Peak Adventure Experiences

Software Design Document

Date: 02/06/2026

Team Members: Makaela Crookes, Yahir Espinoza, Alonso Garcia, Jack Morris

Project Sponsor: Paddy McGarry

Team Mentor: Ogonna Eli

Table of Contents

I. Introduction

Relocating to a new city is one of the most consequential decisions individuals and families make, involving significant financial, emotional, and lifestyle considerations. Despite the availability of online tools such as cost-of-living calculators and city-ranking websites, many prospective movers struggle to gain an understanding of what daily life in a new location will feel like. This gap contributes to widespread relocation regret and uncertainty, highlighting the need for more immersive and experiential decision-making tools.

The relocation industry encompasses a wide range of services designed to assist movers, including real estate agencies, moving companies, and digital relocation platforms. The U.S. relocation market is valued at over $20 billion annually, supported by millions of domestic moves each year. Within this industry, decision-making tools such as cost-of-living calculators and city-ranking websites help users compare destinations but often fail to capture the full lifestyle experience that drives satisfaction after a move.

Our project addresses this challenge by designing a web-based virtual relocation experience for the city of Flagstaff. Sponsored by Padraic “Paddy” McGarry, owner of *The Scouting Party*, a Flagstaff-based guided relocation tour service, the system aims to extend the value of in-person city tours to users who may not be able to visit the city physically. By presenting Flagstaff through a gamified, interactive environment, the application promotes familiarity, engagement, and confidence in relocation decisions while encouraging users to connect with The Scouting Party for personalized, in-person tours.

The system is designed for prospective homeowners and relocators who wish to explore Flagstaff virtually. Users will navigate a game-like environment featuring explorable zones of the city, non-player character (NPC) interactions that provide contextual information about neighborhoods and amenities, and Flagstaff-themed mini-games that reinforce learning through play. The application will also guide users toward initiating contact with The Scouting Party to begin the relocation process.

From a functional perspective, the system will support user registration and authentication, gameplay mechanics, progress and score tracking, real-time weather display, audio features, main quests, and the incremental addition of new explorable areas. Performance requirements include responsive map and weather updates within two seconds of data retrieval, gameplay frame rates exceeding 30 FPS, user login and data access within one second, and support for at least ten concurrent users without noticeable lag. Additionally, the interface must be intuitive enough for a new user to learn basic interaction within five minutes.

The system will be implemented as a web application using the Next web framework and the Phaser game framework for the front-end gameplay experience, FastAPI for back-end

services, and PostgreSQL for persistent data storage. It will be accessible through modern web browsers and will integrate third-party services to achieve functional requirements. Development and deployment will adhere to data protection requirements comparable to FERPA standards, ensuring responsible handling of user account information. These constraints define the technical environment within which the system architecture and implementation decisions are made.

II. Implementation Overview

Virtual Flagstaff implements a solution that allows prospective Flagstaff residents to experience Flag with no required time or travel commitments. Players will be able to explore a Flagstaff map, experience the town through mini games and enjoy its history through quests and activities. Our restrictions include time constraints that restrict the potential complexity of the design of the map and trademarks which restrict what can and can not be displayed fully and accurately within the game.

Our solution vision includes the delivery of a scalable, and reliable web app that improves the user's accessibility to the information and experiences available in Flagstaff. To accomplish this our team will be developing Virtual Flagstaff with our client’s goals and expectations in mind in order to establish a technology stack that efficiently designs these objectives.

The technology stack includes a React front-end, with Phaser supporting game rendering and interaction. The back-end manages user authentication, user sessions, and game data. PostgreSQL will be used as a means of user data persistence, and finally an external Weather API will offer real-time in-game weather features.

Using these technologies, Peak Adventure Experiences will develop a web application game where users will have many helpful features available to them. Users will be able to sign up for personalized emails about Flagstaff events, tours, etc. if they so choose. Users can then log into the game wherever they left off previously with the game remembering their character location, their completed or started quests, mini game high scores, and more. The game will include NPCs that offer information about Flagstaff and hand out quests that give purpose to the player's exploration. Quests will encourage users to check out the many features built into the game such as mini games that demonstrate gamified Flagstaff activities. The game will include a helpful UI that shows users how to play/interact with the game as well as UI that shows real time Flagstaff weather. A pause menu will be developed that will allow users to change their volume settings, game controls, toggle tutorial pop ups/the mini map, and more. All of these features will come together to create a fun and interactive online experience of Flagstaff, Arizona.

