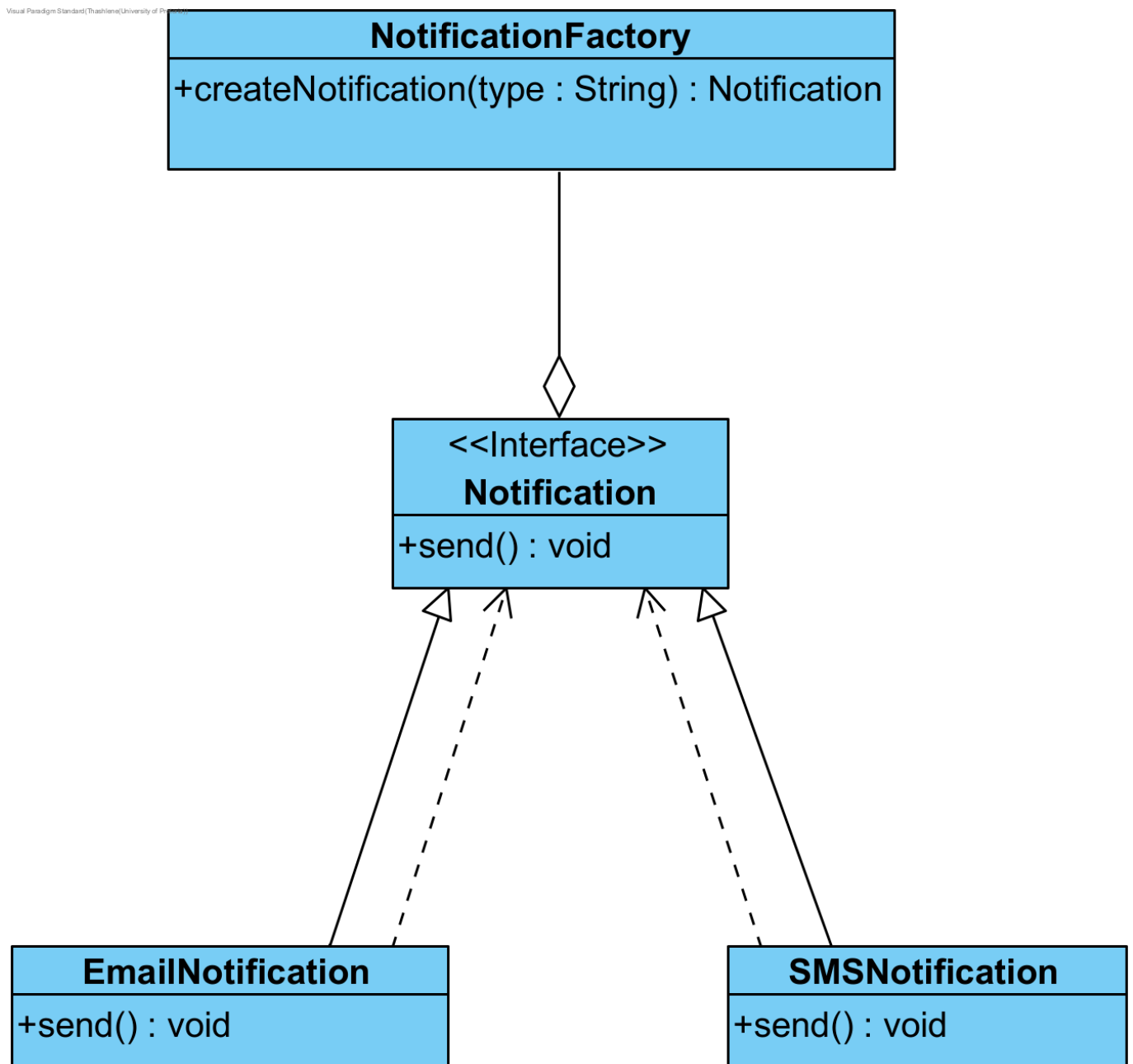
The *ConfigurationManager* class that handles configuration settings for the application.

