CSCC01

Documentation

Team: Characters

Eric Zhou
Ayazhan Bauyrzhankyzy
Maria Gotcheva
Sung Ha Hwang
Jacob Matias

Table of Contents

Backend

/backend/models

Contains the model schemas of the data that is stored in the mongoose database

Story.js
This file stores a model schema for user-submitted stories. This model schema includes the following fields:

- storyTitle: String, required
    - This field stores the title of the story.

- storyText: String, required
    - This field stores the entire story text.

- storyType: String, required
    - This field stores the type of submitted story.
    - If the story is submitted to become a Character candidate, this field takes the value ‘characterCandidate’.
    - If the story is submitted to receive custom apparel, this field takes the value ‘customApparel’.

- storyDate: Date
    - This field stores the date that the story was submitted.
    - This field’s default value is the value returned by Date.now().

- storyStatus: String, required
    - This field stores the current status of the submitted story in the Character candidate evaluation process (includes: ‘new’, ‘interviewing’, ‘validated’)
    - This field’s default value is ‘new’

- author: Schema.Types.ObjectId
    - This field stores a reference to the user id, which is the author id.

- isDeleted: boolean
    - This field stores a boolean value of if the story is deleted or not.

User.js
This file stores a model schema for user accounts. This model schema includes the following fields:

- firstName: String
    - This field stores the first name of the user

- lastName: String
    - This field stores the last name of the user
- email: String, required,
    - This field stores the email of the user account
- password: String, required,
    - This field stores the password of the user account
- type: Number
    - This field stores the user type.
        - 1=default, 2=character, 3=employee

Product.js
This file stores a model schema for products created to represent a character's story. This model schema includes the following fields:

- productImage: String
    - This field stores the name of the image file of the product

- productName: String
    - This field stores the name of the product

- productType: String
    - This field stores the type of apparel of the product

- productPrice: Number
    - This field stores the price of the product

- productDescription: String
    - This field stores the description of the product

- productInventoryAmount: Number
    - This field stores the amount of the product that is in stock

- productStory: Schema.Types.ObjectId
    - This field stores the ObjectId of the story item in the story model that is represented by the product

Order.js
This file stores a model schema for user orders. This model schema includes the following fields:

- transactionDate: Date
    - This field stores the date on which the order was checked out

- purchasedBy: Schema.Types.ObjectId

    - This field stores the ObjectId of the user that checked out the order

- shippingInfo: {firstName: String, lastName: String, address: String}
    - This field stores the first name, last name and address that the order was shipped to

- billingInfo: {firstName: String, lastName: String, address: String, paymentMethod: String}
    - This field stores the first name, last name, and address that the order was billed to, as well as the method of payment

- products: Array containing {pid: Schema.Types.ObjectId, productImage:String, productPrice: Number, productDescription: String, itemCount: Number} objects
    - This field stores an array of JSON objects representing all products in the order. Each array object consists of the product information to be displayed in the order details page and the quantity ordered.
- isFulfilled: Boolean
    - This field stores the boolean for whether or not an order has been fulfilled by the employee.
    - Default value: false

/backend/routes

user.routes.js

POST /users/

- Description: This endpoint is to add a user account into the database
- Body Parameters:
    - email: String
    - password: String
- Body Example
```
{
        "_id": "62a117beead14963dfc16bae",
        "email": "abc@gmail.com",
        "password": "abc123"
}
```
- Expected Responses:
    - 200 OK - For a successful post
    - 400 BAD REQUEST - If the request body is improperly formatted or missing required information
    - 500 INTERNAL SERVER ERROR - If save or add was unsuccessful

GET /users/

- Description: This endpoint is to get users with a specific query
- Example Call:

    - GET http://localhost:4000/users/?email=abc123@gmail.com
- Response Body Example:
```
[{
        “_id”: “629fe6e5cf6530b7ec5487bd”
        "email": "abc@gmail.com",
        "password": "abc123"
},
{
        "_id": "62a117beead14963dfc16bae",
        "email": "abc1@gmail.com",
        "password": "abc1234"
},...]
```
- Expected Responses:
    - 200 OK - For a successful get
    - 500 INTERNAL SERVER ERROR - If get was unsuccessful

