|  |  |
| --- | --- |
| Creative Layout. Top View of White Model Plane, Airplane Toy on Pink Pastel  Background. Flat Lay Wi' Photographic Print - jchizhe | AllPosters.com  **FareTide**  CST-451 Capstone Project Design | Caroline Macauley  Grand Canyon University  Instructor: Professor Amr Elchouemi  Revision: 2  Date: March 16th, 2025 |

**ABSTRACT**

Many travelers face challenges when determining the optimal time to purchase airline tickets, often resulting in uncertainty about whether to buy now or wait for better prices. Existing tools on the web can provide flight information and basic fare comparisons, but they frequently lack advanced predictive capabilities and integrated insights into fare trends. This can leave users guessing if they are getting the best deal or if waiting might offer a better price, leading to potentially missed opportunities or unnecessary expenses.

FareTide is designed to address these issues by offering a comprehensive platform that consolidates flight information from various airlines and uses historical data to predict the best times to purchase tickets. By leveraging advanced machine learning algorithms and real-time data integration, FareTide provides users with detailed fare trend insights, helping them make well-informed decisions. The website’s features will include easy search and filtering features, allowing even less seasoned travelers to access all possible flights that fit their needs. Detailed flight listings will populate a clean, simple search results page that enables users to make comparisons. When users finds a flight that interests them, they can click to view more specific details such as plane models or seat classes. The standout feature, fare price predictions, will be provided on the selected flight’s purchasing options page. The goal of FareTide is to ensure that every click on the website guides users smoothly towards a confident airline ticket purchase.

|  |
| --- |
| **History and Signoff Sheet** |

**Change Record**

|  |  |  |
| --- | --- | --- |
| **Date** | **Author** | **Revision Notes** |
| November 17th, 2024 |  | Initial draft for review/discussion |
| March 16th, 2025 |  | Amr Elchouemi |
|  |  |  |

|  |
| --- |
| **Overall Instructor Feedback/Comments** |

**Integrated Instructor Feedback into Project Documentation**

Yes  No

**Project Approval**

Professor Amr Elchouemi

**TABLE OF CONTENTS**

[Design Introduction 4](#_Toc182761188)

[Design Introduction 4](#_Toc182761189)

[Detailed High-Level Solution Design 5](#_Toc182761190)

[Software Requirements 5](#_Toc182761191)

[Hardware Requirements 5](#_Toc182761192)

[Proof of Concepts 6](#_Toc182761193)

[Logical Solution Design: 7](#_Toc182761194)

[Physical Solution Design: 8](#_Toc182761195)

[Detailed Technical Design 9](#_Toc182761196)

[General Technical Approach: 9](#_Toc182761197)

[Key Technical Design Decisions: 9](#_Toc182761198)

[Sitemap Diagram: 10](#_Toc182761199)

[User Interface Diagrams: 11](#_Toc182761200)

[Component Design: 11](#_Toc182761201)

[Service API Design: 13](#_Toc182761202)

[NFR’s (Security Design, etc.): 17](#_Toc182761203)

[Operational Support Design: 17](#_Toc182761204)

[Other Documentation: 18](#_Toc182761205)

[Appendix A – Technical Issue and Risk Log 19](#_Toc182761206)

[Appendix B – References 20](#_Toc182761207)

[Appendix C – External Resources 22](#_Toc182761208)

[Appendix D – UI Diagrams Continued 23](#_Toc182761209)

[Appendix E – Component Designs Continued 25](#_Toc182761210)

# Design Introduction

This section provides an introduction to the technical design, loosely covering the machine learning model, backend API, frontend interface, and data management. It also includes a Deliverable Acceptance Log detailing key project documents and their locations.

## Design Introduction

FareTide predicts flight ticket prices using a machine learning model trained with historical data in CSV files. The backend, built with Python, Pandas, and scikit-learn, serves predictions via a Flask REST API and integrates the Skyscanner API to allow users to search for real-time flights and access purchasing options on third-party sites. The frontend, developed with ReactJS, provides an intuitive user interface. The system supports continuous model retraining with updated datasets. It can be hosted on a local laptop with standard hardware specifications, including at least 8GB of RAM, a multi-core processor such as an Intel i5 or comparable, and at least 600GB of available disk space for data storage. Note that this program does not use a database as all data is managed through CSV files. This document has sections below where more in depth details about the technical design of FareTide can be explored.