Visual Paradigm Standard (Thashlene/University of Pretoria))



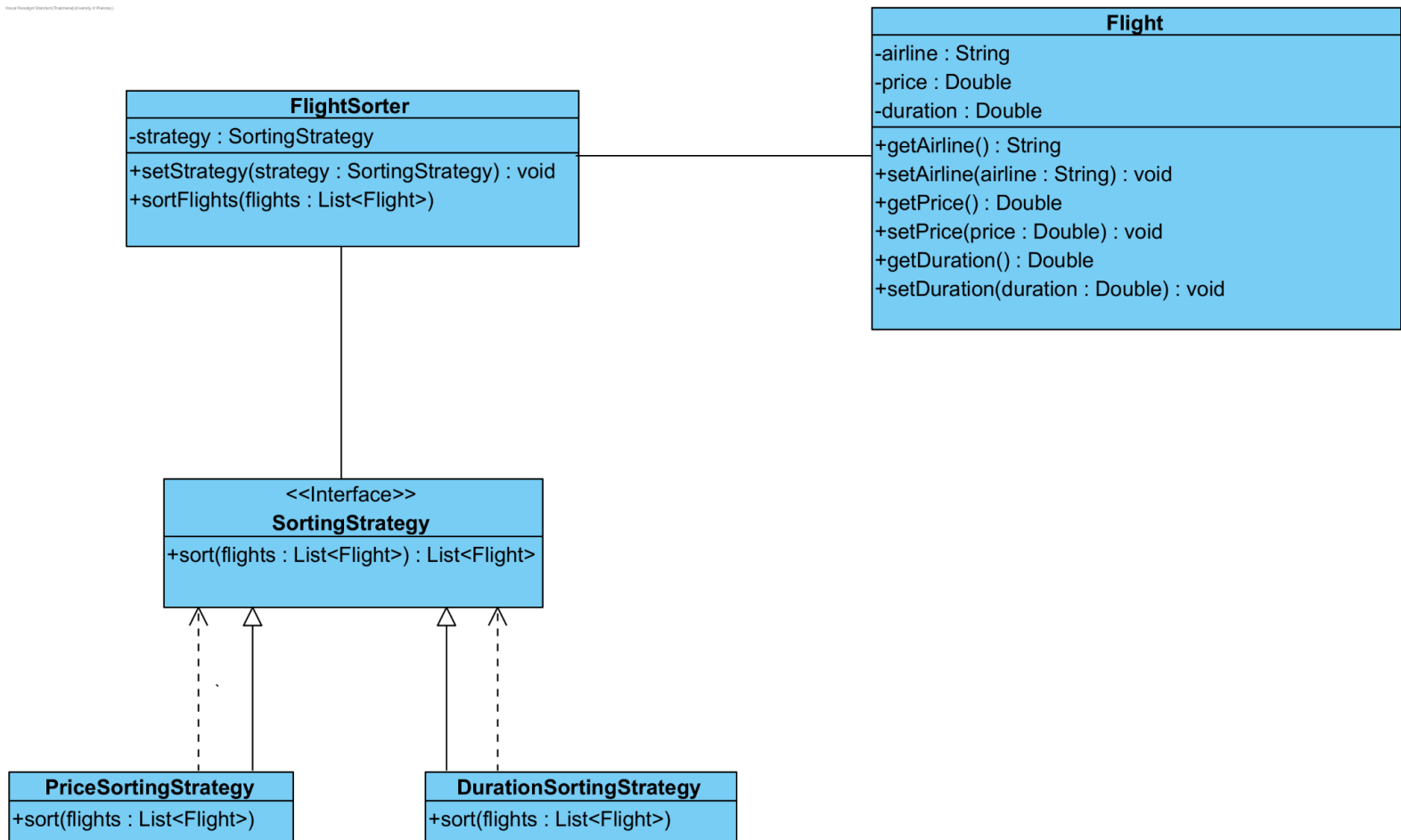
2. Factory Pattern

The *NotificationFactory* class to create different types of notifications.



