Software Design Document

Tourism Insights Dashboard

Group: Coding-is-Life

11-27-2024

Contents

1. Introduction

The Tourism Insights Dashboard is an interactive platform designed to provide insights into tourism trends, weather and traffic forecasts, and economic impact by combining data from multiple APIs, including Digi Traffic, Statistics Finland and Visit Finland.

1.1 Purpose

This document provides a detailed design for the Tourism Insights Dashboard application, outlining its architecture, components, and specifications. It serves as a guide for developers and stakeholders involved in the project.

1.2 Scope

The application will assist users in planning their trips based on the insights provided in the dashboard considering various aspects. The integration of tourism data and weather forecast ensures that users can make informed travel decisions.

1.3 Audience

- Software developers
- Project managers
- System architects
- Quality assurance teams
- Stakeholders

1.4 Definitions, Acronyms, and Abbreviations

- **UI:** User Interface
- **API:** Application Programming Interface
- **MVC:** Model-View-Controller
- **JavaFX:** A software platform for creating rich client application

2. Objectives

The objective of this application is to create an interactive platform that analyzes tourism trends and provides regional economic factors. Additionally, it aims to provide real-time weather data and forecasts for these tourism destinations, enabling travellers to make well-informed decisions. It will also offer insights into tourism statistics, helping users to find out visitor trends and popular locations over time.

2.1. Functional Requirements

- **Travel Planning:** Users can get tourism insights based on locations and seasons.
- **Weather Forecasting:** Users can view weather and traffic conditions.
- **Economic Impact:** Users can view economic impact data related to tourism.
- **Data Visualization:** Data will be presented using various charts, graphs and tables for easy interpretation.
- **User Preferences:** Users can save preferred time frames, locations and other settings.

2.2. Non-functional Requirements

- **Performance:**

  The application should minimize the time to load travel and weather data.

- **Scalability:**

  The architecture supports easy extension. Future additions could include:

  - Add more detailed tourism insights (e.g., event-based data).
  - Integrate historical weather data for comparison.
  - Add user authentication for customized reports and saved locations.

- **Usability:**

  The interface must be user-friendly and accessible to a wide range of users.

- **Reliability**

  Ensure that the data used for visualizations (ex: trip data, visitor data) is always accurate and consistent.

2.3 Key Features

- **Data Retrieval:**
  - Retrieve weather and traffic related data from https://www.digitraffic.fi.
  - Fetch statistical data on travel and tourism trends from Statistics Finland.
  - Retrieve economic impact data from https://visitfinland.stat.fi
  - Combine and present travel and weather information in a user-friendly format.

- **Visualization:**
  - Visual representation of economic indicators.
  - Visual representation of tourism statistics and trends at various locations.
  - Present travel and weather information in a user-friendly format.

- **User Interaction:**
    - Input options for locations, travel time and other selections.
    - Toggle between different weather/travel parameters.
    - Save and load user preferences for different parameters for quick access using Java Preferences API.

- **Modular and Scalable Architecture:**
    - Future addition of new APIs and add more data from existing APIs.
    - Designed to be scalable with minimal changes required when integrating new services.

3. System Architecture

3.1 High-Level Overview

The **Tourism Insights Dashboard** application is designed to provide insights into various aspects of tourism, including economic impact, visitor and travel statistics, traffic data, and weather conditions. The main components in this application are organized according to the **Model-View-Controller (MVC)** architecture, which separates data handling, user interface, and control logic.

3.1.1 Components:

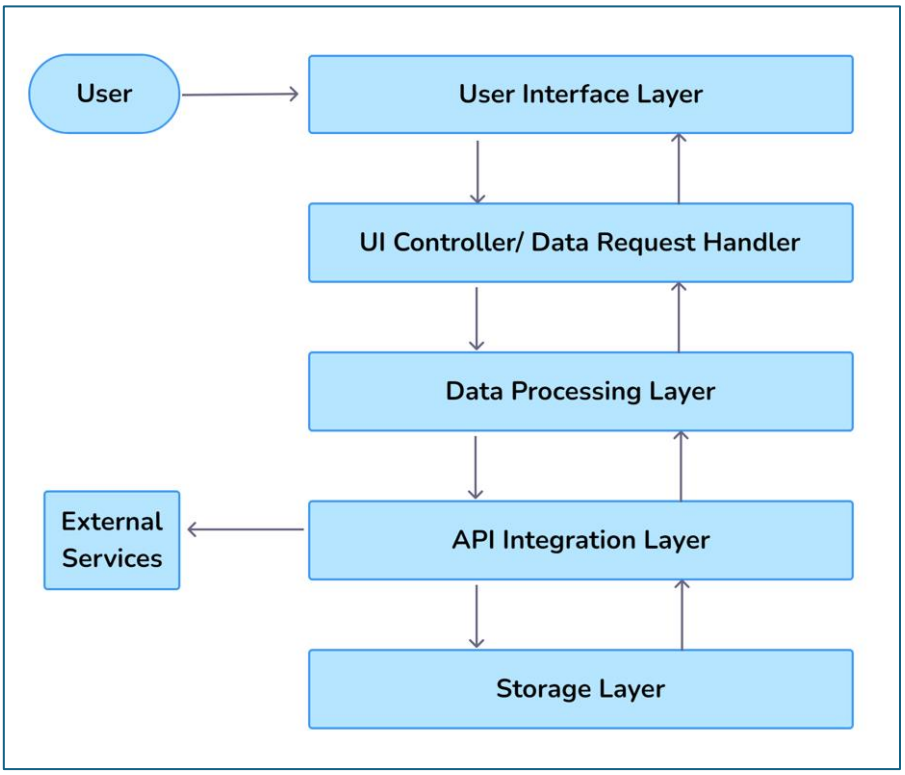
- **Controllers**: Manage user input, handle navigation between views, and update the UI based on data processing.
- **Models**: Represent data structures (e.g., WeatherData, VisitorStatistics, TripStatistics).
- **Views**: Present UI components and visualization to the end user.
- **Services**: Responsible for data retrieval from external APIs (e.g., EconomicImpactService, TrafficService, WeatherService) and for processing the data.