See a [flowchart](#_Other_Documentation:) that gives insight into how the user will flow through the website’s features for further understanding of website navigation and features.

# Detailed High-Level Solution Design

This section outlines the software and hardware requirements.. It provides proof of concepts for these technologies showing their suitability and alignment with project goals. Logical and physical solution designs detail system architecture, data flow, and user interactions for implementing FareTide.

## Software Requirements

Table below details the necessary software and their versions needed to complete this project.

|  |  |
| --- | --- |
| **Software Category** | **Technology or Tool** |
| Back-end Language | Python version 3.1 |
| Back-end Framework | Flask version 2.3 |
| Front-end Language | JS |
| Front-end Framework | ReactJS |
| Imported ML Related Libraries | Panda, scikit-learn |
| IDE | VS Code |
| Historical Flight Dataset | CSV file from kaggle |
| Real-time Flight Data - Third Party API | Skyscanner |
| Task Management | Jira |
| Version Control | Github private repository |

## Hardware Requirements

Table below details the necessary hardware and their versions needed to complete this project.

|  |  |
| --- | --- |
| **Hardware Category** | **Technology or Tool** |
| CPU | Intel i7, Ryzen 7, Apple M1 (or comparable) |
| RAM | 32 GB |
| Available disk space (storage for CSV file) | 600GB |

## Proof of Concepts

Table below provides a detailed inventory of the technologies selected for the project to include frameworks, third-party libraries, and APIs. Each row includes the purpose (found in the description) and rationale for the technology being chosen against alternatives. Associated Proof of Concepts (POCs) designed to validate its suitability, cost-effectiveness, and ability to solve the problem. The purpose of the POCs is to mitigate risks identified in the project’s risk assessment and ensuring the selected technologies align with the solution’s goals. Please note all technologies being used already provided and are free.

|  |  |  |
| --- | --- | --- |
| **Proof of Concepts** | |  |
| **Description** | **Rationale** | **Results** |
| 1. Back-end Framework : Python Flask V Python Django | Flask is more lightweight and rates higher in community for integration with ML libraries and learnability | Python Flask |
| 1. Front-end Framework: ReactJS V Angular | Developer prefers ReactJS to Angular based on preferences from previous experience | ReactJS |
| 1. ML Library for building and deploying ML Models: scikit-learn V Tensorflow | scikit-learn is designed for beginners and rated as easier to learn by community | scikit-learn |
| 1. Library for preparing datasets for ML: Pandas V Dask | Pandas has most community support and tutorials. Rated by community as easiest learning curve | Pandas |
| 1. IDE: VS Code V Jupyter Notebook | Developer has previous experience with VS Code IDE | VS Code |
| 1. Task Management Tool: Jira V Trello | Developer is more established with Jira and wants to learn more about tool | Jira |
| 1. Version Control: Github V BitBucket | Github provides more community support and developer is already established | Github |
| 1. Historical dataset CSV Files: Kaggle V AI generated CSV | Kaggle Datasets are already created and tested with sufficient data | Kaggle CSV Files |
| 1. Third party API for live flight fare data: Skyscanner V Google Flights | Skyscanner API Provides sufficient amount of requests and correct data types for free | Skyscanner API |
| 1. Hosting: Local Hardware or Rented Hardware (cloud) | Local hardware is chosen for this project as the program is not intended for public consumption at this time. | Local Hardware/Machine |

## Logical Solution Design:

A screenshot of a phone

Description automatically generatedThis Logical Solution Design diagram below illustrates how the FareTide system is structured to provide flight predictions and real-time data to users. At the top, the Presentation Layer represents the user interface, built with ReactJS, where users can search for flights using components like search bars and buttons. The Application Layer, powered by a Flask API written in Python, processes user requests and sends responses in JSON format. The lowest layer, Business/Service Layer handles the core functionality to include the prediction model using historical flight data (stored in CSV files) integration with the Skyscanner API for live flight data and tools like Pandas for data processing. The layers interact seamlessly to deliver predictions and flight information to the user.