III. Architectural Overview

While the implementation overview outlines what Virtual Flagstaff will deliver and how it will be built with the technologies discussed, a deeper understanding of how these components interact is essential. The following architectural overview will detail the structural design of the system while illustrating how the front-end, back-end, database, and external dependencies will interact with one another to create a scalable and secure web application.

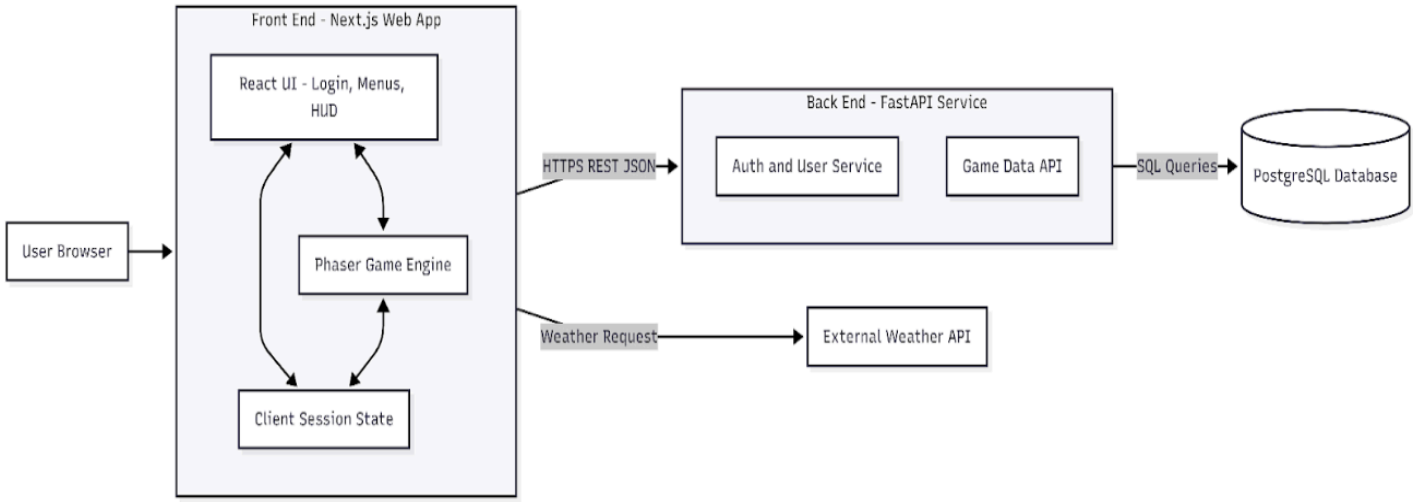

Figure 1: Virtual Flagstaff Architecture

Front-End: The primary responsibility is to deliver the full interactive web experience to the user's browser. React UI handles all the routing, login/registration, menus and allows everything to flow into the Phaser game engine where all the real time gameplay takes place. Also stores session information such as important account information and game data such as active quests, game currency, and high scores.

Back-End: Responsible for securely managing game and user data. Within the Back-End are the Auth and User services which are responsible for handling login/registration, session validation, and password hashing. The Game Data API provides endpoints for player progression, high scores, and game currency

Database: Responsible for storing user accounts (email, phone number, username, hashed password), game progression (unlockables, quest state, last zone), mini game scores, and survey data.

External Dependencies: Weather API - provides the current conditions in Flagstaff which gets displayed in the UI/game world

IV. Component-Level Design

Virtual Flagstaff is intended to be a website where users can create accounts and play a fun virtualized experience of the city of Flagstaff. In order to create the front-end and back-end for Virtual Flagstaff, numerous key components will need to be created for this virtual experience. The following components will need to be created:

Player: This component is how the user will interact with the game, as it will be the user's main source of control. The Player class is the player character. The Player class should contain at least a control component, an asset key, its current position, the scene it is in, and a State Machine class. Player classes for different scenes should iterate upon this base class, adding features that are relevant to that specific scene. The player class should contain functions to set the interacted object and return it, and a class to get the physics body of the object.