GET /users/:id
- Description: This endpoint is to get a user account with the specific id
- Example Request:
    - GET http://localhost:4000/users/62a117beead14963dfc16bae
- Response Body Example:
```
{
        "_id": "62a117beead14963dfc16bae",
        "email": "abc@gmail.com",
        "password": "abc123"
}
```
- Expected Responses:
    - 200 OK - For a successful get
    - 404 NOT FOUND - If there is no user in the database with that id
    - 500 INTERNAL SERVER ERROR - If get was unsuccessful

PUT /users/:id
- Description: This endpoint is to update a user account into the database with the specific id
- Body Parameters:
    - email: String
    - password: String
- Body Example
```
{
        "email": "abc@gmail.com",
        "password": "abc123"
}
```
- Example Request:
    - PUT http://localhost:4000/users/62a117beead14963dfc16bae

- Expected Responses:
    - 200 OK - For a successful update
    - 400 BAD REQUEST - If the request body is improperly formatted or missing required information
    - 404 NOT FOUND - If there is no user in the database with that id
    - 500 INTERNAL SERVER ERROR - If update was unsuccessful

DELETE /users/:id
- Description: This endpoint is to delete a user account from the database with the specific id
- Example Request:
    - DELETE http://localhost:4000/users/62a117beead14963dfc16bae
- Expected Responses:
    - 200 OK - For a successful deletion
    - 404 NOT FOUND - If there is no user in the database with that id
    - 500 INTERNAL SERVER ERROR - If delete was unsuccessful

story.routes.js
POST /stories/
- Description: This endpoint is to add a user-submitted story into the database
- Body Parameters:
    - storyTitle: String
    - storyText: String
    - storyType: String
    - storyStatus: String
- Body Example

```
{
        "_id": "f2a192b3cad14963daa113ae"
        “storyTitle”: “My First Story”
        “storyText”: “This is my story about how I love animals.”
        “storyType”: “characterCandidate”
        “storyStatus”: “rejected”
}
```

- Expected Responses:
    - 200 OK - For a successful post
    - 400 BAD REQUEST - If the request body is improperly formatted or missing required information
    - 500 INTERNAL SERVER ERROR - If post was unsuccessful

PUT /stories/:id
- Description: This endpoint is to update the data of a specific story accessed through an ID
- Body Parameters:

  Depending on what is updated, following are possible parameters:

    - storyTitle: String
    - storyText: String
    - storyType: String
    - storyStatus: String
- Body Example
  ```
  {
          "_id": "be3d764b3cad14963dfc113se"
          “storyStatus”: “rejected”
  }
  ```
-
- Expected Responses:
    - 200 OK - For a successful put
    - 400 BAD REQUEST - If the request body is improperly formatted or missing required information
    - 500 INTERNAL SERVER ERROR - If put was unsuccessful

GET /stories/
- Description: This endpoint is to get all user-submitted stories from the database with a specific query
- Example Call:
    - GET http://localhost:4000/stories/
- Response Body Example:
  ```
  [{
          "_id": "000092b3ddd14a63afc11ffe"
          storyTitle: "Example Story One",
          storyText: "This is my first example.",
          storyType: "customApparel",
          storyDate: "2022-06-11T00:11:20.163Z",
          storyStatus: "new", …
  },
  {
          "_id": "aca19ff3cad12963df001dee"
          storyTitle: "Example Story Two",
          storyText: "This is my second example.",
          storyType: "characterCandidate",
          storyDate: "2022-06-11T00:11:20.163Z",
          storyStatus: "new", …
  }]
  ```
- Expected Responses:
    - 200 OK - For a successful get
    - 500 INTERNAL SERVER ERROR - If get was unsuccessful

GET/stories/:id
- Description: This endpoint is to fetch a specific story from the database using its ID

- Example Request:
    - GET http://localhost:4000/stories/f2a192b3cad14963daa113ae
- Response Body Example:
  ```
  {
          "_id": "a6a192b3cad14963dfc113fe"
          “storyTitle”: “My First Story”
          “storyText”: “This is my story about how I love animals.”
          “storyType”: “characterCandidate”
          “storyDate”: “2022-06-11T00:11:20.163Z”
          “storyStatus”: “new”
  }
  ```
- Expected Responses:
    - 200 OK - For a successful get
    - 404 NOT FOUND - If there is no story in the database with that id
    - 500 INTERNAL SERVER ERROR - If get was unsuccessful

