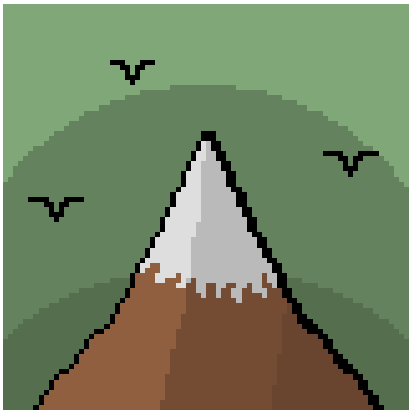
Peak Adventure Experiences



Tech Feasibility

Date: 10/24/2025
Team Members: Makaela Crookes, Yahir Espinoza, Alonso Garcia, Jack Morris
Project Sponsor: Paddy McGarry
Team Mentor: Ogonna Eli

Table of Contents

I. INTRODUCTION

In 2024, nearly 70% of Americans reported regrets about their relocation decisions, with 21% seeking a change in lifestyle and 18% pursuing a better quality of life [1]. Relocating is often a stressful process that involves evaluating numerous factors such as cost of living, weather, commute times, and local job opportunities. These challenges highlight a growing need for tools that can help prospective movers make confident, informed decisions about where to live.

Traditional relocation support relies heavily on realtors, who provide clients with housing options and basic information about local amenities. While this helps, it rarely captures the lived experience of a new city. Our client, Paddy, owner of The Scouting Party, aims to bridge this gap by offering guided tours that help prospective residents experience the authentic feel of Flagstaff before making a move. However, in-person tours can be limited by scheduling, travel costs, and seasonal availability, leaving many potential clients without access to this valuable experience.

To address these challenges, our team is developing an interactive, gamified relocation platform that virtually showcases the experience of living in Flagstaff. The environment will feature mini-games, interactive non-playable characters (NPCs), and dynamic locations designed to convey essential information about daily life, like local weather and cultural landmarks. This immersive approach aims to provide users with an engaging and informative experience that supports confident, stress-free relocation decisions.

Our solution will enhance The Scouting Party's ability to engage clients and build confidence in their relocation decisions. By providing an accessible, virtual preview of the Flagstaff lifestyle, prospective residents can explore and learn at their own pace — reducing stress, uncertainty, and post-move regret. Ultimately, this tool is expected to expand client reach, strengthen decision-making satisfaction, and position The Scouting Party as a pioneer in digital relocation experiences.

Now that we have established the motivation and goals of our project, we turn our attention to its technical feasibility. At this early stage, our team is focused on identifying the major technological challenges, evaluating potential solutions, and determining the most effective approaches for implementation.

In this Technological Feasibility Analysis document, we begin by examining the key technological challenges anticipated during development in Section II. In Section III, we conduct a technology analysis, where we compare possible software tools, frameworks, and design approaches suitable for our gamified relocation platform. Section IV covers technology integration, describing how the selected components will interact within the final system

architecture. Finally, Section V presents our conclusions and summarizes the next steps for project development.

II. TECHNOLOGICAL CHALLENGES

Developing an interactive gamified relocation experience introduces several challenges that must be addressed early in development to ensure a smooth scalable development. For this project we will integrate web technologies, dynamic game elements, and real time data. Each of these components must integrate seamlessly while still maintaining strong performance standards. Major technological challenges include:

- **Front-end**
  - **Robust web framework**
    - To manage routing, dynamic loading, and API communication while hosting
  - **Asset optimization:**
    - Managing and compressing models, textures, and UI elements to maintain fast loading times across all devices.
  - **Cross platform compatibility:**
    - Compatibility for the app with both desktop and mobile devices is essential.

- **Back-end**
  - **Extracting API data**
    - Obtaining API data like real-time weather will help create the setting of the game
  - **Scalability**
    - Making a scalable workflow will help make more features to the project, like sending customer data to the client
  - **Efficiency**
    - Providing the users with an efficient product will allow them to play the product and provide a smooth experience

- **Databases**
    - **User login capabilities**
        - Allowing users to have the option to log in and receive further promotions for Flagstaff helps The Scouting Party expand their business

    - **Data integrity**
        - Having a database that maintains data integrity to ensure the correctness, consistency and reliability of the stored data promotes trust in the database

    - **Scalability**
        - Being able to handle increased demand upon potential expansion without losing efficiency is key for growing to handle new locations

- **Game Design**
    - **An Interactive 2D/2.5D map**
        - Selecting the appropriate rendering engine will be crucial in handling graphics and assets

    - **Game-like UI and Quest System Integration**
        - Set up a UI that allows users to interact with key gameplay features such as NPC dialogues, quest progression, and item collection.

    - **Responsive Cross platform performance**
        - Be able to perform as well as or better than desktop implementation

It is necessary to address these technological issues to maintain a reliable and immersive experience for Virtual Flagstaff. The front-end will deliver responsive visuals and engaging interactions, while the back-end creates a reliable and consistent data stream for gameplay and dynamic content. The database must maintain accuracy while providing scalability and security as the user's engagement continues to grow. Lastly, the game design will fully incorporate visual assets, interactions, and optimization of performance into an engaging experience, giving users an immersive experience of our virtual world.

III. TECHNOLOGY ANALYSIS

In this section, we evaluate several candidate technologies intended to address the core technical challenges of our system: building a front-end, back-end, database, and game design. Section A examines front-end frameworks that provide scalability, modularity, and responsive performance. Section B evaluates back-end technologies that enable reliable and efficient real-time data processing. Section C analyzes database solutions designed to maintain easy data

access in support of real-time features. Finally, Section D reviews game design frameworks for implementing an interactive 2D/2.5D environment with game-like elements while preserving cross platform compatibility.

A. Front-end

1) Intro: The client envisions a virtual world where players can engage with NPCs, play an assortment of mini games, and interact with different points of interests found throughout the gamified relocation experience. In order to achieve this, the front-end must balance performance and immersion, providing a seamless experience across both desktop and mobile devices. Selecting the right front-end frameworks and rendering engines are crucial not only for achieving the desired visual quality but also for ensuring scalability and maintainability.

2) Desired characteristics: For the front end of this project, the development team identified several critical characteristics necessary to deliver an immersive, smooth, and accessible experience for users exploring the gamified relocation world. Since our system will blend web design, interactive 2D/2.5D rendering, and real-time user engagement, the front-end framework must support both visual fidelity and technical scalability. The following characteristics are prioritized:

- **Support real time rendering**
    - Smooth animation layers with basic depth effects to create a sense of immersion.

