**Next Level**

**Team 3**

**Joseph Chen, Christopher Hyman, Trenton Reeves, Michelle Song, Richard Stump, Nicholas Zogbi**

#### Purpose:

Making games can be complex and intimidating, especially to a beginner or layperson. Whether from scratch or with an engine, there is a steep learning curve, and producing one requires immense dedication and effort. Furthermore, some people may even just want to play from a large library of custom content. Next Level is a platforming game which provides users a simple means to create their own levels/assets, share them, and play content curated by others while being easier to use than alternatives since it requires no coding, minimal effort, and not as much creativity due to preprogrammed rules and usage of prebuilt and shared assets.

#### Design Outline:

**High Level Overview**

This project is a web based platforming game, that allows users to create levels and play user created levels. This game will be based on the client-server model in which the server will handle a large number of clients using postgREST.

Using the database to store and load data for the server. The server will be able to handle requests from the client and give feedback to them with the aid of the database.



#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

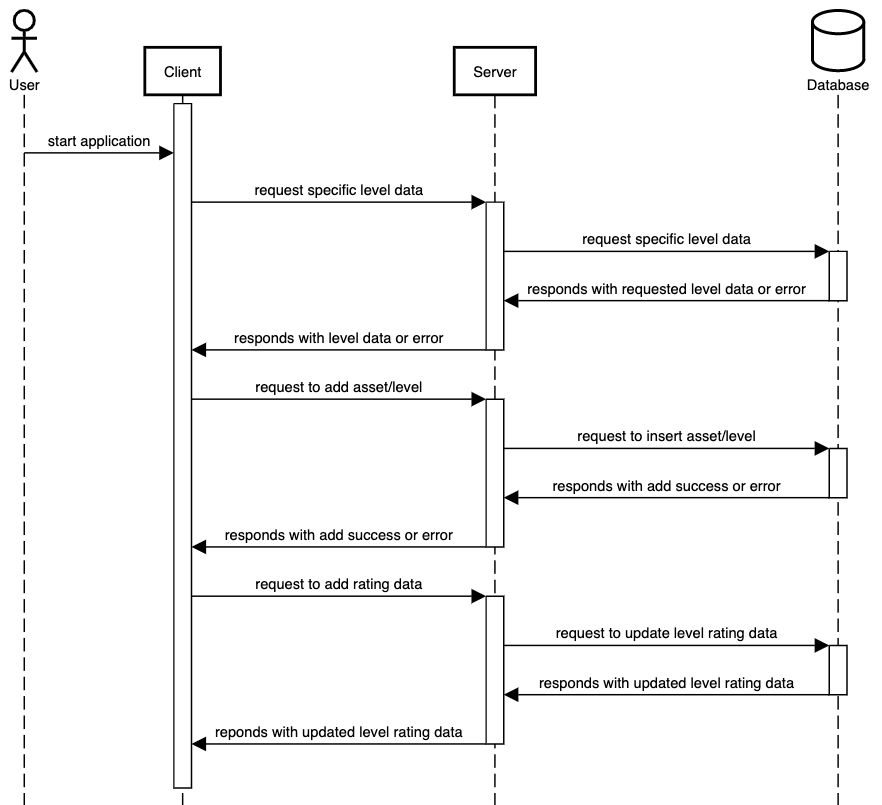
#### 

#### 

#### 

#### Sequence of Events Overview

This sequence of events shows the basic interactions between user, client, server, and database. The user begins the sequence by opening the application and then logging in. When the user enters in their information, the client sends a request to the server which sends the query to the database and receives data from the database. After logging into the application, the client can send requests to the server for different events such as searching for a level given tags/author/name, adding an asset or level, and adding a rating to a level. These requests are passed from the client to the server and then queried to the database. If the request was a change and is valid the database is updated, else the database returns the result to the server which in turn returns it to the client.

****

#### Design Issues:

#### Functional Issues

1. What information do we need for signing up?

* Option 1: Username and password
* **Option 2: Username, password, and email**
* Option 3: Username, password, email, and phone number

Choice: Option 2

Justification: When creating an account, a username and password are obviously needed to separate each unique user from others. An email may be recorded for further forms of verification in case the user forgets their username or password. This allows the user to receive an email containing their username if forgotten, or receive an email allowing them to reset their password if forgotten. A phone number could be another good form of verification, however this costs the user when receiving a text message. We plan on having our game be completely free.

2. Can the user create custom tags for their custom levels?

* Option 1: Yes
* **Option 2: No**

Choice: Option 2

Justification: Creating custom tags may be a nice feature, however it will be hard for us to create a system that filters levels by tags if this were the case. There would be too many possibilities on our filter list and would create an extremely long filter list for the user which would be very hard for them to use. By only allowing users to choose from premade tags, this makes our filter system much cleaner and easier for the user to use.

3. What type of rating system are we going to use?

* **Option 1: 0.5 increments from 0 to 5 stars, have average stars on the map rounded to nearest 0.5**
* Option 2: Number score up to 5, have average score be rounded to nearest 0.1
* Option 3: 1 increments from 0 to 5 stars, have average stars on the level rounded to nearest whole number
* Option 4: Thumbs up and thumbs down

Choice: Option 1

Justification: When it comes to rating systems in video games, many games tend to use stars to describe how good something is. Some may use stars to represent how good an item is, or some may use stars to represent how good levels are. We plan on using stars to make Next Level a game with great aesthetics. Along with this, having the average rating rounded to the nearest 0.5 will allow our stars to look much more neat as compared to having numbers such as 0.6 or 0.3 that would just make the stars look unnatural. Having increments of one also provides not enough variety in rating options. Some levels may be extremely close to perfect but not there yet, hence a 4.5 star rating would be the best scoring option rather than a 4 star. Lastly, having a thumbs up and thumbs down rating system is not really descriptive of the rating of the level. This is because it simply says whether a map is good or bad, not how good it is or how bad it is. Along with this, if we used a thumbs up and thumbs down rating system, it would be hard to rate maps that are average, not good or bad.

5. How are we going to manage reported assets?

* Option 1: Delete it instantly
* Option 2: Leave it in the game until it is reviewed; remove if it is inappropriate
* **Option 3: Remove it, then place it back in the game if it passes the review**

Choice: Option 3

Justification: We plan on removing it first since there is a possibility that the asset is inappropriate, but keep the asset stored in the database. This allows an inappropriate asset to be removed from the game as soon as possible. If the asset turns out to not be inappropriate, we will place it back into the game to be used like normal. Leaving it in the game until it is reviewed may follow the “innocent until proven guilty” idea, this leaves the possibility of an inappropriate asset being left in the game longer than it should be. Our game is designed for people of all ages, so we do not want anything inappropriate in our game. Lastly, deleting the asset instantly seems unfair since it is not checked or reviewed for being inappropriate.

6. How are levels with reported assets going to recognize and deal with reported assets?

* **Option 1: Assign each asset with a UUID. Level metadata contains a list with UUIDs of all the assets the level contains. When an asset is removed, find all levels containing this UUID and remove the asset from the level. Replace the asset with a default asset from the same category. (Character asset reported and removed, replaced with a character asset).**
* Option 2: Assign each asset with a UUID. Level metadata contains a list of UUIDs of all the assets the level contains. When an asset is removed, find all levels containing this UUID and remove the asset from the level.

