**DnDeity**

**Team 22 -Design Document**

Benjamin Crumbacher, Nicholas Fordyce, Christian McKain, Riley Turnbull

# **Index**

* **Purpose** 1
  + Functional Requirements 2
  + Non-Functional Requirements 2
* **Design Outline** 3
  + High-level overview 3
  + Hosting 3
* **Design Issues** 4
  + Functional Issues 4
  + Non-Functional Issues 5
* **Design Details** 7
  + Database Class Diagram 7
  + Sequence Diagrams 9
  + UI Mockups 12

# **Purpose**

The average Dungeons and Dragons player really enjoys playing DnD, but unless they really like pencil and paper, they probably don’t like the constant calculating and writing and erasing that also comes with playing the game. Not only is this tedious, but it also takes up a lot of time. Thus there is a need for a tool that can do this bookkeeping for you.

That’s where DnDeity comes in. Our app is designed to remove the tediousness and time it takes doing the boring administrative stuff and instead focus on what the players really want: to play the game. It will provide tools for the dungeon master to create the games and invite players, as well as create maps and administrate the game. This removes the tedious tasks while maintaining the interaction with the players that makes DnD fun. The app also will automatically update things for the players, minimizing erasing and flipping through pages of the DnD player handbook.

**Functional Requirements**

1. **Users can create a DnDeity account**
   1. As a user, I would like to be able to register for a DnDeity account.
   2. As a user, I would like to be able to login to DnDeity.
   3. As a user, I would like to be able to reset my password.
2. **Users can create characters**
   1. As a player, I would like to be able to create a character sheet and set details such as a character’s name, ability modifiers, level, class abilities, and description.
   2. As a player, I would like to update and delete my existing character sheets.
   3. As a player, I would like to view and update my inventory.
3. **Users can create maps**
   1. As a dungeon master I would like to be able to make a map.
   2. As a dungeon master I would like to add monsters and events to maps.
4. **Users can create and join games**
   1. As a player, I would like to be able to join a game.
   2. As a dungeon master I would like to be able to make a lobby to host a game.
5. **Users can play a game of DnD**
   1. As a player, I would like to see a list of spells available to me and use them.
   2. As a player, I would like my character to be able to perform attacks and actions available to me during a game.
   3. As a dungeon master I would like to see what spells the players use.
   4. As a dungeon master I would like to see the results of player rolls.
   5. As a dungeon master I would like to have initiative order sorted for me.
   6. As a dungeon master I would like to add effects to the players.
   7. As a dungeon master I would like to update the stats of monsters in fights.

**Non-Functional Requirements**

1. **Usability**
   1. As a user, I would like for the app to be easy to use.
   2. As a user, I would like for the app to have a visually appealing user interface.
2. **Security**
   1. As a developer, I would like for API requests to require authentication.
   2. As a user, I would like for other users to not be able to modify or delete my data.
3. **Scalability**
   1. As a developer, I would like for our application to be scalable.
   2. As a developer, I would like for it to be easy to add new content and features.
4. **Client**
   1. The client will be a React web application.
   2. The client will allow users to access and use the different features provided by DnDiety.
5. **Server**
   1. The server will handle requests from our frontend.
   2. The server will access our database to retrieve needed information.

# **Design Outline**

**High-level overview**

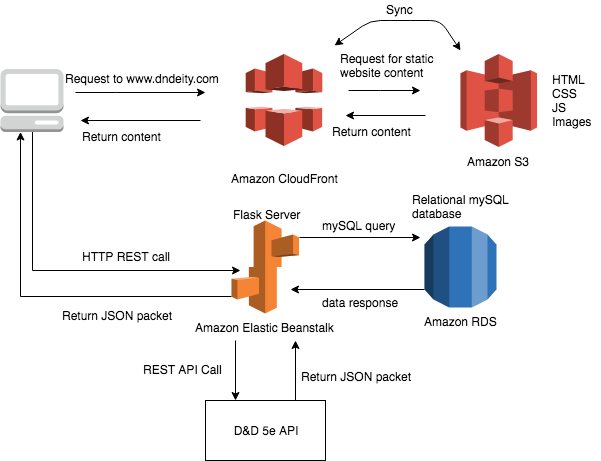
DnDeity will use a client-server architecture. There can be many clients connected to the server. The client will request and send data to the server. When the client needs information from the database, the server will query the database. When the client needs to send data to the database, the server will update the database.