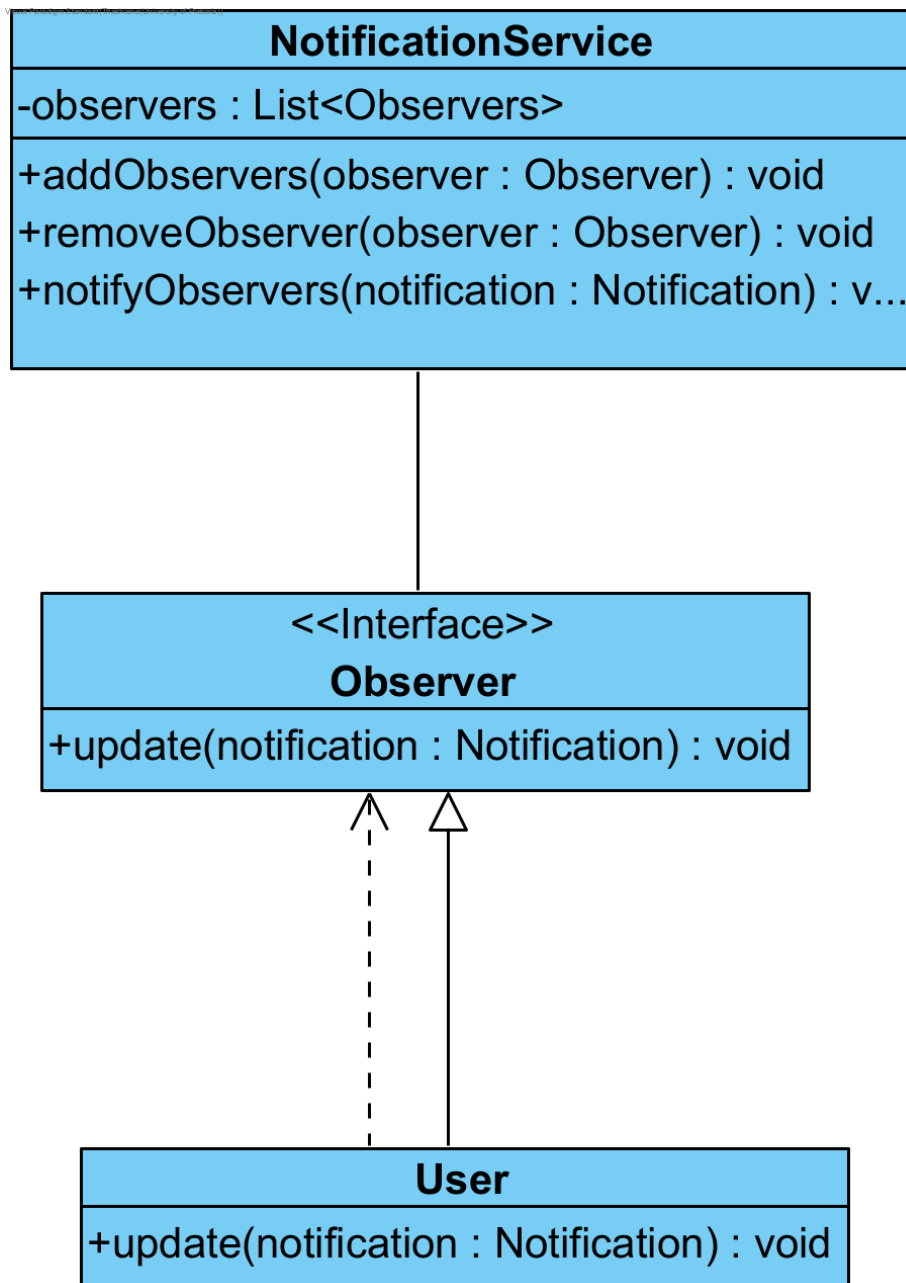
3. Strategy Pattern

The *SortingStrategy* for sorting flights and accommodations.



