Team Project: Beav-Guesser

Product Description

Abstract:

Our project is a take on the game Geoguesser adapted to the OSU campus. Our game will be a web-based application where users are shown a 360 image from around the Oregon State University Campus and asked to guess the location of the image taken. Our major developmental goals are, a user authentication system, user profile, leaderboard system, and integrated locational data.

Goal:

The goal of this project is to help students engage more with the OSU campus by playing a game while familiarizing themselves with the different classrooms and buildings.

Current practice:

The inspiration for this project, Geoguesser, is designed around trying to guess the general area around the world that the player is shown, which limits how accurate guessing can be within smaller areas and distances.

Novelty:

This would be a project that works on a more local scale, one that is more applicable to the current living environment that the group and other students at OSU are currently familiar with, thus encouraging a more observant approach to everyday student life.

Effects:

Students at Oregon State University will seek this game for simple, but engaging entertainment. If the product is successful, students will have more knowledge about the OSU campus and its landmarks.

Technical approach:

Create a website connected to a custom database of locations and the Google Street View API. We serve a random picture or location from the database to the user. The is linked to a point on the map. The user inputs a point on the map. This is analyzed, and the user is given a certain amount of points for how close they are. These points are stored in the database and added to the leaderboard, which is displayed at the end of the game.

Risks Assessment:

1) Risk: There may be limits to the Google Maps API
   Risk Level: Low
   Impact: Medium
   Evidence: Too many requests during testing
   Steps: Make meaning contributions during testing and be aware of the amount of requests made
   Detection Plan: We will monitor API requests and log it to prevent going over the limit

Mitigation Plan: Halt testing that involves the API and continue implementing features since locations are stored in a database

2) Risk: Version Control Conflicts
   Risk Level: Medium
   Impact: Medium
   Evidence: Developers forgetting to use or not using branches, not pushing/pulling
   Steps: Ensure everybody is using their corresponding branch when implementing features or complicated code.
   Detection Plan: Check code for conflicts before merging and monitor commits on the GitHub repository.
   Mitigation plan: Document added/new code, check commit log, and revert back to most recent code before the conflict.

3) Risk: Leaderboard Inaccuracies
   Risk Level: Low
   Impact: Medium
   Evidence: If the leaderboard does not update correctly, players may see outdated or incorrect rankings.
   Steps: Ensure that the leaderboard is automatically updated based on new data.
   Detection Plan: Regularly inspect leaderboard data and compare it with actual player scores.
   Mitigation plan: Allow manual leaderboard recalculations

4) Risk: Frontend-Backend Communication Issues
   Risk Level: Medium
   Impact: High
   Evidence: Incorrect API responses or misaligned request formats may cause the game to break.
   Detection Plan: Implement logging for API requests and responses, and conduct frequent integration tests.
   Mitigation plan: Display error messages when the back-end fails

5) Risk: Data Retrieval Issues
   Risk Level: Medium
   Impact: High
   Evidence: potential API limits and implementation of incorrect data retrieval methods.
   Steps: Test data retrieval from the database and revise if incorrect .
   Detection Plan: Frequently test data retrieval and implement data validation checks.
   Mitigation plan: optimize database queries

6) Risk: User Authentication Failures
   Risk Level: Medium
   Impact: High

Evidence: If authentication mechanisms are weak, users may experience login issues or unauthorized access.
Steps: Use secure hashing for passwords
Detection Plan: Monitor login attempts
Mitigation plan: Provide alternative authentication methods

Project Schedule:

Identify milestones (external and internal), define tasks along with effort estimates (at granularity no coarser than 1-person-week units), and identify dependences among them. (What has to be complete before you can begin implementing component X? What has to be complete before you can start testing component X? What has to be complete before you can run an entire (small) use case?) This should reflect your actual plan of work, possibly including items your team has already completed.

To build a schedule, start with your major milestones (tend to be noun-like) and fill in the tasks (tend to start with a verb) that will allow you to achieve them. A simple table is sufficient for this size of a project.

Milestone	**Tasks**	**Effort Estimate**	**Dependency**
Setup Project	Establish team rules, assign team roles, set up everyone's systems, setup GitHub	1 week	None
Frontend Development	Develop UI design, Interactivity and client-side logic	3 weeks	Project planning complete
Backend Development	Setup server side, connect to database, set user authentication, game logic	4 weeks	Frontend and database development must be started
Database Development	Create tables for user, leaderboards, and guesses	2 weeks	Project planning complete
Debugging/Testing/ Changes	Do a complete run and walkthrough on the website, fix any changes needed	2 weeks	All of development must be complete.

Demonstration	Prepare documentation and presentation	1 week	Project is complete and ready.

