EA Login

Email: aw. Password: secret

Wube Software Login

Email: wube. Password: secret

Valve Software Login

Email: gn. Password: secret

Microsoft Login:

Email: bg. Password: secret

**Advanced JavaScript Report MERN stack CA2. Matthew Pantaleon**

# Introduction

The functionality of the MERN stack application is to emulate a steam game library. Customers are able to read about and add games to their library. Companies are able to manage their games through employee accounts on the application, whilst the employee can have their personal library.

# User Stories

The first three user stories can be applied to both regular users and company users. As the experiences for both are the same in this regard.

1. As a User I want to be able to add games to my Library.
2. As a User I want to see all details about a game.
3. As a User I want to be able to filter the store and games based on search and Genre.

These next few only apply to company users as they need to be able to manage the company games.

1. As a Company Employee I want to add new games to the game store for the company.
2. As a Company Employee I want to edit an existing game in the game store.
3. As a Company Employee I want to delete an existing game from the game store.

# Wireframes

Login Wireframe:

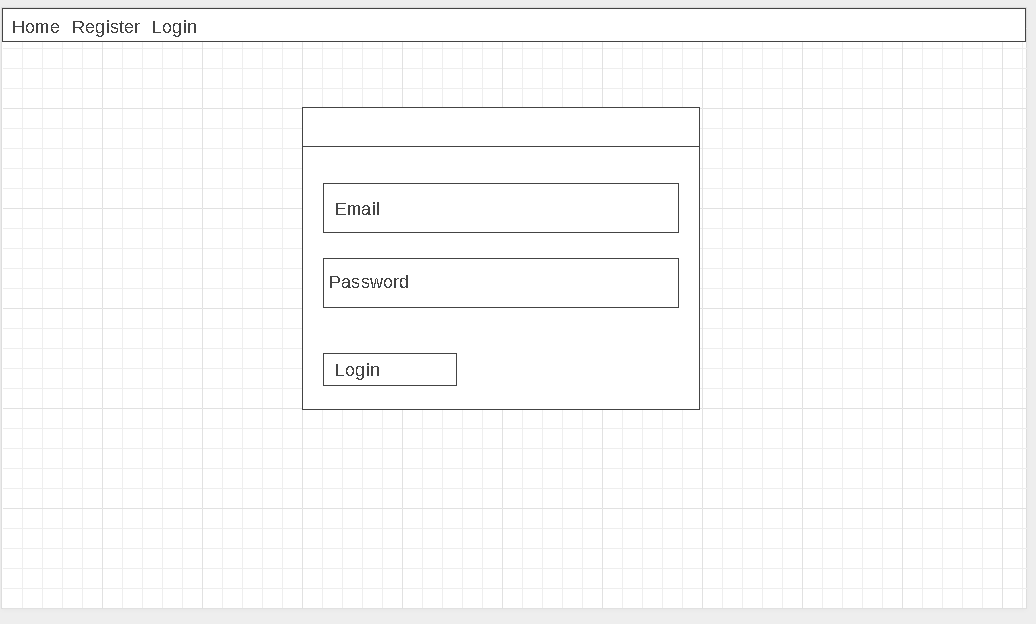


Figure 1 Login Page

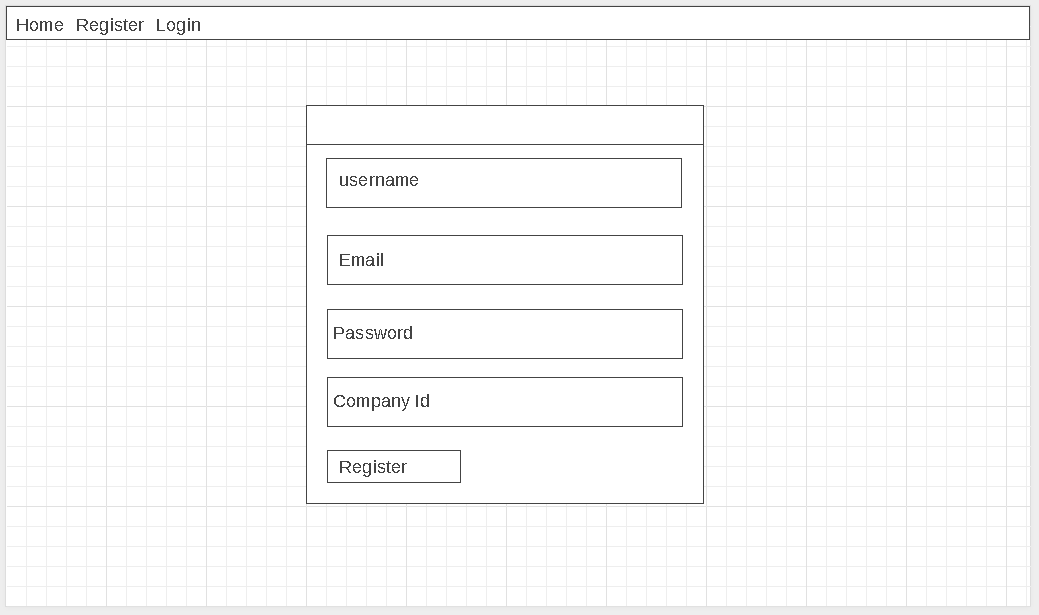


Figure 2 Register Page



Figure 3 Store Page

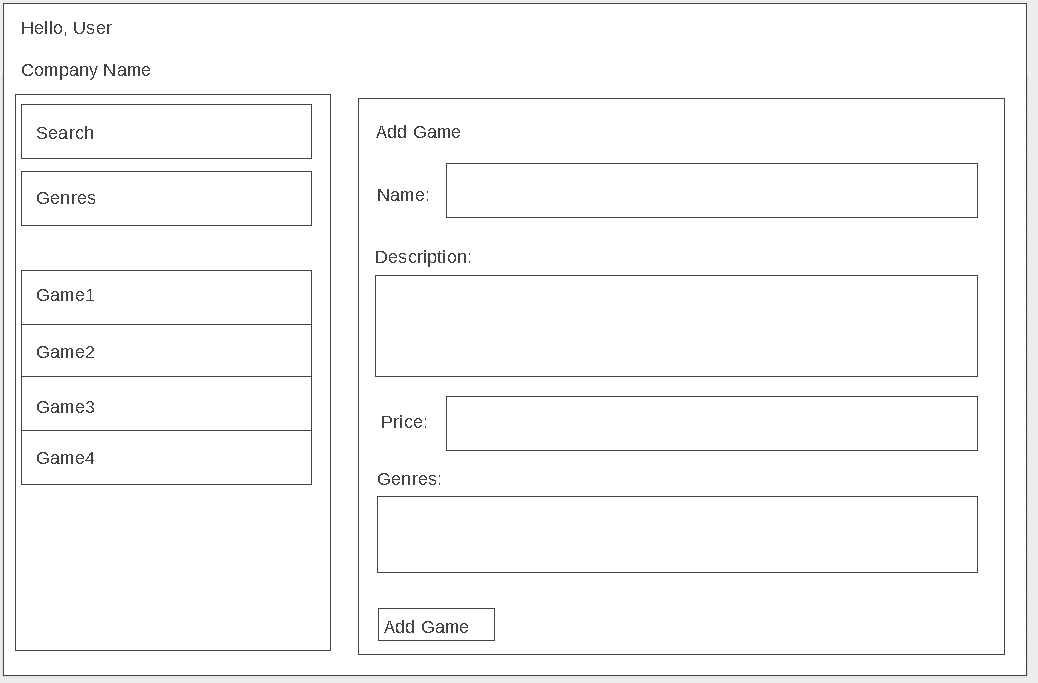


Figure 4 Add Game Panel

Edit Game would be a slightly different variant with the information already filled out.