## Physical Solution Design:

A computer and a cloud with a symbol

Description automatically generated with medium confidenceThis Physical Solution Design diagram below shows how the FareTide system operates when hosted on a local device. Users access the application through a browser connected to the same local network as the hosting device and then using the hosting device's local IP ( http://192.168.x.x:3000). The front end is built with ReactJS and runs on port 3000 meanwhile the back end is powered by a Flask API and runs on port 5000. The backend handles flight predictions and retrieves data from a local CSV file and the Skyscanner API via the internet. This design ensures seamless communication between system components and supports users on the same network.

# Detailed Technical Design

*The Detailed Technical Design section provides a comprehensive breakdown of the system's technical architecture. Subsections include the General Technical Approach, explaining the integration of machine learning and real-time flight searches. The Key Technical Design Decisions that outlines tool and framework choices. Flow Charts and Process Flows to aid in visualizing user interaction paths. Sitemap Diagram that details website navigation. UI Diagrams which present the user interface. Component Design that will explain system components. Service API Design that documents API functionality and security. Each subsection ensures clarity and support for efficient system implementation.*

## General Technical Approach:

The design approach integrates ML powered predictions based on historical data with real-time shopping and purchasing capabilities. By leveraging Flask, Python, and the Skyscanner API, the development team can deliver both functionalities in a single web app. This efficiently utilizes publicly available data and prevents development team from having to create each component of this application from scratch.

Meeting discussions explored include using Tailwind CSS and Jupyter Notebook as additional tools. A 2TB external hard drive is available for additional storage for large CSV files (historical datasets). Out of scope ideas include creating a custom dataset, cloud hosting, and publishing as a Chrome extension to add enhancements post deadline.

## Key Technical Design Decisions:

The FareTide project design includes technologies like Python, Flask Rest API back-end, Pandas, and scikit-learn as in today’s market they are valuable tools to understand and most relevant in the field of machine learning. ReactJS is included in the design as JavaScript is a common language for front-end and is reported to integrate well with Flask. All the tools and technologies were chosen for relevance to topic and availability of free and abundant resources. The program will be ran locally on a developer laptop as this project is being developed for education and testing purposes and is not yet intended for mass public consumption. For refreshers on tools, technologies and Proof of Concepts see [tables](#_Detailed_High-Level_Solution) above.

## Sitemap Diagram:

A screenshot of a computer

Description automatically generatedThe sitemap is a diagram designed with the purpose of showing how the user will navigate from page to page and experience the website’s features. The flow begins in the initial ‘Main Search’ container. The user is then able to progress to a list of matching flight results via two separate paths. The path to the left is for one way flight searches and the path to the right is round trip flight searches which entail two search result pages instead of one. Both paths show ‘Specific Flight’ detail pages for an individual selected flight. On those pages a share option is available (seen in orange star container) that will provide a link for users and then return to the ‘Specific Flight’ page it originated from. When the user selects a purchasing option they will then exit the site and enter a new tab with the selected purchasing vendor. In the blue box are vital notes to understanding the symbols and connections on the diagram.

## User Interface Diagrams:

UI diagrams, also known as wireframes, are intended to provide both the client and development team a peak at the finished screen the user will experience. The below wireframes [Found in appendix](#_Appendix_D_–) are considered low fidelity as they lack real products, marketing and images. Keep in mind the sitemap found above that details the user’s flow through the site’s functionalities. The user maintains the ability to return to previous page or initial search page on every wireframe/page created below.

