INDY-3 – Flight Data

Final Report

CS 4850 Section 1 – Fall 2024

Sharon Perry - November 12, 2024

Team Members:

Jackson Fuller

Vance Peterson

Michael Pietrzak

Kelvien Pyryt

Website: https://indy-03-flight-data.github.io/

GitHub: https://github.com/Indy-03-Flight-Data/nwkraft-web-app/tree/main

Lines of Code: 1,155

Number of Components: 10

Total Man Hours: 185

1.0 Overview

The NWKRAFT flight data app is a software solution designed to provide pilots and aviation enthusiasts with easy access to NWKRAFT flight checklists and airport information. This application will offer a dashboard where users can search for airports, view relevant flight checklists, and mark their favorite airports for quicker access. The app will be available on both web and Android mobile platforms.

1.1 Project Goals

1.1.1 User-friendly Dashboard

Objective: Create a clear, organized dashboard that allows access to all available features, including airport search, checklist data, and favorite airports.

Design Considerations: The dashboard should be intuitive and easy to use, allowing for simple navigation to key features with few clicks and clean data presentation.

1.1.2 Search Functionality

Objective: Provide efficient searches to allow users to find specific airports by name, ICAO code, or location quickly.

Expected Features: Autocomplete suggestions, filtering, and sorting options to help users find the desired information with minimal delay.

1.1.3 Favorites Feature

Objective: Enable users to mark and save frequently accessed airports as favorites for repeated access.

Expected Features: A “Favorites” page where users can view and access their saved airports. This feature should improve the user experience by making data readily available.

1.1.4 Cross-platform Compatibility

Objective: Ensure that the application is accessible across different device types, specifically web browsers for desktop and mobile, as well as an Android app.

Considerations: The user interface should adjust to different screen sizes and orientations, ensuring functionality and usability across multiple platforms.

1.1.5 Scalable Architecture

Objective: Build a scalable backend to support future growth in both user count and data volume.

Implementation: Cloud-based solutions (e.g., Heroku-hosted PostgreSQL) to allow easy scaling and maintain application efficiency as demand increases.

1.2 Definitions and Acronyms

NWKRAFT: A checklist containing key flight planning data required by pilots, covering NOTAMs, Weather, Known ATC delays, Runway lengths, Alternates, Fuel requirements, and Takeoff/Landing information.

API: Application Programming Interface, used to retrieve flight checklist data and airport information from NWKRAFT and FAA sources.

UI: User Interface, the graphical layout through which users interact with the app.

GIT: Version control system, used to track changes to source code and facilitate collaborative development.

JSON: JavaScript Object Notation, a lightweight data format used for data exchange between frontend and backend services.

1.3 Assumptions

Internet Connectivity:

Users will require an internet connection to access the application and retrieve data, as the app relies on APIs (e.g., NWKRAFT and FAA) to fetch data.

API Reliability:

The NWKRAFT API and other external data sources (e.g., FAA API for NOTAMs) will provide reliable data for the application. Downtime or unreliability in these services could affect the availability of up-to-date information.

Data Privacy and Compliance:
The application will ensure that users' information is securely handled.

User Familiarity with Aviation Terms:
It is assumed that users will have a basic understanding of aviation-related terms, such as ICAO/IATA codes and NOTAMs. This is due to the expected audience (pilots and aviation enthusiasts).

Cross-Platform Performance:
The app is expected to perform consistently across both the web and Android platforms, with little-to-no need for specialized knowledge from users to operate it on different devices.

2.0 Functional Requirements

2.1 User Authentication

Account Creation:

Registration Form: Users must be able to create an account by filling out a registration form that includes fields for username and password.

Validation: Enforce input validation to ensure usernames are unique and passwords meet the set requirements (e.g., minimum length, inclusion of numbers and symbols).

Error Handling: If an error occurs during registration (e.g., username already taken), the system will display an error message.

