User Documentation

22 October 2023 13:40

User Documentation:

Getting Started:

1. Landing Page:
   - On the landing page you can view all the games as an unauthorized user, but you can also log in.
   - There is a filter slider for registered, in-progress, and completed games.
   - To join a game, you must be logged in.
2. Game Details:
   - Click on a game to access the "About Game" page.
   - The "About Game" page displays game information, including title, description, rules, missions and a map.
   - You can join the game by clicking the "Join Game" button.
3. Game Page:
   - After joining a game, you will be redirected to the game page.
   - The game page displays the map with mission markers and gravestones.
   - Mission markers provide information about available missions.
   - Gravestones indicate where players have been killed by a zombie.
4. Navigation Bar (Game Page):
   - The navigation bar on the game page allows you to access different sections:
     - "Map" - View the game map.
     - "Manage Squad" - Create or view squads.
     - "Bite Code Actions" - Perform actions related to bite codes.
5. Manage Squad:
   - Under "Manage Squad," you can either:
     - "Squad Registration" - Create a new squad or see details of an existing one.
     - "Squad Details" - View information about a squad. You can choose to join or leave the squad, and if you've joined the squad, you can leave a marker for the squad members to see.
6. Leaving a Marker:
   - You can leave a marker on the map to share your location with your squad.
   - Enter X and Y coordinates and submit your marker.
7. Bite Code Actions:
   - For humans, access bite code features.
   - For zombies, enter a bite code to turn a human into a zombie and click "Kill". If the bite code exists for a player in the game, you can add a description to the kill.
8. Chat Feature:
   - Access the HvZ chat, including "Global Chat," "Zombies Chat," and "Human Chat."
   - Use the text box to send messages.
   - You receive messages sent from other users.

Admin Documentation:

Creating and Editing Games:

1. Landing page:
   - As an admin, you start on the "Landing Page", but you have an extra option to "Create new game".
2. Dashboard Page:
   - As an admin, you have access to a "Dashboard" button in the navbar which will send you to the dashboard page.
   - On the dashboard page you have an overview of all admin feature's such as; create game, edit game, and manage players.
3. Creating a Game:
   - Follow these steps to create a game:
     - Step 1: Provide a title and game description.
     - Step 2: Provide an Image URL for the game (the image users see on the landing page)
     - Step 3: Add missions and rules for the game.
       - Missions require a "Mission name", "Mission description", and X- and Y Coordinates.
       - Rules require a title and a description.
     - Step 4: Choose a map for the game.

        - Step 5: Submit.

4. Editing Game Information:
    - When selecting a game from the "Landing Page", admins have an "Edit Game" button which allows them to edit a specific game.
    - Use the "Edit Game" page to make changes to game details, mission markers, and rules.
        - Step 1: You can change the title and description.
        - Step 2: You can edit the image URL for the game.
        - Step 2: Editing Mission Markers:
            - Edit existing mission markers or add new ones.
            - Enter mission name, description, and location (X and Y coordinates).
        - Step 3: Editing Rules:
            - Modify game rules by adding or editing them.
            - Provide a rule title and description and click "Submit Rule."
        - Step 4: You can change the map for the game.
        - Step 5: Submit.

User Manual

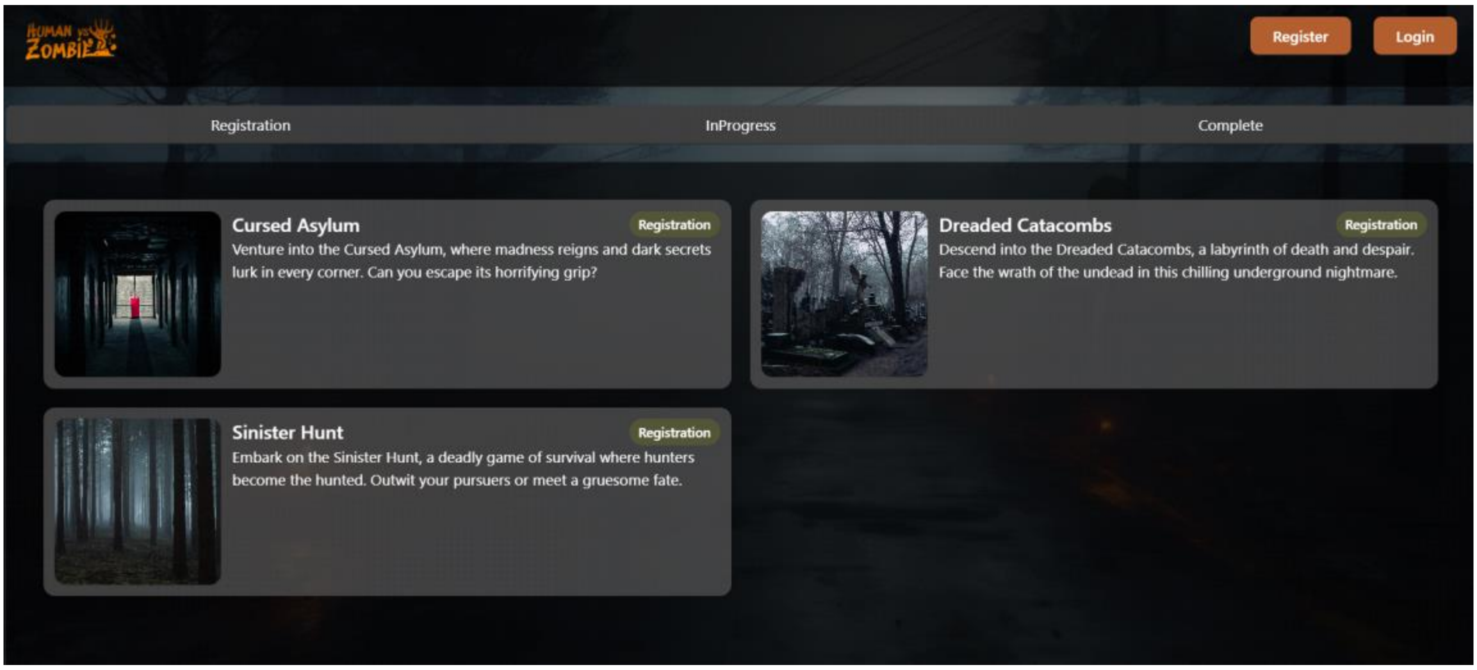
25 October 2023 15:25

Manual for Players/Users

Landing Page:

Upon launching the application, the first screen you'll encounter is the Landing Page. Here, you have several options for interaction:

- **Navbar Options:** You can navigate through the landing page by clicking on the following Navbar options:
  - Registration
  - InProgress
  - Complete
- **Games:** You can also explore different games on the Landing Page, but please note that you won't be able to join them unless you are logged in

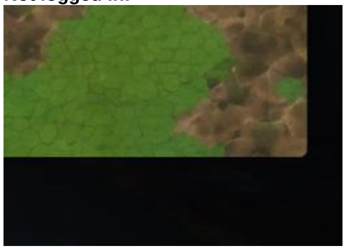


Cursed Asylum
Venture into the Cursed Asylum, where madness reigns and dark secrets lurk in every corner. Can you escape its horrifying grip?

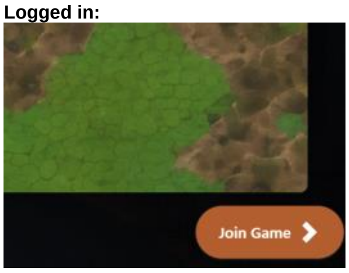
Dreaded Catacombs
Descend into the Dreaded Catacombs, a labyrinth of death and despair. Face the wrath of the undead in this chilling underground nightmare.

Sinister Hunt
Embark on the Sinister Hunt, a deadly game of survival where hunters become the hunted. Outwit your pursuers or meet a gruesome fate.

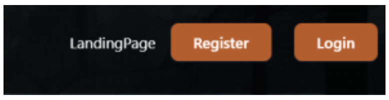
Not logged in:



Logged in:



You can login by either register an account, or login if you already have an account. The options are in the top right corner on the landing page:

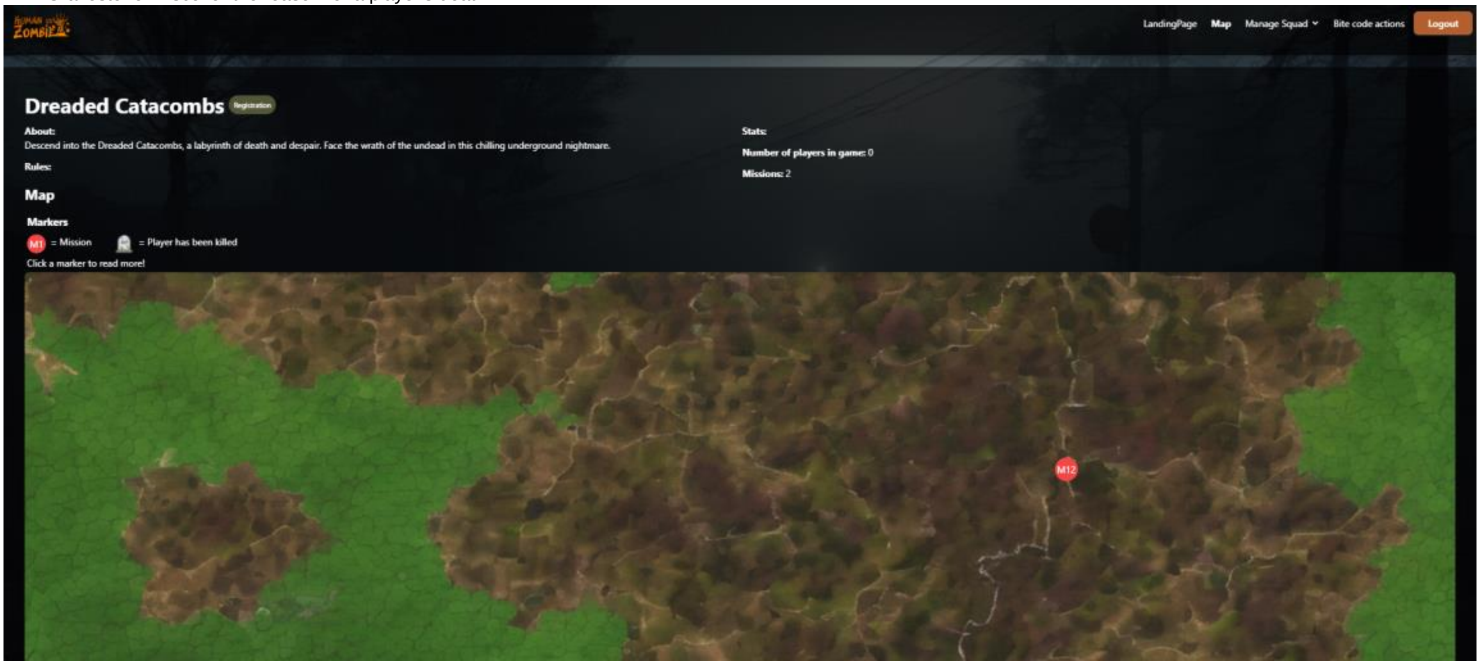


After logging in, you can participate in games. Here's what you can do:

