API

Documentation

Restaurant App

Table of Contents

Models

User.js

- *Name* : String : Stores the user's full name.
- *Email* : String : Stores the user's Email.
- *Password*: String : Stores the user's password : hashed using **bcrypt.**
- *Address*: String : Store's the user's physical address.
- *Longitude*: Number :First part of the coordinates.
- *Latitude*: Number : Second part of the coordinates.
- *Order*: String : Stores the id of the user's order so it can be looked up later on.
- *isAdmin*: Boolean : (default: false) : Gives the user admin privileges.
- *isEnabled*: Boolean : (default: true) : Specifies if the account is enabled or not.

Branch.js

- *Name*: String : The name of the branch.
- *Longitude*: Number : First part of the coordinates.
- *Latitude*: Number : Second part of the coordinates.

Category.js

- *Name*: String : Stores the name of the Category.
- *Image*: String: Stores the path to the category's image which stored locally.
- *Items*: [] : Stores the ids of different items belonging to this category for quick look up.

Item.js

- *Name*: String: Stores the name of the item.
- *Price*: Number : Stores the price of the item.
- *Image*: String: Stores the path to the item's image which stored locally.
- *Category*: String: Stores which category it belongs to.

Order.js

- *Status*: String: Stores the status of the order (default: pending). Admin can change this to complete or rejected.
- *Owner*: String: Stores the id of the owner of this order.
- *Total*: Number: Stores the total price of the order.
- *Store*: String: Stores the id of the branch responsible for this order.
- *Items*: []: Stores the ids of the items added to this order.

Routes

Auth.js

POST '/register'

- JSON body { name, email, password }
- **Joi** is used to validate this input making sure the email is valid and password reaches the minimum length required.
- Checks if the email already has an account and prevents him from registering again if so.
- **bcrypt** is used to hash the password before it is stored.
- Once complete, **jwt** is used to generate access and refresh tokens using the user's id and isAdmin status. Both ACCESS_TOKEN_SECRET and REFRESH_TOKEN_SECRET are stored in a **dotenv** file for privacy.
- Returns the access and refresh tokens so they can be used while Postman testing.

POST '/login'

- JSON body { email, password }
- **Joi** is used to validate this input making sure the email is valid and password reaches the minimum length required.
- Checks for email presence in database.
- Hashes the password entered using **bcrypt** and checks for a match for the given email.
- Checks if the account is disabled.
- Once complete, **jwt** is used to generate access and refresh tokens using the user's id and isAdmin status. Both ACCESS_TOKEN_SECRET and REFRESH_TOKEN_SECRET are stored in a **dotenv** file for privacy.
- Returns the access and refresh tokens so they can be used while Postman testing.

POST '/genToken'

- JSON Header {auth-token: Bearer (enter REFRESH token here and remove parenthesis) }
- Requires the presence of "auth-token" field in the header, where its value is "Bearer" followed by a space and then the refresh token of the logged in user.
- i.e.: Bearer nflgsfulrgelgvrlabhuk7itogylu;i
- This is used to generate a new access once it expires.
- Denies access if the refresh token is not valid.

- Once the refresh token is verified by **jwt** a new access token is generated and is returned to be used for postman testing.

DELETE '/logout'

- JSON Header {auth-token: Bearer (enter access token here and remove parenthesis) }
- Removes all present refresh tokens for this user. Hence once the access token expires the user is required to login again with his credentials.

verifyToken.js

- Middleware used to verify users on the app.
- Requires the presence of "auth-token" field in the header, where its value is "Bearer" followed by the access token of the logged in user. i.e.: Bearer nflgsfulrgelgvrla
- The access token is then verified by **jwt** and returns the user's id and isAdmin status to be used in other requests.
- This function appends a user value to the request containing id and isAdmin values.
- i.e.: req.user._id and req.user.isAdmin

userAccount.js

GET '/user/profile'

- JSON Header {auth-token: Bearer (enter access token here and remove parenthesis) }
- Uses the returned id and isAdmin status from the verification process to retrieve the information related to this account.

PATCH '/user/profile'

- JSON Header {auth-token: Bearer (enter access token here and remove parenthesis) }
- JSON Body {name, password, address, longitude(Number), latitude(Number)}
- Update the user with the entered information

adminAccount.js

GET '/admin/orders'

- JSON Header {auth-token: Bearer (enter access token here and remove parenthesis) }
- Checks if the user is an admin using the isAdmin property.
- Returns all the present orders.