Login and Logout:

Username and Password Login: Users can log in with their registered username and password through the login page.

Session Management: After logging in, users will stay logged in through a secure session until they log out or their session times out due to inactivity.

Logout Option: Users will have the option to log out from any page, terminating their session.

2.2 Airport Search Functionality

Search by Airport Code or Name:

Input Fields: Allow users to search for airports by entering the ICAO code, IATA code, or airport name.

Autocomplete Suggestions: As the user types, the search bar will display autocomplete suggestions using dynamic queries to the backend, helping users find airports faster and with fewer errors.

Display Search Results:

Result List: Present search results in a list format with basic information for each airport, such as name, location, and code.

Error Handling:

Invalid Search Terms: Display clear error messages for invalid input or if no results are found.

2.3 Checklist Data Retrieval

View NWKRAFT Checklists:

Checklist Access: After selecting an airport from the search results, users can view the NWKRAFT checklist associated with that airport.

Information Structure: Checklists will be organized by categories (e.g., NOTAMs, runway conditions), and displayed in an expanded format for easy navigation.

User Selection of Checklist Items:

Customization Options: Users can select specific checklist items they want to view, such as fuel requirements or weather conditions through checkboxes.

User-Friendly Display: Present the checklist data in a readable and organized format, with icons or colors to highlight important items.

Error Handling:

Data Unavailability: If checklist data is unavailable or the API call fails, display a message explaining the issue and suggesting alternative actions.

2.4 Favorites Feature

Add and Remove Favorite Airports:

Favorites Button: Each airport search result or checklist page will have a button to add or remove the airport from the user's favorites list.

Instant Feedback: Provide immediate visual feedback (e.g., a filled star icon or confirmation message) to indicate that the airport has been added/removed from favorites.

Manage Favorites List:

Favorites Page: The app will include a "Favorites" section where users can view, organize, and access their saved airports.

Data Persistence:

Storage: User favorites are saved in the PostgreSQL database, ensuring that favorites remain across sessions and devices.

2.5 Responsive Design

Web Application Responsiveness:

Device Compatibility: The web app should be fully responsive, adjusting layout and elements to provide a smooth user experience on desktops, tablets, and mobile browsers.

Dynamic Layouts: Use flexible grid systems and media queries to ensure content remains easy to read on all screen sizes.

Mobile Application Responsiveness:

Screen Size Compatibility: Ensure the Android app functions well on devices with different screen resolutions, orientations, and sizes.

Touch Interaction Optimization: Design for touch interactions, including larger buttons and minimized text input to improve user experience on mobile devices.

3.0 Non-Functional Requirements

3.1 Performance

Concurrent User Support: The application must support at least 1,000 concurrent users, ensuring smooth functionality under heavy usage without lagging or crashing.

Response Time:

Search and Data Retrieval: All search queries and data retrieval actions, including NOTAM and checklist fetches, should be completed within 2 seconds or less.

Page Load Times: The launch page and other essential pages (login, dashboard, etc.) should load within 1-2 seconds with a basic internet connection.

Data Processing Efficiency: The backend should handle and process multiple API calls efficiently. This includes optimizing queries to the PostgreSQL database hosted on Heroku.

Server-Side Caching: Implement server-side caching for frequently accessed data (e.g., airport and NOTAM information) to reduce the load on the NWKRAFT API and improve response times for users.

3.2 Reliability

Uptime Requirements: The app should maintain a minimum uptime of 99.5%, aiming for continuous availability to ensure reliability for users who depend on it for flight planning.

Data Consistency:

Synchronization: Ensure data across the web and mobile applications remains consistent, especially for user accounts and favorites.

Database Integrity: Use transaction management within the PostgreSQL database to avoid data corruption, specifically for critical operations such as user registration.

Monitoring and Logging:

Performance Monitoring: Use monitoring tools (Heroku's monitoring services) to track system health, uptime, and performance metrics.