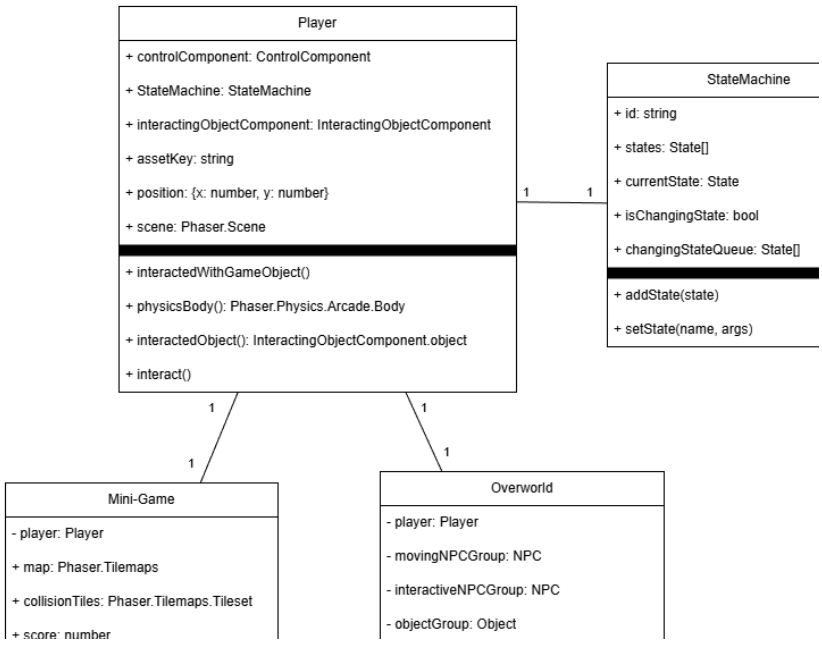

Figure 2: Front-end Player Class Diagram

NPC: This component is how the game will handle and control non-player characters. The NPC class is a modular class that creates non-player characters. Some of these can be interacted with while others aren't. They can be set to be able to move around areas. The NPC class should at the very least contain an asset key, its current position and scene, and a State Machine. NPCs can optionally have interaction components allowing them to be interacted with by the player.

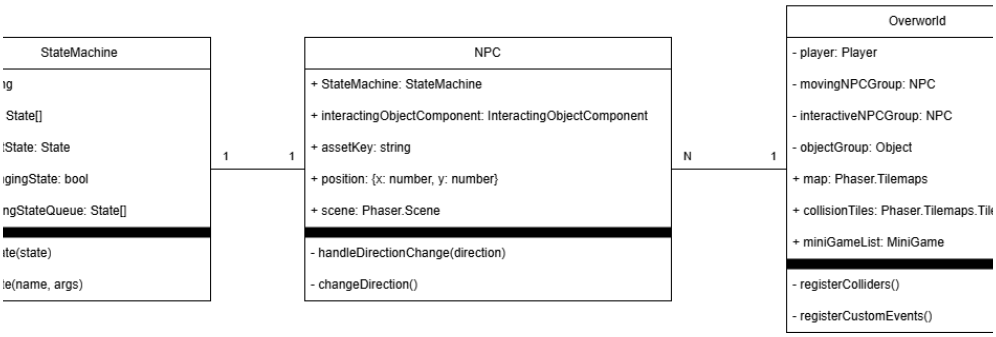

Figure 3: Front-end NPC Class Diagram

Object: This component is how the experience will handle static items. The Object class is a modular class that creates an interactable object. The types of interactions can be broken down into: Dialog, Key, or Mini-game. Dialog objects will open a dialog box, usually to tell the player something about the object. Key objects are objects tied to Quests and will usually disappear once interacted with. Mini-game objects will start their corresponding Mini-games when interacted with. Objects will also contain their asset key, position, and scene data.

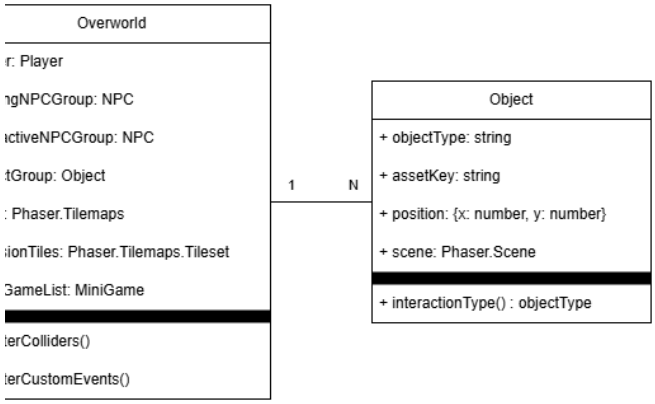

Figure 4: Front-end Object Class Diagram

State Machine: This component is a part of how the experience controls characters by managing what State they are in. The State Machine class is a component for any object that has multiple states. Every instance of the State Machine should have an ID, a map of states it can use, its current state, whether it is changing states, and a list of queued states. It should have functions to add states to use, to set its current state, and to check its current state.