DELETE /stories/:id

- Description: This endpoint is to delete a story from the database with the specific id
- Example Request:
    - DELETE http://localhost:4000/stories/f2a192b3cad14963daa113ae
- Expected Responses:
    - 200 OK - For a successful deletion
    - 404 NOT FOUND - If there is no story in the database with that id
    - 500 INTERNAL SERVER ERROR - If delete was unsuccessful

product.routes.js

POST /products/

- Description: This endpoint is to add a product into the database
- Body Parameters:
    - productImage: String
    - productName: String
    - productPrice: Number
    - productDescription: String
    - productInventoryAmount: Number
    - productStory: mongoose.Schema.Types.ObjectId
- Body Example
  ```
  {
          "_id": "62bdd3b5bb1619c7286c17c8"
          “productImage”: “blue_hoodie.jpg”
          “productName”: “Blue Hoodie”
          “productPrice”: 15.99
          “productDescription”: “This is an example description”
          “productInventoryAmount”: 15
          “productStory”: “62bdd38fbb1619c7286c17c4”
  ```

}

- Expected Responses:
    - 200 OK - For a successful post
    - 400 BAD REQUEST - If the request body is improperly formatted or missing required information
    - 500 INTERNAL SERVER ERROR - If post was unsuccessful

PUT /products/:id

- Description: This endpoint is to update the data of a specific product accessed through an ID
- Body Parameters:

  Depending on what is updated, following are possible parameters:
    - productImage: String
    - productName: String
    - productPrice: Number
    - productDescription: String
    - productInventoryAmount: Number
    - productStory: mongoose.Schema.Types.ObjectId
- Body Example

```
{
        "_id": "62bdd3b5bb1619c7286c17c8"
        “productDescription”: “This is a new description”
}
```

-
- Expected Responses:
    - 200 OK - For a successful put
    - 400 BAD REQUEST - If the request body is improperly formatted or missing required information
    - 500 INTERNAL SERVER ERROR - If put was unsuccessful

GET /products/

- Description: This endpoint is to get all products from the database with a specific query
- Example Call:
    - GET http://localhost:4000/products/
- Response Body Example:

```
[{
        "_id": "62bdd3b5bb1619c7286c17c8"
        “productImage”: “blue_hoodie.jpg”
        “productName”: “Blue Hoodie”
        “productPrice”: 15.99
        “productDescription”: “This is an example description”
        “productInventoryAmount”: 15
        "productStory": "62bdd38fbb1619c7286c17c4"
    },
    {
        "_id": "62bdd3d9bb1619c7286c17ce"
        "productImage": "yellow_hoodie.jpg"
        "productName": "Yellow Hoodie"
        "productPrice": 20.05
        "productDescription": "This is another example description"
        "productInventoryAmount": 13
        "productStory": "62bf864466228ab000573a5c"
    }]
```

- Expected Responses:
    - 200 OK - For a successful get
    - 500 INTERNAL SERVER ERROR - If get was unsuccessful

GET/products/:id

- Description: This endpoint is to fetch a specific product from the database using its ID
- Example Request:
    - GET http://localhost:4000/products/62bdd3d9bb1619c7286c17ce
- Response Body Example:

```
    {
        "_id": "62bdd3d9bb1619c7286c17ce"
        "productImage": "yellow_hoodie.jpg"
        "productName": "Yellow Hoodie"
        "productPrice": 20.05
        "productDescription": "This is another example description"
        "productInventoryAmount": 13
        "productStory": "62bf864466228ab000573a5c"
    }
```

- Expected Responses:
    - 200 OK - For a successful get
    - 404 NOT FOUND - If there is no product in the database with that id
    - 500 INTERNAL SERVER ERROR - If get was unsuccessful

DELETE /products/:id

- Description: This endpoint is to delete a product from the database with the specific id
- Example Request:
    - DELETE http://localhost:4000/productss/62bdd3d9bb1619c7286c17ce
- Expected Responses:
    - 200 OK - For a successful deletion
    - 404 NOT FOUND - If there is no product in the database with that id
    - 500 INTERNAL SERVER ERROR - If delete was unsuccessful