- **Service Façade**: It acts as a single access point which simplifies interacting with different service classes and by using Service Facade, clients of the application can interact with service facade class without needing to manage each individual data service separately. (StatisticsServiceFacade, HomeServiceFacade)
- **Unit Tests**: Tests the crucial functionality of the application and its integration with the other classes and ensures that the operations work as expected.

3.1.2 High-Level Dependencies:

- The application relies on JavaFX for UI rendering and user interaction.
- Libraries such as Gson are used for JSON parsing, Apache HTTPComponents in handling HTTP requests and Apache POI is included for potential Excel handling.
- The **Main Class** initializes the application and facilitates scene changes between different controllers.

3.1.3 High-level Data Flow Diagram



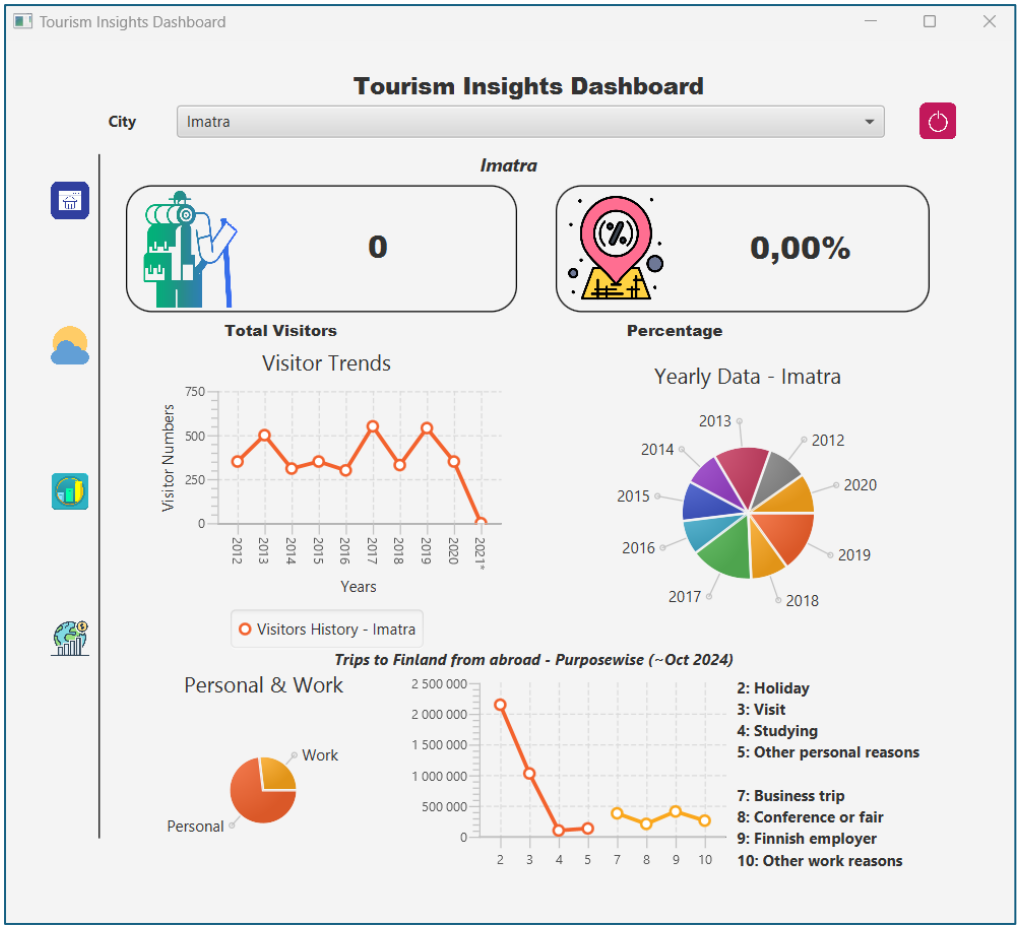
3.2 User Interfaces

3.2.1 Home Screen

The homepage of the application represents a comprehensive UI for visualizing and analyzing tourism-related data. The dashboard is divided into sections, each focused on different aspects of tourism insights.