- **Cross-Platform Compatibility**
    - Must run efficiently in browsers without additional plugins or installations.

- **Lightweight performance**
    - Fast load times, optimized assets, and minimal lag during interaction.

- **Integration with Web Frameworks**
    - Must easily connect with a backend API for dynamic data exchange.

3) Alternatives: To identify the best solution for implementing the Virtual Flagstaff user interface, several front-end technologies were evaluated. Each framework or library provides a unique set of capabilities related to rendering performance, interactivity, and ease of integration with other components such as FastAPI and PostgreSQL. Below is an overview of the primary alternatives considered for the project:

- **Next.js**
  - A react based framework used to create high quality web applications and was developed by the Vercel team in October 2016. Next.js has been used by big name companies such as Nike, OpenAI, and Netflix.

- **Phaser.js**
  - A fast and free open source HTML5 game framework and was developed by Richard Davey in April 2013. Phaser.js has been used by many indie developers for its fast and easy to learn.

- **Three.js**
  - A cross-browser JavaScript library and API that can be used to render 2.5D/3D graphics and was developed by Ricardo Cabello in April 2010 and has been used for Game development, Marketing, and Data Visualization.

- **PixiJS**
  - An HTML5 creation engine that is used to create interactive 2D games and applications. It was originally created by Mat Groves in February 2013. Pixi.js has been used by big tech companies such as Google, Cloudflare, and NVidia.

4) Analysis: With the potential frameworks identified, the team conducted a hands-on evaluation of each option based on the desired characteristics established earlier—rendering capability, ease of use, integration potential, and mobile performance. Each framework was tested using a simple "Hello World" implementation to assess setup complexity, rendering output, and responsiveness.

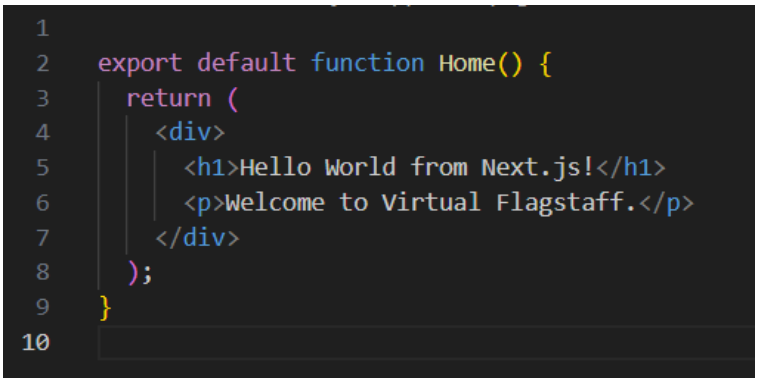


Figure 1. Next.js code

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Phaser Hello World</title>
    <script src="https://cdn.jsdelivr.net/npm/phaser@3/dist/phaser.js"></script>
  </head>
  <body>
    <script>
      const config = {
        type: Phaser.AUTO,
        width: 800,
        height: 600,
        backgroundColor: "#2d2d2d",
        scene: {
          create
        }
      };

      const game = new Phaser.Game(config);

      function create() {
        this.add.text(300, 250, "Hello World", {
          font: "32px Arial",
          fill: "#ffffff"
        });
      }
    </script>
  </body>
</html>
```

Figure 2. Phaser.js code

```javascript
import * as THREE from "https://cdn.jsdelivr.net/npm/three@0.158.0/build/three.module.js";
import { FontLoader } from "https://cdn.jsdelivr.net/npm/three@0.158.0/examples/jsm/loaders/FontLoader.js";
import { TextGeometry } from "https://cdn.jsdelivr.net/npm/three@0.158.0/examples/jsm/geometries/TextGeometry.js";

// Scene setup
const scene = new THREE.Scene();
const camera = new THREE.PerspectiveCamera(75, window.innerWidth / window.innerHeight, 0.1, 1000);
const renderer = new THREE.WebGLRenderer({ antialias: true });
renderer.setSize(window.innerWidth, window.innerHeight);
document.body.appendChild(renderer.domElement);

// Lighting
const light = new THREE.DirectionalLight(0xffffff, 1);
light.position.set(5, 5, 5);
scene.add(light);

// Load font and create 3D text
const loader = new FontLoader();
loader.load("https://threejs.org/examples/fonts/helvetiker_regular.typeface.json", (font) => {
  const geometry = new TextGeometry("Hello World", {
    font: font,
    size: 1,
    height: 0.2,
  });
  const material = new THREE.MeshStandardMaterial({ color: 0x00ffcc });
  const textMesh = new THREE.Mesh(geometry, material);
  textMesh.position.set(-3, 0, 0);
  scene.add(textMesh);
});

// Camera position
camera.position.z = 5;

// Animation loop
function animate() {
  requestAnimationFrame(animate);
  scene.rotation.y += 0.005; // slow rotation
  renderer.render(scene, camera);
}
animate();

// Handle resizing
window.addEventListener("resize", () => {
  camera.aspect = window.innerWidth / window.innerHeight;
  camera.updateProjectionMatrix();
  renderer.setSize(window.innerWidth, window.innerHeight);
});

```

Figure 3. Three.js code

index.html
```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Pixi.js Hello World</title>
</head>
<body style="margin:0; overflow:hidden; background-color:#202020;">
  <!-- Use PixiJS v7 for compatibility -->
  <script src="https://cdnjs.cloudflare.com/ajax/libs/pixi.js/7.3.0/pixi.min.js"></script>
  <script src="src/main.js"></script>
</body>
</html>

```

src / main.js
```javascript
// Create a new Pixi Application
const app = new PIXI.Application({
  width: 800,
  height: 600,
  backgroundColor: 0x1099bb
});

// Add the canvas to the page
document.body.appendChild(app.view);

// Create a text object
const message = new PIXI.Text('Hello, World!', {
  fontFamily: 'Arial',
  fontSize: 48,
  fill: 0xffffff,
  align: 'center'
});

// Center the text
message.anchor.set(0.5);
message.x = app.screen.width / 2;
message.y = app.screen.height / 2;