Error Logging: Log errors and performance issues to diagnose and resolve potential system bottlenecks and/or recurring issues.

3.3 Portability

Platform Compatibility:

Web App: Ensure the web app functions seamlessly across different operating systems and devices, including desktops, tablets, and mobile browsers.

Mobile App: Designed to run on Android OS 10 and above, adhering to Android compatibility guidelines to ensure it functions on various Android devices.

Deployment Flexibility:

Cloud Platform: The app is hosted on Heroku, making it easily transferable to other cloud platforms like Google Cloud or Azure if needed.

Reconfiguration Minimality: Migration to another hosting platform should require minimal reconfiguration, ensuring portability of the backend and database setup with minimal downtime.

3.4 Usability

Intuitive User Interface:

Design Consistency: Ensure consistency in layout, fonts, colors, and icons across both the web and mobile interfaces for a similar user experience.

Ease of Navigation: Clear menus will help users navigate between features such as airport search, favorites, and NOTAM details.

User Assistance and Documentation:

Error Messages: Informative error messages will be used to guide users in case of input errors, login failures, or invalid search queries.

3.5 Security

Data Protection and Encryption:

Data in Transit: All data transfers, including API calls, will be secured with HTTPS to prevent interception and ensure data integrity.

Data at Rest: Sensitive data, such as passwords, will be encrypted and stored securely in the database.

Authentication and Authorization:

User Login and Session Management: Use NextAuth.js with JSON Web Tokens for secure session management. Users should be automatically logged out after a period of inactivity.

Access Control:

Admin and User Roles: If admin roles are added in the future, implement role-based access control to manage access to certain features.

4.0 System Features

4.1 Launch Page

4.1.1 Login Button:

Functionality: Provides a clear entry point for returning users to access their accounts. The login button will redirect users to the login page where they can log in with their username and password.
Positioning and Design: Positioned clearly on the launch page making it visible and accessible. It should be styled in alignment with the app's overall theme, using colors and icons to indicate its purpose.

Feedback: Includes hover effects and click animations to confirm responsiveness.

4.1.2 Register Button:

Functionality: Allows users to create an account by redirecting them to the registration page, where they can input the necessary credentials.

Design and Positioning: Like the login button, the register button will be clearly displayed and distinct to encourage new users to join.

Access Control: Ensures that unauthenticated users are restricted from accessing other parts of the app without registering.

4.1.3 Display App Name and Purpose:

Purpose: Clearly displays the app's name and a brief description to communicate the app's goal immediately.

Design Elements: Styled with a bold font and positioned at the top or center of the page for easy visibility.

Reinforcement: Gives a professional appearance and briefly explains the app's main function, ensuring users understand the app's purpose from the start.

4.2 Registration Page

4.2.1 Enter Username:

Input Field: A dedicated field where users can create a unique username for their account.

Validation: Enforces rules such as minimum/maximum character limits to maintain consistency.

Error Handling: Provides instant feedback for errors, such as "Username already taken" or "Invalid format."

4.2.2 Enter Password:

Input Field: A password field with input masking for security.

Validation: Enforces password strength criteria (e.g., minimum length, use of special characters, numbers).

4.3 Login Page

4.3.1 Login with Username:

Input Field: Users will input their previously registered username into the correct field.

Error Handling: Displays appropriate error messages for unregistered usernames or misspellings.

4.3.2 Enter Password:

Secure Field: Password input with masking for user privacy.

Remember Me Option: Users will stay logged in, even after leaving the site, so long as the JSON Web Token remains valid.

4.4 NOTAM Search Page

4.4.1 Input Airport Codes:

Autocomplete Feature: Offers autocomplete suggestions based on user input, using a dynamic search API to improve ease of use.

Validation: Validates airport codes in real-time to ensure the user's entry matches recognized ICAO/IATA codes.

4.4.2 Prompt User for Required NOTAM Information:

Options for Data Selection: Users can specify what NOTAM information they wish to view, such as runway conditions, weather, etc. via checkboxes.