Key Features:

1. **Filters**:
    - A combo box labelled "City" for selecting a specific location (e.g., Finland cities).
    - Triggers data reload specific to the selected location.
2. **Navigation Icons**:
    - Left-side icons for switching between various sections such as:
        - Home (Default Screen)
        - Weather
        - Tourism Statistics
        - Economic Impact
3. **Graphs and Charts**:
    - A Line Chart visualizing visitor trends over the years.
    - A Pie Chart providing a breakdown of visitors, likely by category or origin.
    - Additional:
        - Purpose-specific charts, including:
            - A second Pie Chart showing the distribution of trips by purpose.
            - A second Line Chart visualizing trends in trip purposes.



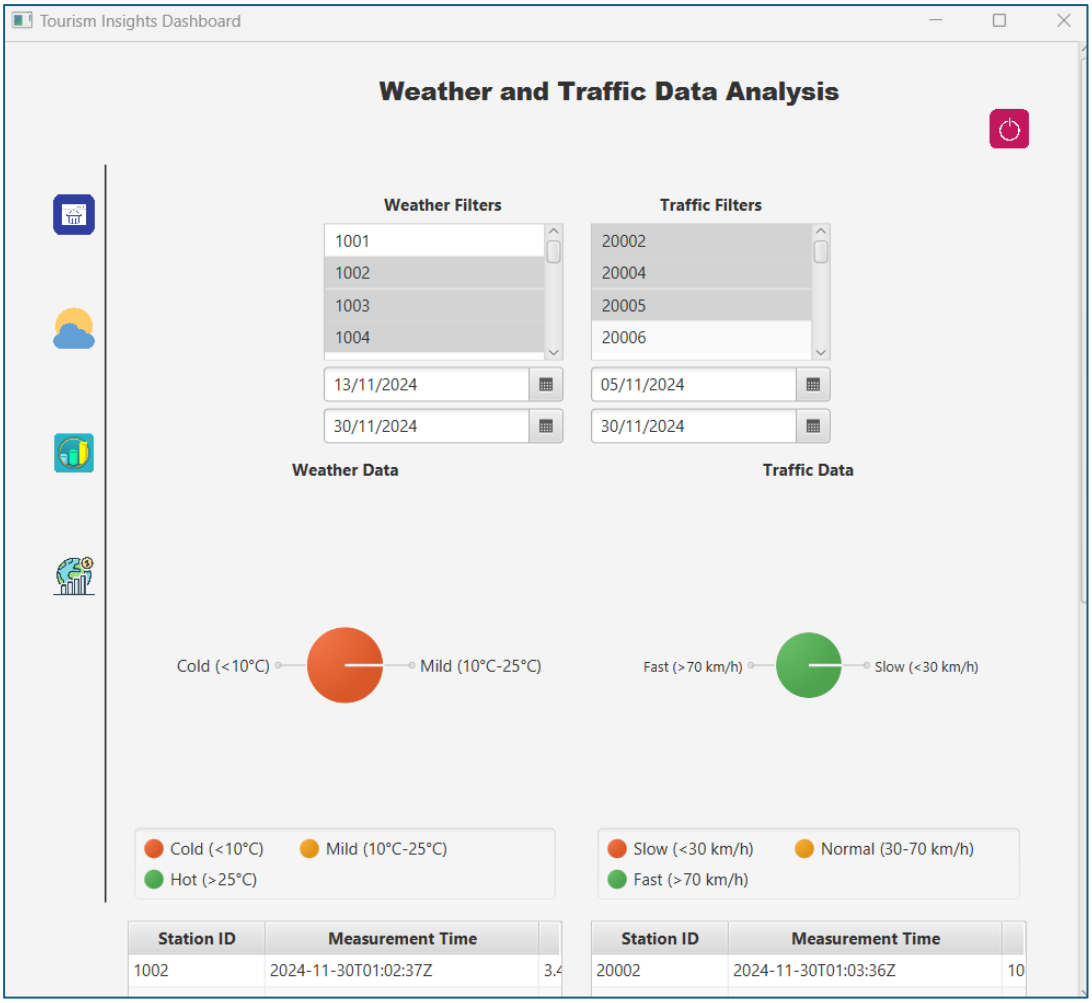
3.2.2 Weather and Traffic Data Screen

Weather and Traffic Data Analysis Screen is designed for users to analyze weather and traffic conditions using interactive filters and visualizations.

Key Features:

1. **Navigation Panel**:
    - Located on the left side with icons for navigating between different sections:
        - Home
        - Weather (Current Screen)
        - Tourism Statistics
        - Economic Impact

2. **Filters Section**:
    - Two vertical filter panels for Weather and Traffic:
        - Users can select weather and traffic stations from a ListView.
        - Includes DatePicker components for specifying start and end dates for data filtering.
3. **Data Visualization**:
    - **Weather Data Section**:
        - A PieChart visualizing weather-related statistics (e.g., temperature distribution, precipitation).
        - A TableView displaying detailed weather measurements with columns like:
            - Station ID.
            - Measurement Time.
            - Temperature, Precipitation, and Wind Speed.
    - **Traffic Data Section**:
        - A PieChart visualizing traffic-related statistics (e.g., traffic volume breakdown).
        - A TableView displaying traffic measurements with columns like:
            - Station ID.
            - Measurement Time.
            - Volume and Speed.



