Software Requirements Specification for FlightsCompare

Team Project

November 9, 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. Besides the validation module, the Java application should include a REST API that includes a custom authentication server (based on JSON Web Tokens), which allows for login via credentials and OAuth2 tokens. The whole application will be deployed via Docker-Compose.

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 Security Enhancements

- Authentication Server with JWT Management: The system will implement a custom authentication server responsible for generating and managing JSON Web Tokens (JWT).
  - The authentication server will issue JWT tokens based on user credentials as well as third-party OAuth2 tokens, allowing for integration with external authentication providers.
  - Tokens generated by the authentication server must be verifiable across all resource servers to ensure secure access to restricted resources.

  - The authentication server will support token invalidation to ensure that compromised tokens can be immediately revoked.
  - All resource servers will validate JWTs against the central authentication server to confirm the token's authenticity and user permissions before granting access.

- Token Expiration and Refresh: JWTs will have defined expiration times, requiring users to refresh their tokens periodically to maintain secure sessions.
- Role-Based Access Control (RBAC): The authentication server will also support role-based access control, allowing different levels of access permissions depending on user roles.

3.3.5 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).

3.3.6 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.

3.3.7 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.

3.3.8 Deployment and Containerization

- Docker Compose for Deployment: The final application will be deployed as a series of Docker containers managed via Docker Compose, allowing for easy orchestration of the different components (e.g., scraper, validation server, database, frontend).
  - Docker Compose will facilitate containerized deployment, enabling consistent and scalable environment setups across development, testing, and production.
  - Environment variables will be used within Docker Compose to securely manage sensitive configurations, such as database connections and authentication server settings.
- Automated Scaling: The system will be designed to scale containers up or down based on user demand, which Docker Compose will handle by supporting parallel deployments of critical services like the scraper and database.

3.3.9 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).

3.3.10 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.

3.3.11 Request Management

- Rate Limiting: To protect against abuse and ensure fair use of resources, the system will implement rate limiting on incoming user requests.
  - Rate limits will be set per user to prevent excessive requests, improving overall system stability.
  - Different rate limits may be applied to specific resources, such as the scraping process, to ensure the backend remains performant and to protect against accidental or intentional overload.
- Caching Mechanism: The system will include caching for frequently accessed data to reduce redundant requests and improve response times.

User Stories for FlightsCompare Application

Introduction

This document outlines the user stories for the FlightsCompare application, focusing on user needs for searching, filtering, and managing flight data, as well as admin functionalities for maintaining system reliability and security.

User Stories

Story 1: As a user, I want to search flights by departure and arrival cities so I can easily find flights to my destination.

Story 2: As a user, I want to filter flights by price range so I can view flights within my budget.

Story 3: As a user, I want to sort flight results by duration to quickly find the shortest flights.

Story 4: As a user, I want to see all layover details for each flight so I can assess the convenience of each option.

Story 5: As a frequent flyer, I want to save specific searches so I can easily re-run them without re-entering details.

Story 6: As a budget traveler, I want to view price differences between multiple sources for the same flight to make the most cost-effective choice.

Story 7: As an admin, I want to manage user accounts and track activity to ensure appropriate use of the application.

Story 8: As a user, I want to save my search preferences so that my filters and settings are applied each time I log in.

Story 9: As a user, I want to sort flights by airline so I can see options only from my preferred carriers.

Story 10: As an admin, I want to receive logs of scraping errors so I can troubleshoot data extraction issues quickly.

Story 11: As an admin, I want to manually trigger or stop scrapers to manage data extraction based on specific needs.

Story 12: As a user, I want secure account authentication via email or third-party options to keep my information protected.

Story 13: As a user, I want the app to provide direct links to booking sites so I can proceed to purchase with ease.

Story 14: As a developer, I want automated tests for each system feature to ensure updates do not break functionality.

Story 15: As a user, I want to receive meaningful error messages if no results are found so I understand how to refine my search.

Story 16: As an admin, I want the platform to handle data validation automatically, removing expired or incorrect flights to maintain data accuracy.

INVEST-Optimized User Stories for FlightsCompare Application

Introduction

This document refines the user stories for the FlightsCompare application using the INVEST criteria to ensure each story is independent, valuable, and testable. Priorities are assigned based on user impact and system functionality.

User Stories with INVEST Criteria

Story 1: **(P1)** As a user, I want to search for flights by selecting departure and arrival cities from a dropdown list, so I can find flights to specific destinations quickly and with minimal input.
Justification: Specifying dropdowns instead of open input fields enhances usability and testability, ensuring users select valid locations. This feature is essential to enable the core function of finding flights, hence given P1.

Story 2: **(P1)** As a user, I want to filter flights by entering a custom price range so I can view flights that fit my budget constraints.
Justification: Allows greater flexibility in filtering options, ensuring the system is testable for both high and low price limits. A high priority as it's a critical feature for budget-conscious users.

Story 3: **(P2)** As a user, I want to sort flight results by duration in ascending or descending order, so I can quickly find the shortest or longest flights based on my preference.
Justification: Making the duration sorting bidirectional increases user control and simplifies testing. This feature enhances user experience but does not prevent core functionality, so it's assigned P2.