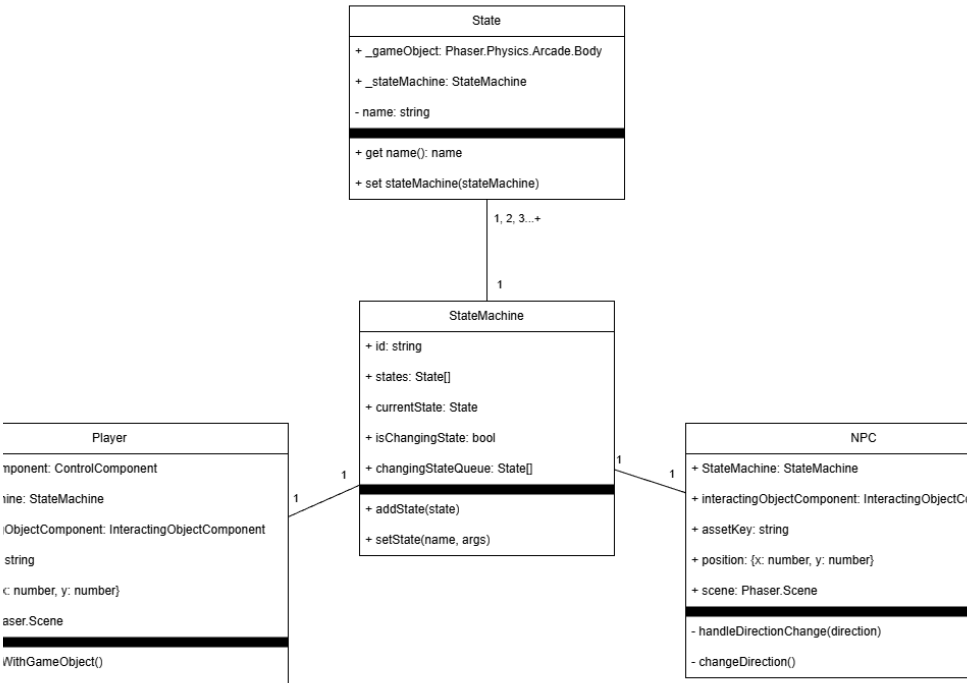

Figure 5: Front-end StateMachine Class Diagram

State: This component is how the experience sets how a character should act or look, it is what state they are in. The State class is a class made to be extendable by specialized states, such as a move state for the Player. The base State class should contain its name, and references to the State Machine and game object it is tied to. The class should have a function to set the State to a State Machine and a get function to return the State's name. Specialized states should expand upon this class to do a specific function, such as moving the object or displaying an idle animation.

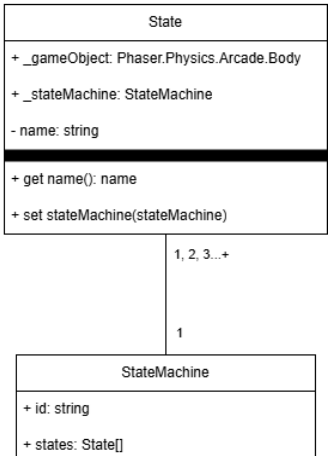

Figure 6: Front-end State Class Diagram

Overworld: This component is what ties the whole experience together, it is the main map that users will move around and interact with. The Overworld scene is the main scene of the experience, and where most other components are used. It should contain, either directly or indirectly via the other classes in it, most if not all of the classes developed. Independently it should contain the map data