orders.routes.js

POST /orders/

- Description: This endpoint is to add an order into the database
- Body Parameters:
    - transactionDate: Date
    - purchasedBy: Schema.Types.ObjectId
    - shippingInfo: {String, String, String}
    - billingInfo: {String, String, String}
    - products: [{Schema.Types.ObjectId, Number}, … ,{Schema.Types.ObjectId, Number}]
- Body Example

```
{
        "transactionDate": "Fri, 08 Jul 2022 05:50:18 GMT",
        "purchasedBy": "62c06f44e1a5a75d3b2f828c",
        "shippingInfo": {
                "firstName": “Jenny Lin",
                "lastName": "Lin",
                "address": "874 WaterFord Drive, Toronto ON, AK3 4U1"
        },
         "billingInfo": {
                "firstName": “Jenny Lin",
                 "lastName": "Lin",
                 "address": "874 WaterFord Drive, Toronto ON, AK3 4U1",
                "paymentMethod": "Visa"
        },
        "products": [
                {
                        "pid": "62c09def9dd01b530ca3b68d",
                        "itemCount": 2
                 },
                 {       "pid": "62c0a0c19dd01b530ca3b6b7",
                         "itemCount": 1
                 }
         ]
}
```

- Expected Responses:
    - 200 OK - For a successful post
    - 400 BAD REQUEST - If the request body is improperly formatted or missing required information
    - 500 INTERNAL SERVER ERROR - If post was unsuccessful

PUT /orders/:id

- Description: This endpoint is to update the data of a specific order accessed through an ID
- Body Parameters:

  Depending on what is updated, following are possible parameters:
    - transactionDate: Date
    - purchasedBy: Schema.Types.ObjectId
    - shippingInfo: {String, String, String}
    - billingInfo: {String, String, String}
    - products: [{Schema.Types.ObjectId, Number}, … ,{Schema.Types.ObjectId, Number}]

- Body Example
```
{
        "_id": "43c06aace1a5a75d34a2828d",
        "paymentMethod": "Paypal"
}
```
-
- Expected Responses:
    - 200 OK - For a successful put
    - 400 BAD REQUEST - If the request body is improperly formatted or missing required information
    - 500 INTERNAL SERVER ERROR - If put was unsuccessful

GET /orders/

- Description: This endpoint is to get all orders from the database with a specific query
- Example Call:
    - GET http://localhost:4000/orders/
- Response Body Example:
```
{
        "_id": "62cb46361e48ae7bf292c3d5"
        "shippingInfo": {
                "firstName": "John",
                "lastName": "Doe",
                "address": "874 WaterFord Drive, Toronto ON, AK3 4U1"
        },
         "billingInfo": {
                "firstName": "Amy",
                "lastName": "Smith",
                "address": "993 Elefor Street, Richmond ON, F31 2LW",
        }
        "paymentMethod": "Visa",
        "transactionDate": "2022-07-08T05:50:18.000Z",
        "purchasedBy": {
                "_id": "62c06f44e1a5a75d3b2f828c",
                "firstName": "jacob",
                "lastName": "matias",
                "email": "jacob.matias@gmail.com"
        },
        "products": [
                {
                        "pid": "62c09def9dd01b530ca3b68d",
                         "itemCount": 4,
                         "_id": "62cb46361e48ae7bf292c3d6"
                },
                {
                        "pid": "62c0a0c19dd01b530ca3b6b7",
                        "itemCount": 2,
                        "_id": "62cb46361e48ae7bf292c3d7"
                }],
}
```

- Expected Responses:
  - 200 OK - For a successful get
  - 500 INTERNAL SERVER ERROR - If get was unsuccessful

GET/orders/:id

- Description: This endpoint is to fetch a specific order from the database using its ID
- Example Request:
  - GET http://localhost:4000/orders/decb3263dae48a89bf292cb1f
  -
- Response Body Example:

```
{
        “_id”: “decb3263dae48a89bf292cb1f”
        "transactionDate": "Fri, 08 Jul 2022 05:50:18 GMT",
        "purchasedBy": {
                "_id": "62cb46361e48ae7bf292c3d5",
                "firstName": "Jenny",
                "lastName": "Lin",
                "email":”Jenny.Lin@gmail.com"
        },
        "shippingInfo": {
                 "firstName": “Jenny Lin",
                 "lastName": "Lin",
                 "address": "874 WaterFord Drive, Toronto ON, AK3 4U1"
        },
         "billingInfo": {
                "firstName": “Jenny Lin",
                 "lastName": "Lin",
                 "address": "874 WaterFord Drive, Toronto ON, AK3 4U1",
                "paymentMethod": "Visa"
            },
            "products": [
                {
                    "pid": "62c09def9dd01b530ca3b68d",
                    "itemCount": 2
                 },
                 {       "pid": "62c0a0c19dd01b530ca3b6b7",
                         "itemCount": 1
                 }
             ]
        }
```

