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.

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

- 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

/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

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.

Methods in this file are:

- getStories
    - This method creates an array of StoriesRow components that are displayed on the page.

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.

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
- 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
- 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 .

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

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