// Add it to the stage
app.stage.addChild(message);

```

Figure 4. Pixi.js code

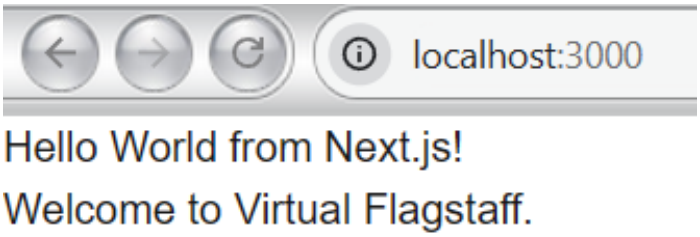


Figure 5. Next.js output

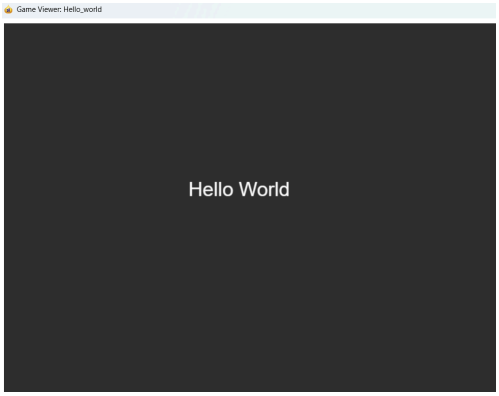


Figure 6. Phaser.js output

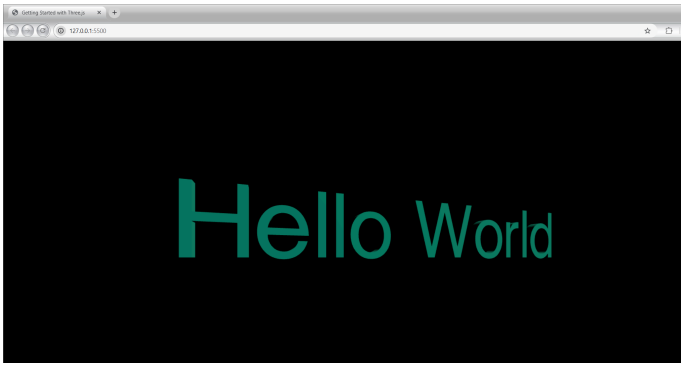


Figure 7. Three.js output

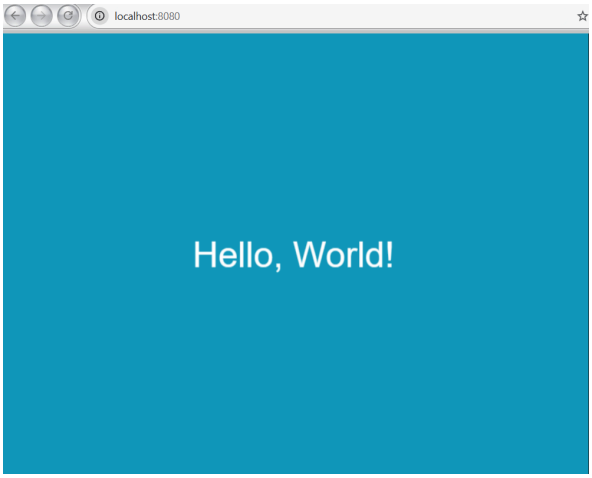


Figure 8. Pixi.js output

5) Chosen Approach: In order to find the framework that best suits the needs of the project, three categories were tested: Rendering capabilities, Ease of use, and Mobile compatibility. Based on the results of testing each framework a conclusion was made that best suits the needs of this project. Some frameworks like Next.js and Pixi.js excelled in some aspects, but fell short in others. However, from testing each framework and grading them, there were two frameworks that excelled in all three categories.

In Table 1, both Three.js and Phaser.js had optimal rendering capabilities, were easier to use in comparison to the other frameworks, and offered easy compatibility with mobile devices. The team believes the combination of Three.js and Phaser.js could serve as a modular architecture for the project, where Three.js can be used to build the core interactive world and Phaser.js can serve as the mini-game layer for points of interest found throughout the interactive world.

Table 1: Front-end comparison table

	Rendering	*Ease of use*	*Mobile compatibility*
Next.js			
Phaser.js			
Three.js			
Pixi.js			

6) Proving feasibility: To verify the feasibility of the hybrid front-end design, the team will develop a prototype that integrates Three.js and Phaser.js within the same web environment.

The Three.js prototype will render a navigable 2D/2.5D map of Flagstaff, allowing users to select locations that dynamically launch Phaser.js mini-games. This will demonstrate smooth transitions between 2.5D exploration and 2D gameplay, as well as efficient resource management between the two rendering contexts. Additional testing will measure frame rate performance, load times, and responsiveness across desktop and mobile devices. Integration with the chosen back end will further validate data exchange for player progress and quest tracking. These evaluations will confirm that the selected technologies can deliver the responsive, accessible, and immersive experience envisioned for Virtual Flagstaff.

B. Back-end

1) Introduction: One of the primary technical challenges identified for this project is the lack of a structured method for collecting and managing customer data once individuals enter the client's business. Currently, there is no back-end infrastructure in place to store customer information, support future engagement, or integrate dynamic features that enhance user experience. To address this, the project requires the development of a backend server capable of securely logging customer data while also supporting additional interactive features, such as displaying real-time weather conditions in Flagstaff. This backend component will act as the core service layer of the application, enabling communication between the user-facing interface and stored business data, and will determine the system's long-term scalability, responsiveness, and maintainability.

2) Desired characteristics: To evaluate back-end technologies for this project, characteristics are necessary to ensure that the final solution can support both the functional and experiential goals of the product. The backend will not only collect and store customer information, but also serve as the engine for interactive features such as real-time weather display and future user-facing modules. The following characteristics were identified as critical for selecting the most appropriate technology:

- **Performance / Fast Response Time**
    - The application must provide real-time feedback to users and respond quickly to interaction events such as data submission or display updates. A slow backend would negatively impact user engagement and could cause users to lose interest before interacting with the experience. Therefore, low latency and high throughput are key priorities.

- **Scalability / Future Expandability**
    - The client has expressed interest in expanding functionality over time, including potentially logging new types of customer interactions, integrating recommendations, or exposing additional tourist-related features. The chosen

framework must support scalable architecture so the system can grow without requiring a full rewrite.

3) Alternatives: To address the need for a backend capable of collecting customer data, integrating real-time features, and supporting scalable future functionality, two viable technologies were identified: Express.js and FastAPI. Both provide modern backend frameworks suitable for API-driven architectures, but they differ in ecosystem, language support, and performance characteristics.

- **Express.js**
  - A lightweight and widely used backend framework built for Node.js. It was originally released in 2010 and is currently maintained under the OpenJS Foundation. It is one of the most commonly used frameworks in JavaScript-based backend development due to its minimalistic design and extensive middleware ecosystem. Express.js was initially considered for this project because it has been taught and used extensively in coursework within Northern Arizona University's Web Development curriculum, making it familiar to student development teams. It is typically used in RESTful APIs, web servers, single-page application backends, and real-time services through integrations such as Socket.io. Its long community history and large package ecosystem make it a proven, production-ready solution.

- **FastAPI**
  - Introduced in 2018 by Sebastián Ramírez and built on top of Starlette and Pydantic. It was recommended early in the project scoping phase as a potential alternative due to its performance advantages and strong developer productivity features. FastAPI has rapidly gained adoption among backend engineers, data service developers, and machine learning practitioners because it enables quick API construction with minimal boilerplate while maintaining high performance. It is also widely used in microservices, data pipelines, cloud-hosted APIs, and systems that integrate with Python-based analytics or machine learning components.

4) Analysis: Each framework was evaluated based on ease of use, adaptability to multiple project components, and overall performance. To test this, a simple endpoint was implemented in both Express and FastAPI to compare code structure, development experience, and runtime behavior.

From the code examples shown in Figure 9 and Figure 10, the Express implementation requires slightly more boilerplate code than the FastAPI example. However, Express provides

more direct control of port selection by default, which can be convenient during development or deployment. FastAPI, by contrast, typically relies on additional server capabilities to configure and run the application, which introduces another layer of tooling but also improves performance in high-concurrency environments.

In Figure 11 and Figure 12, both servers successfully return a response message to the caller, demonstrating that each framework can handle a basic request-and-response interaction. At this small scale, there is no substantive difference in performance, but differences in architecture and development workflow become more relevant as the backend grows in complexity.