3.2.3 Tourism Statistics Screen

This screen mainly visualizes a Tourism Statistics Dashboard for analyzing visitor data, trip patterns, and demographics with interactive filters and visualizations. It is useful for users to get insights in different aspects before planning their trips.

Key Features:

1. **Navigation Panel**:
    - Located on the left side with icons for navigating between different sections:
        - Home

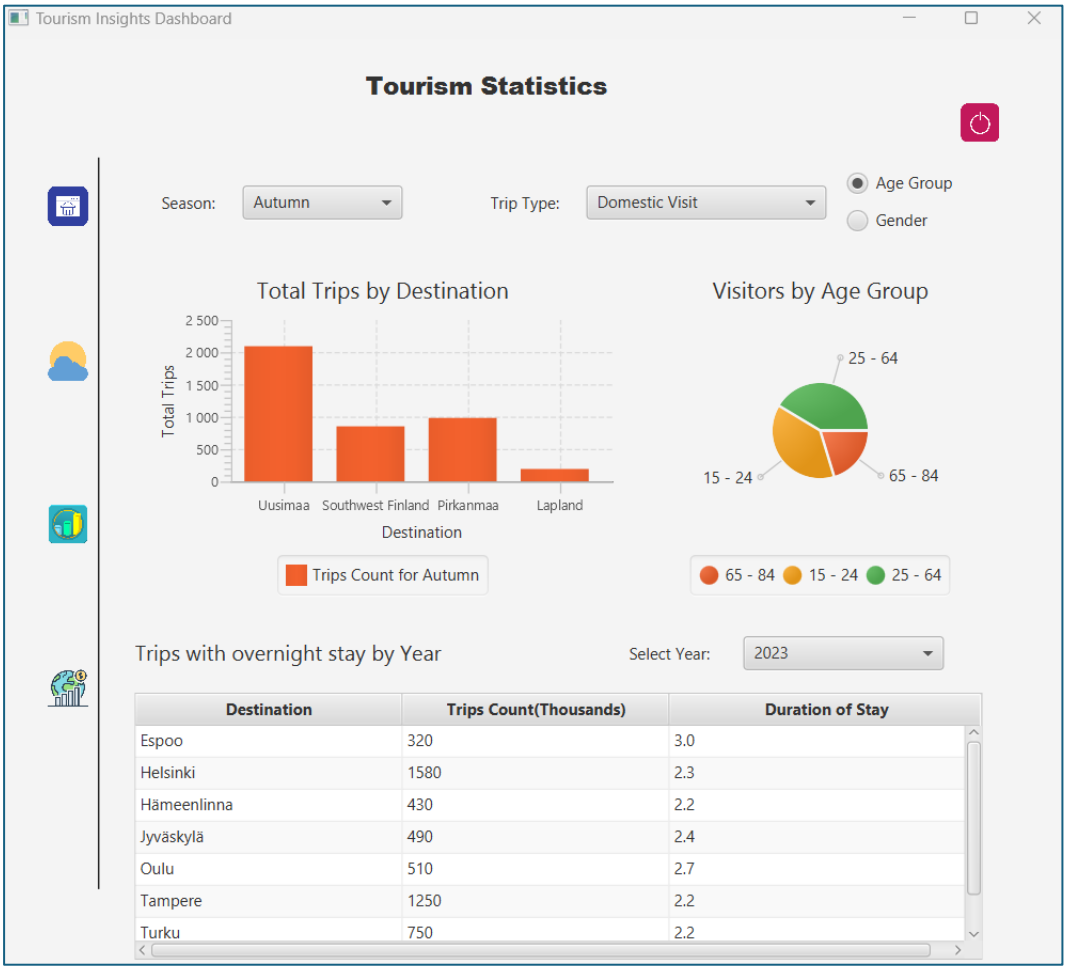
- Weather
- Tourism Statistics (Current Screen)
- Economic Impact

2. **Filters Section**:

- Season Filter:
    - Combo Box for selecting specific seasons (e.g., Summer, Winter).
- Trip Type Filter:
    - Combo Box for filtering by trip types (e.g., leisure, business).
- Demographics Filters:
    - Two Radio Button options for analyzing data by Age Group or Gender.
- Year Filter:
    - A Combo Box for selecting specific years to analyze overnight trips.

3. **Data Visualization**:

    - Bar Chart:
        - Displays trip counts grouped by destinations.
        - Uses a CategoryAxis for destinations and a NumberAxis for trip counts.
    - Pie Chart:
        - Represents the distribution of visitors by demographics (e.g., age groups).
    - Detailed Statistics Table:
        - A TableView to provide detailed information on:
            - Destination: Locations where trips occurred.
            - Trips Count (Thousands): Number of trips to the destination.
            - Duration of Stay: Average duration of the trips.



3.2.4 Economic Impact Screen

This layout defines a Regional Economic Impact of Tourism Dashboard, allowing users to analyze the economic contributions of tourism by products, regions, and other attributes.