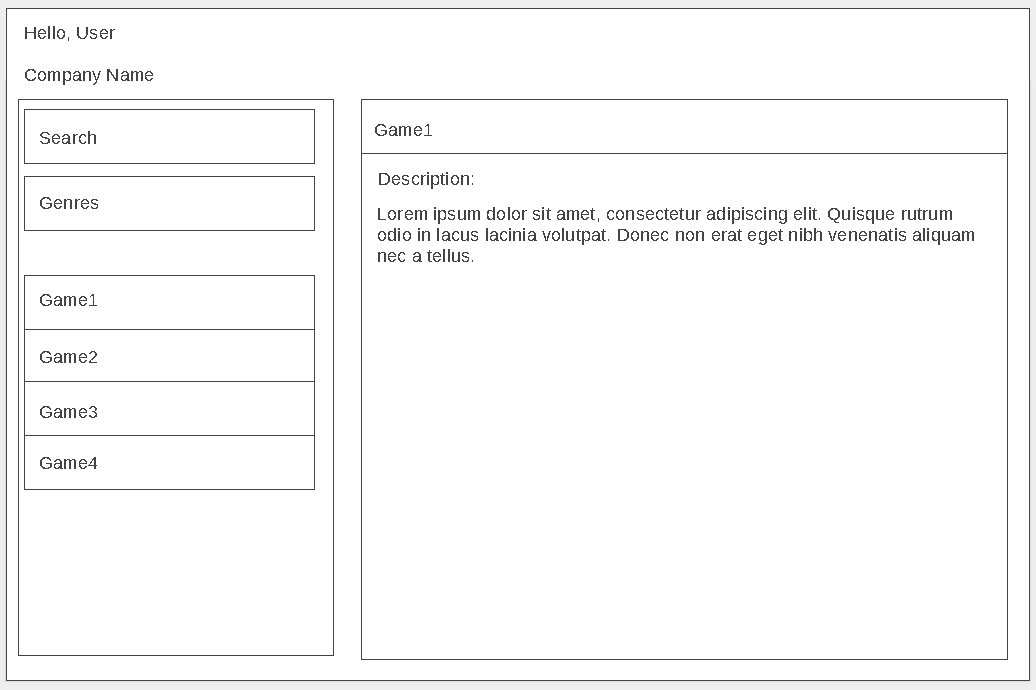


Figure 5 View Game Details

# API Endpoints

This lists all the routes of the application uses sorted by request type.

## Get Routes

**/seed** :

This route clears all games from all libraries but does not delete the library itself. Completely ignores the users collection. It resets the games table to a predefined 10 games, then resets the company table and their respective game Ids, not guaranteed however and resets the genres collection with genres. Finally populates each game’s genres and genre’s games arrays.

Returns *{“message”};*

**/games** :

Returns all games with their genres unpopulated.

Returns: *{games[]};*

**/genres :**

Returns all genres with their games unpopulated.

Returns: *{genres[]};*

**/library/:id** :

This returns the library object of the user logged in with the games array populated.

Returns: *{library{}};*

**/company/:id** :

This returns partial company information (array of game ids and the name) rather than the full object to emulate company security.

Returns: *{gameIds[], “name”};*

## Post Routes

**/games** :

Adds a new game to the games collection and updates the current company with a new game.

Returns *{success: true, newGame{}, companyGames[]};*

**/library/:id** :

Adds a game id to the library of the user logged in and returns back the library object.

Returns: *{library{}};*

**/login** :

When a user logs in successfully it returns their username, company id, token and library id. Else some form validation object.

Returns: *{“token”, “username”, “company\_id”, “library\_id”};*

**/register** :

This is to register a new user to the application. All fields except for company id are required, if there is a company id it will be handled when the user logs in afterwards. This also creates a new library for the new user. Once the registration is complete the page redirects to login.

Returns *{“message”};*

**/check** :

This route is primarily used as a middleware check to see whether or not the user trying to access certain sections of the application is logged in or not. This is implemented in the Login, Register and Main components. If the user is validated to be logged the Login and Register components are restricted, if not then the Main component is Restricted.

Returns: *{boolean};*

## Put Routes

**/games/:id** :

Updates game based on passed id and returns the edited game.

Returns: *{editGame{}};*

## Delete Routes

**/games/:id** :

Finds all libraries that have this game and removes the game id from the library games array and for the user currently logged in, then deletes the game itself. It then returns all games and the user library with games populated.

Returns: *{games[], userLibrary{}};*

**/library/:id** :

This does not delete the library itself but rather removes a game from the games array and returns the library object with games populated.

Returns: *{library{}};*

# Application Functionality

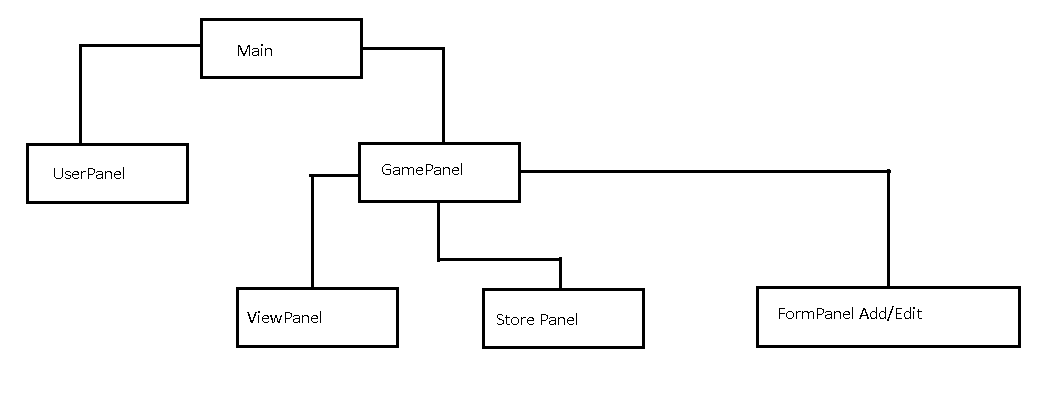


Figure 6 Core Component Structure

In figure 6 shows the main component structure of the application omitting the login and register components as they are independent components. All data necessary data in loaded here through the ApiLoader. The main component checks if the user accessing the page is authorized using the authUser function. This function is is created to be versatile in that it returns a Boolean and you can spcify your own callbacks. Which the main componentDidMount function utilizes.