```
1  const express = require('express');
2  const app = express();
3  const port = 3000;
4
5  app.get('/', (req, res) => {
6      res.send('Hello from Express!');
7  });
8
9  app.listen(port, () => {
10     console.log(`Server listening at http://localhost:${port}`);
11 });
```

Figure 9. Express code

```
1  from fastapi import FastAPI
2
3  app = FastAPI()
4
5  @app.get("/")
6  def read_root():
7      return {"Hello": "World"}
8
```

Figure 10. FastAPI code

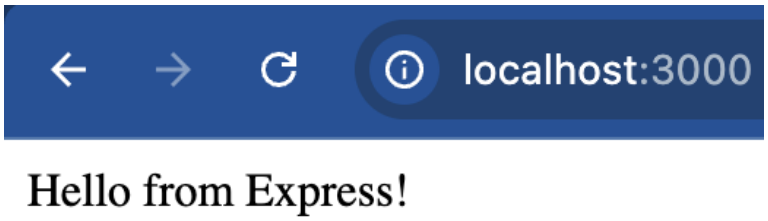


Figure 11. Express output

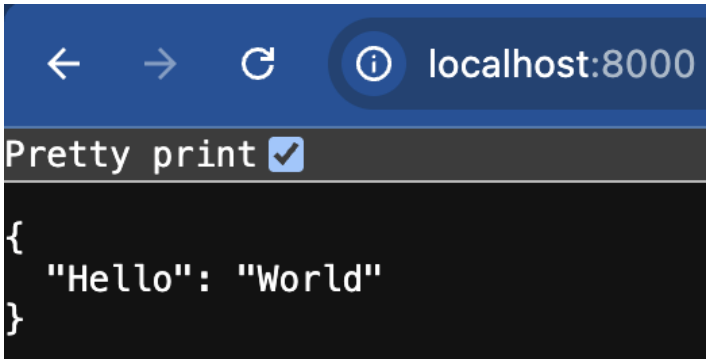


Figure 12. FastAPI output

5) Chosen Approach: Both Express and FastAPI are scalable frameworks capable of expanding as the backend grows in complexity. However, FastAPI demonstrates better performance characteristics during testing, offering lower latency and higher requests per second than Express [2]. While Express is also scalable and widely used in production systems, it required slightly more boilerplate code to achieve equivalent functionality. FastAPI, by contrast, integrates asynchronous performance by default and fits more naturally into the Python ecosystem, making it faster to implement and extend for this project.

In Table 2, the two backend alternatives are compared against the desired characteristics. Both Express and FastAPI satisfy the scalability requirement, as each framework can be extended to support additional routes, services, or modules as the system evolves. However, Express falls short in the performance criterion when compared to FastAPI. FastAPI demonstrates higher throughput and lower latency in typical API workloads, which makes it better suited for real-time interaction and fast response delivery to users. For this reason, FastAPI is the preferred option moving forward, as it meets all of the identified characteristics more effectively for this project.

Table 2. Back-end ranking table

	Express	FastAPI
Scalable		
Efficient		

6) Proving feasibility: To ensure that FastAPI is the most suitable backend framework for this project, further testing will be conducted beyond the initial prototype. Upcoming evaluation will include testing FastAPI's integration with a database for storing customer information, as well as measuring real response time under real-world usage conditions once it is

implemented within the full system. Basic request handling has already been validated, as shown in Figure 12; the next steps will involve connecting the selected frontend to confirm seamless communication between client and server. These tests will help verify that FastAPI can reliably support the interactive features planned for the final product.

C. Databases

1) Intro: The client wants to have accessibility to reach out to prospective customers, as such the use of a database to maintain user information is an important aspect of the application. Not only will the use of a database allow for the storage of user information for the means of extending offers and extra information to clients, but it will offer the opportunity to store a user's progress to allow for a more enjoyable experience.

2) Desired characteristics: The desired characteristics that we wish to see in the database is something that would… Ideally be programmed through SQL commands, SQL is a common and easy to read language that is powerful when dealing with structured data and offers reliability and consistency (two essential aspects of the database). We also wish to see at least some scalability for potential future expansion of the project, as well as a reliable form of data integrity so that we can maintain the accuracy, reliability and consistency of the data being stored. The last prioritized characteristic of the database is that it is open source which allows for free access on top of the flexibility, control, security and transparency commonly offered by open source resources.