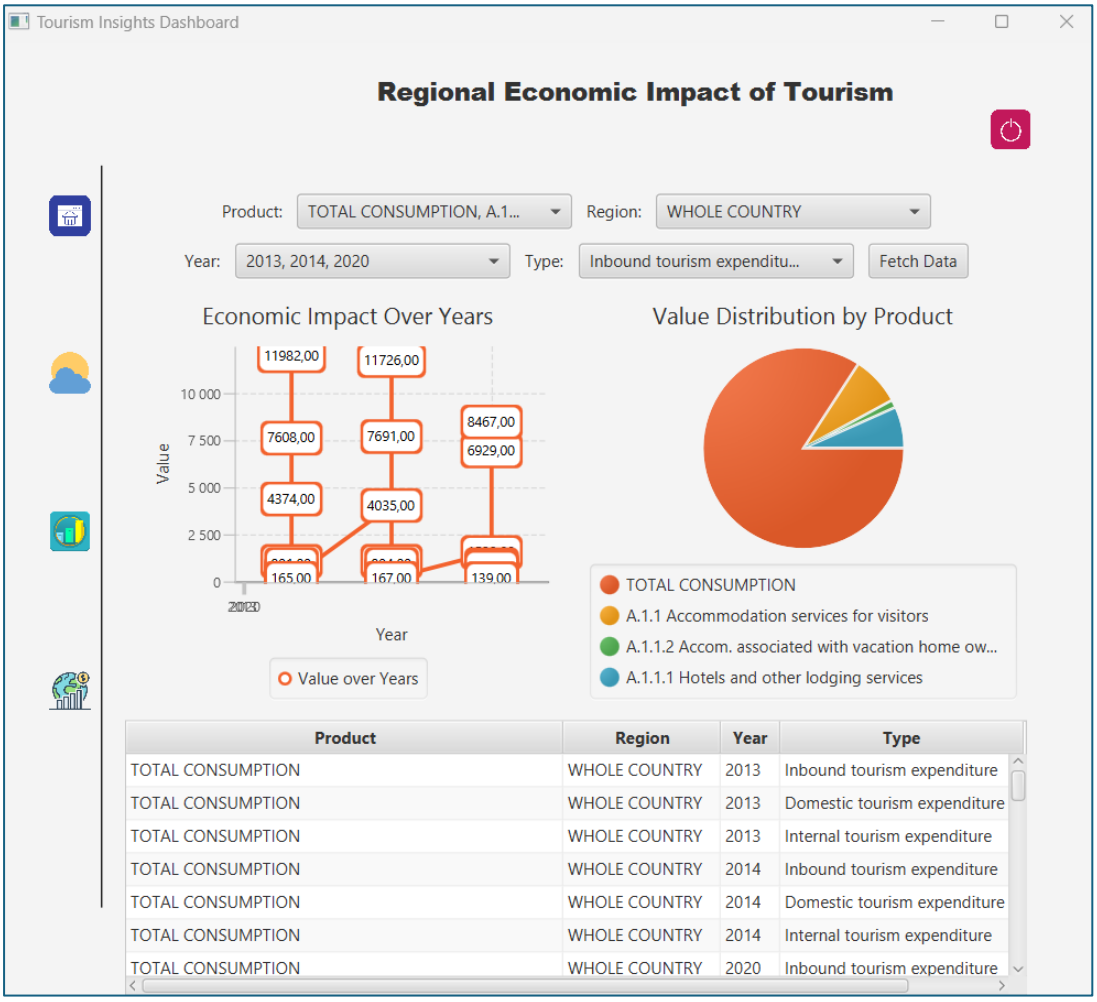
Key Features:

1. **Navigation Panel**:
    - Located on the left side with icons for navigating between different sections:
        - Home
        - Weather

- Tourism Statistics
- Economic Impact (Current Screen)

2. **Filter Section:**
    - Product Dropdown:
        - CheckComboBox allows users to select multiple tourism-related products (e.g., lodging, dining, activities).
    - Region Dropdown:
        - CheckComboBox for selecting multiple regions to analyze their economic impact.
    - Year Dropdown:
        - CheckComboBox for selecting specific years or a range.
    - Type Dropdown:
        - CheckComboBox to filter by type (e.g., direct, indirect, induced impact).
3. **Visualization Section:**
    - Line Chart:
        - Displays the economic impact trend over the years.
        - CategoryAxis for years and NumberAxis for impact values.
        - Useful for analyzing changes in impact over time.
    - Pie Chart:
        - Illustrates the distribution of values by product.
        - Allows quick visualization of economic contributions from different tourism sectors.
    - Data Table:
        - Located below the charts for a detailed view of data.
        - Displays granular data such as:
            - Product: Tourism product categories.

- Region: Impact by regions.
- Year: Annual data.
- Value: Numerical representation of the economic impact.



3.3 Internal Structure and Functions

The application defines interfaces to support modularity and abstraction, allowing for interchangeable components. The main interfaces include:

3.3.1 IDataService<T>:

- **Purpose**: Standardizes data processing across different types of data services (e.g., ForeignVisitorsDataService, TripsByDestinationDataService).
- **Methods**:
    - **processData():** Retrieves and structures data.
    - **filterData(List<T>, String filter):** Allows filtering of processed data based on specific criteria.

