

NWEN304 Group Project

<https://crud-project-nwen-304.herokuapp.com/>

Our CRUD application was made by Pravin Modotholi, Iqbal Wan and Praveen Bandarage. It is a footwear shop. Currently hosted on Heroku - <https://crud-project-nwen-304.herokuapp.com/>

Installation

After cloning the repo, run **npm install** to download all of the packages and dependencies. Then run **npm start** to run the application.

```
npm install  
npm start
```

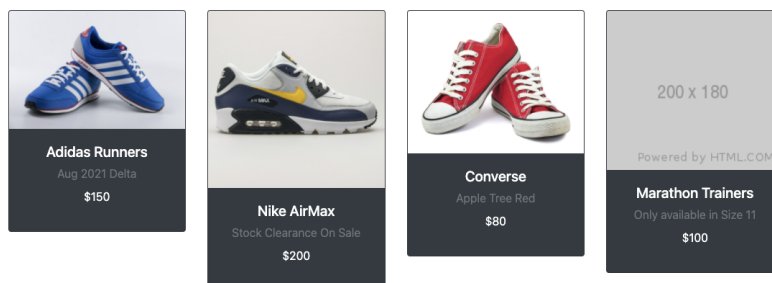
How to use the system

In terms of how to use the web application, first boot up the application on the browser if running locally. Locally the application runs on port 3000 hence the following link can be used to run the app in the browser - <http://localhost:3000>. Once booted any unauthenticated user will be presented with the index page which only displays products.



Index Page

Products



Registration

To view all features of the application, you should first register. To register a user you must click the sign up link from the nav bar or by performing a POST request for the **/auth/register** route. with your name, email and a password. The password must be of length 7 characters minimum, include at least 1 number, 1 capital letter and one special character.

Registration through the application



Sign up

Name

Email address

Password

Confirm password

Registration through a POST API request

```
curl --location --request POST 'http://localhost:3000/auth/register' \
--data-urlencode 'name=Pravin' \
--data-urlencode 'email=pravin@mail.com' \
--data-urlencode 'password=password'
```

```
curl --location --request POST 'http://localhost:3000/auth/register' \
--data-raw '{
  "name": "Pravin",
  "email": "pravin@mail.com",
  "password": "password"
}'
```

If successful a redirect will be performed to the login page, where the user can now login.

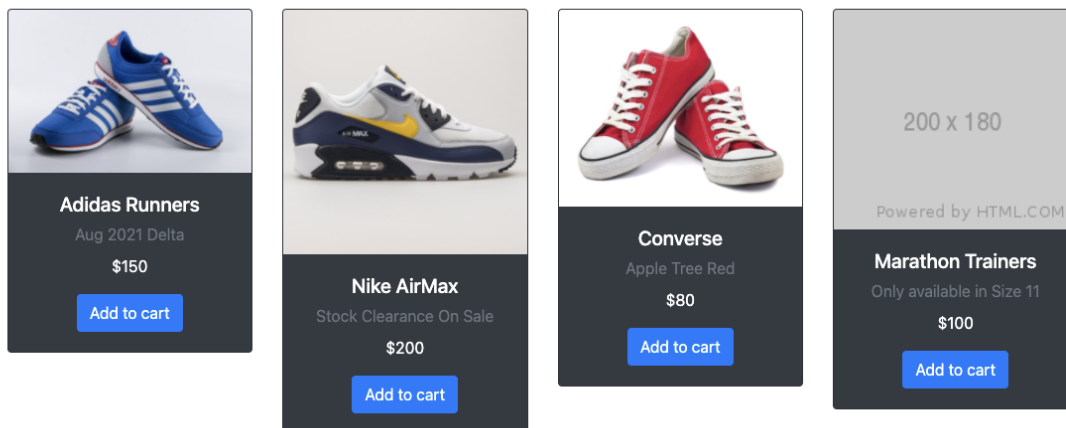
Login

To login simply enter a valid email and password on the login page, if an incorrect value is given then the page would be populated with an alert depending on the error observed. If the login is successful will redirect back to the index page, where an extra add to Cart option will be visible below each product with an extra link to view the cart.



Welcome Normal User | Index Page

Products




Authenticated users have the ability to add items to their cart as those protected routes are now open.