3) Alternatives: In order to make sure the desired characteristics were met, multiple potential technologies were researched and by the end 3 technologies were selected for analysis. PostgreSQL, mySQL, and mongoDB met most of the criteria during the research phase so they were selected as the top 3 alternatives and chosen for further analysis.

- **PostgreSQL**
    - A free and open source object-relational database management system, compatible with several operating systems, offers inheritance and function loading which enhances extensibility reliability and data integrity (Primary Keys, Foreign Keys, Explicit Locks, Advisory Locks, and Exclusion Constraints), numerous constraints that ensure data integrity, supports various useful SQL features (Multi-Version Concurrency Control, SQL Sub-selects, complex SQL queries, Streaming Replication, etc.), compatible with several data types (Structured, Primitives, Customizations, Geometry, and Documents), Highly extensible (JSON/SQL path expressions and Stored procedures and functions).

- **MySQL**
    - An open source relational database management system, highly scaleable, runs on multiple platforms, easy deployment and management, different memory caches to maintain and administer servers, can be configured with any programming language, essential functionalities with high-performance results, complete data security.

- **MongoDB**
    - An open source document-oriented database, can be used to store high-volume data, scalable and flexible database platform, offers flexibility through horizontal scaling and load balancing capacities, faster than an RDBMS, highly scalable, has high availability through replica sets, flexible and adaptable, in case of hardware failure its spread across multiple servers allows the system to continue running (balancing the load or duplicating data), the data model enables users to represent hierarchical relationships and store arrays as well as other complex data structures seamlessly.

4) Analysis: The analysis was done through a brief testing of: the database management tools and their ability to create/populate tables, along with testing how a .js file might be used to insert data into the database. The databases were judged mainly on ease of use.

PostgreSQL offers the pgadmin tool for database management. As seen in Figure 13, it uses SQL to create, populate and show tables. Pgadmin is an easy and somewhat intuitive tool to use. Figures 14 and 15 demonstrate an ability to connect to postgreSQL databases through javascript and use commands in that script to create, populate and draw values from tables. Figure 16 shows that the connection between the code and database successfully transfers the data between the two.

MySQL has the mySQL tool for manual database management. Figure 17 demonstrates the tools ability to (similarly to postgreSQL) create and populate tables with simple SQL commands. The mySQL tool is not as easy to use as pgadmin but offers many similar features. Figures 18 and 19 show the ability to connect to the database through javascript and once again create and populate a table. Figure 20 provides proof of the connection between the code and the database itself.

MongoDB offers a tool called mongoDB compass. This tool uses a JSON/CSV file import or a document insert to populate tables or as mongoDB calls them, collections. Figure 21 shows the formatting of a document and the output provided by mongoDB. Figures 22 and 23 show that you can use a .js file to insert data into the database and that the connection between the two successfully updates the database.

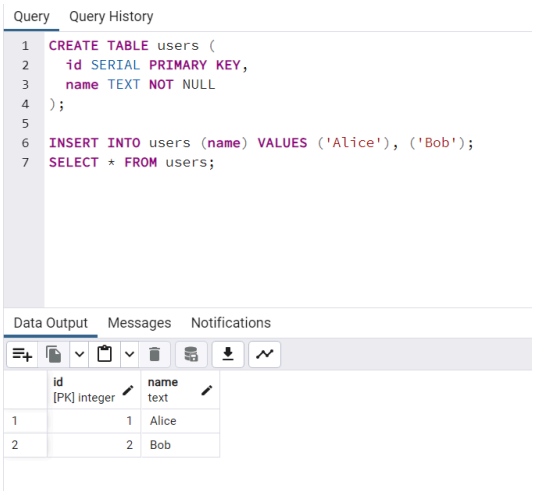


Figure 13. postgreSQL's pgadmin table creation/population

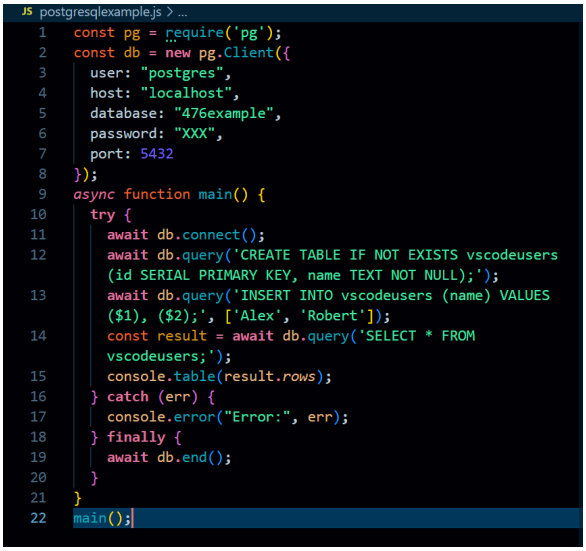


Figure 14. vscode connection to PostgreSQL: table creation/population

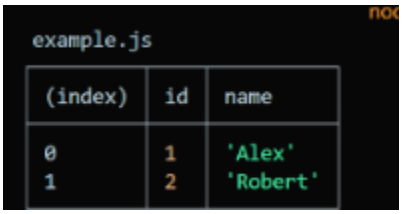


Figure 15. Terminal output of Figure 14

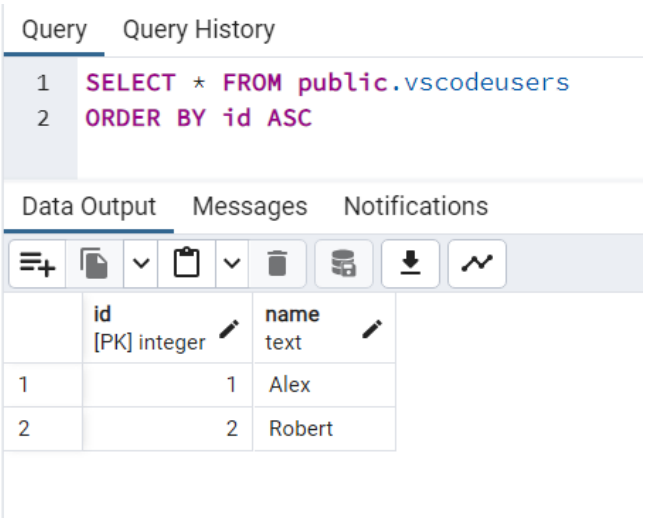


Figure 16. output view of Figure 14 from pgadmin

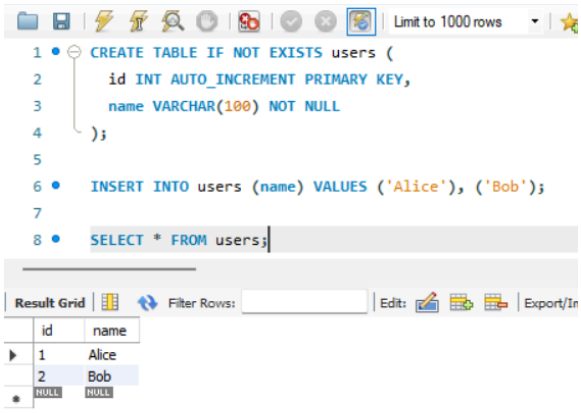


Figure 17. mySQL table creation/population