****

1. Web Client
   1. The client will display the front-end user interface of the system.
   2. The client will display and add/modify/delete database data by making HTTP requests (get/put/post/delete)
2. Flask Server
   1. The server will interface between the client application and the database.
   2. The server will fetch information from the database and send it to the client.
   3. The server will take data from the client and update the database.
3. mySQL Database
   1. The database will store all the information about users, characters, and rooms
   2. The database will respond to mySQL queries.

**Hosting**

The application will be hosted on Amazon Web Services. The static content (HTML/CSS/JS/Images) will be stored in Amazon S3 and cached in CloudFront. When a user makes a request to the URL, CloudFront will deliver the content to the browser. When a user interacts with the application, an HTTP call will be made to the Python Flask server hosted on Amazon Elastic Beanstalk. The Flask server will then interpret the call and interact with the database as necessary, returning a JSON packet with the necessary information. The database will be a relational mySQL database hosted on Amazon RDS. This will store all the information on users, characters, and rooms. The Flask server will also make calls to the D&D API when necessary to populate the database with game-specific statistical and categorical information. This eliminates the need to manually populate these fields, and allows the modification of data if necessary.

****

# **Design Issues**

### **Functional Issues**

1 - How should users login to the app?

Options:

* Option 1: Login using another account, such as their Google or Facebook account
* **Option 2: Create a DnDeity account**

Choice: Option 2

Creating a DnDeity account is the simplest solution. If we used a third party login then we would have to follow that third party’s authentication flow. It is much simpler to store our own usernames and passwords.

2 - How should users join a game?

Options:

* Option 1: Lobby List
* Option 2: Direct link
* **Option 3: Type in a code**

Choice: Option 3:

In order to join a game, users will type in a code that corresponds to room they want to join. We decided not to do a list of lobbies because that would make it difficult for users to find the right game that they want to join. By using a code or link users can go straight to the correct game. One of the reasons we opted to use a code rather than a link because it is simpler to type in. Also, if a user went came from another site using the link then they would have to log in, while if they type in a code while inside the app then they are already authenticated.

### **Non-Functional Issues**

1 - What database software should we use?

Options:

* **Option 1: MySQL**
* Option 2: PostgreSQL
* Option 3: Non-relational database

Choice: Option 1

We choose MySQL to use as our database engine. We opted to go with a relational database over an non-relational database because it allows us to effectively link information together, such as relating a user to multiple characters and maps. When deciding between MySQL and PostgreSQL we choose MySQL, since it is easier to manage and fast.

2 - What programming language should we use for our backend?

Options:

* **Option 1: Python**
* Option 2: Go
* Option 3: Java

Choice: Option 1

We decided to program our backend server in Python. We choose Python over Go because only one of our group members has previously used Go, but everyone has programmed in Python before. We also choose Python because we think it is a simpler language to program in than Java or other languages. By choosing Python we can use the Flask framework, which will let us easily set up a REST API to address requests.

3 - How will we get the necessary Dungeons and Dragons information?

Options:

* Option 1: Manually input necessary details into the database or code using the D&D 5E book as a reference
* **Option 2: Make requests to a D&D 5E API**

Choice: Option 2

For this issue we decided to use an existing API to get the necessary information like monsters and loot. Using an API will save us a ton of time manually inserting data into our database. Furthermore, we will store all the data we get from the API in our database. This will allow us to cut back on excessive calls to their API and allow us to easily modify any data if we need to.

4 - What framework(s) will we use for the User Interface?

Options:

