Overview

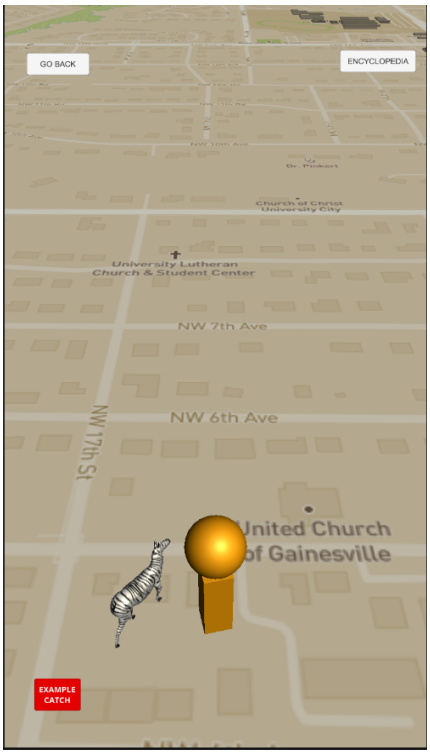
Jei’s Jungle is the working title for our prototype educational location-based game. This game is similar in concept to Pokemon Go, but focused on using real life animal data, with the goal of educating users about different animal species in their area as well as providing them enjoyment. This prototype is being created by Schuyler Gleaves, Czar Parreno, Adam Lizek, and William Cosker as their senior project for Spring 2021.

Jei’s Jungle is a full stack application using NodeJS and MongoDB for the server and backend, and Unity3D for the frontend of the application. It is being built with mobile devices in mind, namely Android and iOS devices. This documentation will be split up into both Server-Side and Client-Side categories, focusing on the overall architecture and requirements to start/expand upon those areas.

Current State

As of writing this documentation, we have a working prototype which has the following core features:

- Retrieves real-life data of animals in a given location
- Instantiates “spawners” of these animals in the database which persist for 60 minutes, these spawners will serve any users in this area the same animals (similar to PokemonGo)
- MapBox integration for displaying users current location on a map overlay
- Display of animals (clickable animated icons) that show up when animals spawn. When these animals are clicked on they are added to user’s Encyclopedia.
- Encyclopedia screen which showcases all of the animals in your collection, as well as images and a brief description of that animal.
- Local cross-platform storage of collected animals.

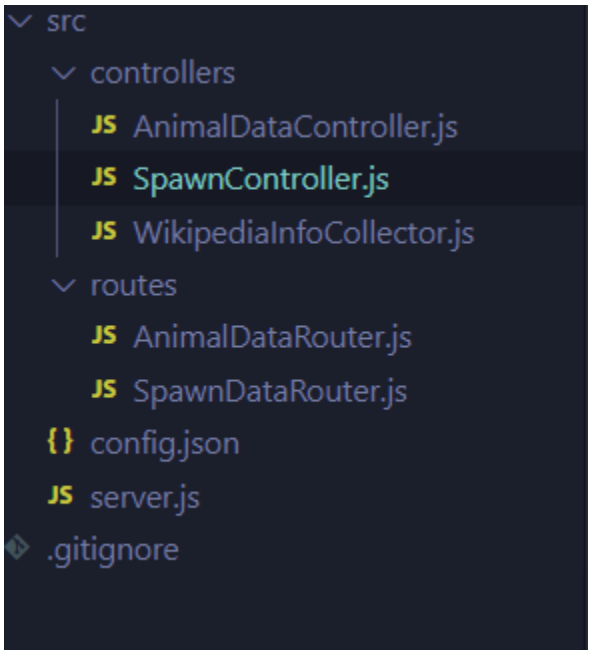


Server-Side

Architecture

Our server is built using NodeJS with ExpressJS as an additional framework. Our server also utilizes MongoClient to interface with our MongoDB database. This server mainly acts as an interface with the Map of Life API and will take in data from users of the application and return them animals and animal spawn locations.

The main entry point into our server is server.js, which instantiates the server and defines the main 2 API routes: get-animals and get-spawn. We chose to separate our server functionality into routes and controllers. We have routes (and routers) for both AnimalData and SpawnData, which handles get-animals and get-spawns, respectively. These routes will call functions from the associated controller when necessary.