```
const mysql = require('mysql2/promise')
async function main() {
  const db = await mysql.createConnection({
    host: 'localhost',
    user: 'root',
    password: 'XXX',
    database: 'mysqlexample'
  });


  try {
    await db.execute('CREATE TABLE IF NOT EXISTS vscodeuser
    (id INT AUTO_INCREMENT PRIMARY KEY, name VARCHAR(100)
    NOT NULL);');
    await db.execute('INSERT INTO vscodeusers (name) VALUES
    (?), (?);',['Alex', 'Robert']);
    const [rows] = await db.execute('SELECT * FROM
    vscodeusers;');
    console.table(rows);
  } catch (err) {
    console.error("Error:", err);
  } finally {
    await db.end();
  }
}

main();
```

Figure 18. vscode connection to mySQL: table creation/population

```
>>
```

(index)	id	name
0	1	'Alex'
1	2	'Robert'

Figure 19. Terminal output of Figure 18

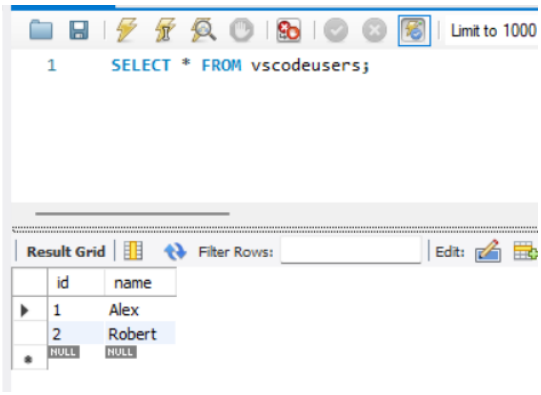


Figure 20. output view from mySQL of Figure 18

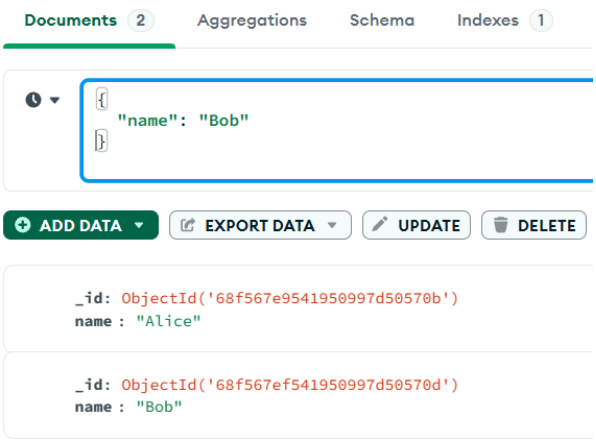


Figure 21. mongoDB compass table creation/population

Figure 22. vscode connection to mongoDB: table creation/population

Figure 23. output view from mongoDB compass of Figure 22

5) Chosen Approach: Through the analysis mongoDB was taken off of the consideration list for potential databases. While mongoDB compass offers intuitive navigation, the required input formats are not as easy to use as SQL queries are, which postgreSQL and mySQL both use. Between postgreSQL and mySQL, both offer the use of SQL that was hoped for, they are both

scalable, they both offer data integrity, and they are both open source free to use tools. The main thing that sets these two databases apart is pgadmin's easier/more intuitive UI.

The following Table 3 shows that the databases were able to hit most of the desired criteria, with only mongoDB falling short on a few of them. The final deciding factor was the ease of use, which postgreSQL offered more than mySQL was able to. Moving forward postgreSQL will be our preferred technology for our database management.

Table 3. Database ranking table

	mySQL	*postgreSQL*	*mongoDB*
SQL Use			
Scalability			
Data Integrity			
Open Source			

6) Proving feasibility: In future analyses of the database, more in depth checks for data integrity will take place along with general testing of how the database will be able to interact with and provide the features we require for the development of Virtual Flagstaff. The most important testing that will take place is checking the compatibility between the database and the other chosen technologies. The testing that has been done to check for ease of use and the existence of the basic requirements/desires will provide a strong foundation for the upcoming validation of the database.

D. Game Design

1) Intro: The client is currently looking to get customers in the door and drive them to the business. To do so we will create an interactive 2D game world on a webpage that users looking to move can play in and learn some facts about Flagstaff in hopes of driving them to our Client's business. In order to create such a game we'll need to use an available game engine or game framework in order to easily and effectively implement our game on a webpage.

2) Desired characteristics: For any game engine or framework, we broke down our characteristics into two categories. The first are the required features, these were called The Basics, as they're something every good framework or engine should natively support. These features were: object and object movement, collision/simple physics, audio control, scene changes, and a movable camera. These are all key to any game and ours would be no different, we needed to be able to make things and move them, detect collisions and have simple physics,

play music and sounds, change scenes when moving between game areas, and have a camera that could move with the player. The second set of categories we called The Advanced, as they were all things we would like the framework to be able to do, but weren't necessarily a requirement. These features were: creating minigames, interactive NPCs, dialogue/dialogue options, and mobile/accessibility controls.