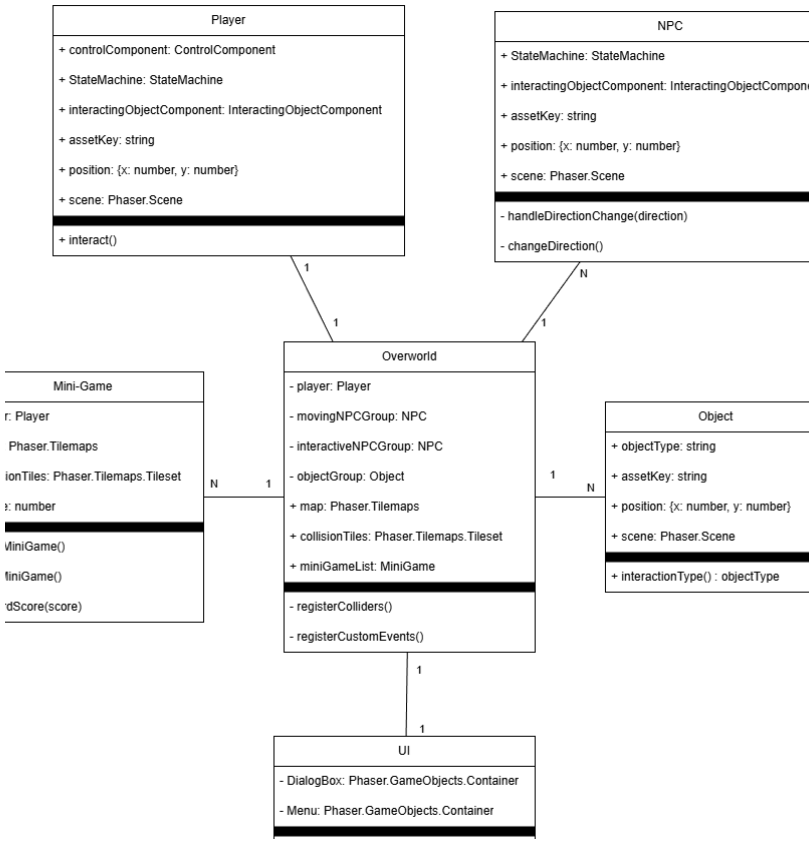

Figure 7: Front-end State Class Diagram

Mini-Games: This component is an individual mini-game, which users can play and enjoy. Each Mini-Game will be different, so this scene class exists as a template. It should contain a version of the Player class, map and score data, and start, end, and score recording functions. Each Mini-Game class will be drastically different from each other, requiring functions and variables unique to themselves. Each will also have ties to the Overworld scene class as they'll be called from the Overworld scene.

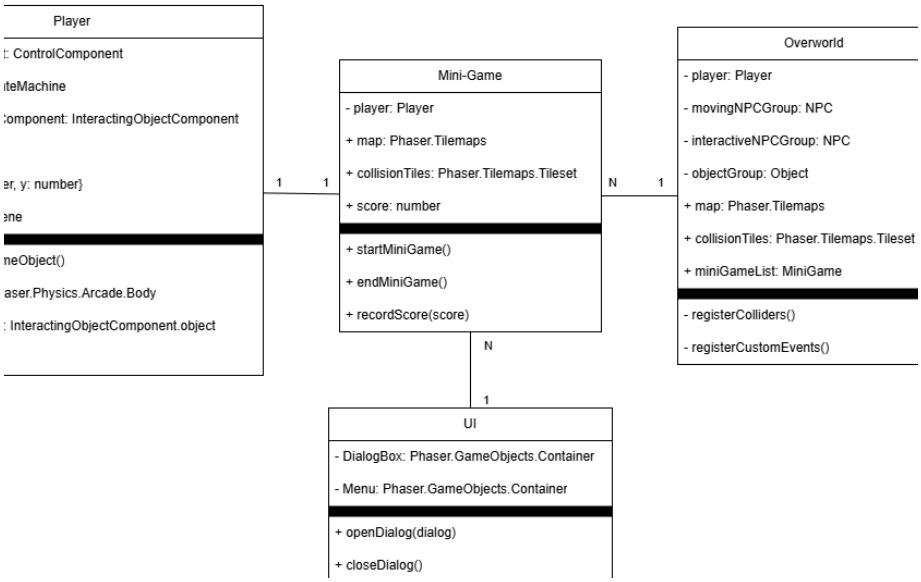

Figure 8: Front-end State Class Diagram

UI: This component is where most of the user interfaces reside, like settings and pause menus. The UI scene is where all the visual user interface should be. The UI scene should persist and be callable across all scenes as every scene will need a user interface. The UI scene should contain at least a dialog box and a menu, with corresponding functions to open and close those UI elements.