4.4.3 Display Detailed NOTAM Information:

Results Display: Shows NOTAM information in a structured and readable format, including details such as event type, location, and duration.
Real-Time Updates: Information is fetched dynamically from the FAA API to ensure all displayed NOTAMs are current.

User Interaction: Users can navigate between different NOTAMs or save important data to favorites.

4.4.4 Display Error Message for Invalid Airport Codes:

Real-Time Validation: Validates input airport codes and displays error messages for invalid codes immediately.

Suggestions: Suggests corrections or similar airports based on partial matches for mistyped codes.

5.0 External Interface Requirements

5.1 API Interface

Data Source: The NWKRAFT API will be the primary data source for airport and checklist information.

Endpoints:

Authentication: Endpoint to validate user credentials for logging in and managing sessions.

Airport Search: Endpoint for querying airport information by ICAO codes, names, or other criteria.

Favorites Management: Endpoints for adding, retrieving, or removing airports from the user's favorites.

NOTAMs and Checklist Retrieval: Dedicated endpoints to fetch NOTAMs and NWKRAFT checklist data for airports.

Security Protocols: All API requests will be made via secure HTTPS connections to ensure encrypted data transmission.

Data Format: JSON will be used for data exchange to maintain consistency between the frontend and backend.

Rate Limiting: The application should remain below the rate limits imposed by NWKRAFT's API to avoid disruptions.

5.2 User Interface

Web Application:

Responsive Design: The interface will adapt to various screen sizes and resolutions, enabling use on desktop, tablet, and mobile devices.

Framework: Developed with Next.js for a fast, interactive user experience. Styling will rely on a combination of CSS and other libraries to maintain a professional look.

Android Mobile Application:

Compatibility: Designed to function on Android OS to provide support for current devices.

Interaction Design: Includes mobile-optimized features like autocomplete for search and swiping for navigation where it is useful.

Data Synchronization: Utilizes API endpoints for data syncing between the mobile app and backend services to keep airport data, checklists, and favorites up to date.

5.3 Cloud Services

Hosting:

Platform: Heroku will serve as the content delivery network to ensure fast loading times and scalability, allowing the app to accommodate a large user base and reducing latency.

Data Storage:

Database Choice: PostgreSQL on Heroku will manage the backend data, including airport information, user accounts, session data, and favorites.

Scalability: The database setup on Heroku is scalable, enabling storage capacity to increase in line with growing user data.

Data Security:

Authentication and Access Control: User data, including account information and personal preferences, will be secured via strict access controls, using JSON Web Tokens for session management.

Encryption: Sensitive data will be encrypted both in transit and at rest to ensure data security.

6.0 Analysis

6.1 Environment

6.1.1 Development Environment

Programming Languages and Frameworks:

Frontend: The web app will be developed using HTML, CSS, and JavaScript frameworks, primarily Next.js and React for dynamic content and the UI.

Mobile App: The Android app will be developed using Java/Kotlin.

Development Tools:

IDE: Development will be performed in Visual Studio Code for the web application and Android Studio for the mobile app. Both provide excellent environments for coding, debugging, and version control.

Version Control: GitHub will be used for collaborative development and version control, allowing team members to work simultaneously.

API Testing: Tools such as Postman or Swagger will be used for API testing to ensure data retrieval is consistent and accurate.

6.1.2 Browser Compatibility

Supported Browsers:

The web application must be compatible with major browsers, including **Chrome, Firefox, Safari, and Edge**.

Browser-Specific Testing:

Ensure that key features such as search, authentication, and data display are consistent across all supported browsers.

Responsiveness:

The design should adjust to different screen resolutions and orientations, guaranteeing accessibility for users on various devices.

Feature Testing:

Test critical features (e.g., search, authentication, and checklists) across browsers to ensure consistent results.