At the moment, we only have 2 API routes in this prototype of Jei's Jungle:

/api/get-animals

This route will make a GET request to the Map of Life API, passing in a longitude and latitude. This longitude and latitude will be provided by the caller as part of the query parameters. This

request will return all of the animals found within a specific radius of the lat and long provided. This radius can either be specified, or will default to the value in config.json.

/api/get-spawn

This endpoint is the key to our server, and links together our database with the MOL API. When a call is made to this endpoint (passing lat and long), it will make an internal call to get-animals to get all animals in a given location. Once these animals are retrieved, we check to see if there is already a spawner at the lat and long in our database - or within a small radius of it. If there is a spawner there, then we will return that spawner, along with 10 animals that are able to spawn in it. If there is not an existing spawner, then one will be created.

This spawner will persist for 60 minutes, and any user who makes a request to get-spawn from a similar lat and long will receive the same spawner and animals. This feeds into the Pokemon Go style collaboration where everyone will receive the same animals in the same location.

Database

For our database, we chose to use MongoDB as our document-based solution.

A “spawner” in our database consists of a timestamp, coordinates, and a list of Animals that will spawn at that location.

```
const newSpawn = {
    "createdAt": new Date(),
    "coordinates": coords,
    "Animals": data
}
```

As explained previously, these spawners will be created when a user provides the coordinates to the server (typically will occur every few minutes on the Unity frontend). This spawner will have the list of animals associated with it, and any other users who provide a similar longitude and latitude will be returned this existing spawner. This spawner will persist for 60 minutes, before it will be automatically deleted by MongoDB (using the createdAt date).

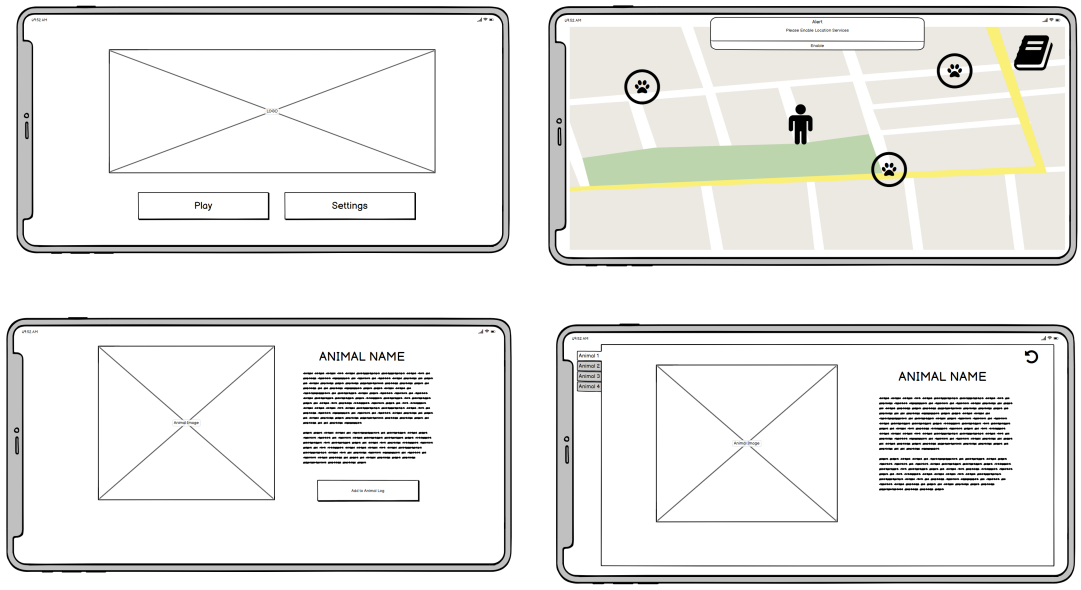
Spinning Up Server

Hosting the server is as simple as running “node server.js” on a publicly accessible machine once you have the required npm packages installed, namely *Express* and *MongoClient*. For the purposes of our prototype, we have set up our server on Heroku. However, given its nature of spinning up and shutting down the server intermittently based on usage, we wouldn’t recommend Heroku going forward. We only utilized it as it was a free and easy solution for the prototype phase of our app. To read more about Heroku and how to set up an account, you can read their FAQ.