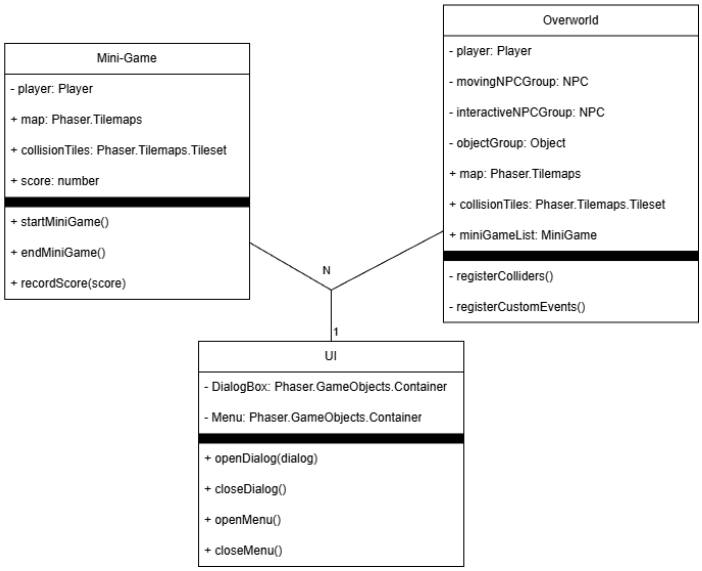

Figure 9: Front-end State Class Diagram

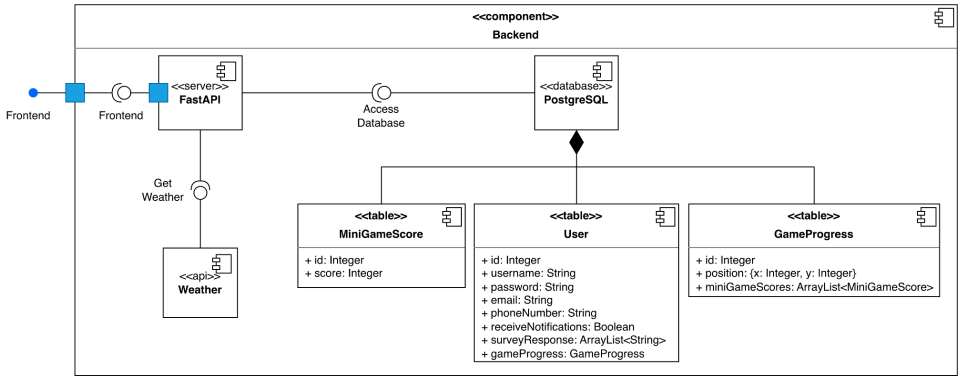

Figure 10: Back-end UML component diagram

V. Implementation Plan

The implementation plan outlines how the proposed system architecture will be realized through phased development, incremental integration, and milestone-driven delivery. Given the system's interactive, web-based, and game-oriented nature, development is structured to ensure early validation of core functionality while allowing parallel work across front-end and back-end components. This approach keeps the design grounded in execution and supports iterative refinement leading up to the final acceptance demo.

V.I. Development Phases and Milestones

The project will be implemented over approximately 12 weeks, concluding on May 1, 2026. Development is organized around three major milestones: Alpha Demonstration I, Alpha Demonstration II, and the Acceptance Demo. Each milestone represents a stable, integrated system state that incrementally expands functionality and user experience.

Time Frame	1/12/2026 - 2/13/2026	2/14/2026 - 3/20/2026	3/21/2026 - 5/1/2026
Phase	Core Systems	Feature Expansion	Final Integration
Main Menu	UI skeleton	UI polishing	UX refinement
	Authentication pages	Mini-map UI	Accessibility
Gameplay	Map	Progress tracking	Mini-games
	NPCs	Interaction	Content polish

Server	Authentication	Survey handling	Optimization
	Weather API	Game data storage	Scalability checks
Database	User tables	Progress & scores	Unlockables
	Survey data	Data integrity	Cleanup
Integration	Full System Communication	Full System Tests	Deployment Readiness
Deliverable	Alpha Demonstration I	Alpha Demonstration II	Acceptance Demonstration

Figure 4: Milestone roadmap

V.II. Phase Narrative and Key Dependencies

V.II.I Core Systems

This phase focuses on establishing the foundational components required for a functional virtual relocation experience. High-priority features implemented during this phase include user registration and authentication, real-time weather display, basic NPC interactions, survey data submission to the client. These elements form the backbone of the system and are necessary to validate the core architectural decisions early.