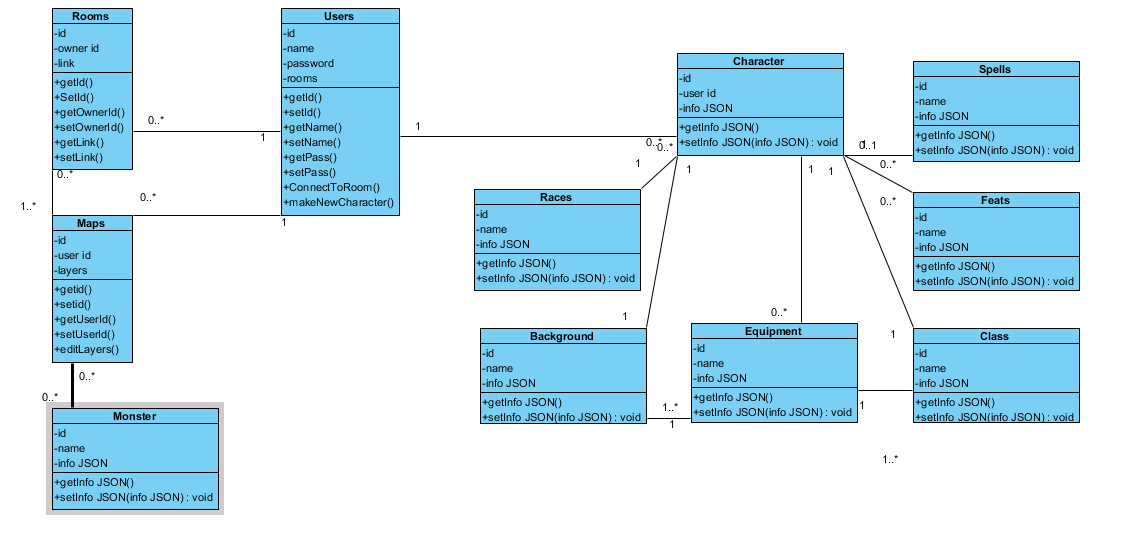
* **Option 1: React**
* Option 2: Angular
* **Option 3: Bootstrap**

Choice: Option 1 and 3

We decided to use React to create and render the UI because although not the easiest of the frameworks it is the most versatile, especially when there is lots of dynamic content on one page, as there will likely be in our app. React also can be used as a base to move into mobile development making it flexible for the case that we decide to develop a mobile companion app, something the other frameworks do not share. Also one of our group members has experience working with React, and since the rest of us have only limited experience with the others, React makes the most sense. We will also use Bootstrap since it has many premade and styled components that will expedite frontend development.

# **Design Details**

**Database Class Diagram**



**Description of Data Classes and their Interactions**

Our data model has been designed to store the data unique to our software as well as data from DnD APIs. Therefore most class objects contain a generic JSON field to store the large amount of text and unique fields found in these DnD objects.

User:

* Represents an individual using the program
* This is where most of the individuals files can be linked back to
* Is first created when a new user makes and account and then is referenced anytime a user logs in

Rooms:

* Represents the play space in which Dungeon Masters and Players interact
* Responsible for assigning the roles of Dungeon Master and Player before game space is loaded
* Manages other games systems like maps and player initiative
* Created when a user makes a new Room, loaded from the server when an existing room is requested

Maps:

* Represents a matrix that will be displayed for Players and Dungeon masters to move pieces, as well as containing graphical info such as backgrounds and walls
* Created when user requests to make a new map
* Updated when user makes changes to the map, or players / the dungeon master moves pieces on the play space

Character

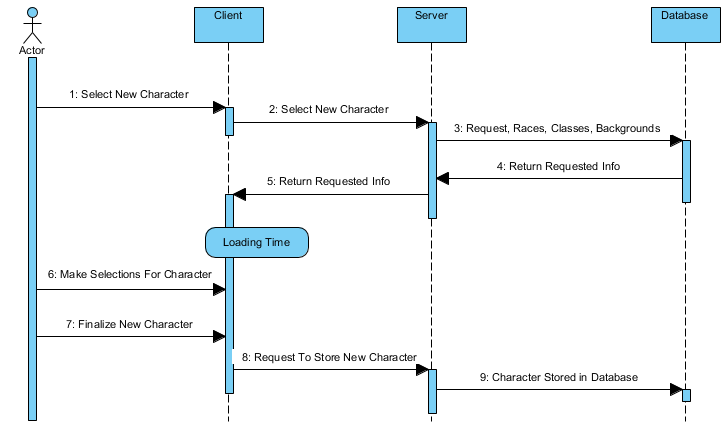
* Represents a user's character sheet, a state block containing information on a specific character such as inventory, level, and race.
* Created when user requests to make a character
* Updated when user selects to manage a character or is referenced in a game session

Spells, Monsters, Background, Feats, Races, Classes, Equipment

* Represents the system in which the api is interpreted by the system, into JSON that the server can interpret.
* The categories Spells, Monsters, Background, Feats, Races, Classes, and Equipment all have their own class for storing information as all the categories don't share any commonalities to make one centralised data management class. Furthermore some classes such as spells and monsters must be requestable quickly in the during a game session
* Created at time of server activation