3.3.2 ISpecificKeywiseDataService<T>:

- **Purpose**: Handles data services requiring a specific key parameter (e.g., year-wise data).
- **Implementation**: ForeignVisitorsYearWiseDataService implements this interface to filter visitor data based on years or specific keys.

3.3.3 DataRetrievable:

- **Purpose**: Defines methods for data retrieval from APIs.
- **Implementation**: Used by StatisticsFinlandDataService for standardized data fetching across different services.

3.3.4 Flow Description:

- **Controllers** (e.g., HomeController) request data through **Service Facade**, which use the interfaces to fetch and process data from multiple services, then return results back to the controllers.
- **Service Façade** (e.g. HomeServiceFacade) encapsulates the interaction with different services and acts as a central point for interacting with different data services.

- **Services** implement these interfaces to fetch data from external APIs, standardize it, and prepare it for use by UI components.

3.4 Component Breakdown

The architecture of the application is based on the Model-View-Controller (MVC) pattern to ensure clear separation of concerns.

3.4.1 Controllers:

- **HomeController**: Manages main navigation and updates UI components based on data from tourism statistics.
- **EconomicImpactController**: Manages economic data selection, fetches data using EconomicImpactService, and updates line and pie charts based on selected filters.
- **WeatherController**: Handles user input for selecting weather and traffic stations, date range, and updates the UI with data from WeatherService and TrafficService.
- **StatisticsController**: Visualize the tourism and visitor statistics based on tourist destinations, different seasons of the year and visitor demographics.

3.4.2 Service Façade:

- **HomeServiceFacade :** Provides single access point for HomeControllers to connect with multiple service classes
- **StatisticsServiceFacade:** Provides single access point for Statistics Controller to connect with multiple service classes

3.4.3 Services:

- **EconomicImpactService**: Retrieves economic impact data from an external API, processes initial data for dropdowns, and handles data filtering for visualizations.
- **TrafficService**: Fetches traffic data based on station and date range selections.
- **WeatherService**: Retrieves weather data using filters for station and date range.
- **VisitorStatisticsDataService:** Extends StatisticsFinlandDataService, defining custom processing for visitor data by mapping age groups and gender. It implements IDataService<VisitorStatistics>.
- **ForeignVisitorsByPurposeDataService:** Retrieves and process data on foreign visitors to Finland, categorized by their travel purpose.
-

3.4.4 Models:

- **EconomicImpactData**: Holds data for economic impact by product, region, year, and type.
- **WeatherData**: Stores weather data with attributes such as station ID, temperature, precipitation, and wind speed.
- **VisitorStatistics**: Represents visitor statistics categorized by age group, gender, trip type, and percentage.
- **TripStatistics:** Represents trip statistics categorized by destination, season and trip count.
- **TouristYearData:** Represents tourists statistics categorized by year and tourists count.
- **TouristData:** Represents tourists statistics categorized by region and tourists count.

3.4.5 Views:

- HomePage.fxml
- Statistics.fxml
- Weather.fxml
- EconomicImpact.fxml

3.5 APIs

- **Statistics Finland**: Provides statistical data on tourism trends and travel information.
- **digitraffic.fi:** Provides traffic and weather-related data for specified regions.
- **visitfinland.stat.fi**: Provides data related to economic impact for tourism.

3.6 Rationale for Design Choices

- **MVC Architecture**: This design enables separation of concerns, which has allowed independent development and testing of UI, services, and data models.
- **SOLID Principles**: Use of SOLID principles ensure that application is maintainable and scalable while reducing complexity and improving productivity, and robustness.
- **Interfaces for Abstraction**: Using IDataService and DataRetrievable interfaces allows flexible data processing and retrieval, supporting potential API source changes without impacting controllers.
- **Generic Programming**: Using generics in IDataService interface supports the implementation classes to use different types of data models, providing type safety and code reusability without needing to write redundant code for different data types.
- **Design Patterns (Façade, Strategy) :** Simplifies by providing a unified interface, making it easier to use the service layer and promotes flexibility by defining different behaviors of service classes.
- **JavaFX**: Chosen for its capabilities in building interactive and responsive desktop applications.
- **Gson** for JSON parsing: Chosen for compatibility with API response structures.
- **Apache HTTPComponents**: Chosen for its ease of use in handling HTTP requests. The library simplifies POST requests and response parsing, essential for connecting to external APIs.
- **Error Handling**: Designed to handle common HTTP errors and data validation issues, allowing robust data retrieval from external sources.
- **Junit:** for unit testing to make sure that system components work as expected.

- **Java Preferences API:** For saving user preferences and providing quick access to desired data.

3.7 Design Patterns and Approaches

1. **MVC Architecture**
   - **Model:** Manages data and business logic (API data processing, caching).
   - **View:** Frontend dashboard displaying data (charts, table views).
   - **Controller:** Backend handling API requests, user interactions, and data management.