1. **Joining a Game:** To join a game, you need to enter a username. Once you've provided a username, you'll be directed to the Map Page, and you will become an active participant in the game.
2. **Map Page:** On the Map Page, you'll find a new Navbar with these options:
   - Landing Page
   - Map
   - Manage Squad (Squad Registration & Squad Details)
   - Bite Code Actions

On this page, you'll see mission markers and gravestones if a player is killed. By clicking on either the mission marker or the gravestone, you can access relevant information:

- Mission marker: Learn about the mission.
- Gravestone: Discover the reason for a player's death



Dreaded Catacombs

About:
Descend into the Dreaded Catacombs, a labyrinth of death and despair. Face the wrath of the undead in this chilling underground nightmare.

Rules:

Stats:

Number of players in game: 0

Missions: 2

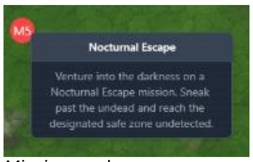
Map

Markers

= Mission = Player has been killed

Click a marker to read more!

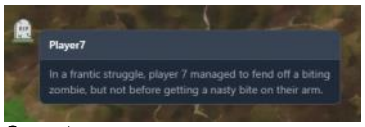
Map Page



Nocturnal Escape

Venture into the darkness on a Nocturnal Escape mission. Sneak past the undead and reach the designated safe zone undetected.

Mission marker



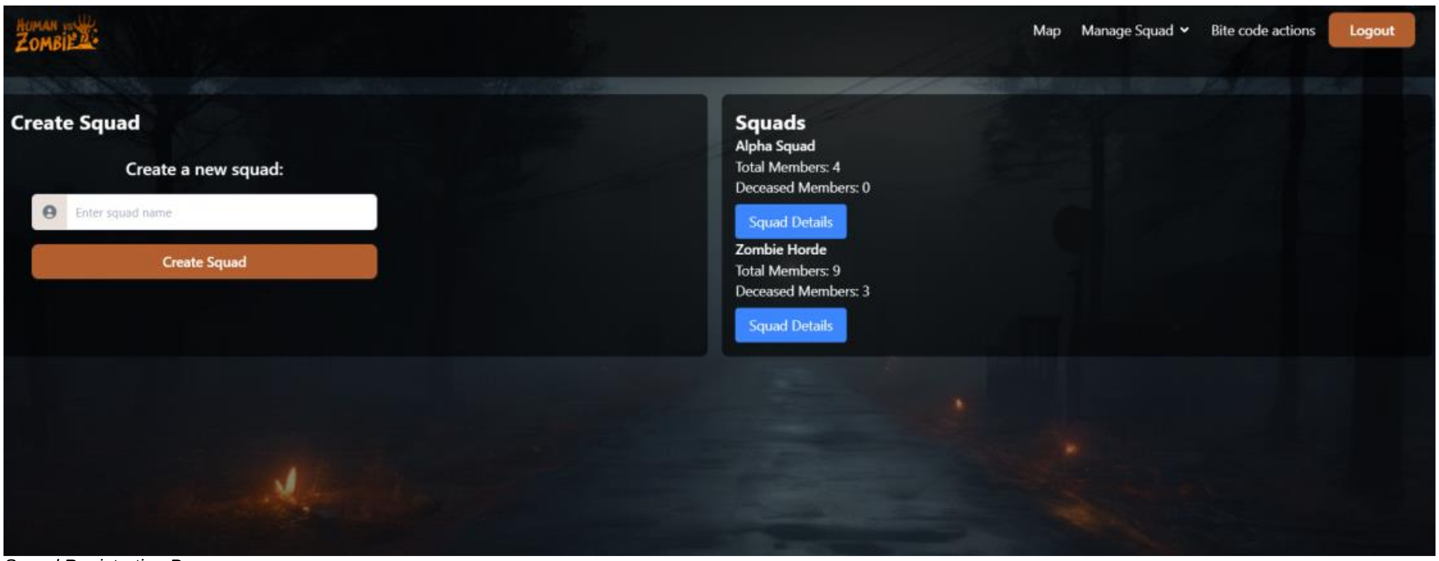
Player7

In a frantic struggle, player 7 managed to fend off a biting zombie, but not before getting a nasty bite on their arm.

Gravestone

Manage Squad:
In the "Manage Squad" section, you'll find two options:

- **Squad Registration:** You can create a new squad or join an existing one (an overview of existing squads is displayed on the right side of the page). After creating a squad, you'll receive a confirmation message, and your squad will appear in the overview.
- **Squad Details:** By clicking on one of the squads in the overview, you can access detailed information about the squad, including Squad Information, Squad Name, Total Members, Deceased Members, and Squad Members. If you join a squad, this information will be updated. You can leave the squad at any time and leave markers for squad members to see as long as you are part of a squad



Create Squad

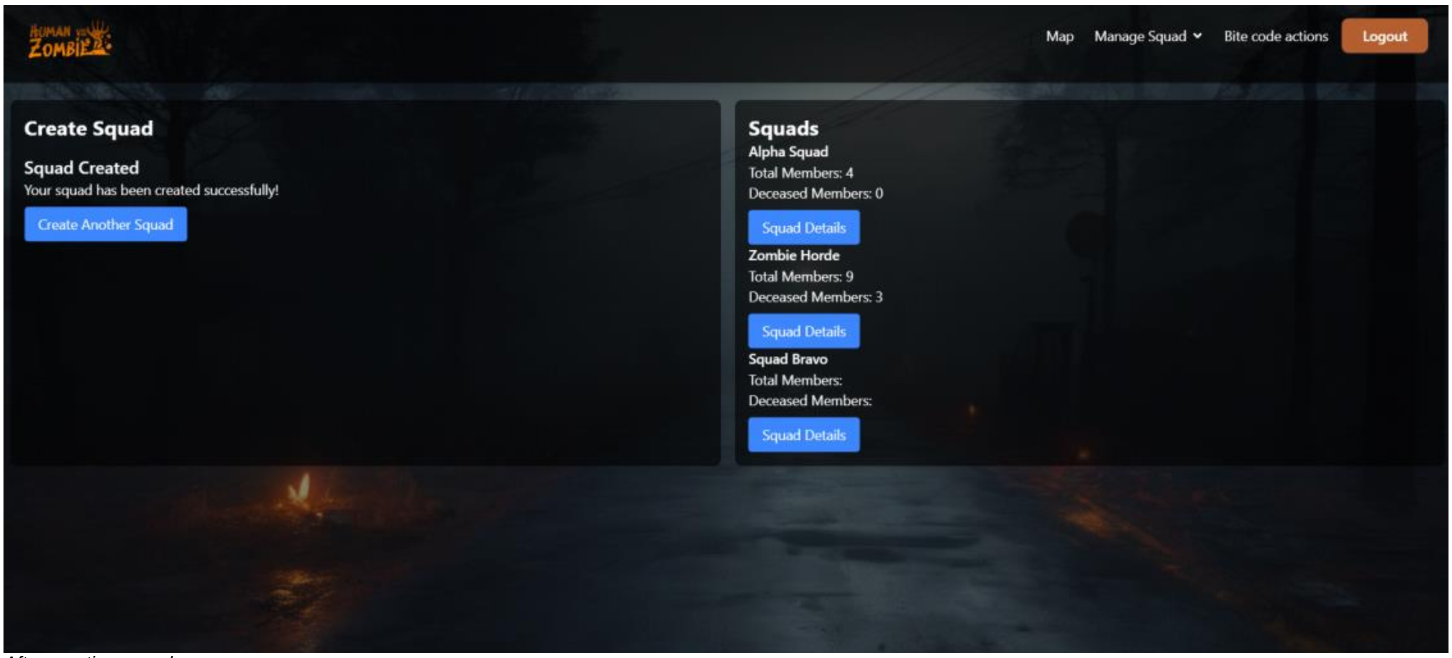
Create a new squad:

Squads

Alpha Squad
Total Members: 4
Deceased Members: 0

Zombie Horde
Total Members: 9
Deceased Members: 3

Squad Registration Page



Create Squad

Squad Created

Your squad has been created successfully!

