1. Project Foundation

Start by clearly defining your project's fundamental elements:

Project Overview

1. Write a clear, concise project title
    a. ARRO (Aircraft Resource Routing Optimizer)
2. List all team members and their roles
    a. Allen Mathew - Frontend
    b. Eric Wong - Frontend
    c. Everett Yan - Frontend
    d. Himanth Bobba - Backend
    e. Jason Xu - Backend
    f. Yannis Fu - Backend
3. State your project's primary goal and objectives
    a. Create multiple competing heuristics. Analyze which provides better routes and why.
    b. Support planning a route for more than one vehicle simultaneously. Any vehicle could go to any location but all locations must be visited just once.
4. Define what is (and isn't) included in your project scope
    a. What is
        i. Front+back end
        ii. Map interface
        iii. Cost and schedule analysis from heuristics
        iv. Interpreting csv files
    b. What isn’t
        i. Intermodal routing
        ii. Real time updates
        iii. Multiple users

2. Understanding Your Users and Stakeholders

Identify and analyze who will use or be affected by your system:

User Analysis

1. Who are your primary users?
    a. Airline Route Planners
2. What problems are you solving for them?
    a. THEIR JOBS
3. What are their technical capabilities?
    a. Uploading a CSV File
4. What are their key needs and pain points?
    a. A consistent method to handle complex data from multiple sources

Stakeholder Analysis

1. List all project stakeholders (professors, clients, end-users)
    a. Boeing
    b. Mia Mohammad Imran
2. Document their expectations and requirements
    a. Have multiple competing heuristics and show which ones perform the best
    b. Support route planning for multiple planes
    c. Calculation of routes must be within a reasonable amount of time
3. Note any conflicts between different stakeholder needs
    a. N/A

3. Functional Requirements

Detail what your system needs to do:

Core Features Document each feature using this format:

Feature ID: F001

Name: [Feature Name]

Description: [Clear, specific description]

Priority: [High/Medium/Low]

User Story: "As a [user type], I want to [action] so that [benefit]"

Acceptance Criteria: [List specific conditions that must be met]

Feature ID	Name	Description	Priority	User Story	Acceptance Criteria
F001	Point picking	User is able to pick points on map to guide routing	1	As a pilot, I want to be able to pick points on the route for places I want to stop at so that I can have a spot to rest at.	-Points on the map are marked and can be clicked on to be added to the route -Points can be added or removed from the route by the user to their liking.
F002	Route calculation	Program will calculate efficient routes	1	As a pilot, I want to be able to	-The most efficient routes are

				calculate the most efficient routes so that I can save both time and fuel costs.	displayed to the user based on the selected heuristics -The routes will tell the user how long the route will take and how fuel expensive they will be
F003	Route display	Program can display calculated routes on map	1	As a pilot, I want to be able to view all my available routes so that I have some options to choose from, whether I want to spend a little less time in the air or save the most I can on fuel costs.	-Multiple routes are displayed to the user so that they may choose the best one according to their circumstances -The trade-offs between each of the displayed routes will be explicitly highlighted

System Behaviors

1. Describe how your system responds to user actions
    a. When the user inputs a CSV file, the system should take the data from that CSV file and calculate multiple routes via the available heuristics.
    b. The multiple routes should be updated and display on the Google Maps visual
2. Define input requirements and expected outputs
    a. Input: A CSV file along with manual supplemental information
    CSV contains location identifiers (assuming gps?) and multiple types of costs (time, fuel, personnel)
    b. Output: A Maps Route & Chart with Cost Comparisons
3. Specify error handling procedures
    a. Pop up Text asking for a Fix

4. Technical Requirements

Specify how your system will be built:

Development Specifications

1. Programming languages and frameworks
    a. Typescript
    b. Nest.js + Svelte

    c. C++
2. Database requirements
    a. N/A
3. API integrations needed
    a. Google Maps (for visuals)
    b. iFlightPlanner API
4. Development tools and environments
    a. NextJS

