

INVENTORY

DESCRIPTION:

A web application designed to store custom created “item” objects for its users to use within their story-based games, primarily TTRPG’s (table-top role-playing-games). Administrators can create these items and assign them to users who play within their games, allowing for a dynamic management of custom items between players.

CONTEXT:

When playing TTRPG’s, it’s common for people to create custom items and abilities to be used within the game that don’t exist within the base rules. After adding a few, it can get cumbersome to keep track of all the items, who has them and their descriptions.

This product will give game masters an environment where they can store their repository of custom created content and then share that with their players, giving an easy system to manage this content besides manually sharing it.

FEATURES:

CUSTOM ITEMS: Administrators will be able to create and manage custom items as with the functionality below:

Create - Administrators can create new items which are stored in their vault,

Read - Administrators and assigned users can view information on items,

Update - Administrators can edit existing items details,

Delete - Administrators can delete items from their vault,

Share - Administrators can assign and unassign items to users to control access on viewing the item,

ACCOUNTS: Administrators will be able to create and manage accounts for themselves and other users with the functionality below:

Create - User can create a new player account ,

Read - Administrators and users can log into their account,

Read - Administrators can see a list of all accounts,

Update - Administrators can edit the items assigned to a user,

OTHER SITE FUNCTIONALITY:

When viewing a list of items, the list can be filtered by an item’s tags or by name (including partial searches).

When viewing a list of items, they are shown in alphabetical order for ease of navigation.

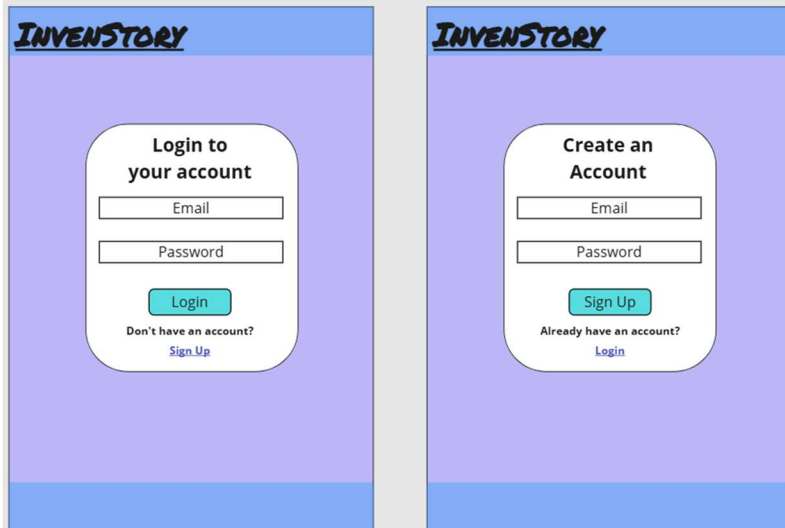
USER INTERFACE:

The Site will start on the login page and the MVP will only have 1 page it goes to when logged in. This page differs depending on the role of the user, with administrators being able to see all items, as well as manipulate them.

Items will be shown in a table, with each cell showing the complete information about that item. Administrators will also see the users who have access to that item, and the buttons for altering the item data.