4. Observer Pattern

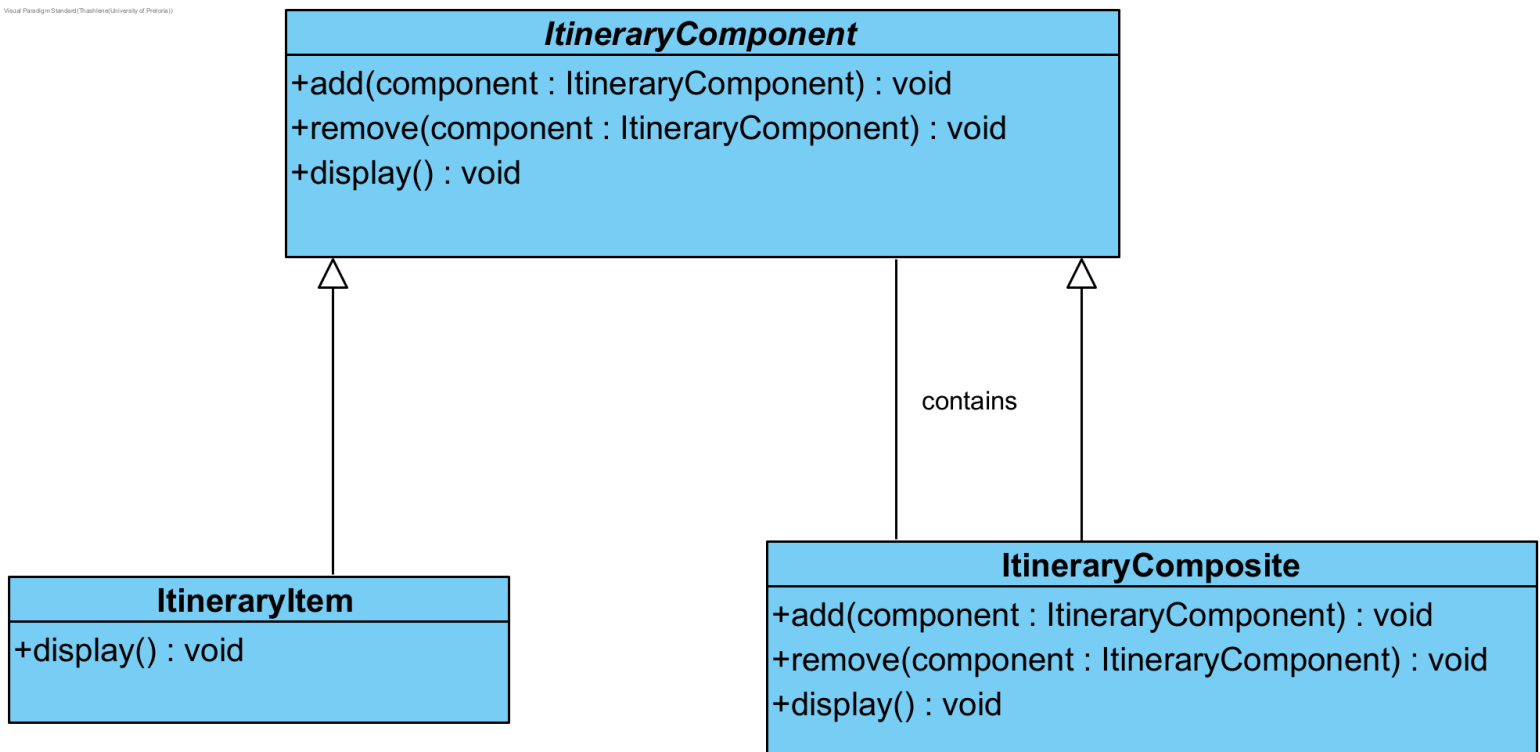
The *NotificationService* to notify users about changes or updates to their itineraries.



5. Composite Pattern

The *ItineraryComponent* to handle hierarchical itineraries.

Visual Paradigm Standard (Toolbars/University of Pretoria)



Constraints

API Limitations

Rate Limits: Free API's often have strict rate limits, meaning the number of requests you can make per minute or per day is restricted.

Data Limitations: Free tiers of API's may provide limited access to data. For example, a travel API might limit the number of cities or the depth of information you can query.

Features: Some advanced features might only be available in paid tiers.

Data Quality and Coverage

Accuracy and Completeness: Free data sources might not be as accurate or comprehensive as paid ones. They may miss important details or have outdated information.

Geographic Coverage: Some free services might have limited geographic coverage, which can be a significant constraint if you aim to create itineraries for less popular destinations.

Performance: Free services may not handle a high volume of requests efficiently, leading to slower response times or even downtime during peak usage.

Concurrent Users: Handling multiple users simultaneously might be challenging with free services due to limited resources.

Integration and Compatibility

Data Formats: Different services might provide data in various formats, requiring additional effort to standardize and integrate them.

APIs and SDKs: Some free services might have limited or poorly documented APIs, making integration more difficult.