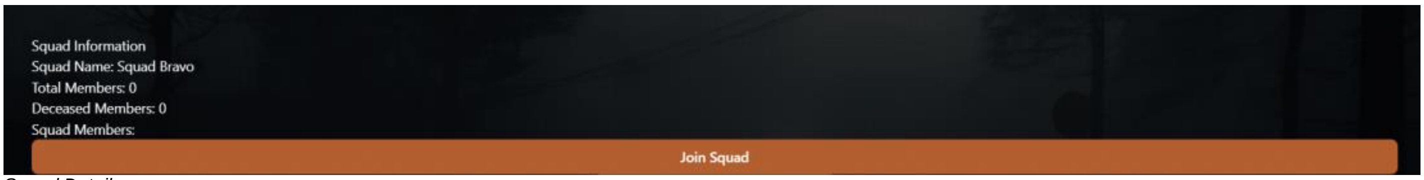
Squads

Alpha Squad
Total Members: 4
Deceased Members: 0

Zombie Horde
Total Members: 9
Deceased Members: 3

Squad Bravo
Total Members:
Deceased Members:

After creating squad

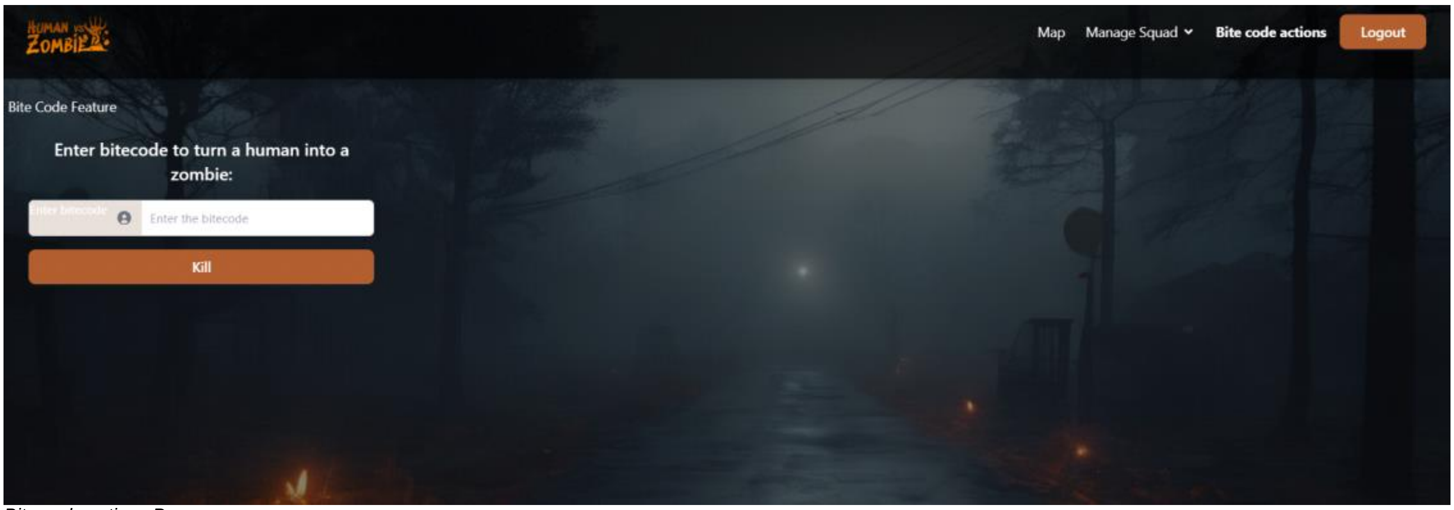


Squad Information
Squad Name: Squad Bravo
Total Members: 0
Deceased Members: 0
Squad Members:

Squad Details

Bite Code Actions:

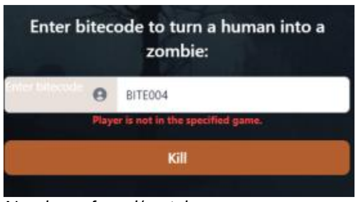
For zombies, the "Bite Code Actions" allow you to enter the bite code of players. If the bite code matches an existing player in the game, that player will be eliminated. However, the bite code must match an existing player; otherwise, you'll receive an error message. If it's a match, you'll be prompted to enter a description for the kill



Bite Code Feature

Enter bitecode to turn a human into a zombie:

Bite code actions Page

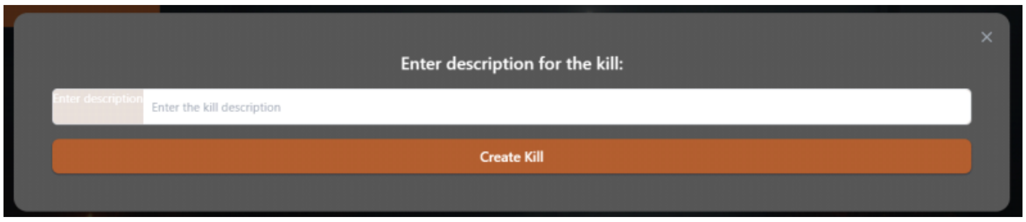


Enter bitecode to turn a human into a zombie:

BITE004

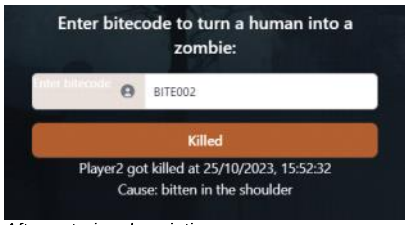
Player is not in the specified game.

No player found/match



Enter description for the kill:

Description pop-up if the bite code match an existing player in the game



Enter bitecode to turn a human into a zombie:

Enter bitecode: BITE002

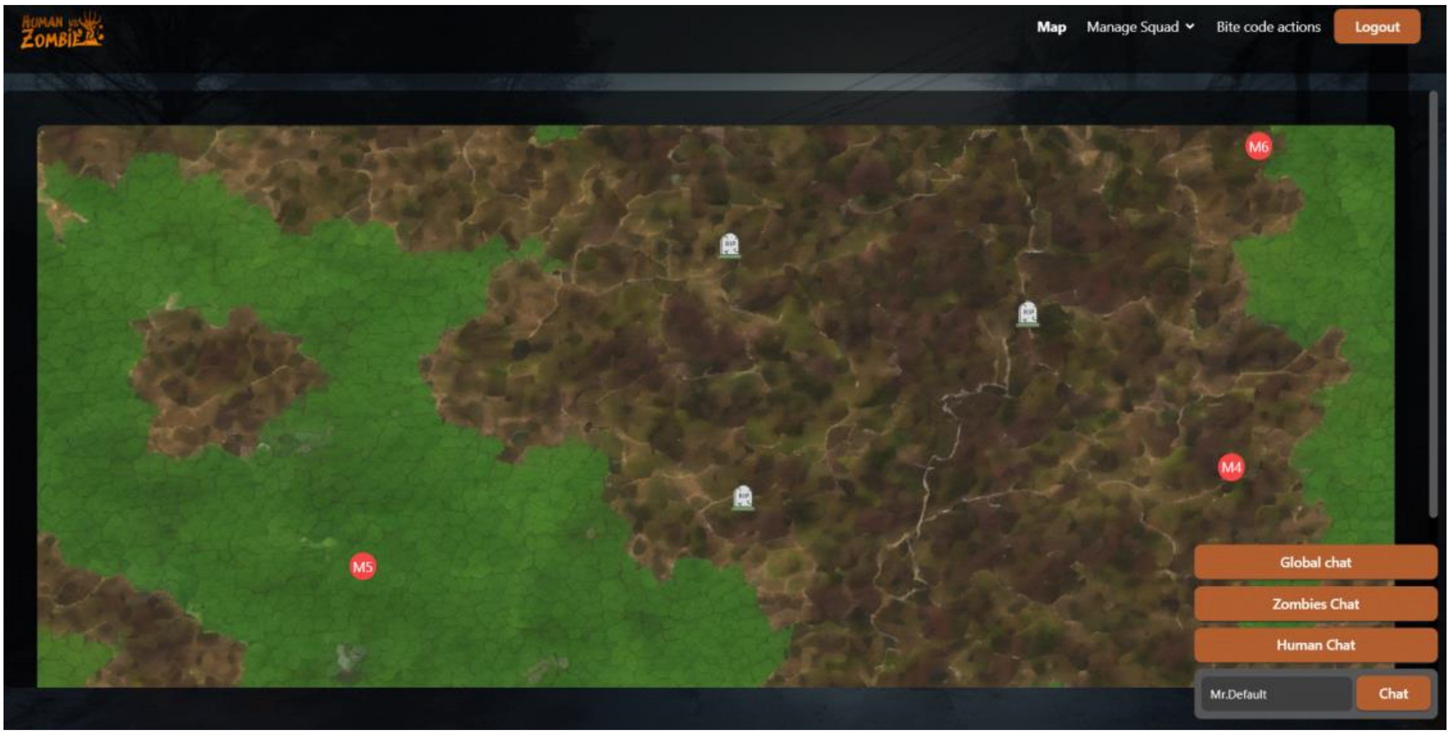
Killed

Player2 got killed at 25/10/2023, 15:52:32

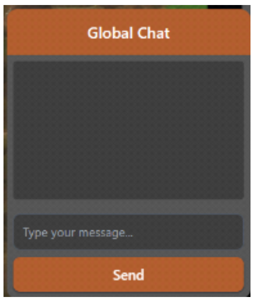
Cause: bitten in the shoulder

After entering description

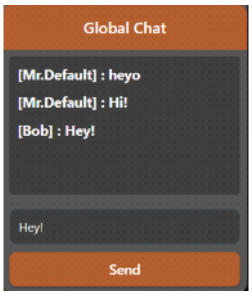
Chat:



The chat has three different chats; *Global Chat*, *Zombies Chat*, and *Human Chat*.
The chat box looks like this (Global Chat):



In the chat, you can send messages by either pressing "Enter" or clicking the "Send" button.



Global Chat

[Mr.Default] : heyo

[Mr.Default] : Hi!

[Bob] : Hey!

Hey!