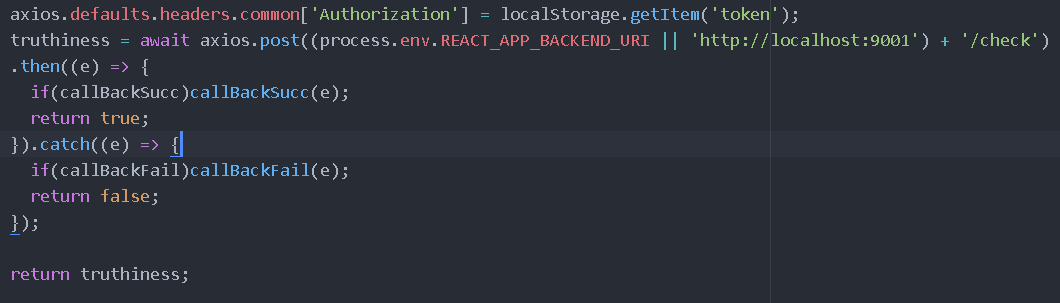


Figure 7 authUser function

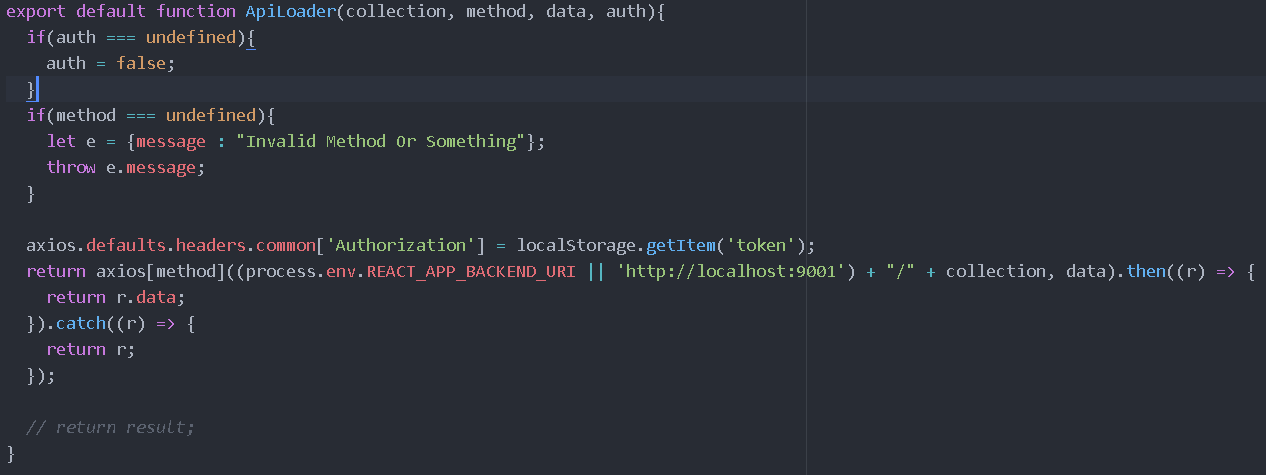


Figure 8 ApiLoader

All requests to the backend server use the ApiLoader function instead of individual axios requests to reduce code replication. Its arguments are the important part of the url, method type string, a data object and an auth Boolean but the auth is not used. However since the function returns a promise object from the original axios call, all successful and catch requests will return in the then callback of the ApiLoader.

The data retrieved from the ApiLoader calls are then passed on to the child components along with various functions to update the main state when an event happens on the appropriate components.

# Reflections

Learned on how express is implemented to create a server with custom routes and authentication but not quite fully understand on how passport itself works.

Learned on how a mongo database structure and terminology, also on how mongoose is able to encapsulate common aspects of database interaction from connection, schemas and models. Mongoose methods are asynchronous. From that expanded my knowledge on how promise objects and asynchronous functions operate. Mongoose methods can be used like promises or supplied with a callback instead.

For the react part reusing the same component for adding and editing a game via props is very useful. I learned that you can chain array filter to array map.

For hosting, it was not very difficult but very easy to mess up. The key things to note were the env variables and buildpacks.

For the mongodb collections not all relational directions are fulfilled. i.e. you can find a game with a company but you cannot find a company using just a game, same thing for libraries. You cannot find all libraries containing the game using only the game.

Github Link: <https://github.com/MatthewPantaleon/MERN-CA2>

Heroku Link (make sure it is http not https): <http://steam-emulator.herokuapp.com>