## Component Design:

The Component Design outlines the primary building blocks of the FareTide web application. Each component is outlined and labeled in colors t in the wireframes [Found in appendix](#_Appendix_E_–). This table directly below is intended to express how the components manage data/state, render content, and interact with each other to create a cohesive user experience. The wireframes area a visual representation of the application's modular structure and roles to provide clarity for developers and stakeholders. Please note that the table below shows the components in colored font that corresponds with the wireframes.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Component** | **Wireframe Section** | **Component Description** | **State Maintained** | **Pseudo Code Logic** |
| **Header** | Top section of all pages with the "FareTide" title/logo | Displays the website title/logo and navigation link to the main search page | None | - Render the title/logo - On click of the title/logo, navigate to the Main Search Page |
| **SearchForm** | Main Search Page - Fields below the Header | Provides input fields for users to specify search criteria (locations, dates, travelers, layovers). | searchInputs: stores user input | - Render input fields for locations, dates, travelers, and layovers.  - On input change, update searchInputs state - On "Find" click validate and submit data |
| **FlightResults** | Departing Flights Page and Returning Flights Page | Displays a list of available flights based on search criteria with sorting options. | Flights: array of flight results | - Fetch flight data based on search criteria - Render each flight result as a row in the results list |
| **FlightDetails** | Specific Flight Page - Detailed flight section | Shows detailed information for a specific flight, including layovers, baggage allowance, departure/arrival times, and price. | None | - Render detailed flight information - Display “Share” Button to allow sharing flight details |
| **CopyLink** | Share Flight Page (Copy link pop-up) | Displays a sharable link for a selected flight | shareableLink: the link to be shared | - Generate or fetch the shareable link - Display the link with a "Copy" button to copy it to the clipboard |
| **SelectedFlight** | Purchase Options Page - Selected flight section | Displays selected departing and returning flight details | None | - Render selected departure and return flight details  - Display total price for both flights |
| **FarePredictions** | Purchase Options Page -Fare prediction section | Displays predicted fare savings and the best date to purchase the ticket | Predictions: best date to buy, predicted savings | - Fetch fare prediction data - Render the best purchase date and predicted savings |
| **PurchaseOptions** | Purchase Options Page -Third party purchase links section | Displays third party links for purchasing selected flights | |  | | --- | | None | | - Render list of purchase options  - On click of an option, redirect to the third party website |

## Service API Design:

This subsection outlines the APIs supporting FareTide. The Skyscanner REST API enables the current flight data retrieval with key parameters like origin, destination, travel dates, and passenger count. JSON requests and responses detail flight pricing, schedules, and durations. Secure access is ensured via API keys and HTTPS encryption. The FareTide Flask REST API provides endpoints for fare prediction and flight search that have been written in house by a developer. Predictions use historical data found in the FareTide REST API for insights like the best date to purchase and potential savings. Meanwhile flight searches integrate Skyscanner REST API data. JSON specifications are provided for seamless third-party integration. When the program is deployed publicly, JWT authentication would secure interactions.

Skyscanner REST API

**Access point**: <https://skyscanner3.p.rapidapi.com/> with keys provided to developers RapidAPI account

**Parameter requirements:**

**Endpoint:** /search

* origin: IATA code of the departure airport (e.g., "JFK").
* destination: IATA code of the arrival airport (e.g., "LAX").
* departureDate: Date of departure in YYYY-MM-DD format.
* returnDate (optional): Return date in YYYY-MM-DD format.
* adults: Number of adult passengers.
* currency: Currency code for pricing (e.g., "USD").

**Detailed JSON data format** **specification that could be used by a third party developer to integrate with the service and API:**

Request: Response:

{

"status": "success",

"data": [

{

"flightId": "ABC123",

"price": 250.00, "airline":

"Delta", "departureTime": "2024-12-01T08:00:00",

"arrivalTime": "2024-12-01T11:00:00",

"duration": "3h"

} ,

{

"flightId": "XYZ456",

"price": 275.00,

"airline": "United",

"departureTime": "2024-12-01T10:00:00",

"arrivalTime": "2024-12-01T13:00:00",

"duration": "3h"

}

]

}

{

"origin": "JFK",

"destination": "LAX",

"departureDate": "2024-12-01",

"returnDate": "2024-12-10",

"adults": 1, "currency": "USD"

}

**Must address how security will be designed (HTTP Basic Auth, OAuth, JWT):**

Utilizes API keys for access control. All requests are made over HTTPS to encrypt data in transit.