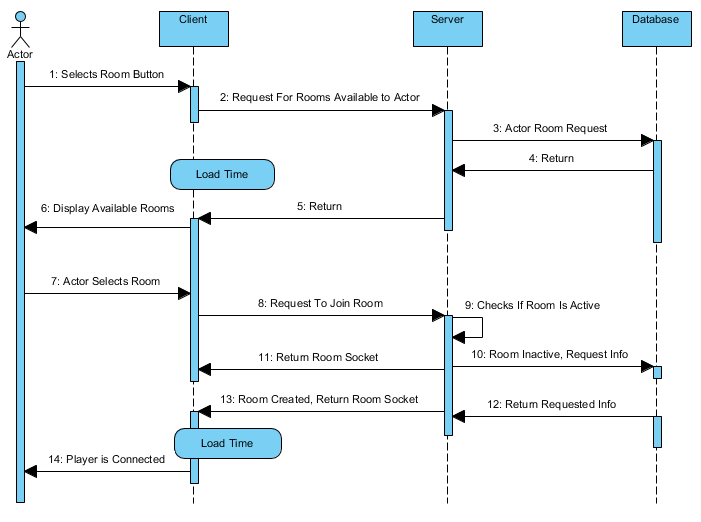
**Sequence of events when then user begins to make a new character**

On the web page a user can select to make a new character, the request is then made to the server than to the database for all the JSON files relevant to making a character (spells, races, equipment, classes etc.). From there the client waits until the user has selected finish character, at which point a new character object will be made using the info provided by the user and uploaded to the database for future use. Furthermore its saved on the client to be shown on the user character page.

****

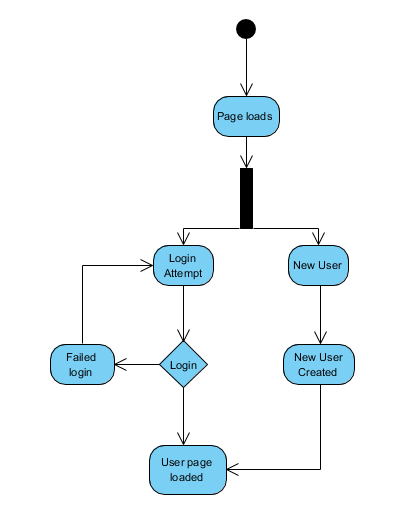
**Sequence of events when the user selects to join a room**

When the user selects to join a room the client will ask the server if the room is loaded, if it is loaded, the client will request the socket to the room. In the event the room is inactive the server will request the room from the database, load it onto the server, then provide the client with the socket and connect the user. The user loads in and the room page is displayed.