Test plan & bugs:

Describe what aspects of your system you plan to test and why they are sufficient, as well as how specifically you plan to test those aspects in a disciplined way. Describe a strategy for each of unit testing, system (integration) testing, and usability testing, along with any specific test suites identified to capture the requirements.

We require that you use GitHub IssuesLinks to an external site. to track bugs that occur during use and testing.

Aspects we plan to test: Database data retrieval, Database data submission, Server connectivity to database, Server connectivity to web page, Webpage user interaction, User to web page connectivity.
The testing will be done via testing suite files which when run, test individual functions as well as full functionality of the program. These test suites will be divided into four files. One file will be used for full system testing and the other three will be used for user to website, website to server, and server to database respectively. Usability testing will be done by asking friends to use and navigate the program without prior information, and by having other members of the group attempt to use the feature given that they did not work on the feature themselves. System testing will be done through one main file, while Unit testing will be done within the other three test suites to test individual functionality

The test-automation we are using is Jest. We choose this as all of our logic is written in JavaScript and Jest is for creating tests for JS and is easily used and installed with npm. Adding a new test is easy. You can add a new file to the __test__ folder which is run by Jest. You can also add a test to one of the existing files. Additionally we are using Github Actions as our CI service. When the CI is run a security analysis is performed on the code. On top of that all the unit tests in Jest will also be run. Any time a member of the team pushes to the main branch on Github it triggers the CI to run.

	Github Actions	Jenkins CI	Travis CI
Pros	Works directly in repo and Github, highly customizable, real-time feedback	Open source, platform independent,	Easy to use, automated testing, free for open source
Cons	Complexity, resources limits, dependency on Github	Complex setup , regular maintenance, resource intensive	Performance issues, limited customization

Documentation Plan:

Outline a plan for developing documentation that you plan to deliver with the system, e.g., user guides, admin guides, developer guides, man pages, help menus, wikis, etc finish the documentation plan.

To allow for ease-of-use and for new developers to come into this project we will be providing these documents.

1. User Guides
    a. Audience: Users/Customers
    b. Content: There will be a small overview of the game at the beginning before you click play. This overview will include how to play the game and how points will be accumulated. For more information on the leaderboard and account creation, there will be a help page that you can click.
2. Administrator Guide
    a. Audience: Admins and Maintenance Users
    b. Content: There will be a pdf within our GitHub repository that outlines how to manage the User profiles like the passwords. It will show ways to update the leaderboard and to add/update locations within the application. There will also be information on how to back up the database.
3. Developer Guide
    a. Audience: Developers
    b. Content: This section will be a guide in the form of a pdf within our GitHub repository. This guide will show how to set up the Beav-Guesser and show the changes within the web server. This includes how to run it locally or on the web server. It will also show the code structure and where every process is done within the code. Our API will be documented with its end dates of use and our database schema and structure will be included. Rules will be in place if there are any changes or a new developer is brought in.

Team Info:

Provide a concise summary of the project team and project artifacts. Specifically:
List each team member and their role in the project.

- Blake - Database
- Joy - API/ data collection
- Kevin Tran - Backend
- Kevin Nguyen - Backend
- Sam - UI Designer/Dev
- Gavin - Frontend
- Lukas - API/data collection

Role Description:

- Database:

Creation of database and integration with application
- API/data collection:
  Collection of photos and integration of Google Maps API's
- Backend Developer:
  Creation of server and backend logic
- Frontend Developer:
  Creation of website and integration of website with backend processes and user interface
- UI designer/Dev:
  Creation of user interface and frontend structure

Communication/Tools:
Github:
https://github.com/blakethomas12/Team-18
Trello:
https://trello.com/invite/b/6785b34faf152936f7a64481/ATTI04781f6d4e3950e8c2c6ecd52edaf43a00ED59BB/pt18-beav-guesser
Discord:
https://discord.gg/cD4kHm3b

List communication channels/tools and establish the rules for communication.
Discord server
Rules:
- Communicate any changes within 6 hours of completion
- Members report their updates via Discord when they can't be present
- Give everyone a chance to talk and express their ideas
- Ask for help when there's an as soon as there is an issue with their task

Use Cases:
1. Blake
    a. Actors: User
    b. Triggers: Change of Username
    c. Preconditions: User exists, new name is different than old name, new name is not taken
    d. Postconditions (success scenario): Username has been changed
    e. List of steps (success scenario)
        1. User requests change of username
        2. User asked for new username
        3. Check if old username = new username, reprompt if true
        4. Check if new username is taken, reprompt if true
        5. Username passes checks and is updated in database
    f. Extensions/variations of the success scenario
        1. User fails check if old name = new name, then corrects name