6.2 User Characteristics

6.2.1 End Users

Target Audience: The primary users of the NWKRAFT flight data app are pilots and aviation enthusiasts who require reliable access to flight planning information.

Technical Skill Level: Users will range from technically proficient to less experienced, so the application must be intuitive with a straightforward interface and clear instructions.

User Goals:

Accessing Flight Data: Users will access airport information, checklists, and NOTAMs to support safe and efficient flight planning.

Customization and Personalization: Users may save frequently accessed airports in their favorites for fast retrieval.

6.2.2 User Roles

General Users:

Functionality Access: General users can access all standard functionalities, including airport search and managing their favorites.

Data Privacy: General users will have access to data relevant to their account and usage history, but no permission to see other users' data.

Future Admin Roles (if applicable):

Content Management: In the future, the app may contain admin roles for content management and user support.

Access Control: Admin roles will require role-based access controls to restrict certain actions, ensuring general users cannot access admin functionalities.

6.3 System

6.3.1 Performance:

Efficiency: The application should handle simultaneous requests efficiently, ensuring that search queries, checklist retrievals, and other operations are performed within an acceptable time limit.

Response Time:

API Requests: Response times for API calls, such as those for airport search and NOTAM retrieval, should be under 2 seconds to avoid disrupting user experience.

Database Interactions: Queries to the PostgreSQL database should be optimized to reduce latency and enhance performance.

6.3.2 Scalability:

User Growth: The system should be scalable to support a growing user base without significant changes to infrastructure. The use of Heroku facilitates automatic scaling to handle increased traffic or data.

Data Volume: The design should accommodate future data expansion, including more airports, user accounts, and saved favorites, ensuring no loss of performance.

API and Data Load: Ensure that the system can handle an increased API load by implementing caching and rate limits to prevent excessive requests to external API's.

6.3.3 Security:

Authentication: Secure authentication mechanisms, including NextAuth.js and JWT for session management are required to protect user accounts and data.

Data Encryption:

In Transit: All data transferred between the app and server (including API calls) should be encrypted via HTTPS to prevent interception.

At Rest: Sensitive user data (such as login credentials) will be encrypted in the PostgreSQL database to ensure security.

Data Privacy and Compliance:

Regulatory Compliance: The app will comply with relevant data privacy regulations, allowing users to manage and delete their data on request.

Access Control: Role-based access control will be implemented if future admin roles are added. This ensures only authorized personnel can access specified functions and data.

7.0 Design Considerations

7.1 Assumptions and Dependencies

- **API Reliability:** The NWKRAFT API will provide consistent and reliable data.
- **User Devices:** Users will access the app from various devices, requiring responsive design for the web and compatibility with Android devices.
- **Internet Connection:** Users need an internet connection to interact with the web and mobile apps.
- **Compliance:** The app must comply with data privacy and security regulations.

7.2 General Constraints

- **Performance:** The system must handle a minimum number of concurrent users with a response time of less than ### seconds for API calls.
- **Scalability:** The design should accommodate future growth in user base and data volume.
- **Security:** Sensitive and confidential data must be encrypted, and access must be controlled.

7.3 Goals and Guidelines

- **User Experience:** Ensure an intuitive and responsive user interface.
- **Maintainability:** The codebase should be modular and well-documented for ease of maintenance.
- **Flexibility:** Design should allow for future enhancements and integration.

7.4 Development Methods

- **Agile Methodology:** Adopt an agile approach with iterative development and regular feedback.

- **Version Control:** Use Git for version control and collaboration.

- **Testing:** Implement unit, integration, and system testing throughout development.

8.0 Architectural Strategies

- **Microservices Architecture:** Utilize microservices for modularity, with separate services for user management, search functionality, and data retrieval.

- **Cloud-Based Hosting:** Deploy the web application on AWS Cloud Front for scalability and reliability.

- **Responsive Design:** Ensure the web app adapts to different screen sizes and resolutions.

