Software Requirements Specification for FlightsCompare

Team Project

October 20, 2024

Contents

1 Introduction

This document serves as the Software Requirements Specification for the FlightsCompare application. The purpose of this document is to outline the requirements for a flight comparison platform that scrapes flight data from various websites, validates the data, and displays it for users to compare prices across different sources.

1.1 Product Scope

FlightsCompare is designed to assist users in finding the best available flight deals by scraping real-time flight data from multiple sources, validating the data for accuracy, and presenting it in an easy-to-use interface for comparison. The primary objectives are:

- Provide users with up-to-date, accurate flight price comparisons from various websites.
- Streamline the process of finding affordable flights by centralizing data.
- Enhance user experience by offering customizable search filters (e.g. airline, layovers, price range).

1.2 Product Value

FlightsCompare provides significant value by saving users time and effort in searching multiple websites for flight deals. With a single query, users can access real-time validated flight data and compare prices, ensuring they get the most accurate and up-to-date information. The platform benefits frequent travelers, budget-conscious users, and anyone looking to optimize their travel expenses.

1.3 Intended Audience

The intended audience for this document includes:

- Project managers and stakeholders responsible for the development and delivery of FlightsCompare.
- Development teams involved in scraping, validation, and user interface components.
- Testers and quality assurance teams who will ensure that the product meets the specified requirements.
- End users who want a reliable and centralized platform for comparing flight prices.

1.4 Intended Use

FlightsCompare will be used by individuals searching for flights across various online platforms. The tool will be beneficial for:

- Budget travelers looking for the best deals across different airlines and booking platforms.
- Frequent flyers who want to streamline their flight search process.
- Travel agencies looking to provide their customers with comprehensive price comparisons.

1.5 General Description

FlightsCompare operates by scraping data from multiple flight booking websites using a Python-based scraper. The data is temporarily stored in a "collector database," validated through a Java-based validation module to ensure accuracy, and then transferred to a final database. The validated flight data is presented to users through a C# application with a user-friendly DevExpress-based frontend. The system provides flight details such as airline, price, duration, layovers, and departure/arrival times, with options for filtering and sorting results based on user preferences.

2 Overall Description

2.1 User Needs

Users need a platform that provides up-to-date and accurate flight information in one place, eliminating the need to visit multiple websites. They need:

- Access to real-time flight prices from multiple sources.
- Ability to filter and sort flight data based on personal preferences such as price, duration, layovers, and departure/arrival time.
- A user-friendly interface to facilitate easy comparison of flights.

2.2 Assumptions and Dependencies

- It is assumed that the websites and APIs from which flight data is scraped remain accessible and allow scraping, or provide an API for data retrieval.
- The system is dependent on reliable internet access for both data scraping and real-time updates.
- External data sources may have rate limits or terms of service that could impact the frequency or volume of data scraping.
- Time zone differences between departure and arrival locations must be managed to display consistent flight times.

3 System Features and Requirements

3.1 System Features

- **Flight Data Scraping**: Scrape flight information from multiple platforms using Python.
- **Data Validation**: Use Java to validate scraped flight data (check for canceled flights, correct prices, and verify availability).
- **Search and Filter Functionality**: Allow users to filter flights based on airline, price, layovers, departure and arrival times.
- **Sorting Options**: Sort flights by price, duration, and other user-specified parameters.
- **Real-time Data Updates**: Provide real-time updates of flight prices and availability as new data is scraped and validated.

3.2 Functional Requirements

The functional requirements describe the key features and behaviors the system must provide to ensure that it works as expected. These features cover the entire lifecycle of data handling, from scraping to user interaction and display.

3.2.1 Data Scraping

- The system must scrape flight data from at least 2 different flight booking websites (e.g., Expedia, Skyscanner, Kayak, Google Flights, and others).
- The scraping process must be initiated automatically at regular intervals or triggered manually by an administrator.
- The system must extract essential flight details, including departure and arrival times, flight duration, airline, layovers, and ticket prices.
- The scraper should handle pagination and dynamically loaded content (e.g., websites using JavaScript to load more data).

- Scraped data must be stored temporarily in the "collector database" for validation.
- The system should log any scraping errors or issues (e.g., website unavailability, changes in webpage structure) for debugging and monitoring purposes.

3.2.2 Data Validation

- The system must check that the scraped flight information is valid (e.g., the flight has not already passed, the flight was not canceled, the data is within acceptable price ranges).
- Invalid data (e.g., old, duplicate, incomplete, or canceled flights) must be removed or archived from the final database.