Send

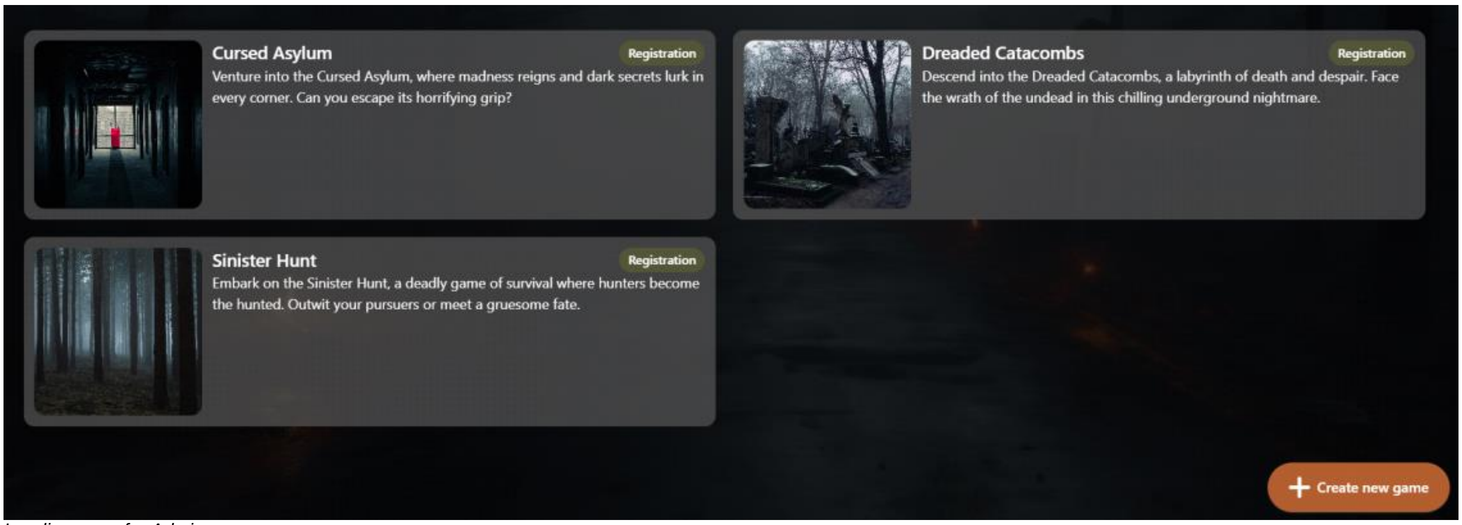
Manual for Admin:

Here you have the options to create a new game, edit a game, and manage players.

Create Game:

As an admin, you will have the same access as the users, and in addition, you'll also have access to exclusive features that are only accessible to admin.

An admin also begin at the landing page:



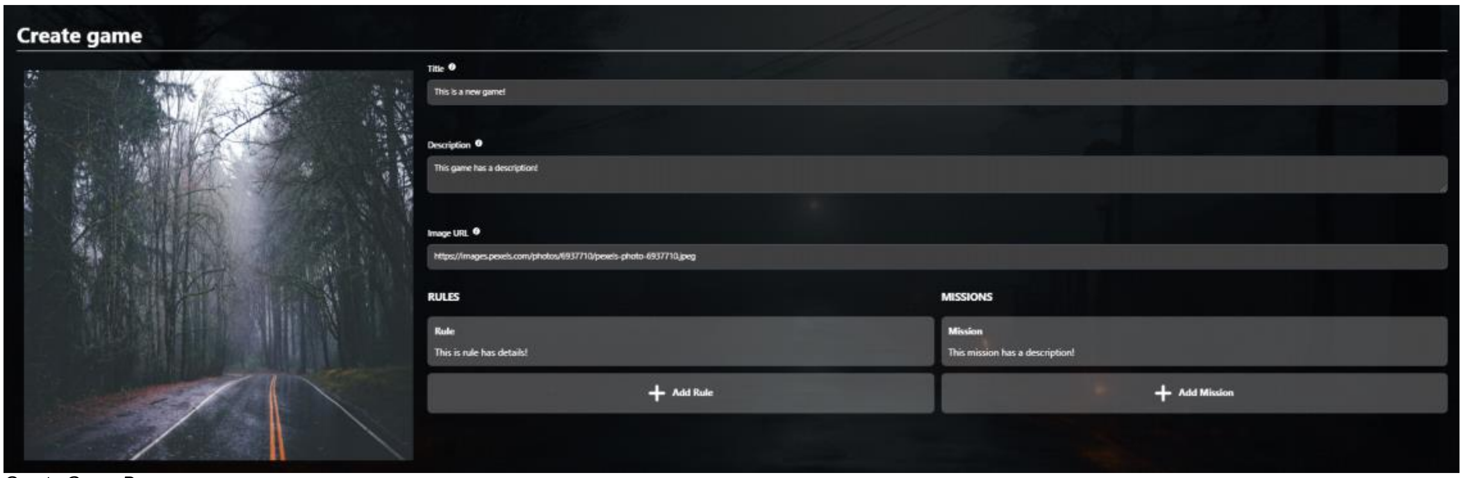
Cursed Asylum
Venture into the Cursed Asylum, where madness reigns and dark secrets lurk in every corner. Can you escape its horrifying grip?

Dreaded Catacombs
Descend into the Dreaded Catacombs, a labyrinth of death and despair. Face the wrath of the undead in this chilling underground nightmare.

Sinister Hunt
Embark on the Sinister Hunt, a deadly game of survival where hunters become the hunted. Outwit your pursuers or meet a gruesome fate.

Landing page for Admin

As you can see, an admin has the same access as a user, however, you have the option to create a game by clicking on the "Create new game" button in the bottom right corner.



Title: This is a new game!

Description: This game has a description!

Image URL: https://images.pexels.com/photos/6937710/pexels-photo-6937710.jpeg

RULES

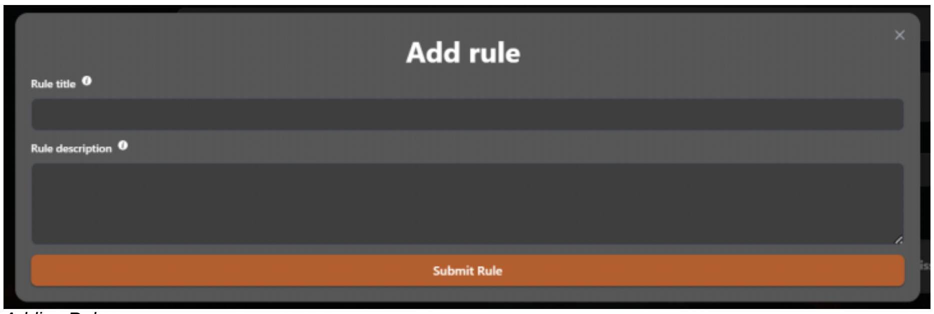
Rule
This is rule has details!

MISSIONS

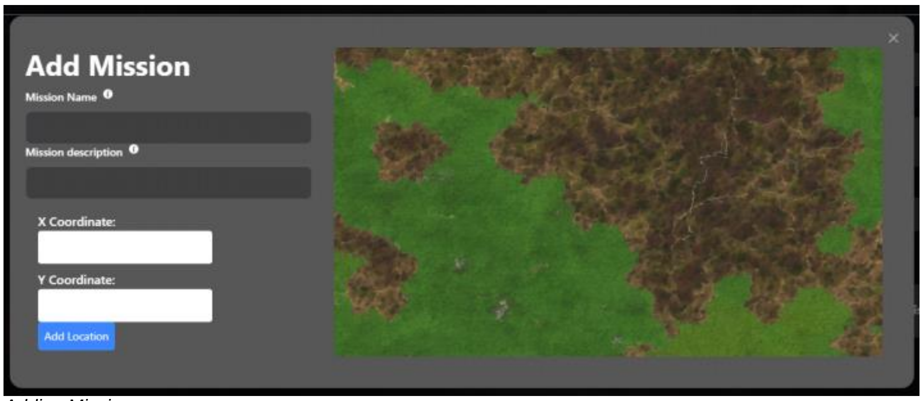
Mission
This mission has a description!

Create Game Page

On the *Create Game Page* you have the choice to create a new game. Here you can add title of the game, an description, image URL, rules and missions.

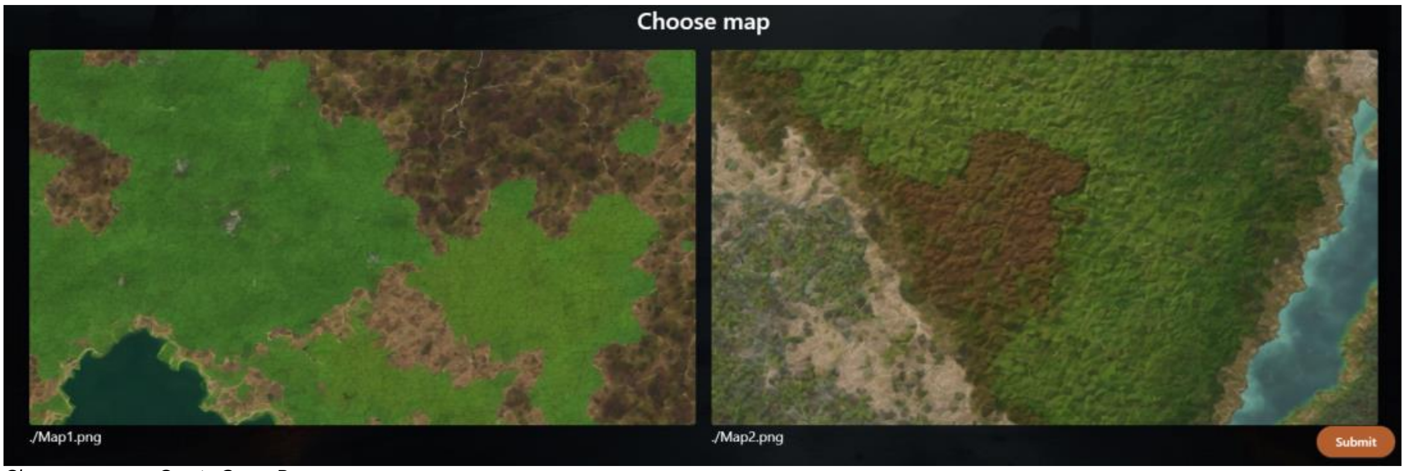


Adding Rule



Adding Mission

You also have the option to choose between maps:



Choose a map on Create Game Page

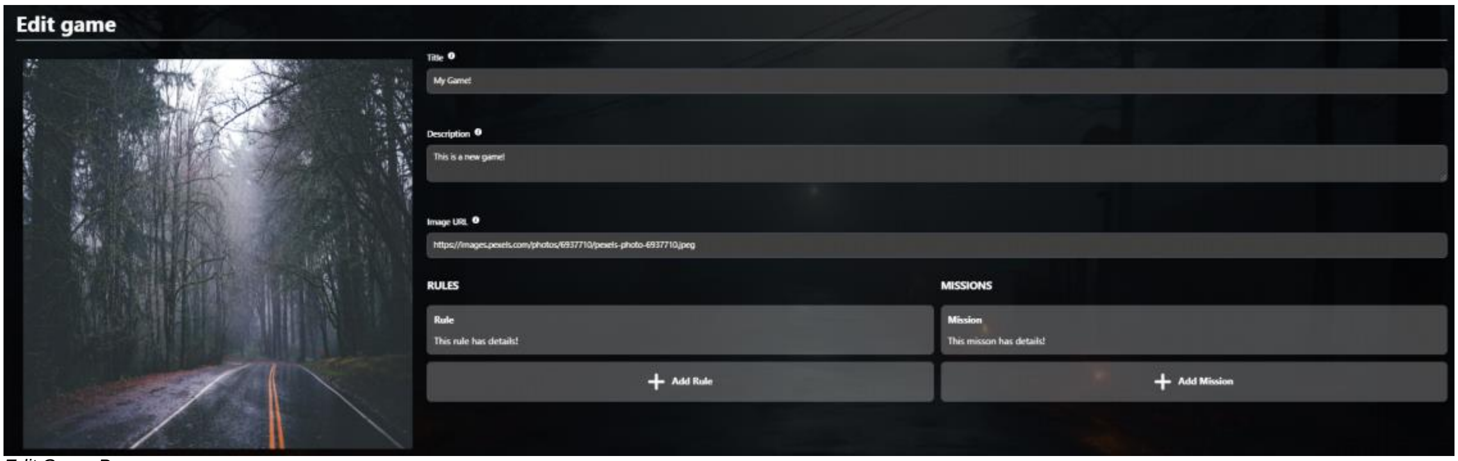
After adding all the fields and choosing a map, you can click on the "Submit" button.

Edit Game:

Admin's do also have the option to edit a game. By clicking on an active game from the landing page, you'll see the option to edit a game through a button called "Edit game"

Edit game button in the bottom right corner

By clicking on it, you'll get sent to the edit game page:



My Game!

This is a new game!

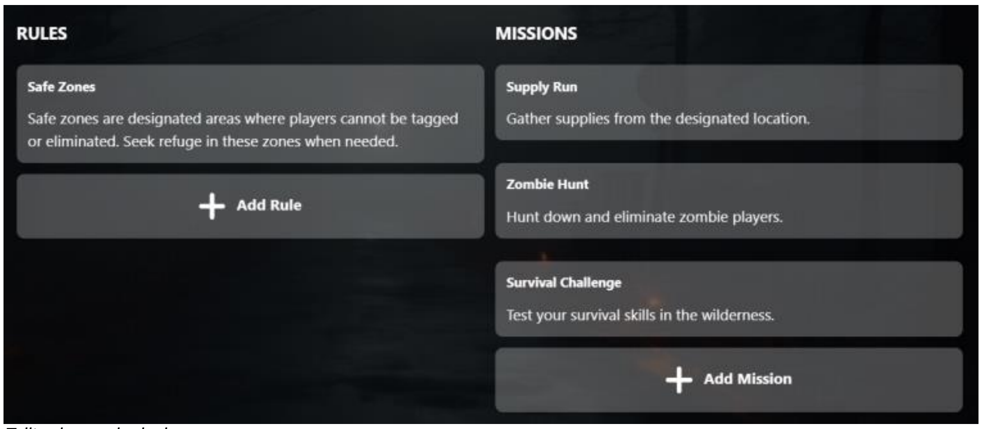
https://images.pexels.com/photos/6937710/pexels-photo-6937710.jpeg

Rule
This rule has details!

Mission
This misson has details!

Edit Game Page

Here you can edit a game. You can edit title, description, image URL, rules and missions, and map.



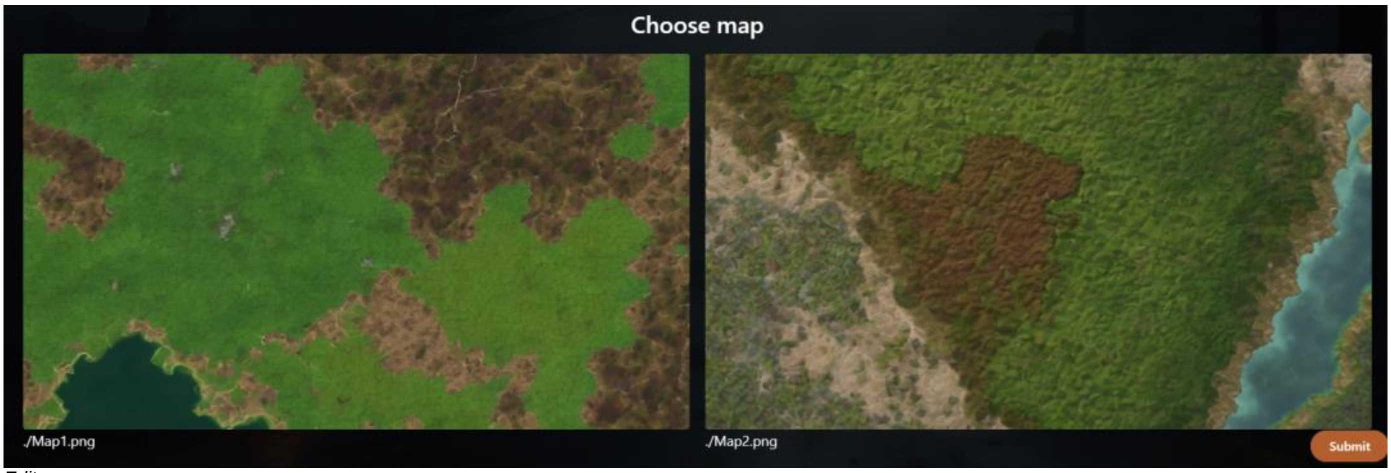
Safe Zones
Safe zones are designated areas where players cannot be tagged or eliminated. Seek refuge in these zones when needed.

Supply Run
Gather supplies from the designated location.

Zombie Hunt
Hunt down and eliminate zombie players.

Survival Challenge
Test your survival skills in the wilderness.

Edit rules and missions

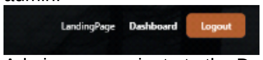


Edit map

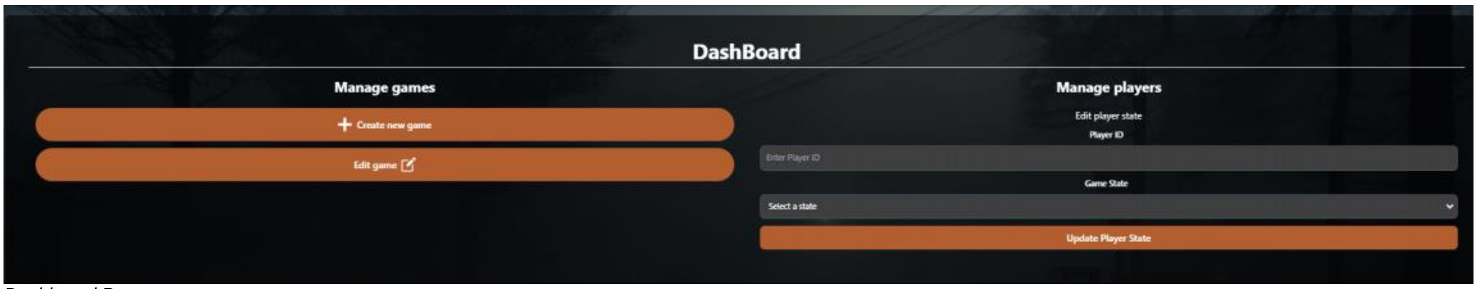
When your done making your edits, you can click on the "Submit" button at the bottom right corner.

Admin Dashboard:

The admin also has a dashboard, which gives an overview of all their features as an admin:

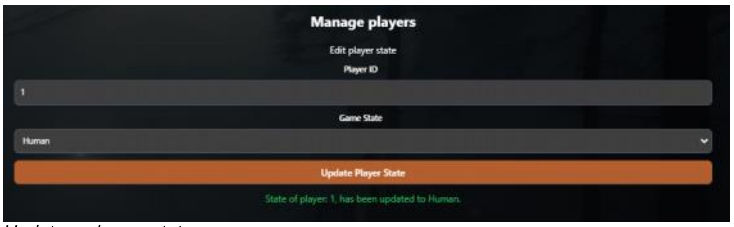


Admins can navigate to the Dashboard through the navigation bar.



Dashboard Page

The admin can also create and edit games on in the dashboard, as well as managing player states.



State of player: 1, has been updated to Human.

Update a players state

The admin can enter the player ID and pick which state they want the user to have, and then submit through "Update Player State".