Legal and Licensing Constraints

Usage Rights: Free services may have restrictions on how you can use their data. Ensure you comply with their terms of service and usage policies.

Attribution: Some free services require attribution, which you must incorporate into your application.

Functionality

Limited Customisation: Free tools and services might offer limited customization options, restricting the ability to tailor the solution to specific needs.

Basic Features: You may only have access to basic features and miss out on advanced functionalities that could enhance the user experience.

Support and Reliability

Support: Free services often come with limited or no support. You might rely on community forums or documentation to resolve issues.

Reliability: Free services might not offer the same level of reliability and uptime guarantees as paid services.

Strategies to Mitigate Constraints

Combination of Services

Use a combination of free services to cover the limitations of individual APIs

Fallback Mechanisms

Develop fallback mechanisms to handle API downtimes or failures by switching to alternative services.

Incremental Development

Start with basic functionalities using free services and plan for future enhancements using paid services if needed.

Service Contracts

Our application leverages a variety of services and tools to ensure optimal performance, seamless integration, and efficient monitoring. Below is a detailed overview of the services we utilize:

1. **Hosting:** We use **Vercel** for hosting our frontend application. Vercel provides a reliable and scalable platform, ensuring that our application is always available and performant. It supports continuous deployment and integrates seamlessly with our GitHub repository, allowing for automatic deployment of new updates.
 2. **Database:** Our application's data is managed using **MongoDB**. MongoDB is a NoSQL database that offers high performance, high availability, and easy scalability. Its flexible schema design allows us to store data in a JSON-like format, making it suitable for our dynamic data needs.
 3. **Flight API:** We rely on **Amadeus** for making flight API calls. Amadeus provides comprehensive travel-related data, including flight schedules, availability, and booking information. This allows us to offer up-to-date flight options to our users, enhancing their travel planning experience.
 4. **Continuous Integration and Deployment (CI/CD):** **GitHub Actions** is our chosen tool for automating actions in our repository. We use it to run tests, build our application, and deploy it. GitHub Actions enables us to implement robust CI/CD pipelines, ensuring code quality and speeding up the development cycle.
-

5. **Monitoring:** To maintain the health and performance of our application, we utilize **PingPong** for monitoring. PingPong provides real-time alerts and insights into the application's uptime and response times. This allows us to quickly identify and address any issues that arise, ensuring minimal downtime and optimal performance.

6. **Issue Tracking:** We track and manage bugs, feature requests, and other project tasks using **GitHub Issues**. This integration allows our team to maintain a clear and organized workflow, ensuring that all tasks are visible and manageable within our development environment.

7. **Dependency Management:** **requires.io** is used to manage our project's dependencies. **requires.io** monitors our dependencies for updates and security vulnerabilities, ensuring that our application is always running with the most secure and up-to-date packages.

These services collectively contribute to the robustness, efficiency, and reliability of our application. By leveraging these tools, we ensure a seamless development process, efficient resource management, and an excellent user experience.

