Software Design Document

Tourism Insights Dashboard

Group: Coding-is-Life

9-27-2024

Contents

1. Introduction

The Tourism Insights Dashboard is an interactive platform designed to provide insights into tourism trends, weather forecasts, and location data by combining data from multiple APIs, including Avoindata.fi (tourism datasets), Statistics Finland and Finnish Meteorological Institute (FMI).

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 trend and provide 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 current and forecasted weather conditions for specified locations.
- **Data Visualization:** Tourist and weather data will be presented using charts for easy interpretation.
- **User Preferences:** Users can save preferred locations and 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**

2.3 Key Features

- **Data Retrieval:**

  Fetch travel and tourism data from Avoindata.fi.

  Retrieve weather data from The Finnish Meteorological Institute's open data.

  Fetch statistical data on tourism trends from Statistics Finland.

  Combine and present travel and weather information in a user-friendly format.

- **Visualization:**

  Visual representation of economic indicators.

  Graphical representation of statistics and trends at various locations.

  Combine and present travel and weather information in a user-friendly format.

- **User Interaction:**

  Input options for locations, and the travel dates.

  Toggle between different weather/travel parameters.

  Save and load user preferences for locations, dates, and weather types for quick access.

- **Modular and Scalable Architecture:**

  Future addition of APIs (e.g., traffic data, air quality).

  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 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.

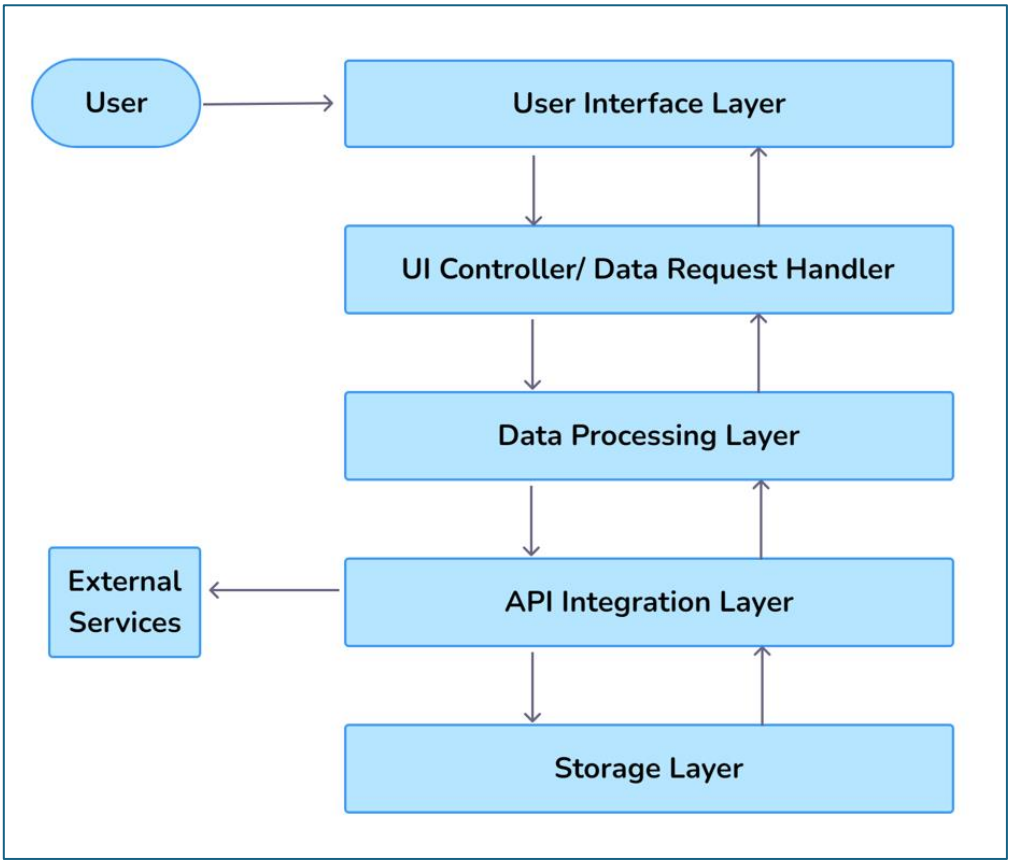
3.1.1 Components:

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

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 Diagram



3.2 Boundaries and Interfaces

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

3.2.1 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.

3.2.2 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.

3.2.3 DataRetrievable:

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

3.2.4 Flow Description:

- **Controllers** (e.g., EconomicImpactController) request data through **Services**, which use the interfaces to fetch and process data, then return results back to the controllers.
- **Services** implement these interfaces to fetch data from external APIs, standardize it, and prepare it for use by UI components.

3.3 Component Breakdown

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

3.3.1 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.

3.3.2 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.

3.3.3 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.

3.3 APIs

- **Avoindata.fi**: Provides tourism-related datasets (e.g., tourist arrivals, accommodations, events).
- **Statistics Finland**: Provides statistical data on tourism trends.