        2. User fails check if name is taken, then enters a free name
    g. Exceptions: failure conditions and scenarios
        1. User does not exist
        2. Database is unreachable(on update or checks)
2. Lukas
    a. Actors: User
    b. Triggers: Start new round
    c. Preconditions:
    d. Postconditions (success scenario): User is served location
    e. List of steps (success scenario)
        i. User starts new round
        ii. Random location is pulled from database
        iii. Corresponding streetview is requested from API
        iv. Correct streetview is shown to user
    f. Exceptions: failure conditions and scenarios
        i. Database is unreachable
        ii. API is unreachable
        iii. Incorrect location
        iv. Incorrect streetview
3. Kevin Tran
    a. Actors: User
    b. Triggers: Submits Location
    c. Preconditions: The round has started and user is presented with the location
    d. Postconditions (success scenario): User guess is recorded and tallied
    e. List of steps (success scenario)
        i. User clicks a location on the map and submits a guess
        ii. Backend received the guess coordinates and the correct coordinates
        iii. Backend calculates the distance between two locations
        iv. Points are rewarded and the user's score is stored in a database.
    f. Extentions/Variations of the success scenario
        i. User are shown two points on the map that correspond to their guess and the correct answer.
    g. Exceptions: Failure Conditions and Scenarios
        i. Invalid Guess
        ii. API unreachable
        iii. Database Unreachable
4. Joy Lim
    a. Actors: User
    b. Triggers: View Leaderboard
    c. Preconditions:
        i. User must be logged in
        ii. System has recorded scores for at least one player
        iii. Leaderboard data is up-to-date
    d. Postconditions (success scenario):

        i. Leaderboard option is displayed in main menu
        ii. Leaderboard will display the top 5 best players' username and scores
    e. List of steps (success scenario)
        i. User selects "Leaderboard" option from main menu
        ii. System will retrieve user scores from leaderboard data
        iii. Scores will be sorted in descending order
        iv. First 5 scores will be displayed to the user
    f. Extensions/variations of the success scenario
        i. User leaves before scores are recorded
    g. Exceptions: failure conditions and scenarios
        i. Leaderboard is empty meaning the database contains no scores
5. Gavin Fifer
    a. Actors: User
    b. Triggers: Login
    c. Preconditions: no account is currently logged in
    d. Postconditions (success scenario): The user is logged into their account
    e. List of steps (success scenario)
        i. User clicks the Login button
        ii. System will display the Username and Password fields
        iii. User fills in Username and Password fields and clicks Submit button
        iv. System validates the Username and Password entered
        v. System returns User to the main page now logged in
    f. Extensions/variations of the success scenario
        i. The Login button is now replaced with the Logout button
        ii. The User can see their Username next to the Logout button
    g. Exceptions:
        i. Invalid Username/Password
        ii. The account is already logged into somewhere else
        iii. Database with Usernames/Passwords cannot be reached
6.
    a. Actors: User
    b. Triggers: New user registration
    c. Preconditions:
        i. User isn't currently logged in
        ii. User provides valid registration details
    d. Postconditions (success scenario):
        i. User account is created and stored in the database
    e. List of steps (success scenario):
        i. User selects the "new account" button from the menu
        ii. System prompts user for username, email, and password
        iii. User enters details and submits
        iv. System checks if submitted details already exist in database
        v. The check passes and the new account is created and stored
        vi. The user is automatically logged into their account

f. Extensions/variations of the success scenario:
    i. User inputs an option profile picture
    ii. A “forgot password” routine is initiated
g. Exceptions:
    i. Username/email already exists in database
    ii. Database is currently unavailable

7. Kevin Nguyen
    a. Actors: User
    b. Triggers: Reset Password
    c. Preconditions:
        i. User account already exists
        ii. User provides the correct email for the account
    d. Postconditions (success scenario): User is able to reset their password through the link sent in their email
    e. List of steps (success scenario)
        i. User clicks on “Forgot Password” in the login page
        ii. System asks for the email connected to the account
        iii. User enters in the email and clicks submit
        iv. System verifies that email exists in the database
        v. A reset link is sent to the email address
        vi. User clicks on the links, enters a new password, and submits it
        vii. System updates the new password in the database
        viii. User is back in the login page and enters their new password and logs in
    f. Extentions/Variations of the success scenario
        i. User enters an invalid email address and a message pops up saying to try again.
    g. Exceptions: Failure Conditions and Scenarios
        i. Database is down
        ii. Email address is not found

Non-Functional Requirements:

1. Databases must be able to grow and serve a growing user base.
2. Photos must not include people's identities
3. Front end must be easy to use and simple to learn

External requirements:

- Since our product is web based the url must be public and accessible to others
- Our project will also include a set of instructions to set up a server and start to program