****

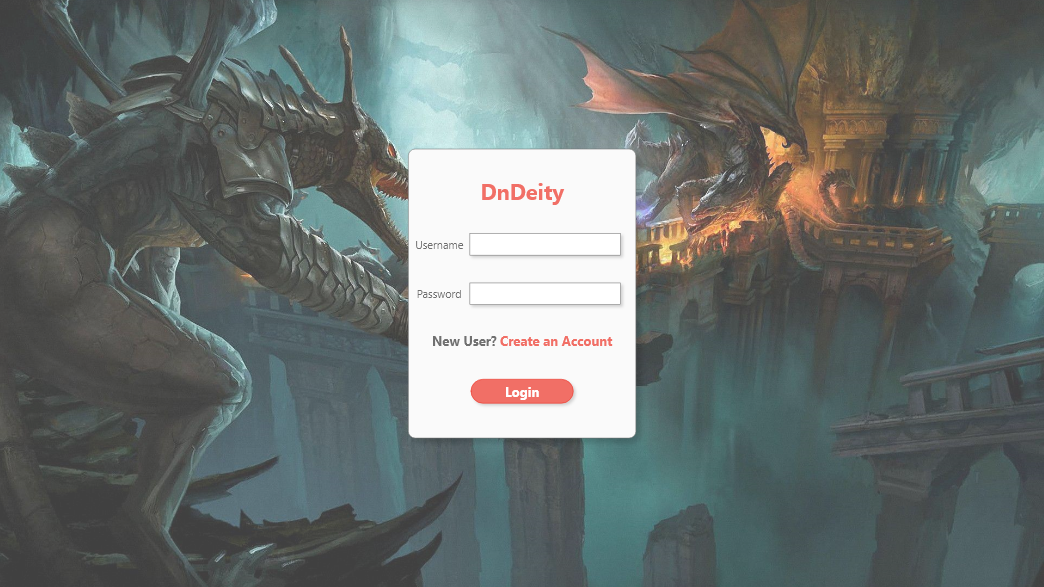
**Action Diagram of the login process**

User loads the page, and either fills in the login info or selects new user, new users fill in the field, then logs the new player in. If the user logs in normally then the server will check login credentials and either accept the login or reject it.