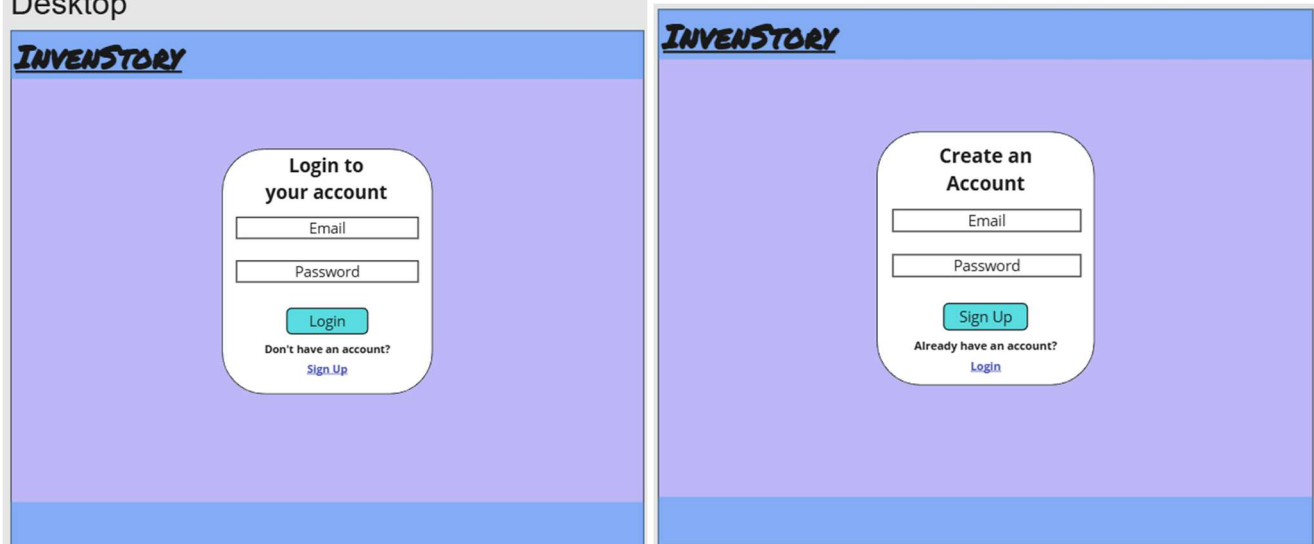
LOGIN/SIGN-UP:

Mobile



The mobile interface consists of two side-by-side screens. Both screens have a blue header with the text "INVENSTORY" in a stylized, bold font. The left screen is titled "Login to your account" and features a white rounded rectangle containing two input fields for "Email" and "Password", a teal "Login" button, and a link "Don't have an account? Sign Up" below it. The right screen is titled "Create an Account" and features a white rounded rectangle containing two input fields for "Email" and "Password", a teal "Sign Up" button, and a link "Already have an account? Login" below it. Both screens have a light purple background and a blue footer.

Desktop



The desktop interface consists of two side-by-side screens. Both screens have a blue header with the text "INVENSTORY" in a stylized, bold font. The left screen is titled "Login to your account" and features a white rounded rectangle containing two input fields for "Email" and "Password", a teal "Login" button, and a link "Don't have an account? Sign Up" below it. The right screen is titled "Create an Account" and features a white rounded rectangle containing two input fields for "Email" and "Password", a teal "Sign Up" button, and a link "Already have an account? Login" below it. Both screens have a light purple background and a blue footer.

VIEW ITEMS (PLAYER ROLE):

Mobile

FILTER

Expands On Click

INVENTORY

LOGOUT

FILTER

NAME:

TAG:

SEARCH

ITEM.NAME

TAGS: ITEM.TAG[0], ITEM.TAG[1]... ITEM.TAG[N]

ITEM.DESCRPTION

ITEM.NAME

TAGS: ITEM.TAG[0], ITEM.TAG[1]... ITEM.TAG[N]

ITEM.DESCRPTION

⋮

⋮

⋮

ITEM.NAME

TAGS: ITEM.TAG[0], ITEM.TAG[1]... ITEM.TAG[N]

ITEM.DESCRPTION

Desktop

FILTER

Expands On Click

INVENTORY

LOGOUT

FILTER

NAME:

TAG:

SEARCH

ITEM.NAME

TAGS: ITEM.TAG[0], ITEM.TAG[1]... ITEM.TAG[N]

ITEM.DESCRPTION

ITEM.NAME

TAGS: ITEM.TAG[0], ITEM.TAG[1]... ITEM.TAG[N]

ITEM.DESCRPTION

⋮

⋮

⋮

ITEM.NAME

TAGS: ITEM.TAG[0], ITEM.TAG[1]... ITEM.TAG[N]