3.2.3 Data Storage and Management

- Validated data must be transferred from the temporary "collector database" to the final database, where it is ready for user display.
- The final database must store flight data in a structured and normalized format to avoid duplication and redundancy.
- Data from the collector database must be wiped after each scraping session to ensure that only fresh data is validated and passed to the final database.
- The system should maintain historical data (e.g., previous flight prices) in the final database for future analytics and comparisons.
- Database backups must be performed regularly to prevent data loss.

3.2.4 Search and Comparison Features

- Users must be able to search for flights based on a variety of criteria, including:
  - Departure and destination cities or airports.
  - Departure and return dates.
  - Number of passengers and flight class.
  - Airline preferences.
- Users should be able to filter search results by price, number of layovers, flight duration, and airline.
- Search results must be displayed in a user-friendly format, allowing users to easily compare flights across different websites.
- The system must show the website link or booking option for each flight, enabling users to proceed directly to the booking platform.
- The system must highlight any price discrepancies between different sources for the same flight, if applicable.

3.2.5 User Account Management

- Users should be able to create an account using an email and password or log in via third-party authentication (e.g., Google or Facebook).
- Registered users must be able to save search preferences and flight alerts for specific routes or airlines.
- User data, such as search history and saved flights, should be securely stored and retrievable during subsequent sessions.

3.2.6 Administrator Functions

- Administrators must be able to start/stop scrapers, and view logs of scraping errors.
- Admins should have access to logs of validation processes, including the ability to review and correct any validation failures.
- Administrators must have control over user management, including viewing user accounts, tracking activity, and disabling or banning accounts if necessary.

3.2.7 External Interfaces

- The system must integrate with third-party APIs (e.g., airport information services) to enhance flight details.
- The scraping module should be adaptable to web scraping and API integration, ensuring data is collected effectively regardless of the source format.

3.3 Nonfunctional Requirements

The nonfunctional requirements describe the qualities the system must have, ensuring it meets performance, usability, security, and scalability standards. These requirements are essential for the user experience and system reliability.

3.3.1 Performance

- The system should scrape data from at least 2 major flight booking websites within 1 hour.
- Data validation in the Java module should complete within 5 minutes after the data is scraped.
- The C# frontend should display results within 5 seconds after the user submits a search query.

3.3.2 Scalability

- The system should be able to scale to support an increasing number of users, websites, and flight data sources without requiring significant reengineering.
- It should support adding new scraping sources (new flight booking sites) with minimal configuration changes.

3.3.3 Security

- User data, including search preferences and settings, must be stored securely in compliance with relevant data protection laws (e.g. GDPR).
- All communication between the frontend and backend should be encrypted using SSL/TLS to prevent data interception.
- The system should be resistant to common security threats, such as SQL injection and cross-site scripting (XSS).
- Access to backend administrative controls should be restricted to authorized personnel only and protected.

3.3.4 Usability

- The user interface (UI) should be intuitive, requiring no more than 5 clicks to complete a flight search and compare results.
- Users should be able to navigate the platform easily with minimal learning, even if they are unfamiliar with the system.
- The system should provide informative error messages and guidance if the user inputs invalid data or if no results are found.
- The frontend must be responsive and adapt to various screen sizes (e.g. desktop devices).

3.3.5 Reliability

- The system should have an uptime of at least 95%, ensuring minimal downtime and high availability for users.
- In case of a failure during the scraping process, the system should automatically retry scraping.
- The system should gracefully handle incomplete or invalid flight data and continue functioning without crashing.
- Backups of the final database must be made at regular intervals (e.g., daily) to ensure that validated data is never lost.

3.3.6 Maintainability

- The system should support automated testing to ensure new updates do not introduce regressions or bugs.
- Any errors or issues within the system should be logged with sufficient details, allowing developers to debug quickly.
- The system should be easy to update, with a well-defined deployment process for pushing updates to the live environment.

3.3.7 Compatibility

- The C# frontend must be compatible with major browsers such as Google Chrome, Mozilla Firefox, Microsoft Edge, and Safari.
- The system should be compatible with modern operating systems, including Windows, macOS, and Linux for the backend server.
- The system should be capable of integrating with external APIs or services to pull in additional data if needed (e.g. airport information).

3.3.8 Data Integrity

- Data stored in the final database must be consistent and accurate. Any discrepancies between the scraped data and displayed data must be eliminated during the validation process.
- If a validation rule flags invalid data (e.g. a flight no longer exists), that data must not be passed to the user-facing application.
- Any partial or incomplete data records should be removed from the collector database during the validation phase to avoid corrupting the final database.