****

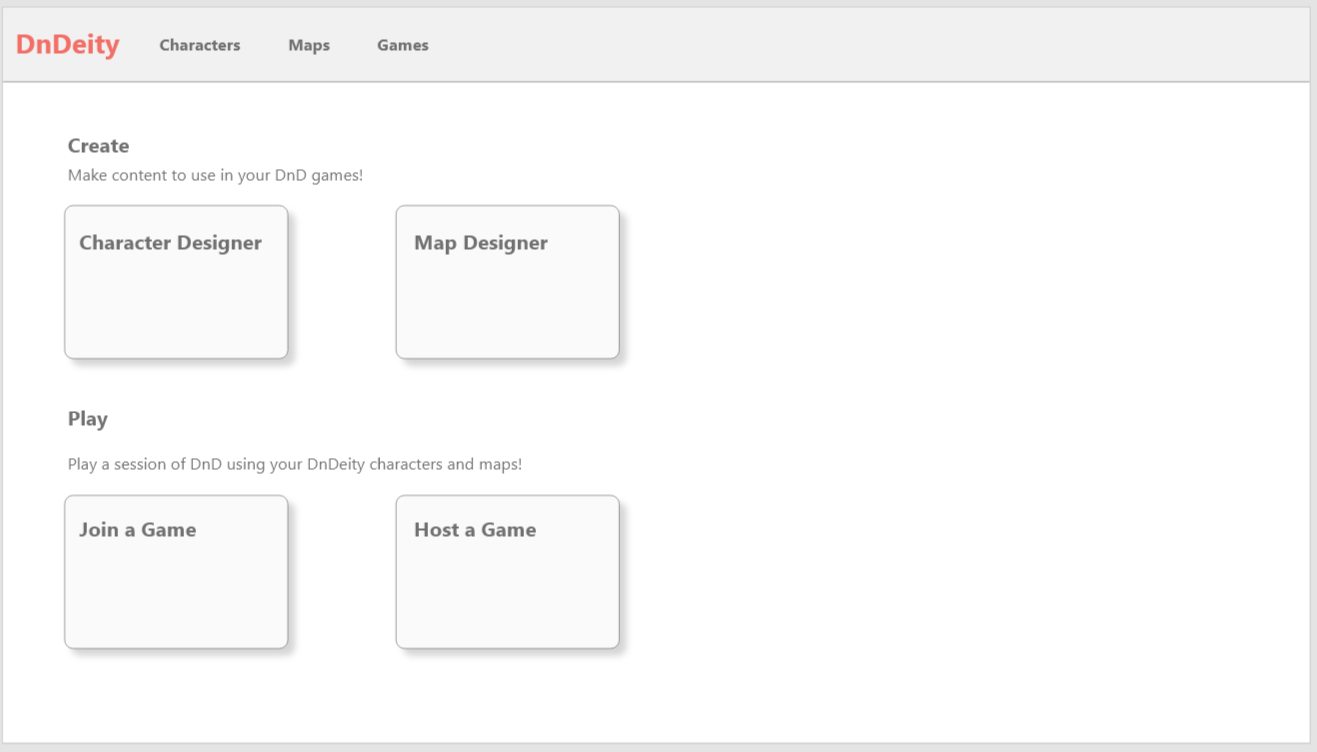
### **UI Mockups**

**Login Page:**



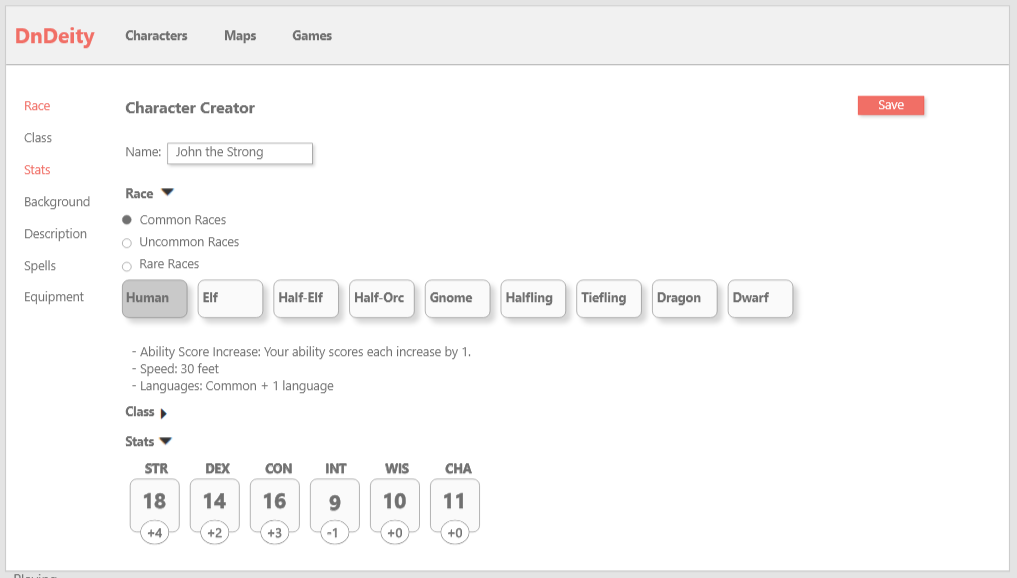
This is the login page for our app. Users will input the credentials for their DnDeity account and then press the “Login” button to go the home page. If the user does not have an account, they can click the “Create an Account” button to register for an account.

**Home Page:**



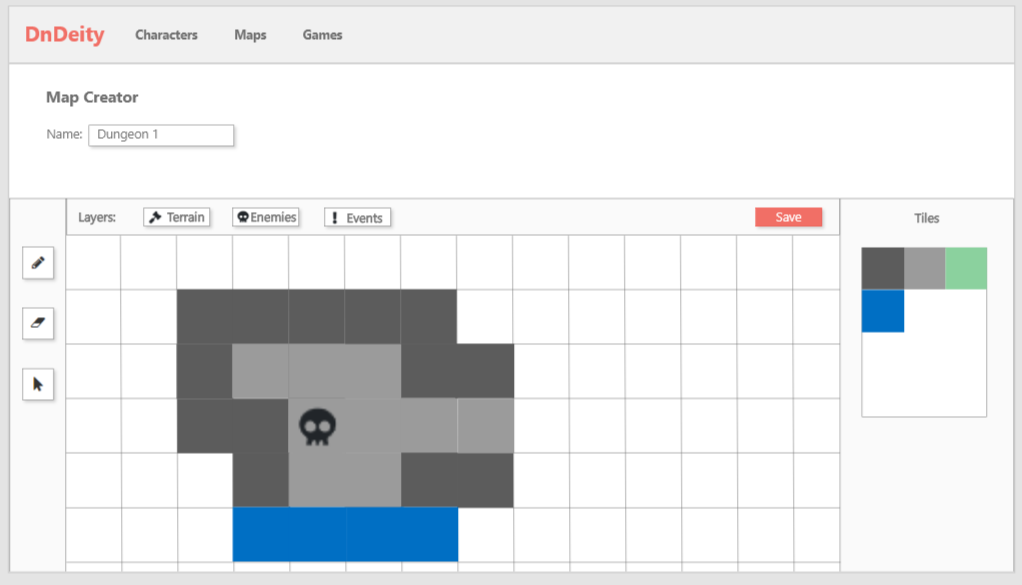
This is the homepage for our application. On the top of the screen is the navigation bar, which users can click to access lists of their characters, maps, and played games. Clicking the “Character Designer” button will take users to the character creation page to create a new character. Similarly, the “Map Designer” will take users to the map creation page to create a new map. The “Host a Game” button allows a dungeon master to create a lobby for other players to join and play the game using their maps. The “Join a Game” button allows players to join games using their created characters.