Choice: Option 1

Justification: By having each asset assigned with a UUID, it will be extremely easy to identify which level contains a certain asset. By using a list to keep track of the UUIDs of the assets in each level, we can easily find whether the asset is in the level or not. We also plan on replacing the reported asset with a default asset of the same category. For example, if a character asset was reported and removed, we would replace that asset in the level with a default character asset. This is because if we do not do that and only remove the reported asset, there is the possibility that the level becomes very empty.

7. What happens if a user submits an unbeatable level?

* Option 1: Allow it
* **Option 2: Require users to beat their level before they can submit it**

Choice: Option 2

Justification: Requiring users to beat their own levels will stop users from submitting troll levels that are impossible to beat. Allowing users to submit unbeatable levels could lead to a decrease in user enjoyment because they could potentially be spending a lot of time on a level only to never be able to beat it. Having a user beat their own level before submitting it will alleviate this problem by making the creator filter their own level.

8. How big of a level can a user create?

* Option 1: Infinitely big level
* **Option 2: Capped by a max level size**

Choice: Option 2

Justification: We choose to cap our levels at a max level size. This helps us manage how much storage is needed to store each level. If users were able to create infinitely big levels, the level could contain so many assets and features that it takes up a huge chunk of memory, being unfair to other users. By capping our levels at a max level size, we can have all our users have the same experience when creating levels.

#### Non-Functional Issues

1. What server are we using?

* Option 1: Spring Boot
* **Option 2: postgREST**

Choice: Option 2

Justification: We are using postgREST as our server, which turns a postgreSQL database directly into a RESTful API. By doing so, it will be very easy for us to make REST calls from our client to our database. Along with this, we do not have much experience with Spring Boot, so postgREST is much easier to learn when it comes to creating our backend. In addition, postgREST has a good security system, in which authorization is implemented through permissions and database roles.

2. What database should we use?

* Option 1: MySQL
* Option 2: NoSQL
* **Option 3: postgreSQL**

Choice: Option 3

Justification: As seen before, we are planning on using postgREST as our server. postgREST directly connects to postgreSQL and turns it into a RESTful API. This leads us to choose postgreSQL as our database since it is designed to work very well with postgREST, as they are connected together. By using postgreSQL, it will be extremely easy to make calls from our client to our database.

3. What frame rate should we use?

* Option 1: less than 30 frames per second
* **Option 2: at least 30 frames per second**
* Option 3: at least 60 frames per second

Choice: Option 2

Justification: Games are basically unplayable if they are below 30 frames per second. Since this is our first time programming a game we are not that experienced with it. Therefore, we want the framerate to make the game playable but also have it obtainable to someone programming their first game, so we want a frame rate of at least 30 frames per second.

4. What framework should we use for Java game development?

* **Option 1: LibGDX**
* Option 2: Slick2d
* Option 3: Our own game engine

Choice: Option 1

Justification: LibGDX is a free, open-source and is very widely used as a cross-platform Java game development framework. Since it has an active community there are many tutorials online. In addition, libGDX comes with a lot of possible parts that we can use in our game such as audio, input handling, graphics, file i/o and storage. Creating our own game engine would also be a possibility, however due to our lack of knowledge in this area, it will be extremely time consuming and lots of extra work in order to create it.

5. What platform should we make this for?

* Option 1: Mobile
* Option 2: Web Browser
* **Option 3: PC**

Choice: Option 3

Justification: For a PC platform, there will just need an application to launch the game. If the web browser was the platform of choice, Javascript would also need to be used which our team does not know. Along with this, our group is most familiar with java, however it is hard to use java to design something to put on the web. A mobile platform would be able to be used for the game but it would require a keyboard as the inputs are recorded with this.

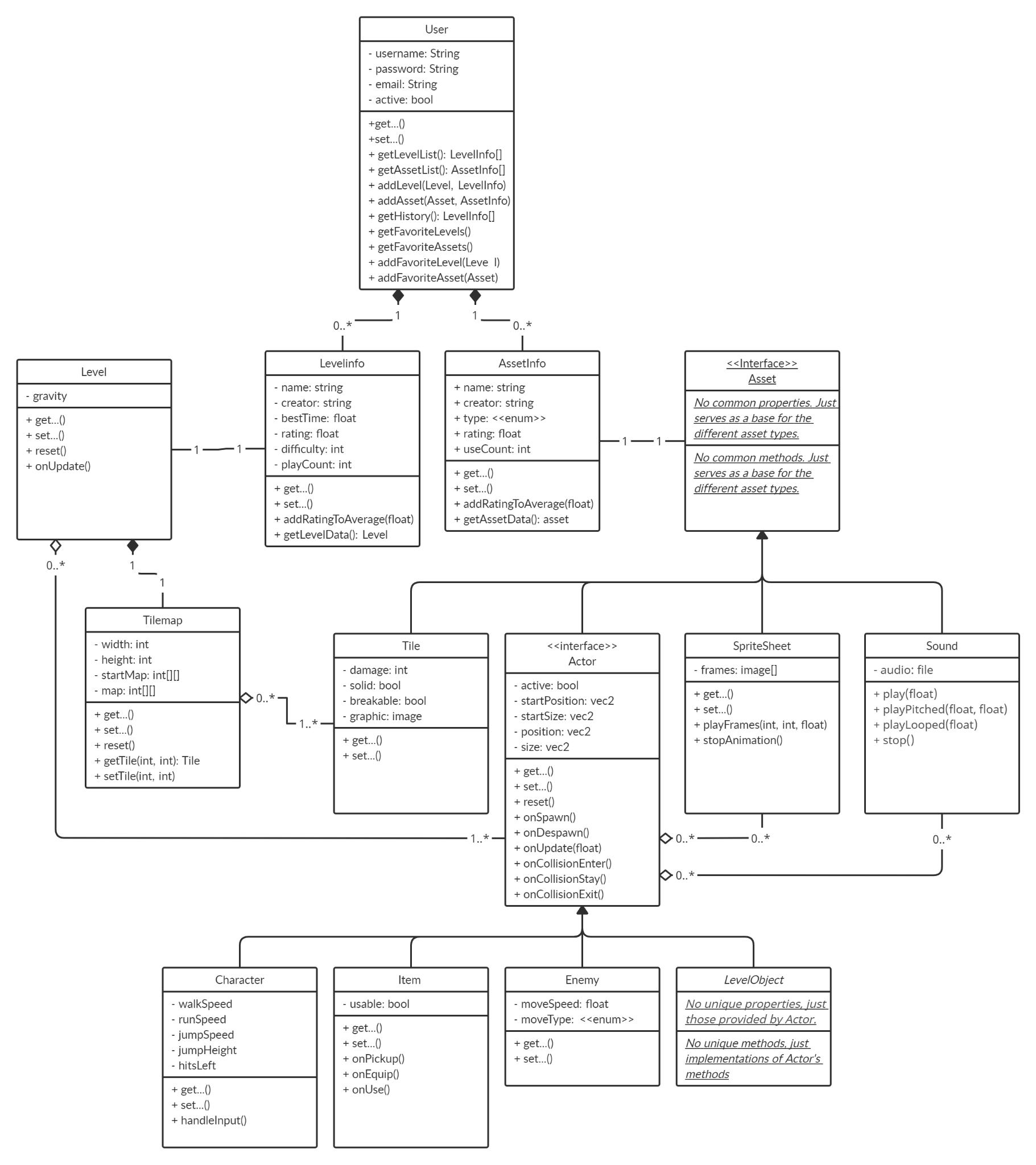
6. How should levels be edited?