3) Alternatives: We had a few options after cutting out the frameworks that didn't possess all The Basics characteristics that we needed. These four remaining options for frameworks were: Phaser, P5.JS, Excalibur.JS, and Kaplay.JS. As for game engines, we ran a comparison between game frameworks and game engines and eventually came to the conclusion that while a game engine would ultimately be easier to create the game in, using a framework would be better for us due to there being so few webpage first engines and that coding using frameworks would ultimately make for a better learning experience that letting the engine handle a lot of the back-end work for us. Along with that it would be much easier for us to implement API and database integrations while using a game framework than using a game engine.

- **Phaser**
    - A Javascript framework that is designed specifically for making 2D web-games. Phaser was created by Richard Davey that we found through forums and from what we found online, it seems to be pretty popular within the web-game development scene. It was released in 2013 and has received continued support and iteration since its initial release, with it currently being called Phaser 4.

- **P5.JS**
    - A Javascript framework that specializes in 2D and 3D rendering on web pages. It was created by Lauren McCarthy and released in 2013. We found out about it in a forum and it appears to be fairly popular in web-game development as you can create your own physics and game engine while not having to develop rendering functions.

- **Excalibur.JS**
    - A TypeScript framework created for web-game development. It was created by Erik Onarheim in 2013. It isn't as popular as the other options but it came up once or twice in forums about web-game development. It still receives updates but it has yet to hit version 1.0.0.

- **Kaplay.JS**
    - A Javascript game development framework. It was created by the community of the now defunct Kaboom.JS library, which was created by Tiga Wu. It was released on May 1st, 2025, so it is the newest of the frameworks, despite this it is

quite in-depth and provides a lot of great features. Kaplay.JS has been quite rapidly growing in popularity, as it appeared very often among the web-game development forums, only behind Phaser in popularity.

Figure 24. Example Game Running in Phaser

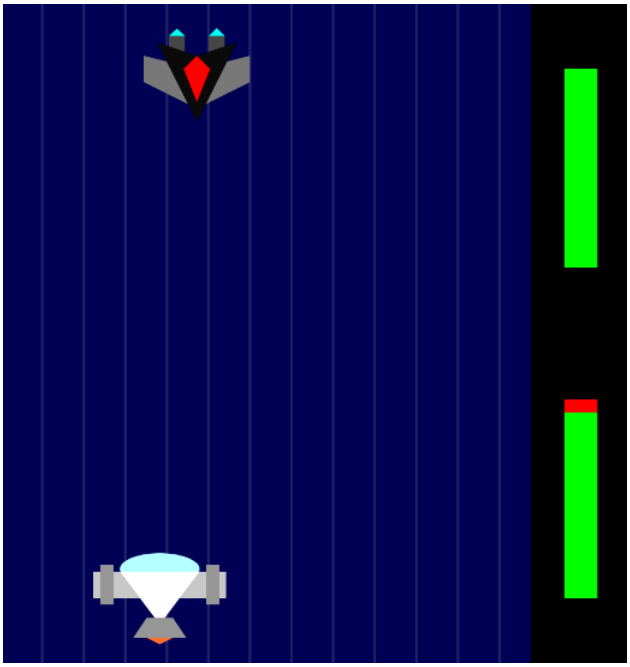


Figure 25. Example Game running in P5.JS

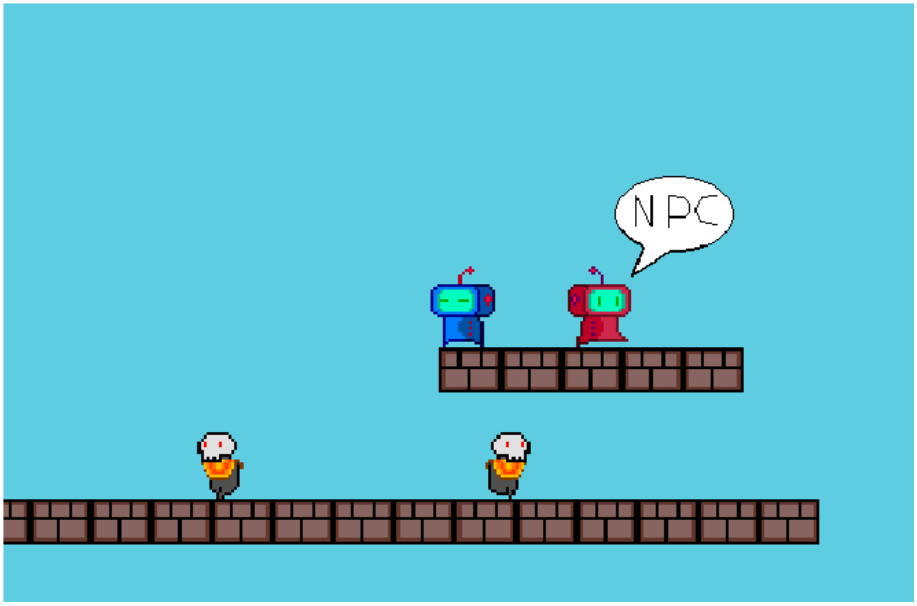


Figure 26. Example game running in Excalibur.JS

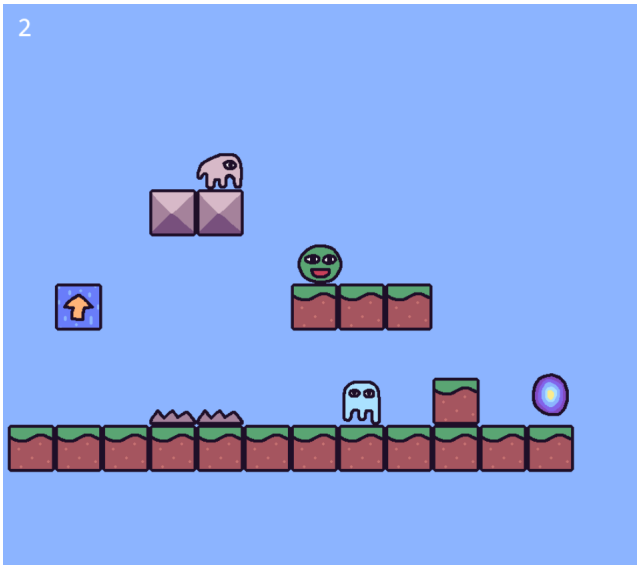


Figure 27. Example game running in Kaplay.JS

4) Analysis: To analyze the four options, we created a table with our requirements, The Basics and The Advanced. Following that we graded each of the categories for each framework by shading each framework red, yellow, or green. A red shading represented below expectation performance, such as missing functions, long implementation, or added difficulty. A yellow shading represents the expected performance. And a green shading represents above average performance, such as easy implementation, expanded functionality or options, and clear documentation or examples.

Table 4. Game Frameworks

		Phaser	P5.JS	Excalibur.JS	Kaplay.JS
The Basics	Objects				
	Movement				
	Collision/Physics				
	Audio				
	Scene Changes				
	Moveable Camera				
The Advanced	Creating Minigames				
	Tile Maps				
	Interactable NPCs				
	Dialogue Options				
	Mobile/Accessibility				

5) Chosen Approach: Table 4 shows the results of our analysis of the four frameworks. P5.JS falls behind in several categories, including in some of the basics, which is partly due to it being more of a rendering tool rather than a proper game framework. Meanwhile Excalibur.JS excelled in some of The Basics and fell behind the The Advanced categories. The Phaser and Kaplay.JS frameworks meanwhile are the two standouts among the four, as neither fall behind in anything and excel in different sections.

As such P5.JS was the first of the four to be cut due to it being more of a rendering tool than a game framework and therefore falling behind in several categories. Excalibur.JS was the next one we cut due to falling behind both Phaser and Kaplay.JS in The Advanced categories, despite doing great in The Basics. This left Phaser and Kaplay.JS as the last two contenders. In the end we decided to go with Phaser due to having more extensive documentation, more examples, and excelling slightly more in The Basics and in a more important Advanced category in ‘Tile Maps’.