Server and database development occurs in parallel with the main menu and gameplay implementation. While one team member focuses on Next.js-based main menu structure and authentication interfaces, another develops the Phaser-based gameplay environment. Two team members collaboratively implement server APIs and database schemas, enabling early integration.

V.II.II. Feature Expansion

With core systems in place, development shifts toward expanding gameplay depth and user engagement. This phase introduces object interactions, a navigable map with a minimap, and improved NPC dialogue flows. Back-end services are extended to support persistent storage of scores and progress, while front-end components integrate these systems into the user experience.

Integration testing becomes more prominent during this phase, as multiple technologies are exercised together. Manual gameplay testing is used to validate end-to-end functionality and uncover cross-system issues.

V.II.III. Final Integration

The final phase emphasizes mini-games, game progress tracking, system stability, performance optimization, and user experience refinement. Additional explorable zones and unlockable content may be added if time permits, and existing content is polished. Performance tuning focuses on maintaining acceptable frame rates in the browser and ensuring responsive back-end interactions under concurrent user load. Deployment configuration and final integration checks prepare the system for demonstration.

V.III. Incremental Delivery Strategy

Each milestone produces a demonstrable, integrated build of the system. Alpha Demo 1 validates architectural feasibility, Alpha Demo 2 demonstrates feature completeness and user engagement, and the Acceptance Demo delivers a minimum viable product that includes a walkable map, NPC interactions, authentication, mini-games, weather display, survey-based customer data acquisition, progress tracking, and a minimap. Lower-priority features such as seasonal events and additional zones are deferred to protect schedule and system stability.

V.IV. Technical Risks, Trade-offs, and Mitigation Strategies

One key technical risk is the team’s limited prior experience with the Phaser game framework. This risk is mitigated by introducing Phaser early in Phase 1, allowing time for learning and refactoring before later milestones. Browser-based performance is another concern, particularly maintaining acceptable frame rates during gameplay. This is addressed through early performance profiling, asset optimization, and limiting scene complexity where necessary.

Concurrency and scalability pose additional risks, as multiple users may access the system simultaneously. While the system is not designed for massive scale, server APIs and database queries are structured to support at least ten concurrent users, with load considerations evaluated during final integration testing. Finally, reliance on multiple frameworks and APIs introduces integration complexity. This trade-off is managed through continuous integration testing and the use of Docker containers.

VI. Conclusion

This project aims to provide resources to *The Scouting Party* so that they can help people interested in relocating to Flagstaff, this project aims to provide a fun and informative experience so that individuals and families that are interested in moving to Flagstaff can learn more while having an enjoyable time, all without having to physically make a trip to the city. Additionally we hope that this experience will drive those who participate to want to connect to *The Scouting Party* for more information and experiences.

In order to achieve these goals, the system will allow users to register and sign in, play through the Virtual Flagstaff Experience, and save their progress and scores for future sessions. In order to make the experience interactive and enjoyable, there will be quests for the players, various explorable areas, retro-styled visuals and audio, real-time weather features, and a number of mini-games, each with their own gameplay and mechanics for players to experience and enjoy. In terms of performance the Virtual Flagstaff Experience must feel responsive to be enjoyable. The project should have internal system load times rarely exceeding two seconds, have gameplay that runs consistently at a minimum of 30 FPS, and have easy to learn systems with no less than 5 minutes for a user to understand the basics.

All of this will run within a web browser as a web application, with the Next web framework and Phaser game framework serving as the systems the project's front-end shall rely on. Next in combination with React will be used to create our webpages while Phaser will be used to create the virtual game experience. Our back-end will serve to store our user data and call the weather API for real-time information. A PostgreSQL database will be used to store login information, game progress, and mini-game scores. This database will communicate with a FastAPI server in order to relay and receive data back and forth from the game, additionally the FastAPI server will communicate with the external weather API so that it can send real time weather information to the front-end so it can be used to augment the game experience.

The plans outlined here will lead to the creation of a modular and easily expandable game experience, which should allow rapid development so that as many users can find ways to enjoy the Virtual Flagstaff Experience as possible. And with an incremental development plan in place there exists a clear roadmap to project completion and developmental success. And in case of difficulties or unforeseen circumstances, there have been mitigation strategies set in place to offset the troubles.