Technical Documentation

23 October 2023 09:25

System Architecture

The HvZ Game Management System is a web-based application that has the following architecture:

Client-Side: The front-end Is developed using React.js, HTML, CSS, Tailwind CSS, and JavaScript.

Server-Side: The back-end is built using .NET (ASP.NET Core Web API and C#). Microsoft SQL Server Management is the chosen database management system, and the code is using Entity Framework code first approach for building the database.

Technologies Used:

Front-end:

- **React:** React has been our front-end framework for building responsive user interface
- **HTML:** HTML is a standard web technology used for structuring our content
- **Tailwind CSS/CSS:** Tailwind CSS has been our framework for CSS to style our content for the web page.
- **JavaScript:** JavaScript has been our front-end programming language for making the interactivity through the web page.
- **Git/GitHub:** Version control and collaboration for code management.

Back-end:

- **.NET (ASP.NET Core Web API and C#):** .NET is our back-end framework for building reliable API endpoints for the frontend.
- **Entity Framework:** Used to build the database through code.
- **Microsoft SQL Server Management:** Microsoft SQL Server has been used for the relational database.
- **Keycloak:** Keycloak has been used for authentication and access control.
- **Swagger UI:** Swagger UI has been used to provide interactive API documentation and testing of endpoints.
- **Postman:** Postman has been used to test API endpoints with JWT Bearer tokens provided in the requests to test authorization.
- **Git/GitHub:** Version control and collaboration for code management.

API Endpoints (handled by Swagger):

The endpoints can also be seen in the Swagger Documentation.

AppUser:

- GET /api/v1/AppUser/subject: Retrieve the current user's "sub" claim from their token.
- GET /api/v1/AppUser/exists: Check if a user exists in the AppUser table, using the "sub" claim from their token.
- GET /api/v1/AppUser: Retrieve all AppUser entries.
- GET /api/v1/AppUser/{id}: Retrieve a specific entry in the AppUser table.
- PUT /api/v1/AppUser/{id}: Update a specific AppUser.
- DELETE /api/v1/AppUser/{id}: Delete a specific AppUser.
- POST /api/v1/AppUser/register: Add a new AppUser with the clients "sub" claim.
- PUT /api/v1/AppUser/{id}/add-player/{playerId}: Add a player to the user.
- PUT /api/v1/AppUser/{id}/update-players: Update an AppUser's list of players.
- PUT /api/v1/AppUser/{id}/remove-player/{playerId}: Remove a specific player from the AppUser's list of players.

Conversation:

- GET /api/v1/Conversation: Retrieve all Conversation entries.
- POST /api/v1/Conversation: Create a new conversation.
- GET /api/v1/Conversation/{id}: Retrieve a specific Conversation entry.
- PUT /api/v1/Conversation/{id}: Update a conversation.
- DELETE /api/v1/Conversation/{id}: Delete a conversation.

Game:

- GET /api/v1/Game: Retrieve all Game entries.
- POST /api/v1/Game: Create a new game.
- GET /api/v1/Game/{id}: Retrieve a specific Game entry.
- PUT /api/v1/Game/{id}: Update a game.
- DELETE /api/v1/Game/{id}: Delete a game.
- GET /api/v1/Game/filterbystates/{gamestate}: Filter games by their state.
- PUT /api/v1/Game/{id}/add-rule/{ruleId}: Add a rule to the game.
- PUT /api/v1/Game/{id}/add-player/{playerId}: Add a player to the game.
- PUT /api/v1/Game/{id}/add-mission/{missionId}: Add a mission to the game.
- PUT /api/v1/Game/{id}/add-conversation/{conversationId}: Add a conversation to the game.
- PUT /api/v1/Game/{id}/update-conversations: Update game conversations.
- PUT /api/v1/Game/{id}/update-rules: Update game rules.
- PUT /api/v1/Game/{id}/update-missions: Update game missions.
- PUT /api/v1/Game/{id}/update-players: Update game players.
- PUT /api/v1/Game/{id}/remove-mission/{missionId}: Remove a specific mission from the game.
- PUT /api/v1/Game/{id}/remove-rule/{ruleId}: Remove a specific rule from the game.
- PUT /api/v1/Game/{id}/remove-conversation/{conversationId}: Remove a specific conversation from the game.
- PUT /api/v1/Game/{id}/remove-player/{playerId}: Remove a specific player from the game.

Kill:

- GET /api/v1/Kill/GetKills: Retrieve all Kill entries.
- GET /api/v1/Kill/{id}: Retrieve a specific Kill entry.
- PUT /api/v1/Kill/{id}: Update a kill.
- DELETE /api/v1/Kill/{id}: Delete a kill.
- POST /api/v1/Kill: Create a new kill.
- PUT /api/v1/Kill/{id}/add-location/{locationId}: Add a location to a kill.

Location:

- GET /api/v1/Location: Retrieve all Location entries.
- POST /api/v1/Location: Create a new location.
- GET /api/v1/Location/{id}: Retrieve a specific Location entry.
- PUT /api/v1/Location/{id}: Update a location.
- DELETE /api/v1/Location/{id}: Delete a location.

Message:

- GET /api/v1/Message: Retrieve all Message entries.
- POST /api/v1/Message: Create a new message.
- GET /api/v1/Message/{id}: Retrieve a specific Message entry.
- PUT /api/v1/Message/{id}: Update a message.
- DELETE /api/v1/Message/{id}: Delete a message.

Mission:

- GET /api/v1/Mission: Retrieve all Mission entries.
- POST /api/v1/Mission: Create a new mission.
- GET /api/v1/Mission/{id}: Retrieve a specific Mission entry.
- PUT /api/v1/Mission/{id}: Update a mission.
- DELETE /api/v1/Mission/{id}: Delete a mission.
- PUT /api/v1/Mission/{id}/add-location/{locationId}: Add a location to a mission.

Player:

- GET /api/v1/Player: Retrieve all Player entries.
- POST /api/v1/Player: Create a new player.
- GET /api/v1/Player/{id}: Retrieve a specific Player entry.
- PUT /api/v1/Player/{id}: Update a player.
- DELETE /api/v1/Player/{id}: Delete a player.
- PUT /api/v1/Player/leaveMarker/{id}: Update a player's location.
- PUT /api/v1/Player/by-bitecode/{biteCode}: Change a player's Zombie field to true.
- GET /api/v1/Player/by-bitecode/{biteCode}: Retrieve player by bite code.
- PUT /api/v1/Player/{id}/update-state: Update a players state; true (Zombie), false (Human)

Rule:

- GET /api/v1/Rule/GetRules: Retrieve all Rule entries.
- GET /api/v1/Rule/{id}: Retrieve a specific Rule entry.
- PUT /api/v1/Rule/{id}: Update a rule.
- DELETE /api/v1/Rule/{id}: Delete a rule.
- POST /api/v1/Rule: Create a new rule.

Squad:

- GET /api/v1/Squad: Retrieve a list of squads.
- POST /api/v1/Squad: Create a new squad.
- GET /api/v1/Squad/filterbygameid/{gameId}: Retrieve squads by gameId.
- GET /api/v1/Squad/{id}: Retrieve a specific Squad entry.
- PUT /api/v1/Squad/{id}: Update squad information.
- DELETE /api/v1/Squad/{id}: Delete a squad.
- PUT /api/v1/Squad/{id}/add-player/{playerId}: Add a playerId to a squad.
- PUT /api/v1/Squad/{id}/remove-player/{playerId}: Remove a player from a squad.
- PUT /api/v1/Squad/{id}/add-game/{gameId}: Add a gameId to a Squad.
- GET /api/v1/Squad/size: Retrieve the size of a squad.

Database Schema

1. AppUser:
   - Represents the users of the HvZ system.
   - It's related to the Player entity.

2. Game:
   - Represents individual HvZ games.
   - Has one-to-many relationships with Conversation, Player, Mission, and GameRule.
     - Allows multiple conversations, players, missions, and rules to be associated with a single game.

3. Conversation:
   - Represents conversations within the HvZ game.
   - Has a one-to-many relationship with Message.
     - A conversation can contain multiple messages.

4. Message:
   - Represents individual messages in a conversation.
   - Belongs to a specific conversation.

5. Player:
   - Represents participants in HvZ games.
   - Has a one-to-one relationship with User.
   - Relates to a single User.
   - Has a one-to-one relationship with Kill.
     - A kill entry can only have one player.
   - Has a one-to-many relationship with Message.
     - A player can send/receive multiple messages.
   - Has a one-to-many relationship with Squad and Location.
     - A player can belong to one squad and be located at one location.

6. Kill:
   - Represents instances of players being killed in HvZ games.
   - Has a one-to-one relationship with Player

7. Mission:
   - Represents missions within a game.
   - Has a one-to-one relationship with Location.
     - Each mission is associated with one location.

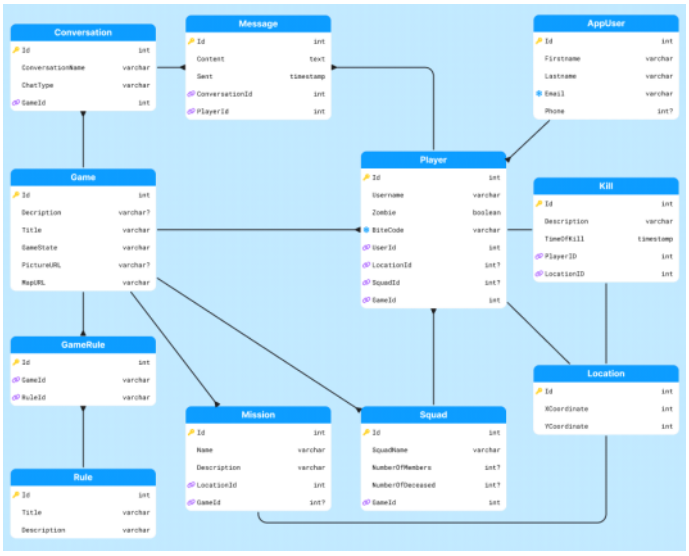
8. Location:
   - Represents locations in the game.
   - Has one-to-one relationships with Mission, Kill and Player.
     - A location is associated to either a specific player, kill or mission

9. Squad:
   - Represents groups of players within a game.
   - Has a one-to-many relationship with Players.
     - A squad can have multiple players.