Login Request through a POST API request

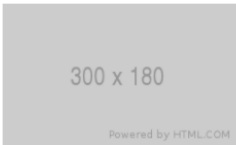





```
curl --location --request POST 'http://localhost:3000/auth/login' \
--data-urlencode 'email=pravin@mail.com' \
--data-urlencode 'password=password'
```

Shopping Cart

A user who is currently logged in is able to view all the products that they had added to their cart. In this page three actions can be performed; increase quantity, decrease quantity and remove product from cart. After decreasing if the quantity comes down to 0 then the product will be removed from the user's cart.

 SHOP

View Cart [Logout](#)

#	Image	Product	Description	Price	Quantity	Total	Action	
1		Marathon Trainers	Only available in Size 11	\$100	1	\$100	+	 -
2		Converse	Apple Tree Red	\$80	2	\$160	+	 -
3		Adidas Runners	Aug 2021 Delta	\$150	1	\$150	+	 -

Total: **\$410**
[Checkout](#)

API request to get a user Cart page:

```
curl --location --request GET 'http://localhost:3000/cart' \
--header 'Cookie: connect.sid=s%3AFmSWAbGN6RQr1amH0F5cIIcKzx5BqnTq.n7bvAXv%2FIIsN
J7sA7Tgs9BozrHimcBqLPahqKxaXnjz8'
```

Note - need to set a header with the a current authenticated user's session ID. This value is assigned when a user logs in and the session related to the current user is saved on the database. To access that session to check for authentication a session ID must which is what's stored in the cookie.

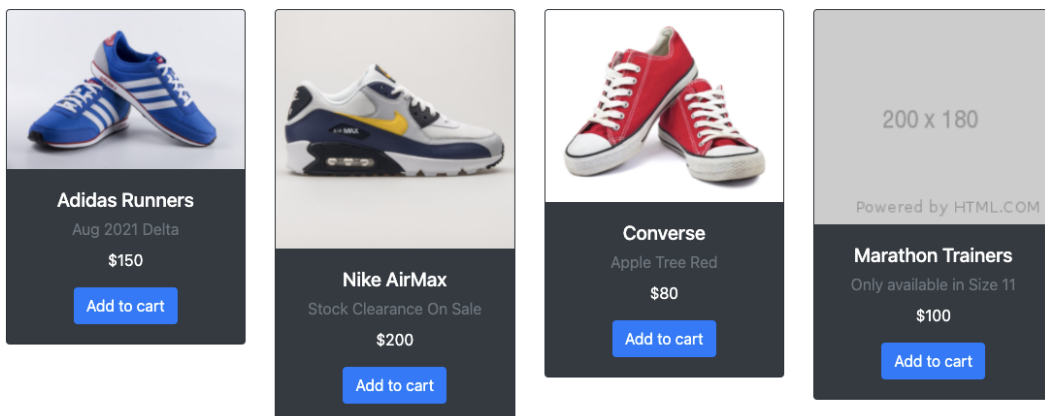
Add Product

This function is only available to admin users, for testing an admin user with the credentials of test@mail.com (email) and password (password) can be used. Admin users will have two more links available in the nav bar. An add product and Edit product link. Admins can also add products to their shopping cart similar to authenticated users.



Welcome Pravin | Index Page

Products



The add product page is just like any other form where you enter the details and the product will be added to the database. This form also allows the option to add an image with the product which will be saved locally on the server and the path of the image will be saved with the product in the database.

API post request for adding a product: Note for this function a sid of an admin will be required.

```
curl --location --request POST 'http://localhost:3000/products/add' \
--header 'Cookie: connect.sid=s%3AFmSWAbGN6RQr1amH0F5cIIcKzx5BqnTq.n7bvAXv%2FI
sNJ7sA7Tgs9BozrHimcBqLPahqKxaXnjz8' \
--form 'name="Test shoe"' \
--form 'description="Test adding a product"' \
--form 'price="355"' \
--form 'productImage=@"path/to/file"'
```

[Add Product](#)[Edit Products](#)[View Cart](#)[Logout](#)

Add Product

Product name

Choose Image (Optional)

No file chosen

Description

Price (\$)

Update Product

This functionality can be accessed on the edit product page. This page contains a dropdown menu of all the products in the database. The user simply selects one to make changes on. Once selected the form values will get autofilled for edits to be performed as shown below.

Edit Products

Adidas Runners

Modifying product "Adidas Runners".