3. Additionally, add the following:

Major Features:

- Location Data
- Leaderboard
- User Authentication
- User Profiles

Stretch Goals

- Multi-User competition
- User Submissions
- Reward System

4. Timeline

Week 1: project idea, role creations, research into roles, tools and common practices
Week 2: basic foundation of website created(basic server uses, interactable front end)
Week 3: database and api integration, data gathering, game logic
Week 4: major features implementation, data gathering
Week 5: finishing major features, data gathering
Week 6: Final touches to all parts, adding in the smaller details(ui fixes, visual components, cleaning up code)
Week 7: Major testing, test all components for bugs
Week 8: Major testing, test all components for bugs
Week 9: Final testing, preparation for final version

5. Software Architecture

1. Server
    a. Job is to serve the html, css, and images needed for the front end and logic of the website.
    b. Pros and cons
        i. Pros
            1. Security
            2. Data management
        ii. Cons
            1. Latency
            2. More complex
    c. Alternative: Non-server based website
        i. Pros:
            1. More simple
        ii. Cons
            1. Security issues
            2. Limited data mangament
2. Database
    a. Job is to store the path to images, users and leaderboard of the game
    b. Data stored: location source paths (google embed links), long and lat of each link, username, password, high score
    c. 3 schema's
        i. User{username, password, high score}
        ii. Location{path, longitude, latitude}
        iii. Leaderboard{username, score}
    d. Pros and cons
        i. Pros

            1. Efficient
            2. More secure
        ii. Cons
            1. Complex configurations
            2. Learning curve
    e. Alternative flat files
        i. Pros
            1. Simple
            2. portable
        ii. Cons
            1. Efficiency
            2. Scalability issues
3. Front-end JS
    a. Job is to communicate with the back-end JS to collect the needed information for the user to play the game, as well as handling interactive components like logging in and pop ups.
    b. Pros and cons
        i. Pros
            1. Interactivity
            2. Performance
            3. versatile
        ii. Cons
            1. Security
            2. compatibility
            3. complexity
    c. Alternative only html/css
        i. Pros
            1. Simple
            2. compatibility
        ii. Cons
            1. Interactivity
            2. functionality
4. Back-end JS
    a. Job is to control the logic of the game, pulling information from the database to send to the Front-end JS. Receives requests from the front-end JS
    b. Pros and cons
        i. Pros
            1. Single language
            2. Asynchronous operations
            3. performance
        ii. Cons
            1. Cpu heavy
            2. Callback management
    c. Alternative python

i. Pros
    1. Readability
    2. libraries
ii. Cons
    1. Performance
    2. Limited multithreading

6. Software design

Server:

The server is the main place of communication between all the rest of the parts. The client first requests files from the server. Then game data is passed back and forth through the server to calculate scores, store final scores, and get photospheres.

The components of the server are:

- Functions to list pages (home, login, leaderboard, profile, error page)
    - Listen for requests and serve the corresponding files
- Functions to call backend JS (database i/o, game logic functions)
    - Listen for request and transfer data to corresponding function to return answer
- Folder with frontend files to serve

Database:

The database is used to store user profile information, like username password, level and high score. It also stores the leaderboard by having a file for every player with their name and score. Lastly it holds the location of photo spheres with their corresponding longitude and latitude.

The database has 3 different schema:

User schema
    Username
    Password
    High score
    Level

Location schema
    Path
    Longitude
    Latitude

Leaderboard Schema
    Username
    Score

Frontend JS

The frontend JS is used to build requests to the server and gather data for the backend JS to use in the functioning of the game. For example the frontend JS will take the input username and password and send a verification request to the server.

Components of the frontend JS

- Login request
    - Sends users input to server for verification
- Gather photo sphere

    - Requests a random location
- Request leaderboard update
    - Request leaderboard information
- Create user
    - Sends information to server to create new user

Backend JS
The backend JS is used to perform the game's logic and interact with the database, things like loading a random location, updating profiles and verifying users are performed on the backend.
Components of backend JS

- Database I/O functions
    - Interact with the database
- find random location
    - Calculate random coordinates
- Update leaderboard standings
    - Updates the database
- Scoring functions
    - Creates scores for game and user level

7. Coding Guidelines
JavaScript:https://developer.mozilla.org/en-US/docs/MDN/Writing_guidelines/Writing_style_guide/Code_style_guide/JavaScript
HTML:https://developer.mozilla.org/en-US/docs/MDN/Writing_guidelines/Writing_style_guide/Code_style_guide/HTML
CSS:https://developer.mozilla.org/en-US/docs/MDN/Writing_guidelines/Writing_style_guide/Code_style_guide/CSS