9.0 System Architecture

9.1 Subsystem Architecture

- **Frontend:**
    - **Web Application:** Built with HTML, CSS, and JavaScript, utilizing frameworks like React or Angular for dynamic content.

    - **Mobile Application:** Developed using React Native for Android, adhering to Material Design principles.

- **Backend:**
    - **API Gateway:** Acts as an interface between the app and NWKRAFT APIs.

    - **Authentication Service:** Handles user login, registration, and session management.

    - **Data Processing Service:** Manages search queries, checklist retrieval, and favorite airports functionality.

- **Database:**
    - **Cloud-Based Storage:** Use AWS RDS or DynamoDB for storing user preferences and other metadata.

10.0 Policies and Tactics

- **Security Policy:** Implement HTTPS for secure data transmission. Use OAuth 2.0 for authentication and authorization.
- **Data Privacy Policy:** Ensure compliance with data protection regulations such as GDPR or CCPA.
- **Backup and Recovery:** Regularly back up user data and implement disaster recovery procedures.

11.0 Detailed System Design

11.1 Classification

- **Web Application:** Client-side application that interacts with the user and communicates with the backend via API calls.
- **Mobile Application:** Android application with similar functionalities as the web app and optimized for mobile devices.

11.2 Definition

- **User Authentication:** Secure login, registration, and password management.
- **Airport Search:** Functionality to search for and retrieve airport data.
- **Checklist Display:** Presenting NWKRAFT checklist data in an organized manner.
- **Favorites Management:** Allow users to mark and access their favorite airports.

11.3 Responsibilities

- **Frontend Developers:** Implement UI/UX, integrate with backend APIs, ensure responsiveness.
- **Backend Developers:** Develop and maintain API services, handle data processing, ensure security.
- **QA Engineers:** Conduct testing, identify and resolve bugs, ensure application reliability.

11.4 Constraints

- **API Rate Limits:** Adhere to rate limits imposed by NWKRAFT APIs.

- **Mobile Performance:** Optimize app performance to ensure smooth operation on a range of Android devices.

11.5 Composition

- **Modular Design:** Divide the application into modular components for ease of development and maintenance.
- **Reusable Components:** Create reusable UI components to ensure consistency across the web and mobile platforms.

11.6 Uses/Interactions

- **User Interactions:** Users interact with the app through the UI to perform searches, view data, and manage favorites.
- **System Interactions:** The front-end communicates with the backend to fetch and display data. The backend interacts with NWKRAFT APIs to retrieve necessary information.

11.7 Resources

- **Development Tools:** IDEs like Android Studio, Visual Studio Code, or WebStorm.
- **Libraries and Frameworks:** React, Angular, Kotlin, Java, and relevant AWS services.
- **Version Control:** GitHub repository for source code management.

11.8 Processing

- **Data Retrieval:** The app processes user requests by querying NWKRAFT APIs and retrieving the necessary data.
- **Data Storage:** User preferences and session data are stored securely in the cloud database.

11.9 Interface/Exports

- **API Endpoints:** Define endpoints for user authentication, airport search, checklist retrieval, and favorites management.
- **Data Formats:** Use JSON for data exchange between frontend and backend.

11.10 Detailed Subsystem Design

- **Frontend:**
  - **Web:** Implement search forms, checklist display components, and favorite management UI.
  - **Mobile:** Design Android-specific interfaces and ensure proper integration with backend services.
- **Backend:**
  - **Authentication Service:** Implement OAuth 2.0, manage user sessions.
  - **API Gateway:** Handle requests and responses, ensure secure and efficient data retrieval.
  - **Data Processing Service:** Manage and process data from NWKRAFT APIs, handle search and favorites functionality.

12.0 Components of the Flight Data Web App

12.1 NOTAM Database Integration

The app will connect to the FAA API to retrieve up-to-date NOTAM information. By relying on this database, the web application can fetch the newest updates directly from the source to guarantee reliable information is presented to users.

Data Retrieval: The app will pull relevant NOTAM data through API calls, providing continuous/live updates.