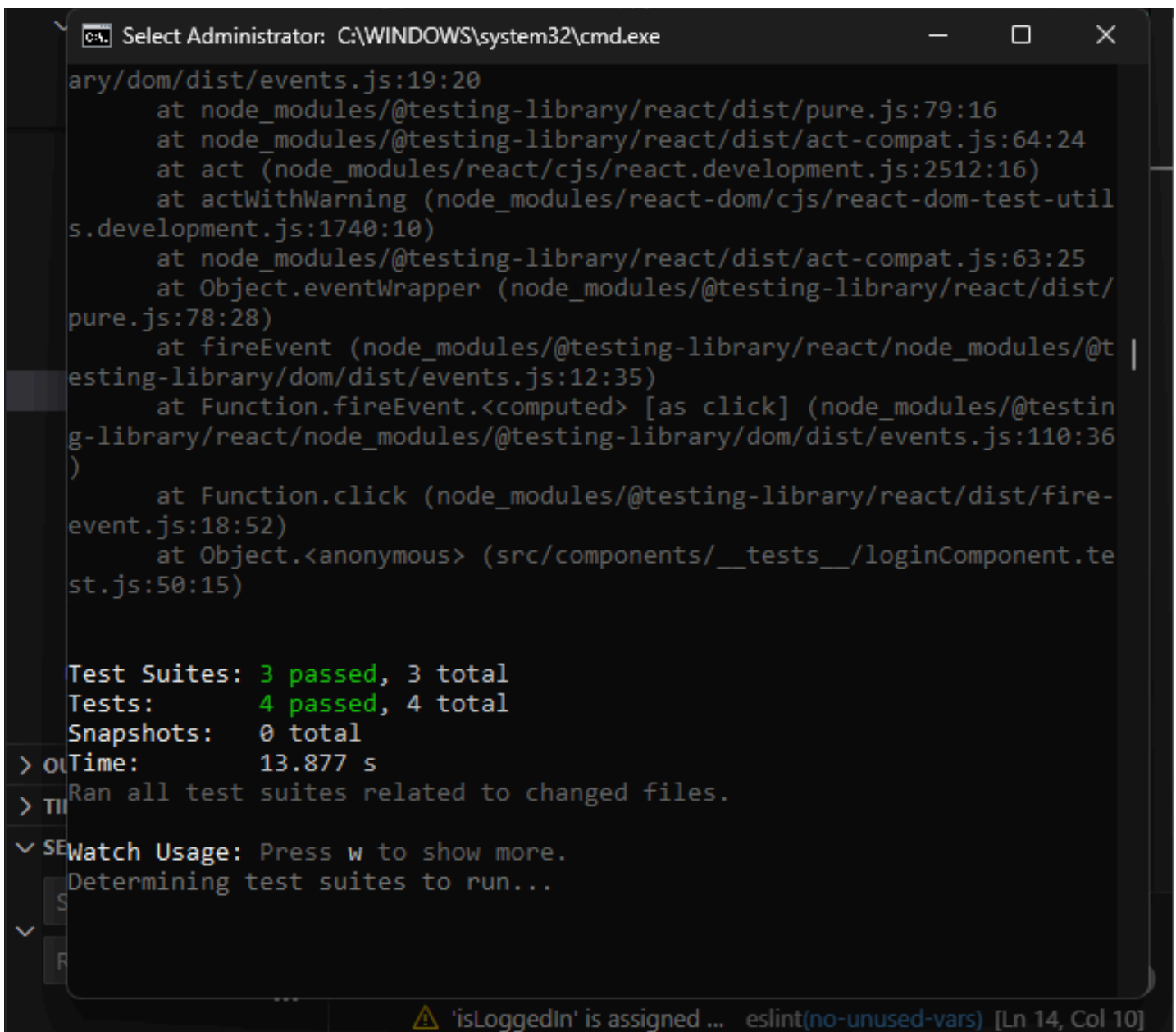
Currently, pending contracts are AeroDataBox, for a student account to make unlimited API requests to gather flight data.

Another pending contract is LekeSlaap for possible API calls to gather data for accommodation.

Currently, we are still searching for things to do in the region of South Africa

Testing

Unit Testing



```

C:\WINDOWS\system32\cmd.exe
ary/dom/dist/events.js:19:20
    at node_modules/@testing-library/react/dist/pure.js:79:16
    at node_modules/@testing-library/react/dist/act-compat.js:64:24
    at act (node_modules/react/cjs/react.development.js:2512:16)
    at actWithWarning (node_modules/react-dom/cjs/react-dom-test-util
s.development.js:1740:10)
    at node_modules/@testing-library/react/dist/act-compat.js:63:25
    at Object.eventWrapper (node_modules/@testing-library/react/dist/
pure.js:78:28)
    at fireEvent (node_modules/@testing-library/react/node_modules/@t
esting-library/dom/dist/events.js:12:35)
    at Function.fireEvent.<computed> [as click] (node_modules/@testin
g-library/react/node_modules/@testing-library/dom/dist/events.js:110:36
)
    at Function.click (node_modules/@testing-library/react/dist/fire-
event.js:18:52)
    at Object.<anonymous> (src/components/__tests__/loginComponent.te
st.js:50:15)

Test Suites: 3 passed, 3 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        13.877 s
Ran all test suites related to changed files.

Watch Usage: Press w to show more.
Determining test suites to run...

...
'isLoggedIn' is assigned ... eslint(no-unused-vars) [Ln 14, Col 10]
```