- Expected Responses:
    - 200 OK - For a successful get
    - 404 NOT FOUND - If there is no order in the database with that id
    - 500 INTERNAL SERVER ERROR - If get was unsuccessful

DELETE /products/:id
- Description: This endpoint is to delete an order from the database with the specific id
- Example Request:
    - DELETE http://localhost:4000/orders/decb3263dae48a89bf292cb1f
- Expected Responses:
    - 200 OK - For a successful deletion
    - 404 NOT FOUND - If there is no order in the database with that id
    - 500 INTERNAL SERVER ERROR - If delete was unsuccessful

receipt.routes.js
POST /receipts/
- Description: This endpoint is to send a receipt email after user paid for the order
- Body Parameters:
    - orderNumber: String
    - orderTotal: Number
    - address: String
    - User: {userID: String, firstName: String, email: String}
- Body Example:

```
{
  "orderNumber": "62cb442641e48ae7bf2934j4qf3",
  "orderTotal": 3000,
  "address": "Toronto, Ontario",
  "user": {
     "userId": "62cb46361e48ae7bf292c4g3",
     "firstName": "Joe",
     "email": "Joe@gmail.com"
  }
}
```

- Expected Response:
  - 200 OK - Email sent successfully
  - 500 INTERNAL SERVER ERROR - If email was not sent successfully

contact.routes.js
POST /contactus/

- Description: This endpoint is to send an email with customer message to company email: contactform.characters@gmail.com
- Body Parameters:
  - name: String
  - message: String
  - email: String
- Body Example:

```
{
  "name": "John",
  "Email": ”john.doe@gmail.com”,
  "message": "How can I share my story?",
}
```

- Expected Response:
  - 200 OK - Email sent successfully
  - 500 INTERNAL SERVER ERROR - If email was not sent successfully

Frontend

/frontend/src/

App.js
This file displays the home page and the main header. Upon render, it retrieves the existing user account stored in localStorage and stores it in its state.
Methods in this file are:

- signIn(data)

    - This method is passed in as a property for SignUp.js and Login.js. Upon successful sign in, this function should be called with the data returned by the database.
    - Stores the user data in the localStorage and the App state, which can be detected by other components
- signOut
    - Signs out a user by removing the user data from the state and the localStorage.
- setToast
    - Creates a toast with a message

/frontend/src/components

SignUp.js
This file displays the sign up page for users to sign up an account. It has three input fields: email, password, and password confirmation. When a user submits an invalid data, error messages will appear below the text input fields. Examples include empty email/password fields, invalid email formats, and password confirmation field doesn't match the password field. Methods in this file are:

- validateLogin
    - This method validates the sign up data entered. This function also sets the warning messages for the text inputs upon validating the sign up info entered.
- onEmailChange
    - This method updates the email variable in the state with the text input field for email.
- onPasswordChange
    - This method updates the password variable in the state with the text input field for password.
- onChangePasswordConfirm
    - This method updates the passwordConfirm variable in the state with the text input field for password confirmation.
- onSubmit
    - This method first validates the form and sends an http post request to the backend with the user account details. Upon successful sign up, the user data is stored in localStorage and the user is redirected to the home page.

Login.js
This is the login screen that gives the user the ability to log in. The page takes 2 text inputs, an email address, and a password which can be submitted after clicking the "Login" button. When a user submits invalid login data, error messages will appear below the text input fields. Examples include empty email/password fields, non-registered email, and invalid passwords.

Methods in this file are:

- validateLogin

    - This method validates the login data entered by making a backend call to find the user via email entered in the database and matches the password entered with the one registered in the database. This function also sets the warning messages for the text inputs upon validating the login info entered.
- onEmailChange
    - This method updates the email variable in the state with the text input field for email.
- onPasswordChange
    - This method updates the password variable in the state with the text input field for password.
- onSubmit
    - This method handles the logic on what to do after the “Login” button is clicked. Upon successful login, the user data is stored in localStorage and the user is redirected to the home page.