* Option 1: Made in Tiled and imported to Level Up
* **Option 2: Build a custom level editor in Level Up**

Choice: Option 2

Justification: Building the level editor into the application is more convenient for the user. They do not need to download external programs in order to use Level Up. A custom level editor would also be tailor-made to the capabilities and limitations of Level Up. This streamlines the user experience and lowers the barrier to entry for people to use our program.

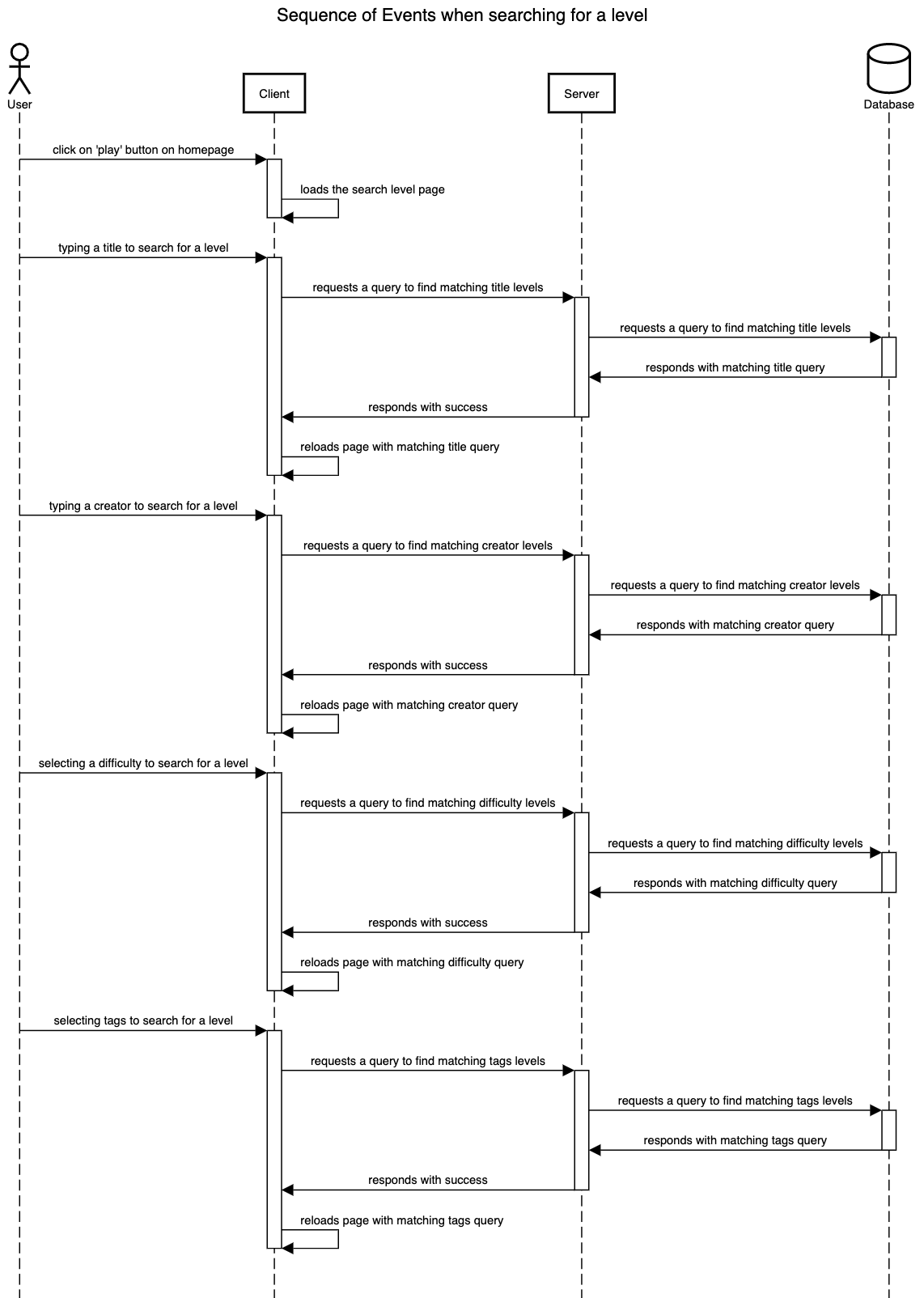
#### Design Details:

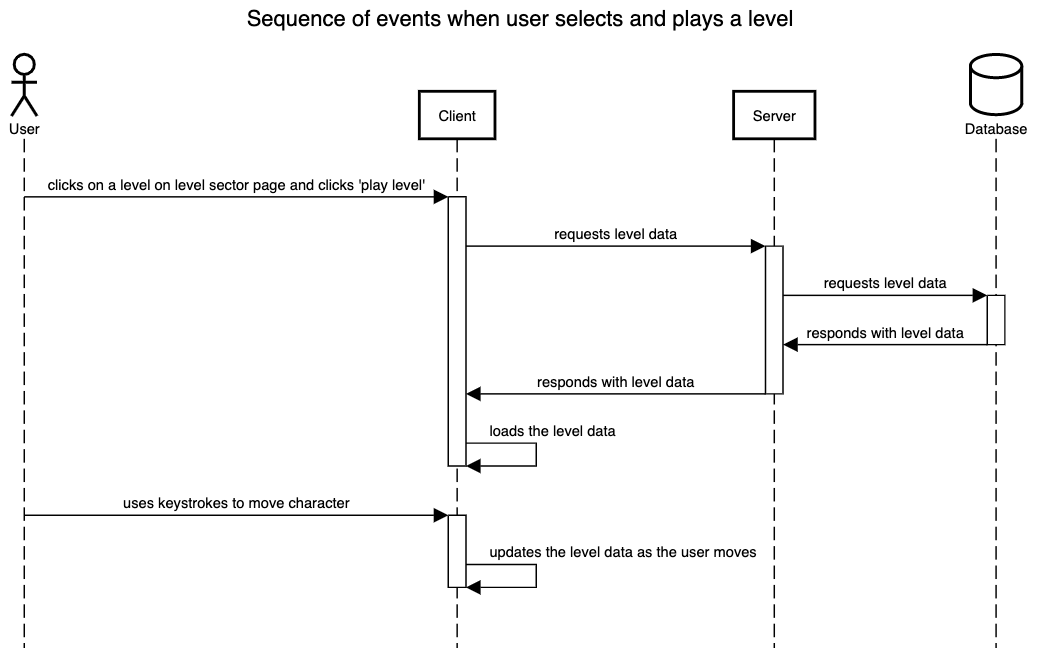
****

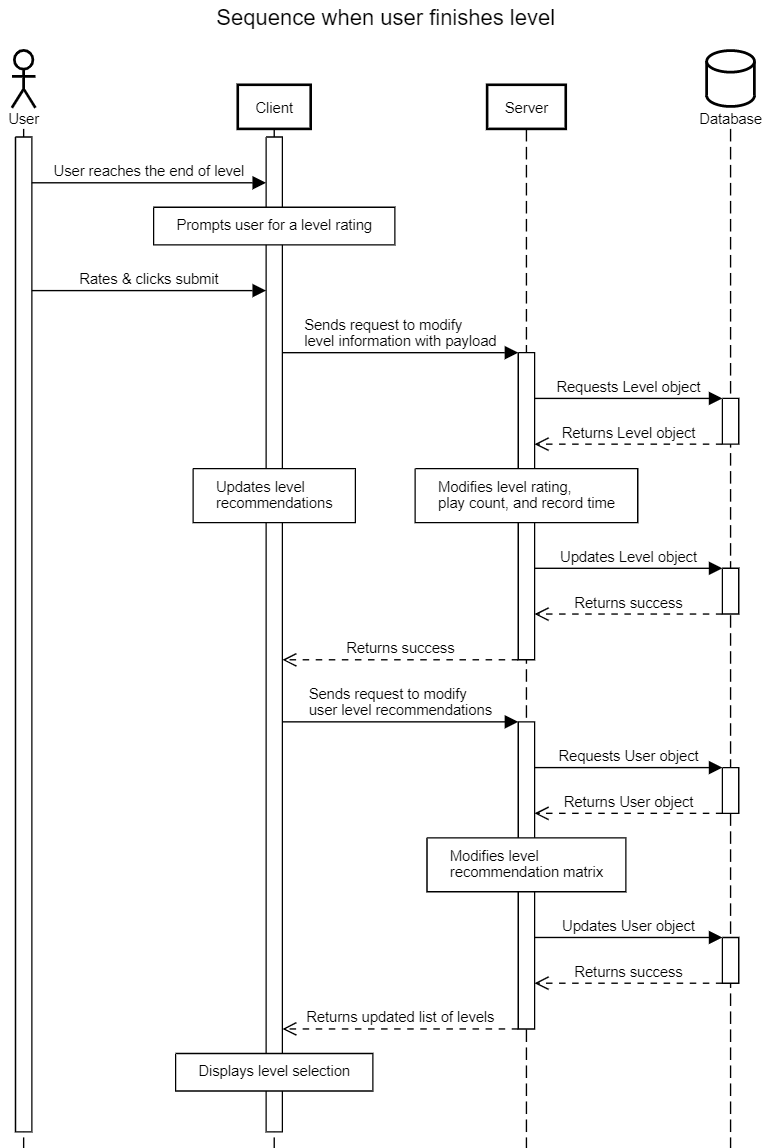
#### Description of Classes and Interaction Between Classes

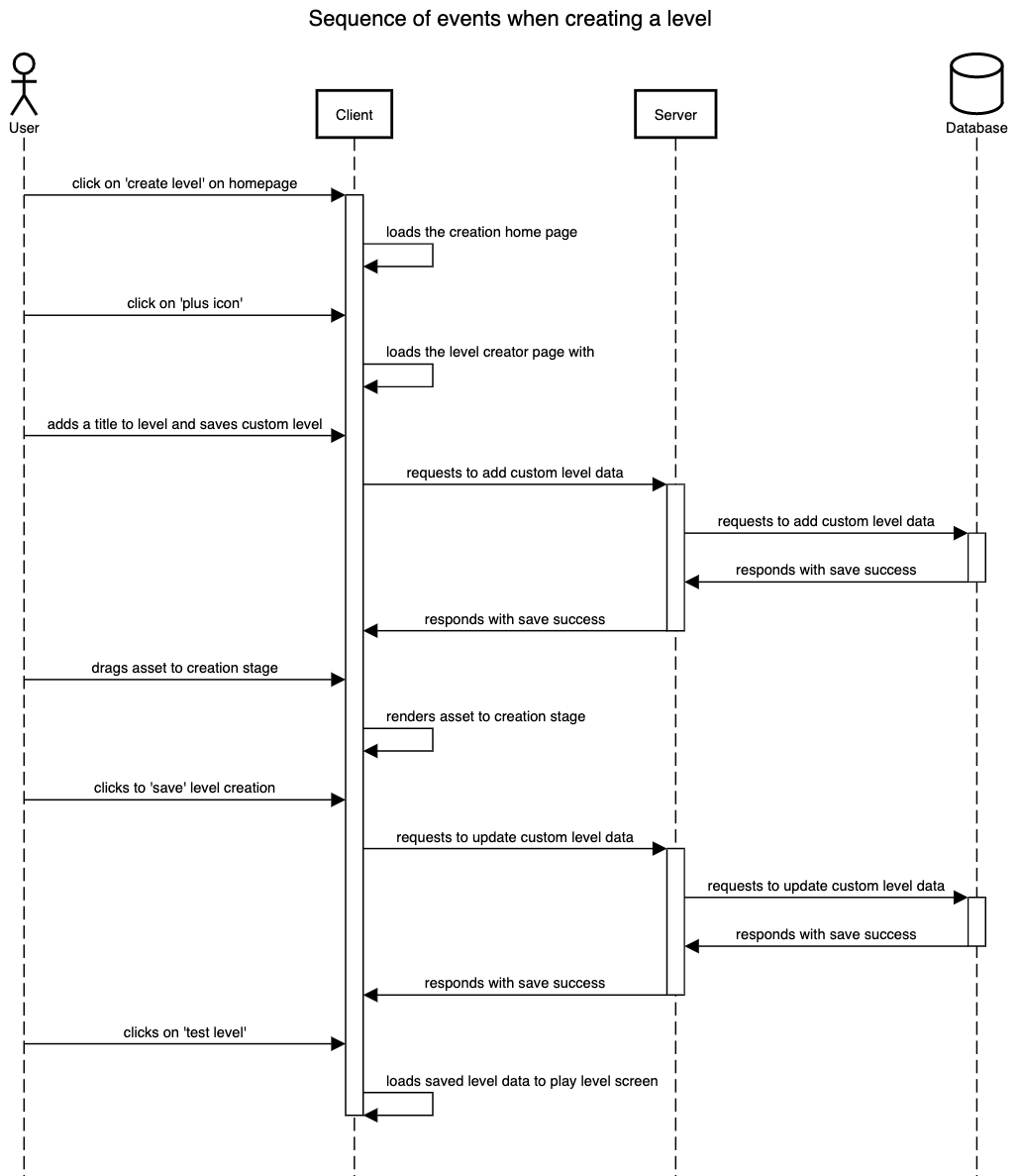
* **User:**
  + Data about users registered with the app.
  + User object is created when someone signs up for New Level.
  + Each user will have a username, password, and email for logging in.
  + Each user will have a list of levels they have created.
  + Each user will have a list of assets they have uploaded.
  + Each user will have a list of favorite levels and assets
* **Level Info:**
  + Contains metadata about one level:
    - The level name
    - The creator of the level
    - The fastest time to complete the level
    - The level’s rating
    - The difficulty of the level
    - The number of plays that the level has
  + Seperate from the actual level data so that only the metadata needs to be loaded when browsing levels to play. The actual level data is loaded when the user selects a level to play.
* **Asset Info:**
  + Contains the metadata for one asset:
    - The asset’s name
    - The creator of the asset
    - The type of the asset
    - The asset’s rating
    - The number of times that the asset has been used
  + Seperate from the actual asset data so that only the metadata needs to be loaded when browsing assets. The actual asset data is loaded separately when added to a level.
* **Level:**
  + A level object is created when someone creates a new level.
  + Each level has exactly one tilemap.
  + Each level has a list of one or more actors that are placed around it.
  + Each level has a gravity parameter that can be set.
* **Asset:**
  + An interface representing an uploaded asset.
  + Asset object is created when someone uploads it.
  + Assets are used to build levels.
  + The asset interface has no properties itself, it just serves as a base to generalize all assets
* **Tilemap:**
  + An asset that represents the background of a level.
  + Contains a list of tiles used in a level (tile sheet).
  + Tiles are placed in a regular grid around the level.
  + A matrix of integers is used to map each cell on the grid to a particular tile. This allows reuse of the tiles around the map to avoid having redundant tile objects.
* **Tile:**
  + A square image used in tilemaps.
  + May have properties such as whether it blocks actors from moving, damages the player, or is breakable.
* **Sprite Sheet:**
  + A collection of images to be used by actors.
  + A range of images, or frames, can be played back at a specific speed to animate actors
* **Sound Effect**
  + A sound to play on command.
  + Can be used by Actors to provide sound effects for the game.
* **Actor**
  + A special asset that defines an object that interacts with the level and other Actors. For example, the player, enemies, items, etc.
  + Utilizes sprite sheets to provide graphics/animation and sound effects.
  + Updates each frame to react to the level and actors around it.
  + Subclasses implement the on..() methods to define their behavior.
* **Character**
  + An actor that describes the attributes for a player
  + Only one should be added to the level.
  + Attributes include the player’s speed, jump height, and number of hits left.
  + Handles input from the keyboard
* **Item:**
  + An actor that the player can pick up and/or use.
  + Can change the behavior of the player when picked up, or add to the number of hits they have left.
* **Enemy:**
  + Actors placed in the level to combat the player.
  + Attributes include a movement speed and an enumeration to select the movement type.
* **LevelObject:**
  + Actors for miscellaneous objects placed around the map, such as moving platforms, springs, etc.