Story 4: **(P2)** As a user, I want to view all layover details, including location, duration, and airline, for each flight option, so I can better assess the convenience of each option.
Justification: Adding specific layover details makes the story more valuable and ensures that relevant flight data is easy to test. Priority is P2 as it enhances but does not affect critical functionality.

Story 5: **(P3)** As a frequent flyer, I want to save specific flight search criteria under a name I choose, so I can quickly re-run saved searches without re-entering details.
Justification: Clear specification of saved search details ensures the feature is testable and reusable. Lower priority since it serves returning users primarily, assigned P3.

Story 6: **(P1)** As a budget traveler, I want to view the lowest available prices from each source in a single list view, so I can select the most cost-effective option at a glance.
Justification: Making the price comparison accessible in a single list simplifies the task of comparing costs and makes this valuable feature easily testable. Assigned P1, as it's crucial for budget-focused users.

Story 7: **(P3)** As an admin, I want to view user activity logs and account status, so I can ensure proper usage of the application and troubleshoot any user-related issues.
Justification: Improved clarity with a focus on two essential admin tasks: monitoring activity and ensuring compliance, making it more testable. Given P3 since it impacts administration rather than end-user functionality.

Story 8: **(P2)** As a registered user, I want my search preferences (e.g., preferred airline, layover options) to be saved automatically so that they're applied each time I log in, saving me time.
Justification: Specifying auto-saving of preferences improves value and testability. Prioritized as P2, as it streamlines the experience for regular users.

Story 9: **(P1)** As a user, I want to sort flights by airline, so I can quickly find flights that match my preferred carriers.
Justification: This sorting option is essential for brand-loyal travelers. Testable and prioritized as P1 to accommodate users with airline preferences.

Story 10: **(P2)** As an admin, I want to receive logs of scraping errors that include timestamps and error types, so I can troubleshoot data extraction issues effectively.
Justification: Providing specific log details ensures the story is testable and valuable for error tracking. Assigned P2, as it impacts backend reliability.

Story 11: **(P2)** As an admin, I want the ability to manually start, pause, or stop data scrapers so that I can control data extraction based on system needs.
Justification: Improves independence and testability by specifying scraper controls. Assigned P2 since it's essential for controlled scraping but not core to user experience.

Story 12: **(P1)** As a user, I want to log in securely using an email and password or via a third-party provider (e.g., Google or Facebook) to ensure my account is protected.
Justification: Adding security and third-party login options makes it independent and testable, and is crucial for security, hence P1.

Story 13: **(P1)** As a user, I want direct links to booking sites from the search results, so I can book flights easily without extra steps.
Justification: A highly valuable feature that makes it testable by ensuring a seamless path to booking. Assigned P1 as it facilitates the transition from search to purchase.

Story 14: **(P2)** As a developer, I want automated tests for each system feature, so I can ensure updates do not introduce new bugs.
Justification: Specifying automated testing increases value and testability. This is crucial for maintaining system stability, hence P2.

Story 15: **(P1)** As a user, I want to receive a meaningful error message if no results are found, so I understand how to refine my search.
Justification: Improves user guidance, making the feature valuable and testable. Priority P1 due to its role in guiding users when they encounter no search results.

Story 16: **(P2)** As an admin, I want invalid or outdated flight data to be flagged automatically during validation so that only accurate information reaches users.
Justification: Automatic data validation improves testability and reliability, ensuring users receive up-to-date data. P2 priority, as it affects system accuracy.

Priority Justification

- **P1 - High Priority:** These stories cover essential functionalities directly affecting user experience, such as flight search, filtering, and secure login. They are crucial for achieving the application's primary goal, which is to provide an effective, user-friendly, and secure flight comparison tool. Stories related to critical user interactions and foundational security were assigned P1 to ensure they are implemented early in the development process.

- **P2 - Medium Priority:** These stories add significant value to the application but do not impact core functionality. They generally enhance user experience or support admin functionalities that improve reliability, such as scraping error logs and sorting features. While not immediately essential, they contribute to a more refined, user-friendly experience, so they are prioritized once core functionality is in place.

- **P3 - Low Priority:** Stories with P3 priority are supportive or optional features that primarily serve specific user needs, like saved searches or detailed activity logs for admins. These features provide additional convenience or monitoring capabilities but do not directly impact the primary use case of the application. As such, they are planned for later sprints or phases, allowing resources to be focused on core and high-value features first.

Story Points for FlightsCompare User Stories

Introduction

This document assigns story points to the FlightsCompare user stories, with justifications based on anticipated complexity, effort, and uncertainty in implementation.

User Stories with Story Points

Story 1: **(P1, 3 points)** As a user, I want to search for flights by selecting departure and arrival cities from a dropdown list, so I can find flights to specific destinations quickly and with minimal input.
Justification: Moderate complexity as it requires setting up a searchable dropdown list populated with a comprehensive list of cities, ensuring a reliable experience for the user. The task has some effort and dependencies, hence assigned 3 points.