**Character Creator:**

****

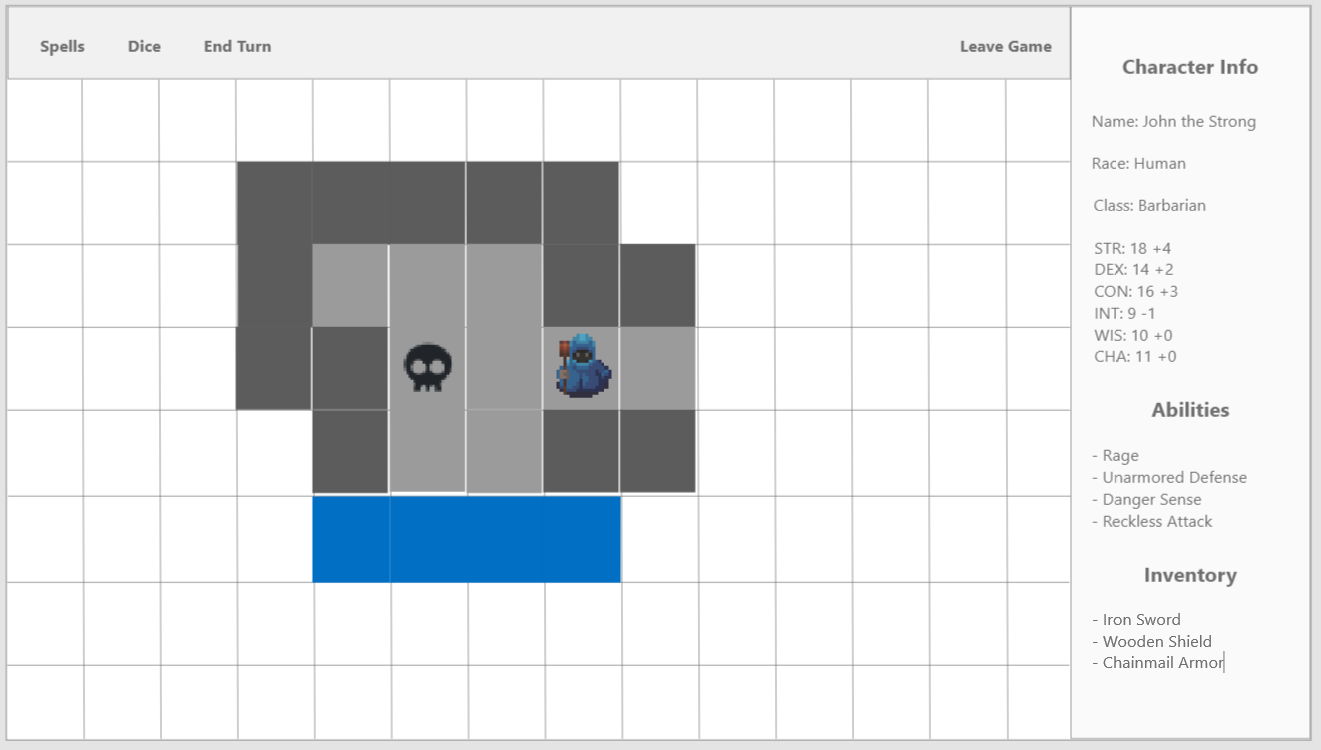
This is the screen where users can create or edit their character sheets. This page is a form that consists of multiple collapsible sections, which can be quickly navigated using the sidebar. Sections that are completed will be marked in color on the sidebar. When a user is finished, they click the “Save” button to save their character.

**Map Creator:**

****

This is the map creator tool where users can make maps to use in their games. The buttons on the left sidebar are for drawing, erasing, and selecting respectively. The buttons on top allow the user to change what part of the map they want to edit. For example, if “Terrain” is selected, then users can draw the dungeon layout. If “Enemies” is selected, then the user can place enemies and edit enemy details. The section on the right of the map will change depending on what layer the user is on. The right section will be a tile selector if Terrain is selected, and a form for editing event/monster details if the other two layers are selected.

**Game View:**



This is the screen where players can play the game. In the center of the screen is one of the maps that the dungeon master created. Players can drag and drop their characters onto different spaces. On the right side is a rollup of their character information. On the nav bar there is a spell button that will allow them to search for spells they can use, a dice feature for rolling different sided die, and an “End Turn” button that will end their turn and signal the next player to start their turn. The “Leave Game” button lets players leave the game when they are finished.