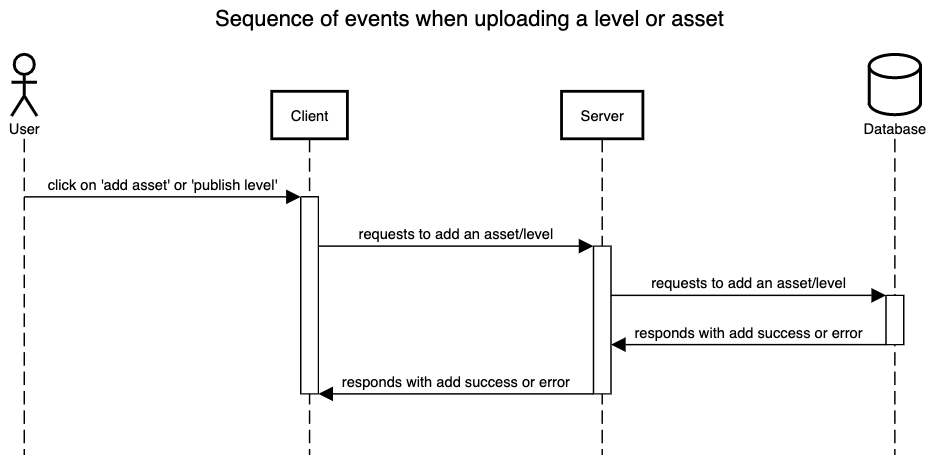
#### Sequence Diagrams





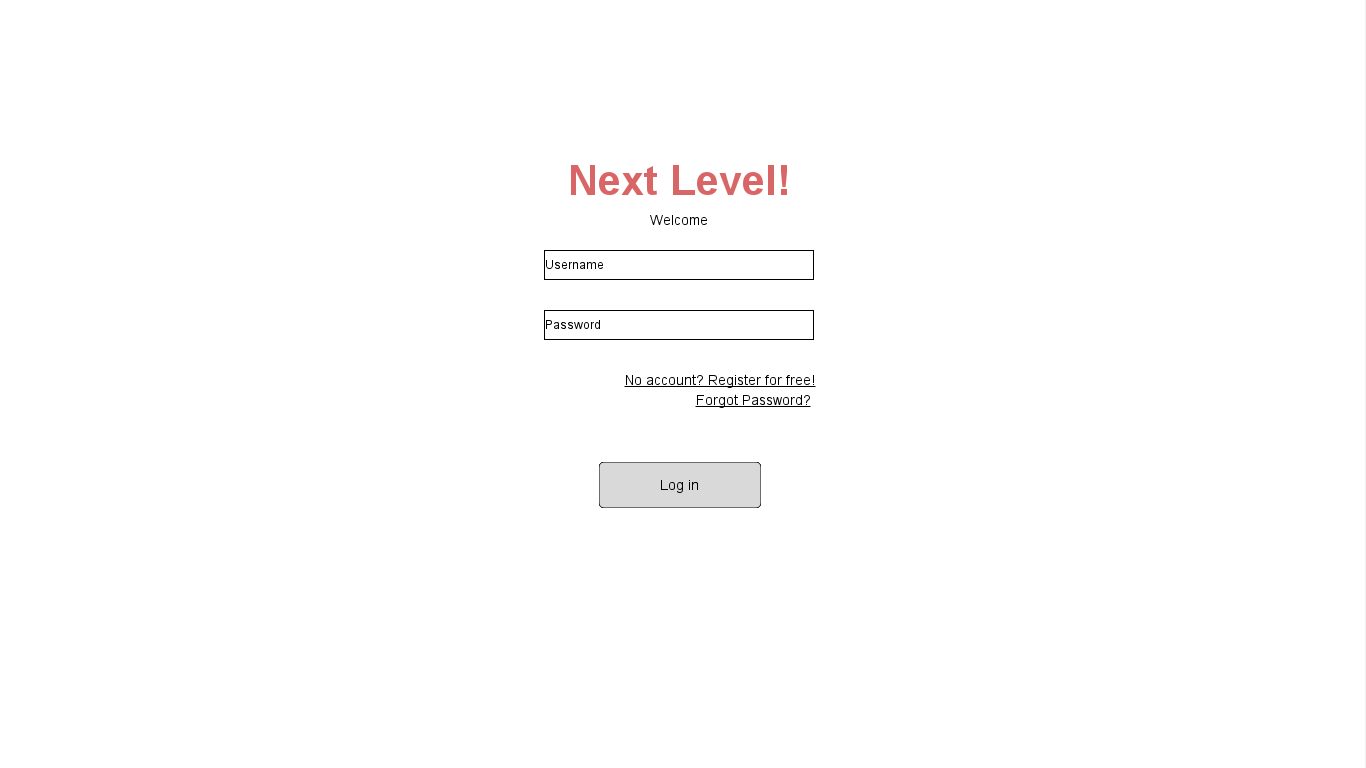


****



#### UI Mockups

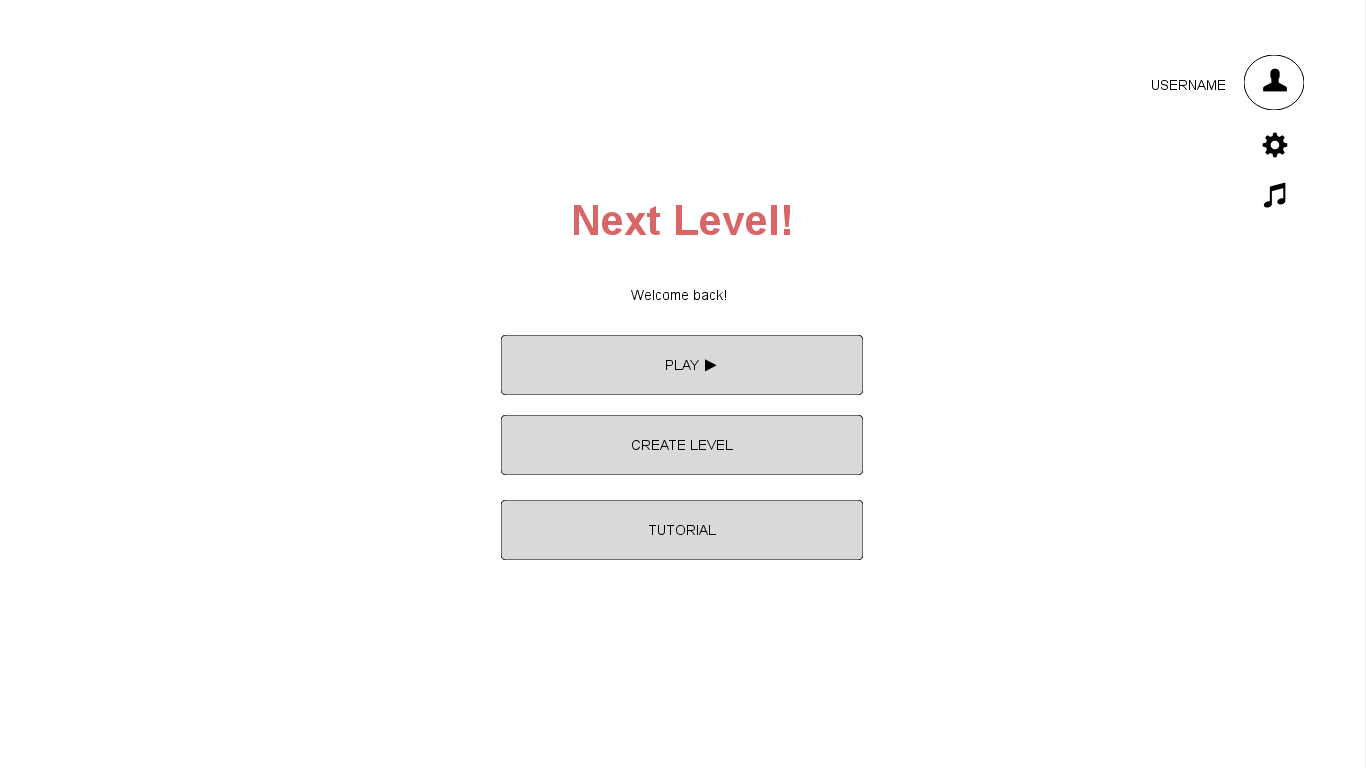
Listed below are a couple UI mockups that we have designed.



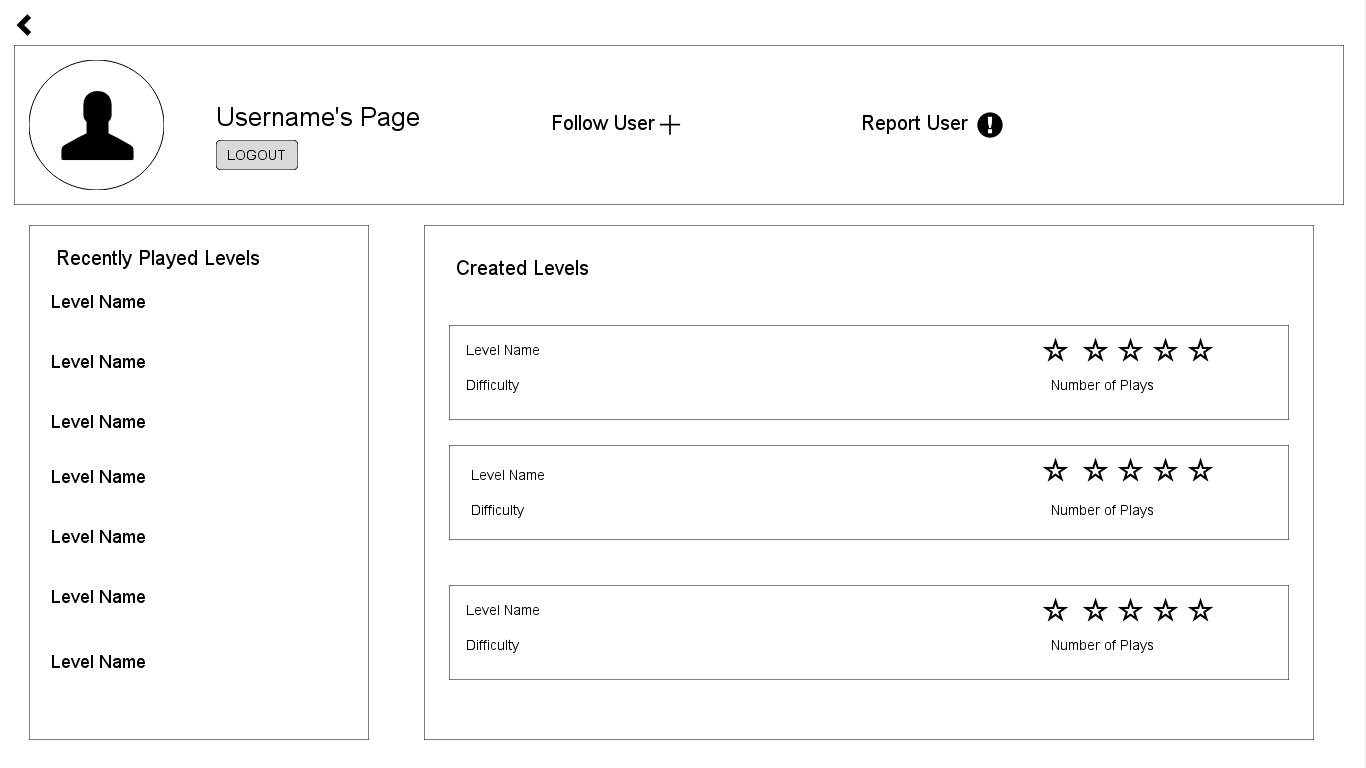
This is the main page of our game. There will be a text box for the user to input their username and password along with a login button. We also have buttons for signing up for a Next Level account or resetting their password if forgotten.