Flask FareTide Back-End REST API

**Access point**: <http://localhost:5000/>

**Parameter requirements:**

**Endpoint:** /predict - Fare prediction based on historical data.

* origin: IATA code of the departure airport (e.g., "JFK")
* destination: IATA code of the arrival airport (e.g., "LAX")
* departure\_date: Date of departure in YYYY-MM-DD format

**Endpoint:** /search - Queries Skyscanner for available flights.

* origin: IATA code of the departure airport (e.g., "JFK")
* destination: IATA code of the arrival airport (e.g., "LAX")
* departure\_date: Date of departure in YYYY-MM-DD format
* return\_date: Return date in YYYY-MM-DD format
* travelers: Number of adult passengers (2)

**Detailed JSON data format** **specification that could be used by a third party developer to integrate with the service and API:**

/predict Request: Response:

{

"best\_date\_to\_buy": "2024-11-20",

"predicted\_savings": 50.00

}

{

"best\_date\_to\_buy": "2024-11-20",

"predicted\_savings": 50.00

}

**/search** **Request: Response:**

{

"flights": [

{

"FlightId": "12345",

"Price": 250.00,

"DepartureTime": "2024-12-01T08:00:00",

"ArrivalTime": "2024-12-01T11:00:00",

"Airline": "Delta",

"Duration": "3h"

}

]

}

{

"origin": "JFK",

"destination": "LAX",

"departure\_date": "2024-12-01",

"return\_date": "2024-12-10",

"travelers": 2

}

**Must address how security will be designed (HTTP Basic Auth, OAuth, JWT):**

JWT Authentication for user authentication if deployed publicly.

## NFR’s (Security Design, etc.):

FareTide's design addresses the Non-Functional Requirement by using a Flask API. The Flask REST API handles requests quickly by processing the CSV data directly to avoid database delays, and also integrates efficiently with the Skyscanner API. Therefore the user should receive both prediction and current flight services in a timely manner. The hardware required for this project is above the necessary amount as well to ensure hardware does not slow down the request. Monitoring tools and reusing components in ReactJS also ensure fast response times especially during light traffic.

|  |
| --- |
| **Non-Functional Requirement** |
| *As a system I should return flight price predictions to user within 5 seconds of being prompted when the web traffic is lite (less than 20 users present on website) So that users are not inconvenienced by long response times* |

## Operational Support Design:

FareTide supports monitoring and logging by using Python's logging module for backend events, Flask-MonitoringDashboard for performance insights, and logging interactions with the Skyscanner API. React Error Boundaries track frontend issues, ensuring efficient debugging and reliable system performance.

## Other Documentation:

Below is a flowchart that visually represents the user’s navigation through the website if they are buying a one way flight. There would be repetition in the ‘displaying search results’ function and the following functions that would be inserted for a round trip flight’s returning flight. Please note to implement FareTide the developers do NOT need to write an algorithm for the fare prediction. This flowchart does not represent the algorithm.

A screenshot of a computer screen

Description automatically generated

# Appendix A – Technical Issue and Risk Log

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Risk Management** | | | | |
| **Event Risk** | **Risk Probability (high, medium, low)** | **Risk Impact** | **Risk Mitigation** | **Contingency Plan** |
| No suitable resources for learning Python | low | Require using a different language/framework | Research resources and speak with mentor | Use Java as alternative if not resolved by set date |
| No suitable resources for learning ML | low | Project fails to have key feature | List as highest priority, research resources and speak with mentor | Change Project topic if not resolved by set date |
| Lacking enough understanding of Python to build website | high | Web app is incomplete | Research resources and speak with mentor | Use Java as alternative if not resolved by set date |
| Lacking enough understanding of Python as it relates to machine learning | high | Project fails to have key feature | Research resources and speak with mentor | Change Project topic if not resolved by set date |
| Inadequate hardware for project | medium | Project stops | Speak with hardware experts and mentor | Acquire necessary material OR adjust project to meet current hardware specs. |
| Cost for access to necessary materials is not within budget | medium | Project stops | Speak with mentor and investigate options | Adjust budget OR adjust project sources |
| Inability to obtain access to data/API to train AI | medium | Project will have no data | Research resources and speak with mentor | Adjust project data sources |