StoriesList.js
This is the Story Statuses page which allows employees to view a list of all the stories that have been submitted and their statuses. When the page is opened it fetches all the stories stored in the database in order to create a list of StoriesRow components which display the title status and a details button specific to that story. Upon click of the details button a new page with the story details will open. It provides the ability to filter the stories by story status and if the apparel is to be custom made or for a character candidate.

Methods in this file are:

- getStories
    - This method creates an array of StoriesRow components that are displayed on the page.
- onCheckNew
    - This method updates the statusNew variable in the state with the opposite boolean value that statusNew currently is.
- onCheckInterviewing
    - This method updates the statusInterviewing variable in the state with the opposite boolean value that statusInterviewing currently is.
- onCheckValidated
    - This method updates the statusValidated variable in the state with the opposite boolean value that statusValidated currently is.
- onCheckRejeced
    - This method updates the statusRejected variable in the state with the opposite boolean value that statusRejected currently is.
- onCheckCustom
    - This method updates the customApparel variable in the state with the opposite boolean value that customApparel currently is.
- onCheckCharacter

    - This method updates the characterCandidate variable in the state with the opposite boolean value that characterCandidate currently is.
- clearFilters
    - This method sets all of the variables statusNew, statusInterviewing, statusValidated, statusRejected, customApparel, characterCandidate in the state to false so that there are no filters applied to the list of stories
- applyFilters
    - This method returns a list of stories with the criteria selected in the filter
- getTitles
    - This method returns a list of titles of stories that match the criteria selected in the filter

StoriesRow.js
This component represents one row in the list of stories displayed on the Story Statuses page. Each row includes the story title, status and a details button that opens a new page with the details of the story.

Methods in this file are:

- goToDetails
    - Upon click of the details button, this method redirects the employee to the Story Details page of the story on that row.

SubmitStory.js
This is the Submit a Story webpage which provides users with an interface to submit their stories in order for them to be later evaluated by Character employees. The page consists of two text fields to allow users to enter their story and story title, radio buttons to determine whether or not the story is being submitted for custom apparel and a submit button to process the submission. User input for all text fields and radio buttons are required, thus attempting to submit an unfilled form will trigger a popup over the topmost field prompting the user to provide an input. If all required input fields are filled-in and the user submits, an alert will pop-up confirming the success of the user’s submission.

Methods in the file are:

- onSubmit
    - This function creates a JSON object using the state variable values and posts it to the database as a new story object. This function also resets the story-related state variables and changes the state of the submissionSuccess variable.
- onChangeStoryText
    - This function updates the storyText variable in the state with the text field input for story.
- onChangeStoryTitle
    - This function updates the storyTitle variable in the state with the text field input for story title

- onCheckYes
    - This function updates the storyType variable in the state with the string “customApparel”
- onCheckNo
    - This function updates the storyType variable in the state with the string “characterCandidate”

StoryDetails.js
This is a page where employees can view submitted stories in full detail, and change story’s status depending on where the story is in its journey. When this page is opened it fetches data of a story using its ID from the database. This page displays the story, its title, status, submission date, and story type. Story type can be either characterCandidate or customApparel based on Character’s choice at the time of submission. The story’s status can be changed to one of the following: new, interviewing, and validated. The status is changed via selection of an option in the dropdown. The change is reflected in the database, and thus, everywhere in the web app where employees can view the status, the latest status is displayed.

Methods in the file are:

- updateStatus
    - This method takes a status string as a parameter.
    - The appropriate string is passed to the method when one of the options in the “Change Status” dropdown is chosen.
    - It uses the PUT request to update the corresponding story’s status in the database.
    - It updates the state of the page to reflect the new status.
- onDelete
    - This method deletes the story by changing the isDeleted field to true.

SubmitProduct.js
This is a page where employees can add products to a character story. They select the character, and input the product name, inventory amount, price, product description and select an image of the product.
Methods in this file include:

- onChangeProductName
    - Updates product name
- onChangeProductType
    - Updates product type
- onChangeStoryCharacter
    - Updates storyCharacter field
- onChangeProductInventoryAmount
    - Updates productInventoryAmount field
- onChangeProductDescription
    - Updates product description field
- onUploadImage
    - Updates image field

- onSubmit
    - Submits the request to add the product to the character story.

ProductDetails.js
This is a page that displays products in more detail. Customers can see the product description, name, price, stock left, and the story behind the product on this page. They can also add products to the cart on this page.