6) Proving feasibility: In order to get a very basic grasp to ensure that Phaser works for what we’ll need, Jack put together a very basic tech demo that used features to test all of the basic features that we need. Along with that Jack found a number of tutorial videos by the

Youtube channel named Scott Westover that have fully functioning example files that display a number of functions that are similar in nature to what we'll require for our own project.

IV. TECHNOLOGY INTEGRATION

With using all these technologies we have to combine them into a sustainable and cohesive product. We'll take all the individual solutions and add them all together into a singular web site, on which the front-end hosts the back-end and displays the game, in which you can sign-in through the back-end and the database to allow for smoother gameplay and score saving. In the end the project's architecture should look something like this:

In Figure 24, starting with the main container, the front-end. The front-end exists to serve the game to our users and cover up the back-end work. The game itself is largely a stand-alone ingredient, however it will communicate with the back end to send certain bits of player data from logged-in players to the database for storage. The back-end's main job is sending and receiving data from the database and the game, a middleman between the game and the database. The back-end needs to communicate with the database to allow users to log-in and then retrieve the logged-in user's data such as their high scores for the game.

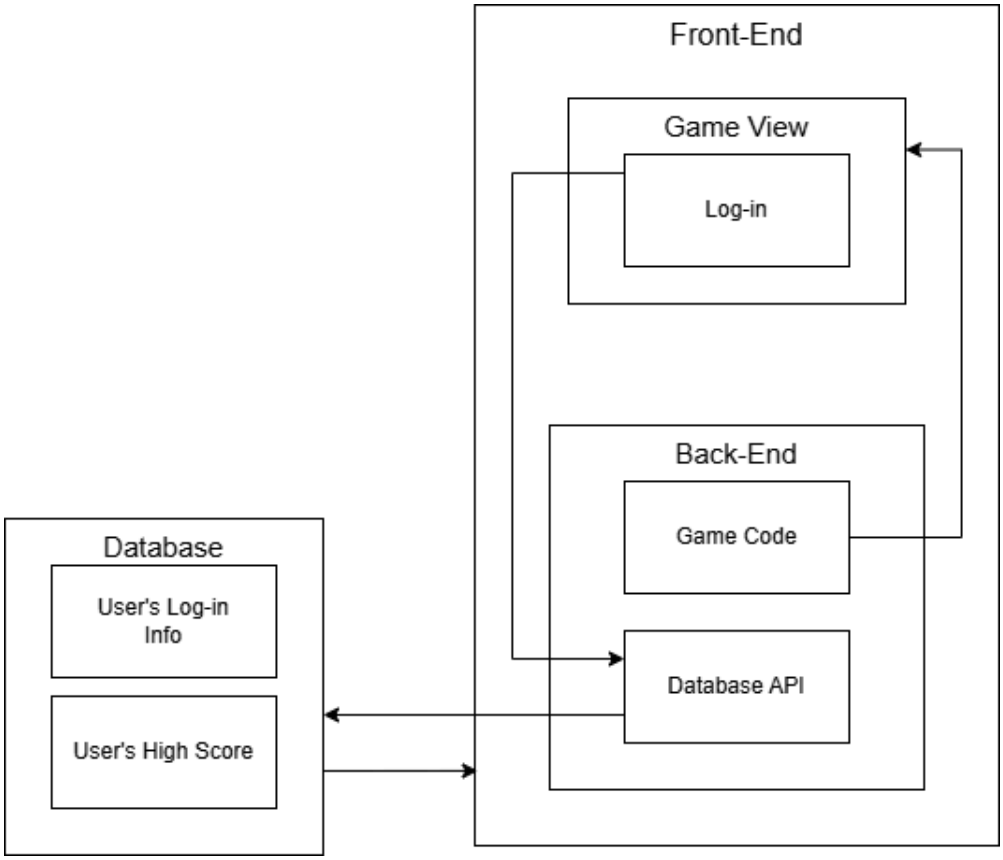


Figure 28. System diagram

V. CONCLUSION

The many factors involved in relocating can cause it to be an extremely stressful process that doesn't always leave people satisfied. Our client Paddy aims to help people avoid blind moves but with the options available now, he can't offer prospective residents the proper experience through tours alone.

The gamified experience that is Virtual Flagstaff is targeting each of these issues and intends to give people peace of mind before they make this big change in their life. The Peak Adventure Experiences team aims to address each problem faced with an in game aspect such as: real time in game weather provided by an API, informative experiences/quests/dialogue with NPCs, mini games that demonstrate activities (such as a ski game for snowbowl), and much more. Some of the technological challenges we intend to address in our technology analysis are: the map requirements, UI and NPCs, a smooth running environment with appealing visuals that will run on desktop and mobile devices, and finally technology that will allow us to implement mini-games. The Peak Adventure Experiences development team plans to use a well tested front-end, back-end, database, and framework to pull this project together. Our team will take

each of these available technologies and mold them into a well developed web application that will provide users with the promised experiences.

Using the Three.js and Phaser.js we can use it to build the core interactive world and the mini-game layer. The FastAPI will allow for real-time interaction that offers fast response delivery to users. Through a database such as postgreSQL we will track users information and progress to provide an enhanced gaming experience, and if authorized by users, offer outside promotions for and about Flagstaff. The game design will use phaser since it has more documentation, examples and more covered in the basic requirements and more in ‘Tile Maps’.

Our team has done thorough testing and research into multiple different potential technologies in order to decide on the ones best fitting of our needs and the experiences we wish to provide our users. Moving forward with these established technologies we will use them to start developing the utmost quality web application that we are able, so that Paddy may offer prospective residents the experiences necessary to move with confidence. This project has much ambition to better the lives of prospective Flagstaff residents, and upon successful development possibly even more locations.

References

[1] J. Dunaway-Seale, “2025 data: 70% of Americans have regrets about moving,” Anytime Estimate, https://anytimeestimate.com/research/moving-trends-2025/.

[2] T. C. Studio, “Express vs FASTAPI: Which scales better under real load?,” Medium, https://medium.com/@thecodestudio/express-vs-fastapi-which-scales-better-under-real-load-cb7c870e3f52.