Quality Requirements

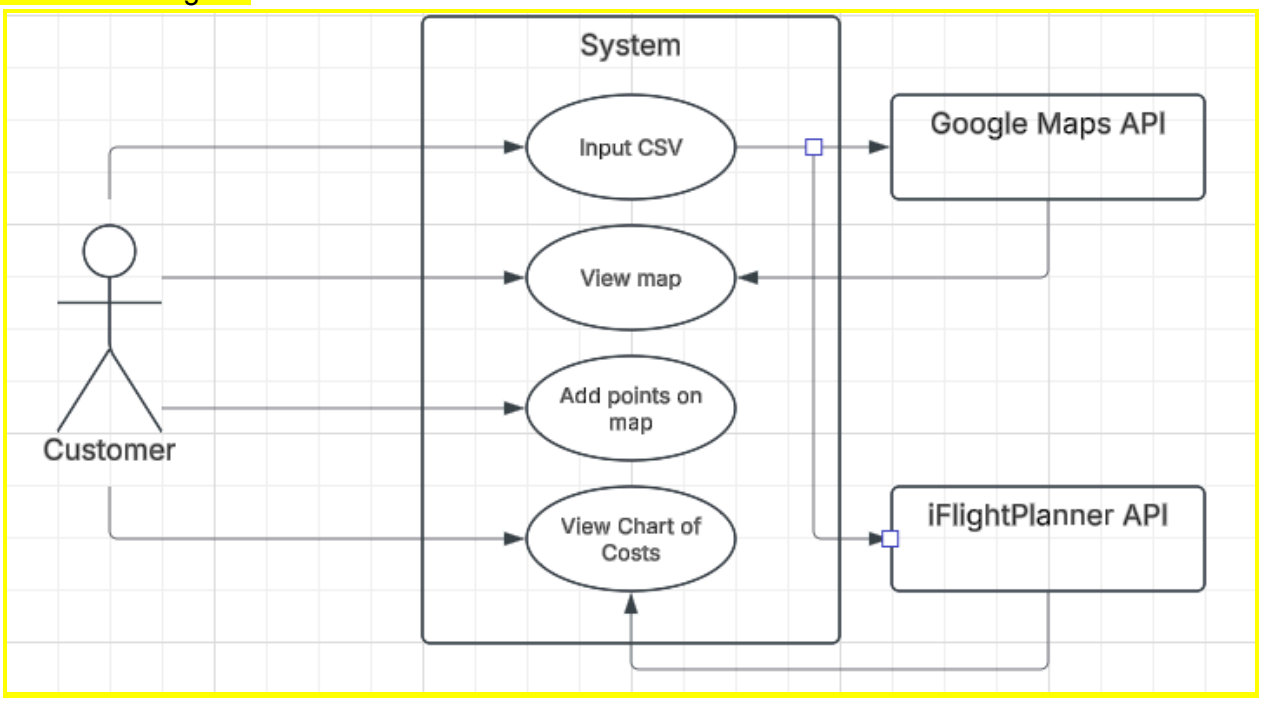
1. Performance expectations (response times, load capacity)
    a. No more than 5 sec for 100k stops
    b. Can support concurrent users
2. Security requirements
    a. Proper memory management (no memory leaks)
3. Reliability standards
    a. Should not produce infeasible routes
    b. Should not produce a route more than double the “cost” of global optimum
4. Compatibility requirements
    a. Should be compatible with every browser

5. Documentation and Diagrams

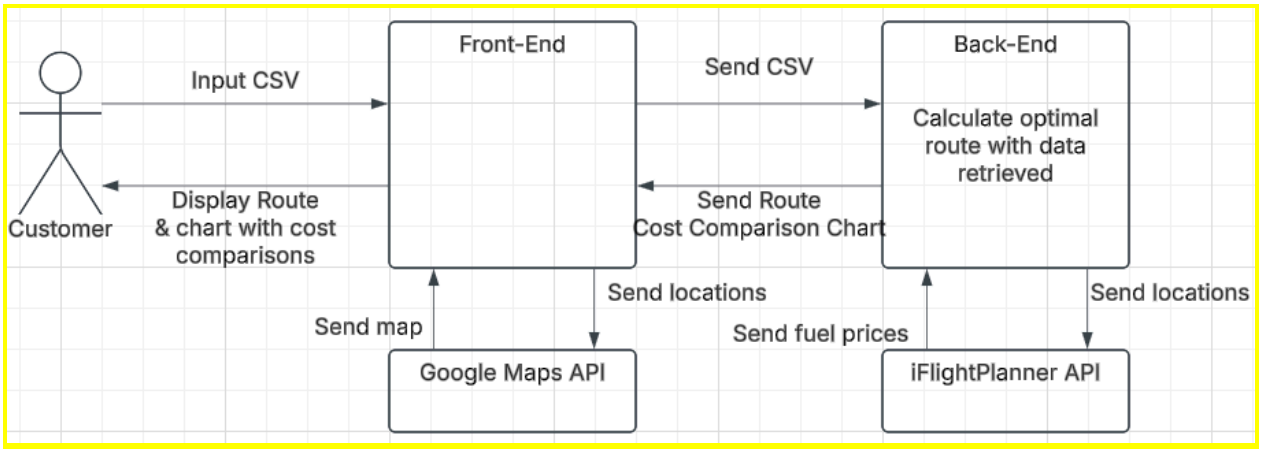
Create visual representations of your system:

Required Diagrams

1. Use Case Diagram



2. Data Flow Diagram



3. Entity-Relationship Diagram
    a. N/A (no database needed)

6. Project Planning Timeline

Organize your development timeline around the planned sprints

- Week 8: Review 2 (Design/Approach)
    - TPS with one heuristic and a single plane is implemented
- Week 12: Review 3 (Implementation)
    - TSP with multiple heuristics

    - Basic functional front end
- Week 16: Deployment and Presentation
    - TSP with multiple heuristics and planes fully implemented
    - Finished UI

7. Validation Strategy

Ensure your requirements are solid:

Review Process

1. Self-review the documentation
2. Peer review by team members
3. Professor/advisor review
4. Stakeholder review (if applicable)
5. Final adjustments based on feedback

Some Advices

Best Practices

1. Use clear, specific language
2. Avoid technical jargon unless necessary
3. Number all requirements for easy reference
4. Keep track of requirement changes
5. Document assumptions and constraints

Common Pitfalls to Avoid

1. Vague or ambiguous requirements
2. Overlooking non-functional requirements
3. Insufficient stakeholder communication
4. Poor documentation
5. Ignoring technical constraints

Submission Guidelines

Final Deliverable Requirements

1. All documentation in PDF format
2. **Please put all team members names**
3. Submit in Canvas (one team member submission is fine)
4. 4-5 pages (you can add more but don't do 10 pages)
5. Create a GitHub repository in CS4091 org and add a folder there and upload it there too.

- In GitHub repo: add your team member names in README file