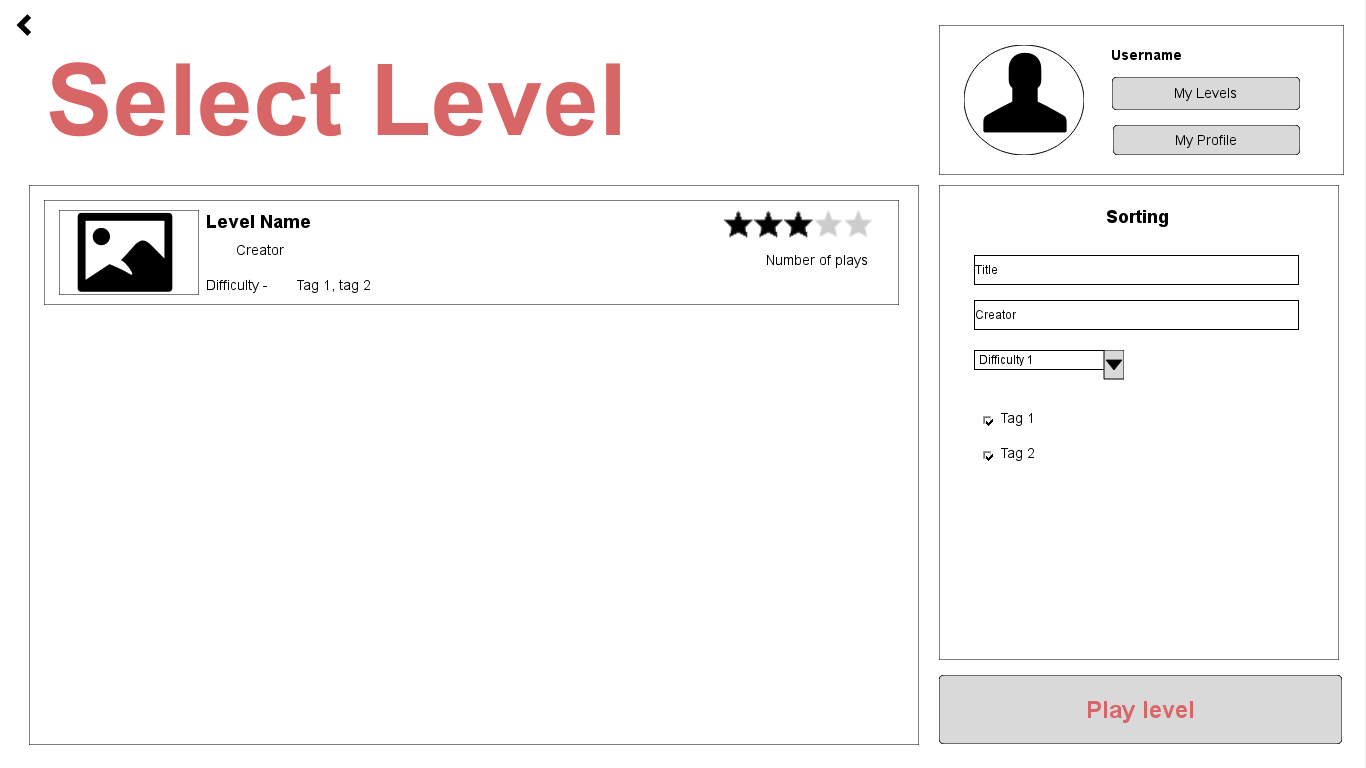
This is the sign up page for signing up for a Next Level account. There will be text boxes for creating a username, entering their email, and creating a password. There will then be a sign up button for confirming the sign up and another button leading back to the main page in case the user accidentally pressed the sign up button on the main page.



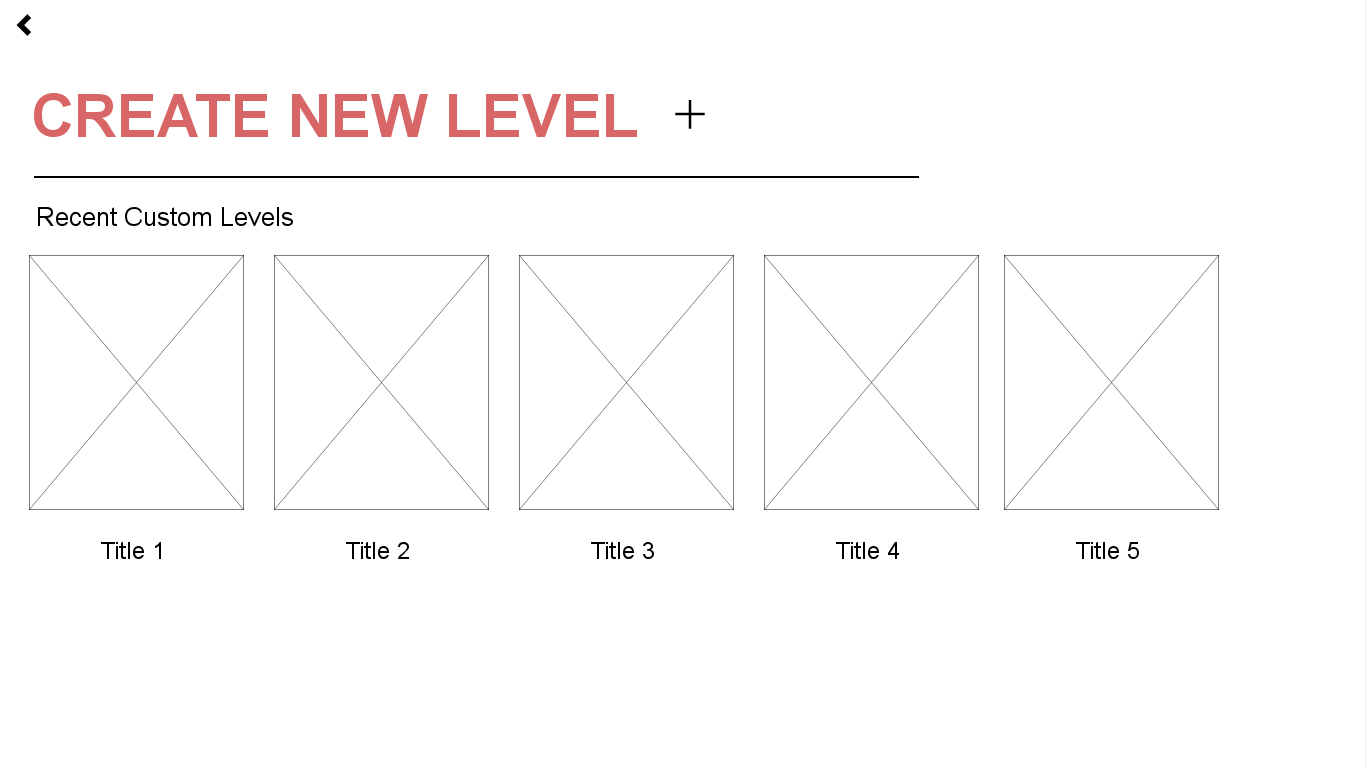
After login/sign up, the user is brought to the home page of Next Level. Here there are options of what they want to do first. On the upper right hand side, clicking on the user icon will prompt users a mini menu: view profile page or logout; settings icon will allow you to change the music volume levels, sound effect volume level, and text size; and the music icon allows users to mute/unmute the music with a click of a button. Play would bring them to the level sector page, create level will send them to the level creation page, and tutorial will show them a brief tutorial and controls of the character.