2. **SOLID Principles**

   a. **Single Responsibility Principle (SRP)**

   Each of the model and service classes used in the application holds a single responsibility, focusing on a specific aspect of handling data, which makes the design modular and compliant with SRP. By having distinct purposes, each class is easy to understand, change, and test independently without affecting other parts of the application.

   b. **Open/Closed Principle (OCP)**

   This is primarily achieved through the use of interfaces and abstract classes, allowing new functionality to be added without modifying existing code. DataRetrievable and IDataService<T> are interfaces that allow for new implementations of data fetching and processing without altering existing classes. StatisticsFinlandDataService abstract class provides common data retrieval logic where specific service classes can extend it to define the corresponding behavior, without needing itself to modify if new services are added.

c. **Liskov Substitution Principle (LSP)**

It maintains the integrity and expected behavior of fetchData, processData and filterData methods defined in DataRetrievable and IDataService<T> interfaces, across the application.

d. **Interface Segregation Principle (ISP)**

The design separates functionality across different interfaces (DataRetrievable for fetching data, IDataService<T> for processing and filtering).

e. **Dependency Inversion Principle (DIP)**

DIP is applied through the use of the IDataService interface as an abstraction layer when declaring instances of any concrete data service implementation.

3. Design Patterns

a. **Façade -** This structural design pattern is used to simplify the interaction between controllers and multiple data services providing a single point for interacting with different data services. For an instance, StatisticsServiceFacade class aggregates all the service connections used for StatisticsController so that clients of the application can interact with a single service class without needing to manage each individual data service separately.

b. **Strategy -** This behavioral design pattern is used by defining a IDataService interface (as a Strategy Interface with processData method) and implementing different strategies to manipulate data based on the external APIs response such as VisitorStatisticsDataService, TripsByDestinationDataService, The specific behaviour (data processing, filtering) is defined by the implementing concrete classes, and it is possible to switch between different strategies easily based on the context.

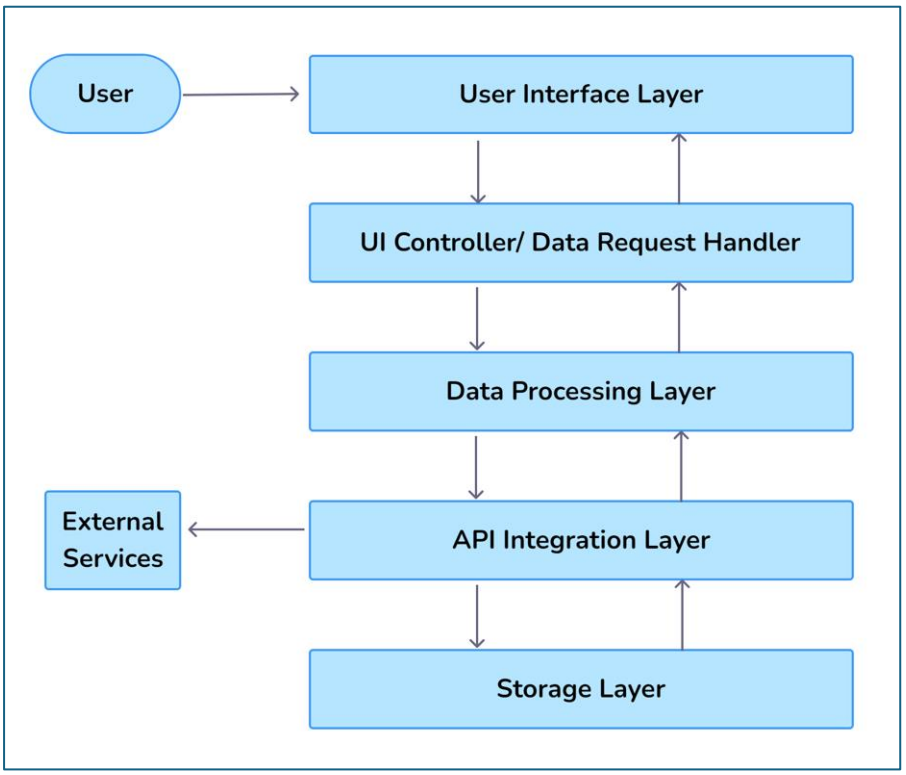
4. **Generics**

Generics is used again in IDataService interface and ISpecificKeywiseDataService interface to make it flexible to use for different types of data models that has been defined in the application and the corresponding model can be defined in the implementation class of the interface.