ITEM.DESCRPTION

ITEM.NAME

TAGS: ITEM.TAG[0], ITEM.TAG[1]... ITEM.TAG[N]

ITEM.DESCRPTION

ITEM.NAME

TAGS: ITEM.TAG[0], ITEM.TAG[1]... ITEM.TAG[N]

ITEM.DESCRPTION

⋮

⋮

⋮

ITEM.NAME

TAGS: ITEM.TAG[0], ITEM.TAG[1]... ITEM.TAG[N]

ITEM.DESCRPTION

VIEW ITEMS (ADMINISTRATOR ROLE):

Mobile

INVENTORY **LOGOUT**

FILTER
NAME:
TAG:
USER: **SEARCH**

ITEM.NAME
TAGS: ITEM.TAG[1], ITEM.TAG[1+1]...
ITEM.TAG[N]

USERS:

- USER[1].NAME
- USER[1+1].NAME
- ...
- USER[N-1].NAME
- USER[N].NAME

ASSIGN

ITEM.DESCRPTION

EDIT ITEM **DELETE ITEM**

ASSIGN USERS:
NAME: **ADD**
CURRENT USERS:
USERS:

- USER[1].NAME
- USER[1+1].NAME
- ...
- USER[N-1].NAME
- USER[N].NAME

CLOSE

EDIT ITEM:
NAME:
TAGS:
DESCRIPTION: **UPDATE**

Desktop

INVENTORY **LOGOUT**

FILTER
NAME: TAG: USER: **SEARCH**

ITEM.NAME
TAGS: ITEM.TAG[1], ITEM.TAG[1+1]...
ITEM.TAG[N]

USERS:

- USER[1].NAME
- USER[1+1].NAME
- ...
- USER[N-1].NAME
- USER[N].NAME

ASSIGN

ITEM.DESCRPTION

EDIT ITEM **DELETE ITEM**

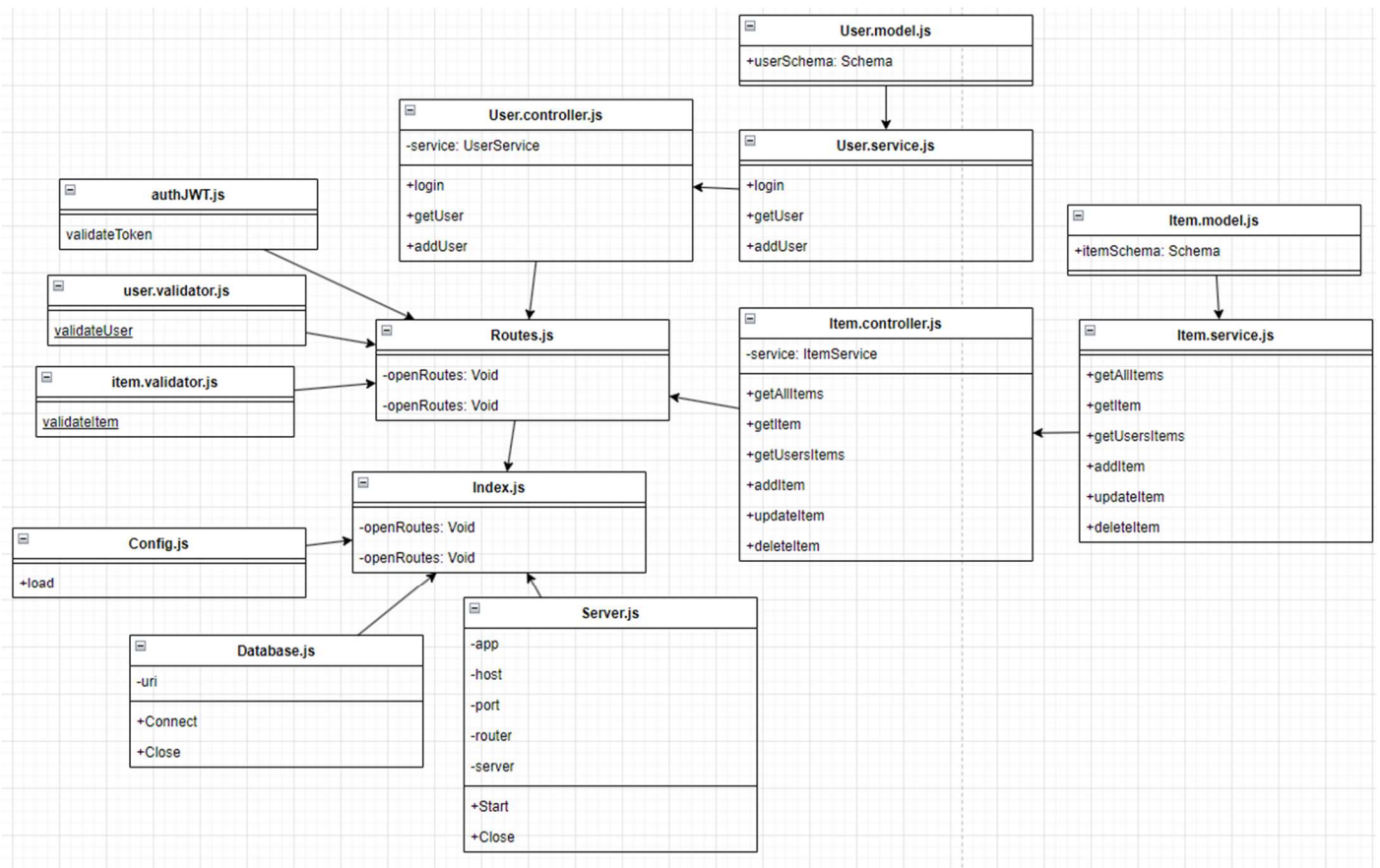
ARCHITECTURE:

The project will consist of 3 main components: the database for which I'll be using MongoDB, the back-end API and the front-end React application.

BACK-END API:

The API shall connect to the database and React application, handling requests for both the item and account data. It shall follow a MSC (model-service-controller) structure and include appropriate validation for incoming requests.

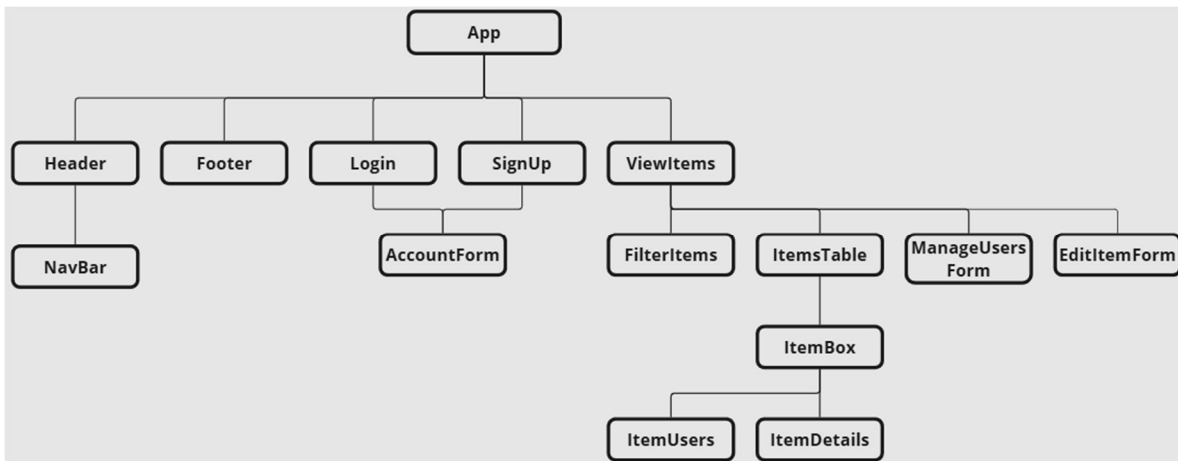
Class Diagram (simplified due to Javascript “duck-typing”):



FRONT-END REACT:

The page shall be developed as a SPA (single-page-application) in react while incorporating Bootstrap elements for styling.