10. GameRule:
    - Linking table representing a many-to-many relationship between Rule and Game

11. Rule:
    - Represents general rules that can be used across multiple games.
      - A game can have many rules, and a rule can have many games.

ER-Diagram showcasing the database relationships and flow



Considerations and Assumptions in Database Design:

1. User:
   - The AppUser entity represents the users of the HvZ system, and it has a one-to-one relationship with the Player entity. This design choice is made to associate each user with their in-game player.
   - Every HvZ system user will has a corresponding in-game player, and the user's token sub claim will be unique for identification. Currently, the "sub" claim is the only thing that is stored for a user. Potential scalability is to have a profile page where the user can provide other details. Furthermore, we can store more information in the database by default by accessing the user's token claims respectively when the register endpoint is called.

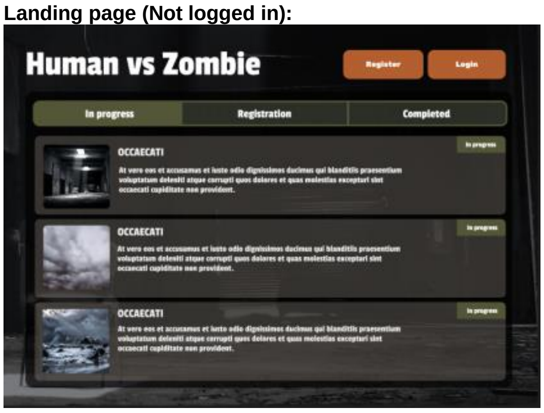
2. Game:
   - A game has conversations, missions, players and rules. By giving game access to other tables we could more efficiently access relevant data in relation to games.
   - In the beginning the table didn't have a MapURL because we thought it could be handled solely on the frontend, but this was added in later stages for easy access.

3. Kill:
   - In the beginning the Kill table had two foreign keys linked to the Player table; with a playerId for the killer and one for the victim. This caused issues when building the database so we moved on to another solution, using a linking table with Roles. The team still encountered issues though when trying to access player locations, so the final version was a simple Kill table referencing Player and Location.
   - In the future the Kill table would ultimately have details about the victim player as well as the killer.

4. Squad:
   - The Squad entity wasn't related to the Game table in the first place, because the team thought we could easily fetch it through the Player table. However, as a fix late in the project, a gameId was added for simpler access. The reason for the change was because the squads were being displayed for all the games. We only wanted to display squads within specific games, thus adding gameId to the Squad table.
   - Squads could in the future have more details, for example "squad leader", "squad score", etc.

Mock-ups (Figma)

Landing page (Not logged in):

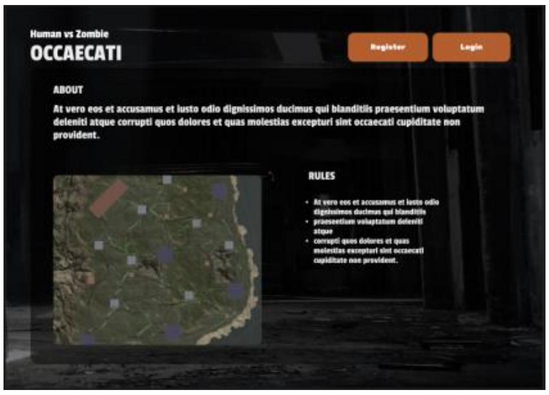


At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis praesentium voluptatum deleniti atque corrupti quos dolores et quas molestias excepturi sint occaecati cupiditate non provident.

At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis praesentium voluptatum deleniti atque corrupti quos dolores et quas molestias excepturi sint occaecati cupiditate non provident.

At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis praesentium voluptatum deleniti atque corrupti quos dolores et quas molestias excepturi sint occaecati cupiditate non provident.

About game (Not logged in):



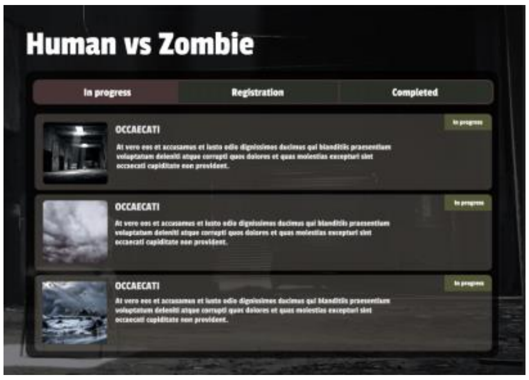
ABOUT

At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis praesentium voluptatum deleniti atque corrupti quos dolores et quas molestias excepturi sint occaecati cupiditate non provident.

RULES

- At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis
- praesentium voluptatum deleniti atque
- corrupti quos dolores et quas molestias excepturi sint occaecati cupiditate non provident.

Landing page (Logged in):



OCCAECATI

At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis praesentium voluptatum deleniti atque corrupti quos dolores et quas molestias excepturi sint occaecati cupiditate non provident.

In progress

OCCAECATI

At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis praesentium voluptatum deleniti atque corrupti quos dolores et quas molestias excepturi sint occaecati cupiditate non provident.

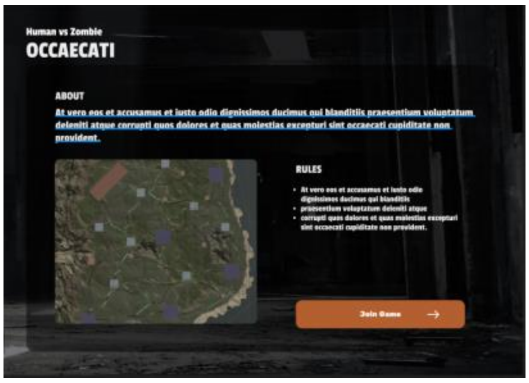
In progress

OCCAECATI

At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis praesentium voluptatum deleniti atque corrupti quos dolores et quas molestias excepturi sint occaecati cupiditate non provident.

In progress

Game details (Logged in):



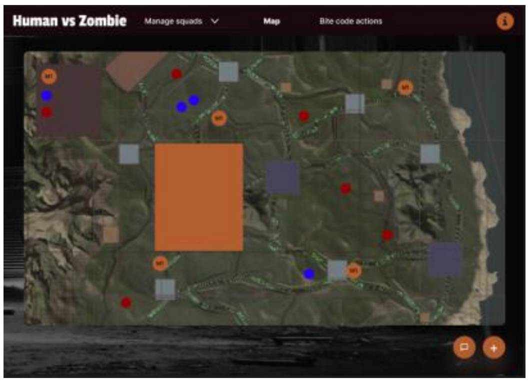
ABOUT

At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis praesentium voluptatum deleniti atque corrupti quos dolores et quas molestias excepturi sint occaecati cupiditate non provident.

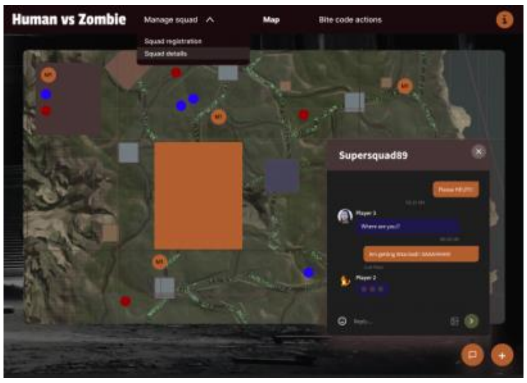
RULES

- At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis
- praesentium voluptatum deleniti atque
- corrupti quos dolores et quas molestias excepturi sint occaecati cupiditate non provident.

Game:



Game with chat:

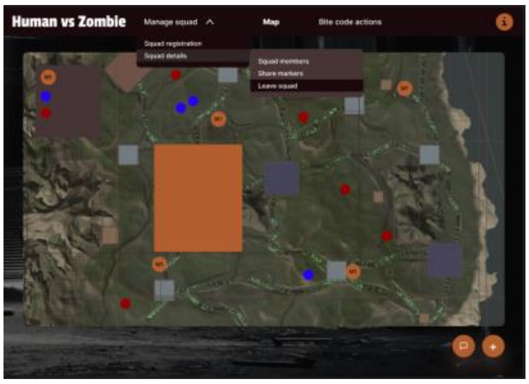


Please HELP!!!

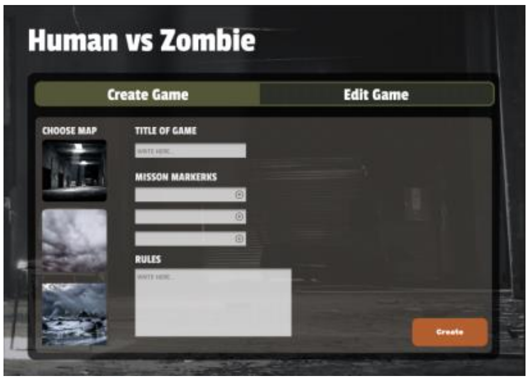
Where are you?

Im getting Attacked! AAAAHHHH

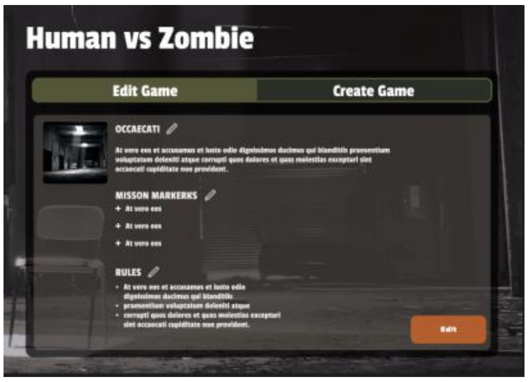
Game with navbar:



Create Game (Admin):



Edit Game (Admin):



OCCAECATI

At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis praesentium voluptatum deleniti atque corrupti quos dolores et quas molestias excepturi sint occaecati cupiditate non provident.