- **Finnish Meteorological Institute (FMI)**: Provides real-time weather forecasts for specified regions.

3.4 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.
- **Interfaces for Abstraction**: Using IDataService and DataRetrievable interfaces allows flexible data processing and retrieval, supporting potential API source changes without impacting controllers.
- **JavaFX**: Chosen for its capabilities in building interactive and responsive desktop applications.
- **Libraries**:
- **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.
- **Apache POI** for potential future integration with Excel data processing.
- **Error Handling**: Designed to handle common HTTP errors and data validation issues, allowing robust data retrieval from external sources.

3.5 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 has 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.

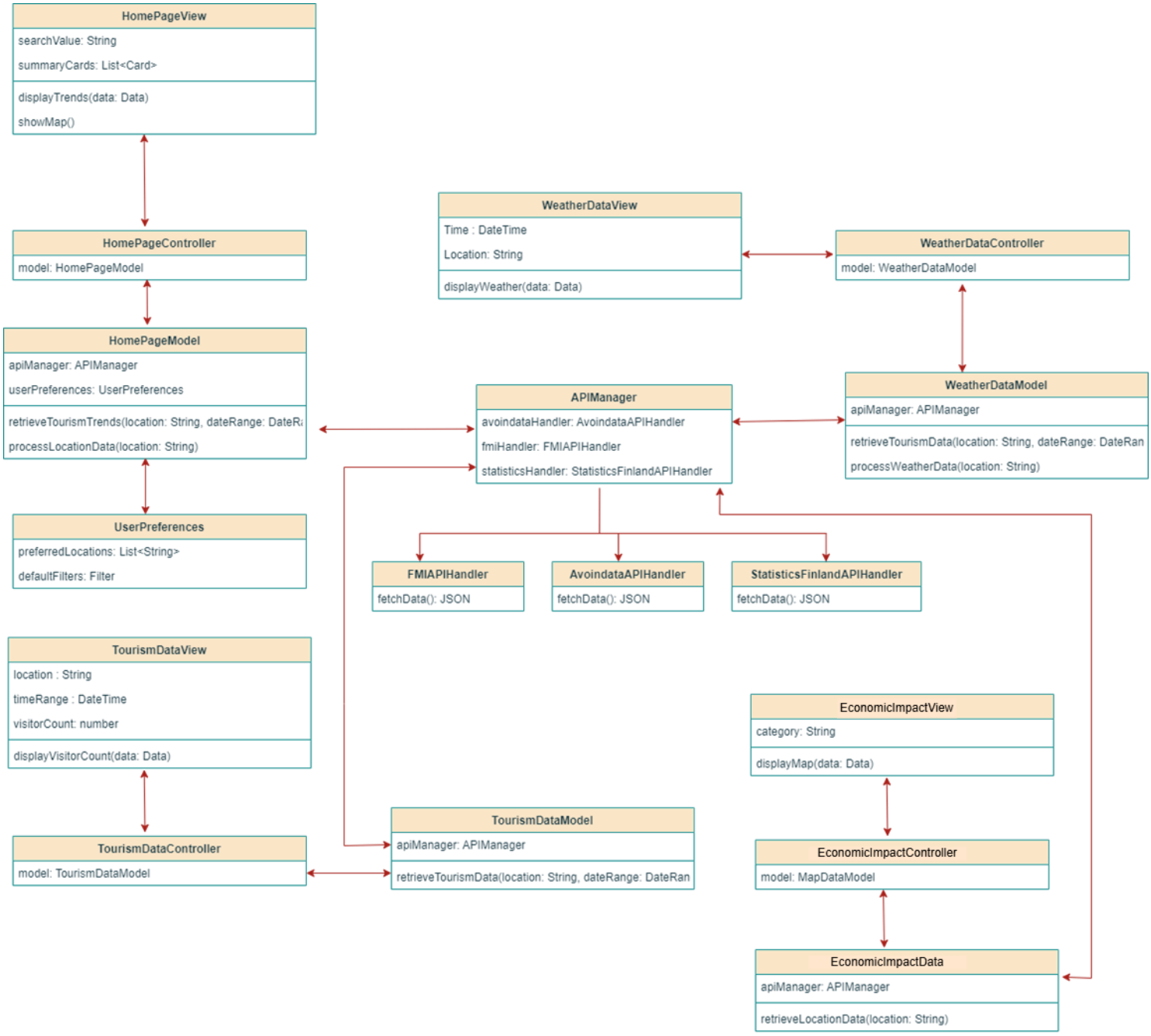
4. Diagrams

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

As for the midterm, the project has completed a substantial portion of the application structure and implementation:

- **Implementation of Design**:
    - The MVC architecture has supported partial implementation by separating data retrieval, processing, and UI handling.
    - The use of interfaces like IDataService and DataRetrievable has allowed flexibility in switching or updating data sources without impacting other parts of the application.
- **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. Conclusion

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