# Appendix B – References

The work on this document is mine in my own words

CST-451 Topic 3 slides and resources found in Mark Reha’s Padlet

*The 5 best logging libraries for python*. Highlight. (n.d.). <https://www.highlight.io/blog/5-best-python-logging-libraries>

CHANDO, M. D. (n.d.). *Chando0185/flight\_price\_prediction\_react\_python*. GitHub. <https://github.com/Chando0185/flight_price_prediction_react_python>

*Draw.io - free flowchart maker and diagrams online*. Flowchart Maker & Online Diagram Software. (n.d.). <https://app.diagrams.net/>

*Flights scraper*. API Hub - Free Public & Open Rest APIs. (n.d.). <https://rapidapi.com/ntd119/api/sky-scanner3/playground/apiendpoint_8207ef83-59d5-436f-aa58-1f9ddcfc4f7>

b

Ganesha, J. (n.d.). *Flight Price Prediction Using Python & React JS | Machine Learning | Tailwind CSS | Flask*. YouTube. <https://www.youtube.com/watch?v=Kf33NwE2jT0>

GeeksforGeeks. (2024, October 8). *Best python libraries for machine learning*. <https://www.geeksforgeeks.org/best-python-libraries-for-machine-learning/>

Naik, K. (n.d.). *Summary Of Live- Implementation Of Flight Fare Prediction Web App Project With Deployment*. YouTube. <https://www.youtube.com/watch?v=6vqGeigbwjc>

Shokrzad, R. (2023, October 27). *Flask vs Django: Deciphering the best fit for your ML deployment*. Medium. <https://medium.com/@reza.shokrzad/flask-vs-django-deciphering-the-best-fit-for-your-ml-deployment-c536dd457a4e>

Simplilearn. (2023, July 24). *Scikit-learn VS TENSORFLOW: A detailed comparison: Simplilearn*. Simplilearn.com. <https://www.simplilearn.com/scikit-learn-vs-tensorflow-article>

*Thinking in react*. React. (n.d.). <https://react.dev/learn/thinking-in-react>

*What is flask-monitoringdashboard?¶*. Flask. (n.d.). <https://flask-monitoringdashboard.readthedocs.io/en/latest/>

YouTube. (n.d.). *How to Create a Flask + React Project | Python Backend + React Frontend*. YouTube. <https://www.youtube.com/watch?v=7LNl2JlZKHA>

# Appendix C – External Resources

|  |  |
| --- | --- |
| **GIT URL:** | *Private Repository. Request access by emailing owner: Carolinerma@gmail.com* |
| **Hosting URL:** | *Not live at this time* |

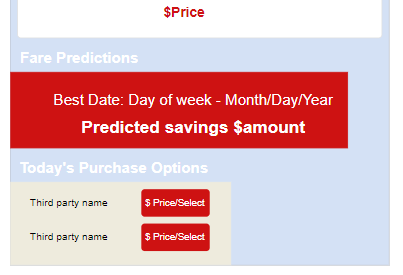
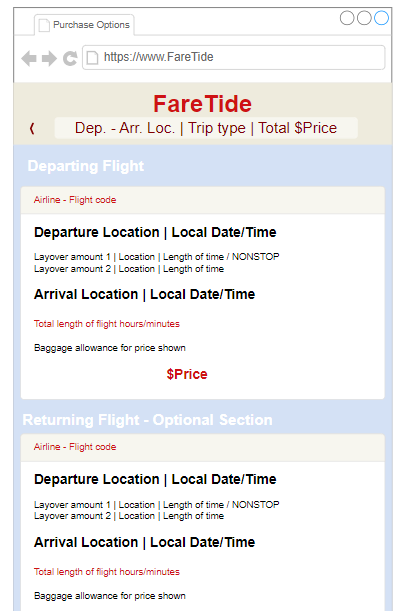
# **A screenshot of a computer Description automatically generatedA screenshot of a search engine Description automatically generated** Appendix D – UI Diagrams Continued

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**



# Appendix E – Component Designs Continued