MISSON MARKERKS

+ At vero eos
+ At vero eos
+ At vero eos

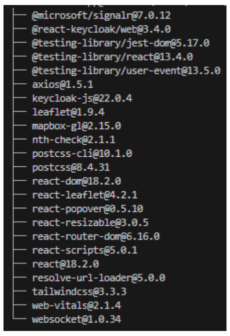
RULES

- At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis
- praesentium voluptatum deleniti atque
- corrupti quos dolores et quas molestias excepturi sint occaecati cupiditate non provident.

Relevant Implementation Details:

Front-end Technologies:

- **Tailwind CSS:** Tailwind CSS is employed to provide a framework for styling and designing the user interface, ensuring a responsive and visually appealing web application.
- **React.js:** React.js serves as the primary front-end framework for building the user interface, enabling the creation of dynamic and interactive components.
- Dependencies:



```
├── @microsoft/signalr@7.0.12
├── @react-keycloak/web@3.4.0
├── @testing-library/jest-dom@5.17.0
├── @testing-library/react@13.4.0
├── @testing-library/user-event@13.5.0
├── axios@1.5.1
├── keycloak-js@22.0.4
├── leaflet@1.9.4
├── mapbox-gl@2.15.0
├── nth-check@2.1.1
├── postcss-cli@10.1.0
├── postcss@8.4.31
├── react-dom@18.2.0
├── react-leaflet@4.2.1
├── react-popover@0.5.10
├── react-resizable@3.0.5
├── react-router-dom@6.16.0
├── react-scripts@5.0.1
├── react@18.2.0
├── resolve-url-loader@5.0.0
├── tailwindcss@3.3.3
├── web-vitals@2.1.4
└── websocket@1.0.34
```

Back-end Technologies (ASP.NET Core - .NET):

- **AutoMapper.Extensions.Microsoft.DependencyInjection (v12.0.1):** AutoMapper is used to facilitate object-to-object mapping in the application, streamlining the data transfer process between different parts of the system.
- **Microsoft.AspNetCore.Authentication.JwtBearer (v6.0.22):** This package is essential for handling JSON Web Token (JWT) authentication, ensuring secure and authorized access to the application.
- **Microsoft.EntityFrameworkCore.Design (v7.0.11):** Entity Framework Core Design is utilized for managing the database schema and migrations in the application's back end.
- **Microsoft.EntityFrameworkCore.Sqlite (v6.0.21):** Sqlite is used as a lightweight and self-contained database engine, allowing efficient data storage and retrieval within the application.
- **Microsoft.EntityFrameworkCore.SqlServer (v7.0.11):** This package provides support for SQL Server as the relational database management system, ensuring data persistence and retrieval.
- **Microsoft.EntityFrameworkCore.Tools (v7.0.11):** Entity Framework Core Tools are used for database migrations, scaffolding, and other database-related operations.
- **Microsoft.VisualStudio.Web.CodeGeneration.Design (v6.0.16):** This package is employed for code generation tasks, making it easier to generate code for controllers, views, and other components.
- **Microsoft.Extensions.Logging (7.0.0):** Microsoft.Extensions.Logging is part of ASP.NET Core and offers a common logging abstraction, allowing flexibility in selecting various logging providers.
- **Newtonsoft.Json (v13.0.3):** Newtonsoft.Json is used for JSON serialization and deserialization, enabling efficient handling of JSON data within the application.
- **Serilog.Sinks.Console (4.1.0):** This package is an extension for Serilog, specifically used for writing log events to the console, which is useful for monitoring log messages during development and debugging.
- **Serilog (3.0.1):** Serilog is used for structured logging in your application. It allows you to log events in a structured format, making it easier to analyze and filter log data.
- **Swashbuckle.AspNetCore (v6.5.0):** Swashbuckle.AspNetCore integrates Swagger UI, providing interactive API documentation for easy exploration of the application's API endpoints.

Back-end Frameworks

- Microsoft.AspNetCore.App
- Microsoft.NETCore.App

Deployment and Configuration

25 October 2023 14:58

Backend Database Deployment in Azure:

We have used Microsoft Azure to host our backend code and database. The backend code is developed in Visual Studio 2022, and the database is managed using Microsoft SQL Server Management 2019.

Database Connection:
The backend application requires a connection to the Azure database. It's crucial to ensure the correct connection string is used.

Steps to Deploy the Backend:

Azure Portal Access:
Ensure you have the necessary permissions to access the Azure Portal to deploy resources.

Database deploying
1. Create a Database: In the Azure Portal, navigate to the Azure SQL Databases service and create a new database.

2. Set Firewall Rules: Configure the firewall rules to allow your backend application to access the database by whitelisting your IP address or IP range.

3. Connection String: Retrieve the connection string for the created database from the Azure Portal. This connection string is essential for your backend application to connect to the database.

Backend Code Deployment:

a. Build the Code: Ensure that your code is successfully built, and all necessary dependencies are included.

b. Publish to Azure: In Visual Studio, right-click your project, choose "Publish," and select Azure as the deployment target. Follow the prompts to publish your code to an Azure App Service.

c. Configure Connection String: In your Azure App Service, navigate to the Configuration settings. Add the connection string obtained earlier as an application setting, naming it appropriately ("DatabaseConnection").

Frontend Deployment on Vercel:

The frontend of our application is hosted and deployed on Vercel, a platform designed for seamless frontend hosting.

Steps to deploy Frontend:

Vercel Account:
Ensure you have access to a Vercel account.

Repository Link:
Connect your frontend code repository (e.g., GitHub) to your Vercel project.

Environment Variables:

Configure any required environment variables in Vercel that your frontend relies on, such as API endpoints.

Automated Deployment:
If you want it is possible to set up continuous integration/continuous deployment (CI/CD) in Vercel to automate frontend deployment from your repository.

Testing:
Test your deployed on backend and frontend to verify that it can successfully connect to the Azure database. Address any connection issues that may arise during testing.

To access our deployed backend application on Microsoft Azure, follow these simple steps:

Open a web browser.

Enter the URL https://humansvszombies.azurewebsites.net/swagger/index.html.

Explore the Swagger documentation to learn about the available API endpoints and interact with the application.

Enter this url to enter our frontend application on Vercel: Human vs Zombie (humanvszombies-lmlo1pb9z-tobias-projects-64a04e43.vercel.app)

Collaboration Tools

23 October 2023 15:33

Collaboration Tools

To facilitate efficient communication and coordination among our team members, we have adopted a well-structured approach using the following collaboration tools:

1. **Jira - Agile Project Management:** Jira serves as our central project management system. We use it to plan sprints, organize our backlog, and prioritize tasks using the MoSCoW method (Must-have, Should-have, Could-have, Won't have). With Jira, we have a clear overview of our project's progress, and it helps us stay aligned with our goals.

2. **Slack - Real-Time Communication:** Slack is our go-to platform for real-time communication. Slack has been helping our team members to discuss tasks, share updates, and seek quick assistance. This real-time communication streamlines our workflow and fosters a sense of collaboration, even when working remotely.

Version Control - Git

23 October 2023 15:24

Version Control

In our project, we have effectively used version control to manage the source code and track changes efficiently. We have implemented a well-structured Git workflow, and here's how we've managed version control:

1. **Git Bash and PowerShell:** We've utilized Git Bash as the primary terminal for managing the front-end codebase and PowerShell for the back-end. This combination has allowed us to work seamlessly with our respective codebases.
2. **GitHub Collaboration:** We've leveraged GitHub as our collaboration platform. GitHub provides a centralized location for streamlines collaboration among team members. Our workflow includes:
    - **Branching Strategy:** We create feature branches based on tasks and user stories. This aligns with our agile development process and allows us to work on specific features or fixes independently. These branches are typically created through Jira, where we manage our sprints, backlog, and MoSCoW prioritization.
    - **Commit Messages**: Each commit conveys the purpose and changes made to ensure that team members can easily understand the updates. This practice aids in code review and maintaining a clean codebase.
    - **Pull Requests:** Before merging code into the main branches, we use pull requests to review, discuss, and ensure code quality. This process encourages collaboration and helps maintain a robust and bug-free codebase.
    - **Code Reviews:** Code reviews are a crucial part of our version control process. They help identify issues, provide suggestions for improvement, and ensure that our code meets quality standards.
3. **Integration with Jira:** Our integration with Jira, an agile project management tool, helps us manage tasks, sprints, backlog, and prioritize features (MoSCoW). This integration streamlines our development process and aligns our code changes with project objectives.

Code Review

23 October 2023 15:39

Code Review

Our commitment to maintaining code quality extends beyond version control. We've also implemented a robust code review process. This process complements our version control strategy and ensures the codebase's integrity and consistency.

- **Code Review Process:** Our code review process is conducted entirely within our version control platform, which is GitHub in our case. It's an integral part of how we manage our codebase.
- **Commit Messages:** Each commit conveys the purpose and changes made to ensure that team members can easily understand the updates. This practice greatly aids in code review and maintaining a clean codebase.
- **Pull Requests and Code Reviews:** Before merging code into the main branches, we use pull requests and code reviews to review, discuss, and ensure code quality. This process encourages collaboration and helps maintain a robust and bug-free codebase. Our code review process follows best practices and emphasizes:
    - **Thorough Review:** We conduct a thorough line-by-line review of code changes within the version control platform.
    - **Feedback and Collaboration:** Reviewers can provide feedback directly on the specific lines of code, highlighting issues, suggesting improvements, or asking questions. This feedback loop promotes collaboration among the team members.