Component Hierarchy:



JSON STRUCTURES:

Item: {

```

  _id: MondoDB id,
  name: String,
  description: String,
  tagList: [{
    tag: String
  }]
}
```

Account: {

```

  _id: MondoDB id,
  userName: String,
  email: String,
  password: String,
  role: String,
  assignedItems: [{
    itemId: String
  }]
}
```

RESTFUL ROUTING:

CUSTOM ITEMS:

Request: GET “/item/allItems”

Header: JWT access-token

Response: 200 {allItems[{{item}}]}

Request: GET “/item/:id”

Header: JWT access-token

Response: 200 { item }

Request: GET “/item/userItems/:userid”<- Gets item list from user, then returns those items

Header: JWT access-token

Payload: { account }

Response: 200 { allItems[{{item}}] }

Request: POST “/item”

Header: JWT access-token

Payload: { item }

Response: 201 { item }

Request: PUT “/item/:id”

Header: JWT access-token

Payload: { item } <- Send & return updated item details

Response: 204 { item }

Request: DELETE “/item/:id”

Header: JWT access-token

Response: 200 { item }

ACCOUNTS:

Request: GET “/user/:userid”

Header: JWT access-token

Response: 200 { account }

Request: POST “/auth/login”

Payload: { account }

Response: 200 { account, access-token: String }

Request: POST “/auth/createUser”

Header: JWT access-token

Payload: { account }

Response: 201 { account }

Request: PUT “/user/update”

Header: JWT access-token

Payload: { account } <- Send & return updated account details

Response: 204 { account }

TECHNOLOGIES:

The product will be developed in Javascript, using React to structure the HTML for the front-end visual components.

BACK-END API:

For development, the following dependencies will be used:

- Bcrypt,
- Dotenv,
- JSON Webtoken
- Mongoose

For testing, the following dependencies will be user:

- Chai & Chai-HTTP,
- Express & Express-Validator,
- Mocha
- Sinon
- Supertest

FRONT-END REACT:

For development, the following dependencies will be used:

- Bootstrap,
- JSON Loader,
- React & React-Dom & React-Router-Dom,
- PopperJS,

For testing, the following dependencies will be user:

- Axios,
- ESLint,
- Testing-Library,
- Vite,
- Vitest,

DEPLOYMENT:

The database shall be stored as a MongoDB database and deployed using MongoDB's ATLAS. A valid connection key will be needed to connect to the database.

GitHub pages will be used to deploy the front-end React application.

Noderender will be used to host the API, which can be linked directly to my Git page for deployment.

ADDITIONAL FEATURES: (BEYOND MVP)

If time provided, I've listed below a set of additional features ranked in order of priority that are desired in the product but are not essential for the MVP.

- Administrator can create new administrator accounts and edit an accounts emails, passwords and usernames.

Create - Administrators can create both new Administrator and User accounts ,

Update - Administrators can edit the account details of user's,

Delete - Administrators can delete a user's account,

Routes for Feature:

Request: POST "/auth/createAdmin"

Header: JWT access-token

Payload: { account }

Response: 201 { account }

Request: DELETE "/user/delete"

Header: JWT access-token

Payload: { account }

Response: 200 { account }

- Administrators can see a view of all users and from there, view an individual users items assigned to them
Read - Administrators can view information on all items assigned to any users,
- Administrators are able to create "games" which users can be assigned to. This also includes the ability to make items public within those games.
- Ability for a user to create an account without an administrator doing it for them. Administrators can then assign new users to a game.
- A third tier or priority, "game owners". These users can be given this priority by an administrator and they're able to create a new game, assign players to said game and assign items within their games.