Story 2: **(P1, 5 points)** As a user, I want to filter flights by entering a custom price range so I can view flights that fit my budget constraints.
Justification: Price filtering requires input validation, integration with the data processing module, and efficient filtering. Moderate complexity and interdependencies justify 5 points.

Story 3: **(P2, 2 points)** As a user, I want to sort flight results by duration in ascending or descending order, so I can quickly find the shortest or longest flights based on my preference.
Justification: Sorting is straightforward and primarily a UI change with minor backend support. Since it's a low-complexity task, it's assigned 2 points.

Story 4: **(P2, 3 points)** As a user, I want to view all layover details, including location, duration, and airline, for each flight option, so I can better assess the convenience of each option.
Justification: Retrieving and displaying layover details requires minor data handling and front-end layout adjustments, with a moderate effort. This warrants a 3-point assignment.

Story 5: **(P3, 8 points)** As a frequent flyer, I want to save specific flight search criteria under a name I choose, so I can quickly re-run saved searches without re-entering details.
Justification: Saving custom searches requires backend storage, data persistence, and UI elements to recall these searches, adding moderate complexity and interdependencies. Thus, it's given 8 points.

Story 6: **(P1, 5 points)** As a budget traveler, I want to view the lowest available prices from each source in a single list view, so I can select the most cost-effective option at a glance.
Justification: Requires aggregating and displaying the lowest prices per source, which involves backend filtering logic and front-end formatting. It's a moderate effort, so it's assigned 5 points.

Story 7: **(P3, 5 points)** As an admin, I want to view user activity logs and account status, so I can ensure proper usage of the application and troubleshoot any user-related issues.
Justification: Implementing logs involves capturing user activity and displaying it in a secure admin view, with some effort in backend tracking and access controls. Given 5 points for its moderate complexity.

Story 8: **(P2, 3 points)** As a registered user, I want my search preferences (e.g., preferred airline, layover options) to be saved automatically so that they're applied each time I log in, saving me time.
Justification: Saving preferences involves minor backend support for data storage and retrieval, with simple UI adjustments. Low-to-moderate effort results in 3 points.

Story 9: **(P1, 2 points)** As a user, I want to sort flights by airline, so I can quickly find flights that match my preferred carriers.
Justification: Simple sorting logic, primarily front-end work with minor backend modifications. Assigned 2 points for minimal complexity and effort.

Story 10: **(P2, 5 points)** As an admin, I want to receive logs of scraping errors that include timestamps and error types, so I can troubleshoot data extraction issues effectively.
Justification: Error logging requires robust backend support for capturing and storing error details, with moderate complexity in formatting the data for admin access. Given 5 points.

Story 11: **(P2, 8 points)** As an admin, I want the ability to manually start, pause, or stop data scrapers so that I can control data extraction based on system needs.
Justification: This feature requires substantial backend control logic and a reliable admin interface, with high complexity in controlling processes dynamically. It's assigned 8 points.

Story 12: **(P1, 8 points)** As a user, I want to log in securely using an email and password or via a third-party provider (e.g., Google or Facebook) to ensure my account is protected.
Justification: Security and integration with third-party authentication increase complexity. Implementing OAuth requires substantial backend work and careful testing, hence 8 points.

Story 13: **(P1, 3 points)** As a user, I want direct links to booking sites from the search results, so I can book flights easily without extra steps.
Justification: Adding booking links to the UI is straightforward, with minimal backend integration. It's assigned 3 points due to low complexity and effort.

Story 14: **(P2, 3 points)** As a developer, I want automated tests for each system feature, so I can ensure updates do not introduce new bugs.
Justification: Adding automated tests for the existing features requires moderate effort and impacts codebase stability. This straightforward task is given 3 points.

Story 15: **(P1, 2 points)** As a user, I want to receive a meaningful error message if no results are found, so I understand how to refine my search.
Justification: Basic error messaging with contextual help is simple to implement, with low complexity and minimal backend changes. Assigned 2 points.

Story 16: **(P2, 8 points)** As an admin, I want invalid or outdated flight data to be flagged automatically during validation so that only accurate information reaches users.
Justification: Requires a validation module with rules to identify outdated data, with substantial backend logic and moderate testing. Given 8 points for moderate complexity and high impact on data quality.

Summary of Story Point Justifications

Story points were assigned to each user story using planning poker, considering complexity, effort, and uncertainty associated with each story. Points range from 2 (simple) to 8 (more complex) as follows:

- **Low Complexity (2-3 Points):** Stories with limited dependencies and straightforward implementation requirements, such as sorting flights by airline or adding basic error messages, were given 2-3 points. These tasks generally require minimal backend work and simple UI modifications, making them easily achievable within a sprint.
- **Moderate Complexity (5 Points):** Stories that involve more integration with backend data handling or data manipulation, such as filtering by price range or viewing scraping error logs, were assigned 5 points. These

tasks include moderate levels of effort, some backend integration, and dependency management, making them slightly more complex but still manageable.

- **Higher Complexity (8 Points):** Stories that require substantial backend or security work, or new modules, such as implementing secure third-party login, enabling manual control of scrapers, and performing automated data validation, were assigned 8 points. These stories generally involve complex logic, integrations, and extensive testing, making them the most complex and time-consuming stories in this set.