Methods in the file are:

- displayStoryDescription
    - This method displays the story description
    - If the story description is shorter than 258 characters it displays the story
    - If it is longer than 258 characters, it displays the first 258 characters with an option to read more
    - If customer chose to read more, they can hide the description again by pressing show less
- checkValue
    - Checks that the quantity is between 1 and inventory amount
- onQuantityChange
    - Updates the quantity amount
- onDelete
    - Deletes the product or prompts the user that the character story is not deleted thus cannot delete the product.
- onSubmit
    - This function is triggered by the add to cart button
    - It gathers all the necessary data that needs to be added to the cart
    - It formats the data as per our standard and writes it to cartProduct in local storage, which is later accessed by the Cart page

ShoppingCart.js
This component represents a page where users are able to view all items that they have added into their shopping cart from the website store. Each item added to the cart will be displayed in the form of a listings, where each listing provides users with a brief summary about the product they have added to their shopping cart. This page also provides a final price summary which contains pricing details for the items in the cart altogether (includes final subtotal, tax, and total from all products).

Methods in the file are:

- getInitialValues():
    - Returns the initial state values for the component's itemCount and subtotal states
- updateCart():
    - Updates the quantity field for products in localStorage and returns an updated copy of the cartProducts

    - If the quantity is equal to 0, then the element with pid pid will be removed from the cartproducts state, and removed from the object with key “cartproducts” in localStorage.
- updatePriceSummary():
    - Invokes updateTotal() and is primarily used as a wrapper for readability
- updateTotal():
    - Computes and returns the subtotal, tax and total, in that respective order, for all items in the shopping cart.

ShoppingCartItem.js
This is a child component of the ShoppingCart component and represents a single item in the list of items that are displayed on the page represented by ShoppingCart.js. Each listing displays details about the item including the item name, item description, item quantity, item price, item quantity, and item subtotal. Moreover, this component allows users to increment and decrement (or completely remove) the object from their shopping cart.

Methods in the file are:
- updateSubtotal():
    - Updates the item’s subtotal state.
- updateQuantity():
    - Updates the item’s quantity state to some new quantity.
    - Invokes related functions to update the item’s subtotal.
- onIncrement():
    - Invokes the updateQuantity function in order to increment item quantity by 1.
- onDecrement():
    - Invokes the updateQuantity function in order to decrement item quantity by 1.
- removeItem():
    - Removes an item from localStorage that has been added to the user’s cart .
- onCheckout():
    - Prompts the user to
        - Add items to cart
        - Login if not logged-in
        - Redirects to Checkout page if logged in

ProductCard.js
This component represents one product in the grid of products displayed on theStore page. Each product includes the product name, price and image and a view product button that opens a new page with the details of the product.

Methods in this file are:
- viewProduct
    - Upon click of the view products button or any other part of the product card, this method redirects the user to the Product Details page of the product.

ProductStore.js
This is the Store page which allows users to view all the products that have been submitted in a grid format. When the page is opened it fetches all the products stored in the database in order to create a list of ProductCard components which display the name, price, image and a details button specific to that product. Upon click of the button a new page with the product details will open.

Methods in this file are:

- listProducts
    - This method creates an array of ProductCard components that are in stock, and displays them on the page in a grid depending on the filters selected.

ProductEdit.js
This component displays the form to edit the information about the product. Here, the user can change the image, name, type, price description, and inventory amount of the product.

Methods in this file are:

- onSubmit
    - Sends a put request to the products route to change the product information
    - Sends a post request to upload a new product image to the database.

ProfileInfo.js
This is a page that displays the user profile in detail. Users can see/edit their first name, last name, email. Users can also edit their password on this page.

ProfileEdit.js
This component displays the form for the user to change their password.

Methods in this file are:

- verifyProfile
    - This method validates the user data entered. This function also sets the warning messages for the text inputs upon validating the user info entered.
- saveProfile
    - This method first validates the form and sends an http put request to the backend with the user account details. Upon successful update, a toast should appear indicating the update has been successful

PasswordEdit.js
This component displays the form for the user to change their password.