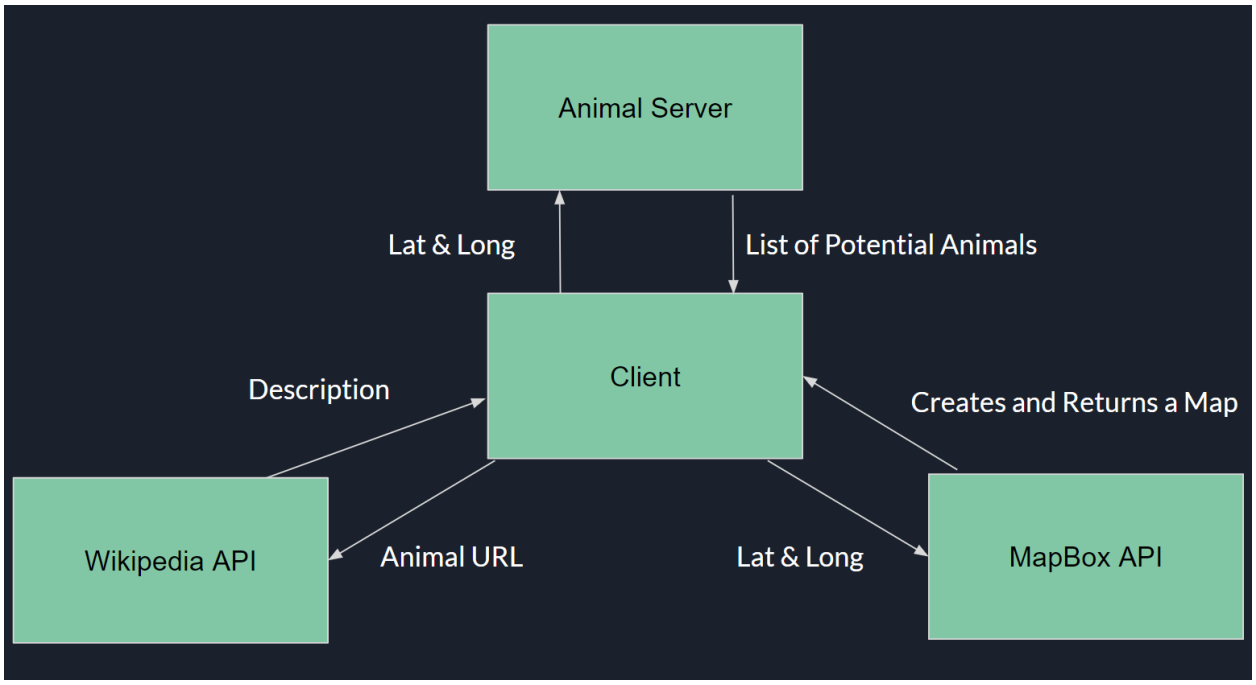
Connecting to the MongoDB database will require having the correct authorization URI, which is kept on the server in our config file. We did not upload these credentials to github, but will give them to Jeremiah to distribute if necessary.

Client-Side

UI Wireframes



Data Flow



Shown here is a brief overview of the main data flowing between our client, backend server, and the various additional API's that we are interacting with. The client in this case will be the user

on their phone, and these API calls will be made periodically to update the server with the position and retrieve the currently available animals. The additional API's serve to provide the Map (MapBox API), as well as providing the descriptions and images of animals (Wikipedia API).

Architecture Overview

Our app is structured around a singleton GameControl object, which interfaces with the rest of our scripts. At the moment, the core functionality of our app comes from the AnimalDataHandler script, which is responsible for saving and loading animals from local storage, as well as making requests to the server to retrieve animal spawner data. Once stored, the EncyclopediaLoader script will populate our encyclopedia page to act as an interface to present this data to the user.

Build Details

We are building this app on Unity Version XXXX, namely using the Android and iOS SDK's as those are the primary platforms we would like to support.

Plans for Expansion

One area that this project is in need of expansion is on the UI side. At the time of writing this documentation, we have made a simple UI for prototyping purposes, as well as set up the framework for how we will animate animal objects on screen. This is a decent start, but will obviously need to be expanded upon.