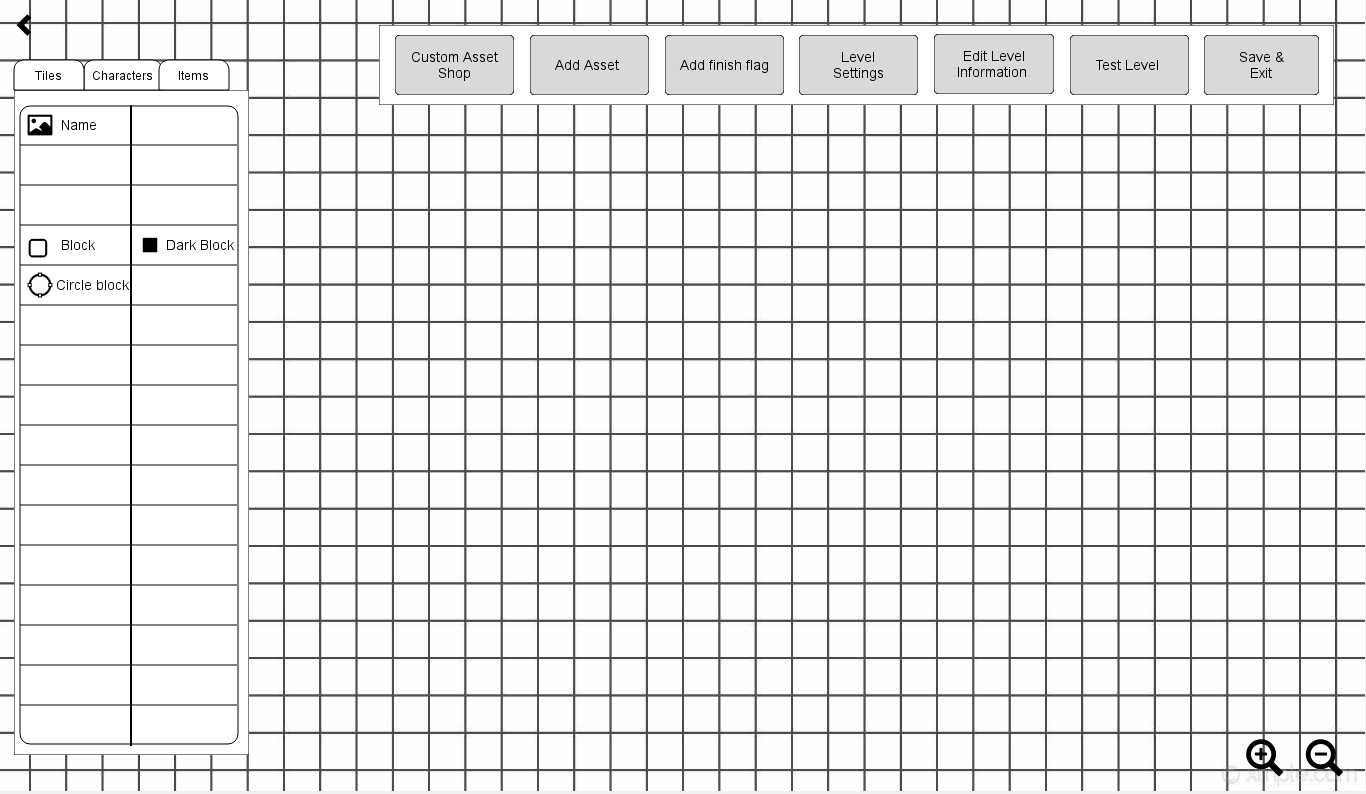
This is the profile page of a user. On the top of the page is the user’s username with their profile picture next to it. Along with that, there is a follow user button for other users to follow that user. There is also a report user button to report users for inappropriate behavior such as an inappropriate level name. The back arrow takes the user back to the home screen.



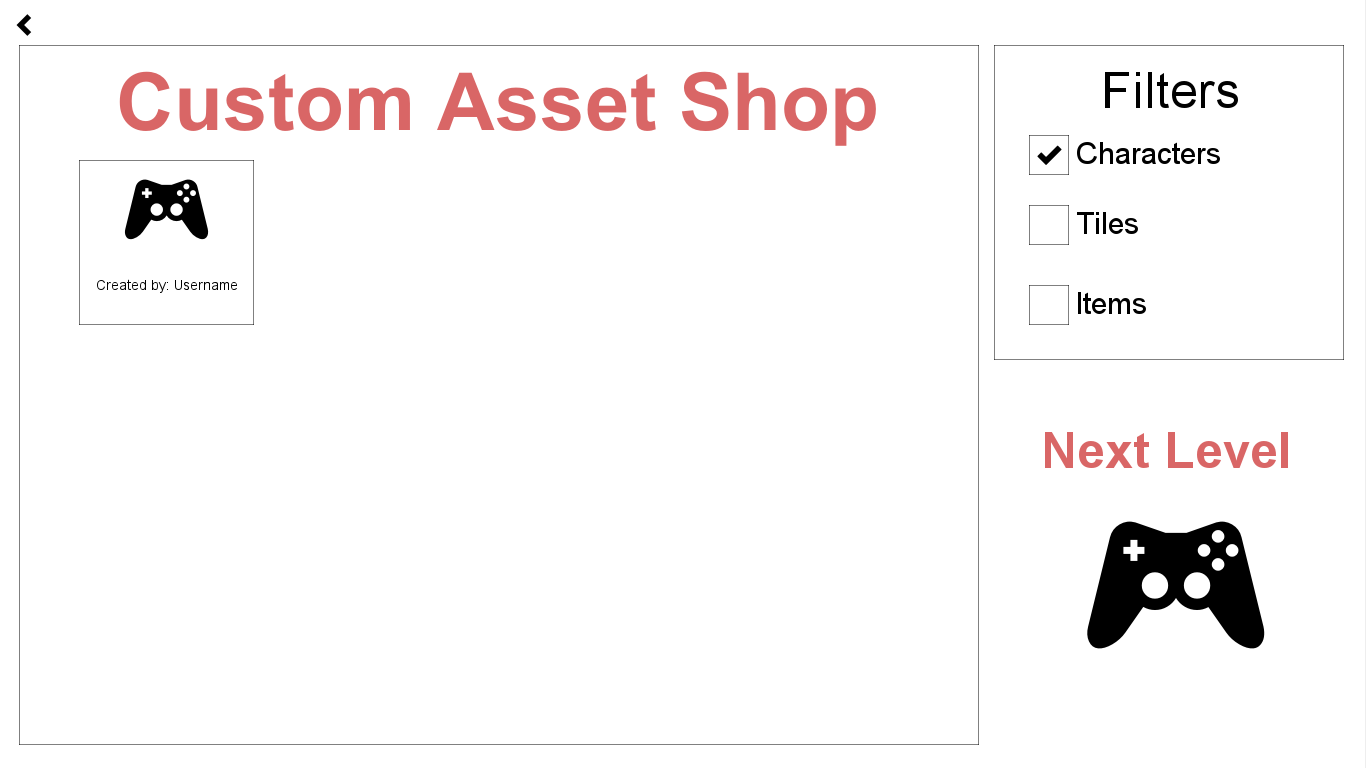
This is the level selection page. On the left is a space for levels with a picture of a part of the level, the level name, creator username, difficulty, rating, number of plays, and tags that describe the level. On the top left shows the user’s profile picture, username, a button leading to levels created by the user, and a button leading to the user’s profile page. Under that is a filter section for filtering the levels you want to find based on difficulty or tags. On the bottom right is a button to start playing the level. The back arrow takes the user back to the home page.



After clicking the create level button on the home page, the client will bring the user to the creation home page. Here the user can decide on whether to work on an existing project or start a new one. The recent custom level projects are shown with a snapshot of the level and the title. Clicking the plus button will create a new project. The back arrow sends the user back to the home page. When clicking a recent custom level, will bring up a mini menu that allows the user to edit the level, publish, or delete.



This is the level creator page. On the left is where the user can select assets they want to place in their level. There will be a tiles section, a characters section, and an items section. On the top right is a section for actions for the user to choose from. They can do things such as set tags or choose a difficulty for their level. They can also access the custom asset shop as well as adding a new asset. They can also save and exit from the level editor to continue editing later on. On the bottom right is a zoom in and zoom out button to make it easier for the user to add tiny details to their level. The back arrow sends the user back to the creation home page.



This is the custom asset page where users can choose from custom assets that have been uploaded by the playerbase. On the left is the custom asset shop where it lists the assets and the username of the creator. On the right is a filters section to filter by what type of assets the user is looking for. The back arrow brings the user back to the level creator page. The next level icon and logo sends the user back to the home page, however it will not save the user’s custom level.