Data Filtering: Users can filter NOTAMs based on their type, allowing for a cleaner interface that only shows information requested by the user.

12.2 User Interface (UI) for Efficient Search and Display

The UI is designed to make navigation easy, allowing users to quickly locate the NOTAMs they need. The design should be clear and accessible on both desktop and mobile devices.

Search Functionality: A search bar allows users to look up NOTAMs by either airport codes or names.

Sorting and Filtering Options: Users can sort NOTAMs by the type of NOTAM and save airports to their favorites.

Responsive Design: The web app's design should ensure a seamless experience across multiple devices.

13.0 Modes of Operation for Enhanced Flexibility

User Search Modes

The web app provides multiple search modes to accommodate various user needs:

Airport Search: Allows pilots to enter a specific airport code to retrieve relevant NOTAMs for that location.

User Favorites: Enables users to save airport codes as favorites, ensuring they are readily available in the future without requiring a search.

14.0 Implementation Steps for Setup

1. **Environment Setup**:

   Set up the development environment, including accessing necessary tools such as Node.js and Heroku.

2. **API Key Acquisition**:

   Get an API key for the FAA API to access data. Configure the key within the application and add to .gitignore for development.

3. **Front-End Development**:

   Build the search and display functions with React. Incorporate UI components for login, search, and favorites.

4. **Back-End Integration**:

   Develop a backend to handle API requests and process user data. Include API data retrieval and JSON parsing for real-time updates.

5. **Testing**:

Test functionality across multiple platforms to guarantee compatibility and a smooth user experience.

6. **Deployment**:
   Host the application on a web server. Confirm secure access via JSON Web Tokens per login instance.

15.0 Software Test Plan

Requirement	High Priority	Low Priority
Create account	X	
Log in to account	X	
Pull NOTAMs data via API	X	
NOTAMs data displayed in a list	X	
Pull weather data via API	X	
Weather data displayed in a list	X	
Allow user to save airports		X
Allow user to remove saved airports		X
Dynamic search field		X
Search field allows search by ICAO code	X	
Search field allows search by name		X
Forgot password selection		X
User logout option	X	
User selection option for which data they want to view		X
Login should take less than 2 seconds to load	X	
Search field data should take less than 2 seconds to load	X	

Search results should take less than 2 seconds to load	X	
Airport static data should be cached in the database		X
User roles		X
Known ATC delays data available		X
Runway lengths	X	
Alternate routes		X
Fuel requirements		X
Takeoff and landing distances		X

16.0 Software Test Report

Requirement	Pass	Fail	High Priority	Low Priority
Create account	X		X	
Log in to account	X		X	
Pull NOTAMs data via API	X		X	
NOTAMs data displayed in a list	X		X	
Pull weather data via API	X		X	
Weather data displayed in a list	X		X	
Allow user to save airports	X			X
Allow user to remove saved airports	X			X
Dynamic search field	X			X
Search field allows search by ICAO code	X		X	
Search field allows search by name	X			X
Forgot password selection		X		X
User logout option	X		X	
User selection option for which data they want to view		X		X
Login should take less than 2 seconds to load	X		X	
Search field data should take less than 2 seconds to load	X		X	

Search results should take less than 2 seconds to load	X		X	
Airport static data should be cached in the database	X			X
User roles		X		X
Known ATC delays data available		X		X
Runway lengths		X	X	
Alternate routes		X		X
Fuel requirements		X		X
Takeoff and landing distances		X		X

17.0 Conclusion

The NWKRAFT Flight Data web app fulfills the primary goal of providing pilots with access to flight planning information, including NOTAMs and airport data. The project has met several goals, including search functionality, a favorites feature, and cross-platform compatibility for both web and Android devices. The architecture allows for easy scalability and ensures users' data remains secure. The use of cloud-hosted components ensures excellent performance and removes the need for hardware upgrades in order to scale up.