PATCH '/admin/handle'

- JSON Header {auth-token: Bearer (enter access token here and remove parenthesis) }
- JSON body {email, status(Boolean)}
- Checks if the user is an admin using the isAdmin property.
- Finds the user of the entered email and changes the accounts status (Boolean) based on the entered status (Boolean). i.e.: changes isEnabled Boolean of the user to the inputted status.

PATCH '/admin/orderStatus/:orderId'

- JSON Header {auth-token: Bearer (enter access token here and remove parenthesis) }
- JSON body { status(String)}
- Finds the order by ID and changes the status (String) of the order.

branches.js

GET '/branch/'

- JSON Header {auth-token: Bearer (enter access token here and remove parenthesis) }
- Checks if the user is an admin using the isAdmin property.
- Returns all available store branches.

POST '/branch/'

- JSON Header {auth-token: Bearer (enter access token here and remove parenthesis) }
- JSON Body {name(String), longitude(Number), latitude(Number)}

- Checks if the user is an admin using the isAdmin property.
- Saves the new branch to the database.

PATCH '/branch/:id'

- JSON Header {auth-token: Bearer (enter access token here and remove parenthesis) }
- JSON Body {name(String), longitude(Number), latitude(Number)}
- Checks if the user is an admin using the isAdmin property.
- Updates the branch specified by the id.

DELETE '/branch/:id'

- JSON Header {auth-token: Bearer (enter access token here and remove parenthesis) }
- Checks if the user is an admin using the isAdmin property.
- Deletes the branch specified by the id.

categories&Items.js

GET '/category/'

- Return all Categories.

POST '/category/'

- JSON Header {auth-token: Bearer (enter access token here and remove parenthesis) }
- Form-data {name, image (Type=file)}
- Checks if the user is an admin using the isAdmin property.
- Uses **multer** to manage uploading files using the upload function.
- Adds a new category.

PATCH '/category/:categoryId'

- JSON Header {auth-token: Bearer (enter access token here and remove parenthesis) }

- Form-data {name, image (Type=file)}
- Checks if the user is an admin using the isAdmin property.
- Finds category with categoryId and updates the name and image.

DELETE '/category/:categoryId'

- JSON Header {auth-token: Bearer (enter access token here and remove parenthesis) }
- Checks if the user is an admin using the isAdmin property.
- Deletes the category entry with _id as categoryId.

POST '/category/item'

- JSON Header {auth-token: Bearer (enter access token here and remove parenthesis) }
- Form-data {name, image (Type=file),price ,category }
- Checks if the user is an admin using the isAdmin property.
- Adds new item with the above fields to the Item collection.
- Adds the id of this item to the items array in the corresponding category entry in the category collection.

PATCH '/category/item/:itemId'

- JSON Header {auth-token: Bearer (enter access token here and remove parenthesis) }
- Form-data {name, image (Type=file),price ,category }
- Checks if the user is an admin using the isAdmin property.
- Updates item with the above fields to the Item collection.

DELETE '/category/item/:itemId'

- JSON Header {auth-token: Bearer (enter access token here and remove parenthesis) }
- Checks if the user is an admin using the isAdmin property.
- Deletes item from item collection with _id itemId.
- Removes the item from the items array in the corresponding category in the categories collection.

GET '/category/:categoryId'

- Access the category with categoryId and get the items array containing array of all items in this category.
- Retrieve all items in this items array.
- Return the items retrieved.
- (Filter by category)

GET '/category/item'

- Retrieves all items to scroll through.
- (Pagination)

GET '/category/item/:name'

- Search for the item with the name entered.
- (Search)

order.js

POST '/order/'

- JSON Header {auth-token: Bearer (enter access token here and remove parenthesis) }
- Gets the user id by req.user._id retrieved from the token.
- Get the user coordinates and all available store coordinates.
- Locate closest store to the user via algorithm.
- Set a new order with the user's id and the store's id.
- Link the order to the user.

PATCH '/order/:itemId'

- JSON Header {auth-token: Bearer (enter access token here and remove parenthesis) }
- Locate item by itemId.
- Locate order by saved order id.
- Insert item id into the items array in the order.
- Increment total price by the price of the item.

DELETE ‘/order/:orderId’

- JSON Header {auth-token: Bearer (enter access token here and remove parenthesis) }
- If order status is not pending reject request
- Else delete order.
- Remove order id from the user’s indormation.