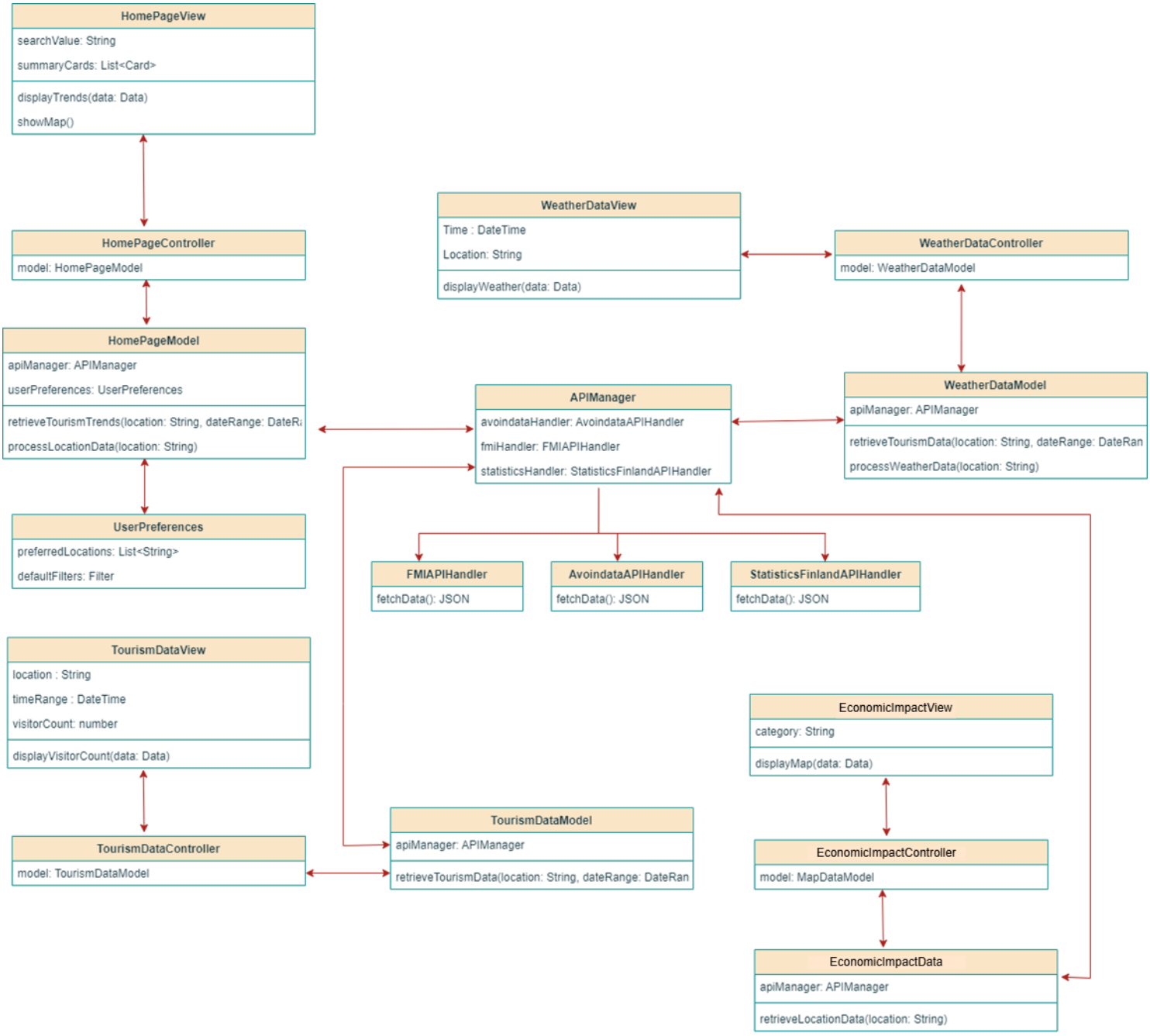
4. Diagrams

4.1 Data Flow Diagram

1. **User interacts** with the dashboard (selects region, date).
2. **Frontend sends requests** to the backend.
3. **Backend fetches data** from APIs (Avoindata.fi, FMI, Statistics Finland).
4. **Backend processes and merges** data.
5. **Frontend displays data** on graphs and charts.



4.2. UML Class Diagram



5. Future Enhancements

- Add more detailed tourism insights (e.g., event-based data).
- Integrate historical weather data for comparison.
- Add user authentication for customized reports and saved locations.

6. Self-Evaluation

- **Implementation of Design**:
    - The MVC architecture has supported implementation by separating data retrieval, processing, and UI handling.
    - The use of interfaces like IDataService and DataRetrievable has allowed flexibility in switching or processing data sources without impacting other parts of the application.
    - Use of SOLID principles ensure that application is maintainable and scalable while reducing complexity and improving productivity, and robustness.
    - Use of design patterns simplifies the system by providing a unified interface, making it easier to use the service layer and promotes flexibility by defining different behaviors of service classes.
    - Using unit tests, it ensures that individual components work as expected, enabling early bug detection, simplifying debugging, and supporting safe code refactoring.
- **Challenges**:
    - Some APIs required specific data processing and error handling to ensure data compatibility, which necessitated adjustments in the design.

    - Real-time data retrieval added complexity in maintaining a responsive UI, especially with larger datasets.

- **Future Changes**:
    - **API Caching**: To reduce API calls and enhance performance, caching will be implemented for frequently accessed data.
    - **Extended Filtering**: Additional filters may be added to enhance data granularity.
    - **Error Handling Enhancements**: Adding more comprehensive error handling for API interactions to improve robustness.

7. Use of AI Tools

1. AI tools were used to brainstorm and find out suggestions in order to come with an idea to implement the system based on the functional requirements for the group project.

2. It is also used to define and prioritize a set of system features and relevant functionalities for the selected application, based on similar applications available.

3. Further it was used to troubleshoot the errors during implementation phase. This helped to save time the development type.

4. Sometimes AI was also used to suggest fixes or improvements during development of the application.

8. Conclusion

The Tourism Insights Dashboard will offer valuable insights by integrating tourism, weather, and location data using modular architecture. The use of design patterns like MVC, Factory, and Observer will help in building a scalable and maintainable application.