Methods in this file are:

- savePassword

    - This method validates the password. This function also sets the warning messages for the text inputs upon validating the password info.
- verifyPassword
    - This method first validates the form and sends an http post request to the backend with the new password. Upon successful update, a toast should appear indicating the update has been successful

Checkout.js
This is a mock component for temporary use where users can enter shipping, billing and payment information in order to checkout their order. There products in the order consists of all items that the user has added into their shopping cart, prior to checkout

Methods in this file are:

- updateShippingFirstName
    - Updates the shippingFirstName state
- updateShippingLastName
    - Updates the shippingLastName state
- updateShippingAddress
    - Updates the shippingAddress state
- updateBillingFirstName
    - Updates the billingFirstName state
- updateBillingLastName
    - Updates the billingLastName state
- updateBillingAddress
    - Updates the billingAddress state
- updatePaymentMethod
    - Updates the paymentMethod state
- sendEmail
    - Makes Post request that sends Receipt email
- onBack
    - This method is an onClick handler that sends users back to the shopping cart page
- onSubmit
    - This method is an onClick handler that retrieves information entered into the checkout form by the user, and uses this information in order to store a new order into the database. This method resets all state fields after the post request is made.

OrderHistory.js
This is a page that displays all previous orders made by the user. The information is displayed as a table, where each row provides information for a single order. This includes the order number, date on which the order was made, the total charge of the order, as well as a hyperlink that users can click in order to view more details corresponding to the order.

Methods in this file are:

- computeTotal
    - This method computes and returns the total charge (tax-included) for products included in a single order.

OrderDetails.js
This is a page that displays a detailed overview of order information for a single order made by a user. This information includes order number, date on which the order was made, complete shipping information, complete billing information, and an order summary that includes details about each item in the order, as well information pertaining to the monetary charges.

Methods in this file are:

- onBack
    - This method is an onClick handler that sends users back to the orderHistory page
- updateTotals
    - This method takes in an itemSubtotal and adds it to the subtotal state. This method also recomputes the tax state and total state, taking into account the new subtotal.

OrderDetailsItem.js
This is a component that is used on the OrderDetails.js page in order to display information about a single item in the order. The information that is displayed by the component includes an image of the item, an item description, the quantity of items purchased in the order, as well as the subtotal (pre-tax).

Methods in this file are:

- updateTotals
    - This method computes the subtotal for a single product in order summary in the orderDetails page, and references OrderDetails.js' updateTotals method in order to update the total charges belonging to the entire order.

ContactUs.js
This is a page that displays a Contact Us form. The page takes three text inputs: name, email, message. The message can be submitted using the "Submit" button. If email has an incorrect format a message will be displayed and the message will not be submitted. For successful submission all fields must be filled out. When message is successfully submitted, a toast message appears and an email to contactform.characters@gmail.com is sent.

Methods in this file are:

- onChangeMessage
    - Updates the message state with the user submitted message.
- onChangeName
    - Updates the name state with the user submitted name.

- onChangeEmail
    - Updates the email state with the user submitted email.
- onSubmit
    - This method is an onClick handler that retrieves information entered into the contact form by the user, and uses this information to send the email to company email.
    - This method resets all state fields after the post request is made.

Home.js
This is the home page, which is first shown when the site loads. It provides the user with information on who Characters are and their mission. It also gives an overview on how the story submission process works. The page contains a button that directs the user to the store, as well as a button that brings the user to the contact us form.

CustomerOrders.js
This is a page that displayed all orders that have been checked out by any user and is only visible to Employee users. The orders are arranged in a list, where each row provides a brief description of the order. Orders can be filtered using the filter options for fulfilled and unfulfilled orders. Moreover, employees can click the view details button that corresponds to an order entry in order to be directed to the orderDetails page for an order, where the employee can modify the order to be fulfilled (if the order has not already been fulfilled).

Methods in this file are:

- onCheckFulfilled
    - This method reacts to the checkbox onclick for the ‘Fulfilled’ filter option, and updates the filterByFulfilled state accordingly
- onCheckUnfulfilled
    - This method reacts to the checkbox onclick for the ‘Unfulfilled’ filter option, and updates the filterByUnfulfilled state accordingly
- clearFilters
    - This method clears all filters by resetting all filter states to false
- computeTotal
    - This method computes and returns the total charge (tax-included) for products included in a single order.
- getFilteredOrders
    - This method applies the activated filters to all orders and returns the result
- displayFilteredOrders
    - This method displays the all filtered orders that are returned by the getFilteredOrders method.