Product name



Choose New Image (Optional)

No file chosen

Description

Price (\$)

All values can be edited and once complete the update button should be clicked. Note to perform this action the user will need to have admin rights.

```
curl --location --request PUT 'http://localhost:3000/products/edit' \
--header 'Cookie: connect.sid=s%3AFmSWAbGN6RQr1amH0F5cIIcKzx5BqnTq.n7bvAXv%2FIsNJ7sA7Tgs9BozrHimcBqLPahqKxaXnjz8' \
--form 'name="Test shoe"' \
--form 'description="Test adding a product"' \
--form 'price="355"' \
--form 'productImage=@"path/to/file"'
```

Delete Product

Delete works in a similar fashion to update where the user selects a product from the dropdown and instead of clicking update the delete button is clicked. Both of the actions will provide visual feed whether it was successful or not.

```
curl --location --request DELETE 'http://localhost:3000/products/delete' \
--header 'Cookie: connect.sid=s%3AFmSWAbGN6RQr1amH0F5cIIcKzx5BqnTq.n7bvAXv%2FIsNJ7sA7Tgs9BozrHimcBqLPahqKxaXnjz8' \
--form '_id="3434532"'
```

What the interface is (both for the web application and with REST API)

As mentioned previously, the interface for our web application and REST API is a footwear shop which allows authenticated users to add products to their shopping cart and admin users to add shoes, view shoes, edit shoes, remove shoes and add shoes to cart. The REST API enables users to complete these functions and functions/requests for this can be viewed above.

What error handling has been implemented in your system(both for the web application and with REST API)?

With the web application, we provided thorough error handling as well as helpful tips to users whenever they encountered an error. For example during registration users will only see the register button once all fields have been filled out.

Sign up

Name

Email address

Password

Confirm Password

No registration button has been added. Similarly we provide feedback to the user based on the password they have provided and only show the registration button once it meets the specs of being of the specified length of 7, a numeric character and a special character. For example a valid password would be “password192@g” however “password2” would not be sufficient.

Insufficient password

Sign up

Name

Email address

Password should contain a special character, a numeric character and be at least 7 characters long

Confirm Password

Sufficient password

Sign up

Name

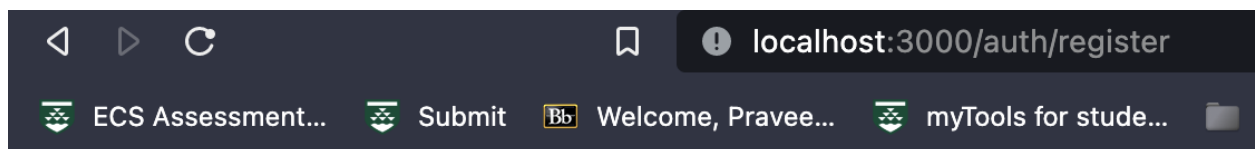
Email address

Password

Confirm password

Register

As I had already used praveenbandarage@gmail.com I encountered this error which told the user, the email was in fact already used.



Email already exists

For the login page similar visual feedback is displayed to the user when the request doesn't meet the required criteria. Possible error include "incorrect email or password", "password" is not allowed to be empty" and "email is not allowed to be empty".

Sign in

"email" is not allowed to be empty

Email address

Password

Sign in

Sign in with Google

This was achieved by converting out validation function into middleware functions. This way the request will only proceed to the next middleware function once its successfully validated. This can be seen in our access management code section, which contains the code for registerValidation.

Throughout our web application and REST API we utilised similar error checking to provide users with feedback and responses when they had provided either an invalid input or were unauthorised. For example if a user attempts to remove a product that does not exist or a product which is not in their cat, they encounter a 400 error representing a client error.

```
64  * to 0. Will return a bad request if the specified product is not
65  * found in the database and/or the user's cart.
66  * @param {Object} req The request object
67  * @param {Object} res The response object
68  */
69  module.exports.removeFromCart = async (req, res) => {
70    // Check is given product exists
71    const product = await Product.findById(req.body.productID);
72    if (!product) { res.status(400).send('Product Does not exist'); }
73
74    // Get user mongoose model
75    const user = await User.findById(req.user._id);
76
77    // Find product to remove and check if present in cart
78    const index = user.cart.findIndex(element => product._id.equals(element.productID));
79    if (index < 0) { return res.status(400).send('Product not in cart'); }
80
81    // Decrement quantity, if quantity is 0 then remove item from cart
82    user.cart[index].quantity--;
83    if (user.cart[index].quantity === 0) { user.cart.splice(index, 1); }
84  }
```

The test cases for frontend and the test scripts (e.g. a list of CURL commands / POSTMAN Requests) for the server end of your web application / service.

Please refer to above CURL commands which can be used to test the frontend, which can also be done via POSTMAN.

Database Design and Access Management Code

The current database uses MongoDB. This is NoSQL

We have developed two databases to store our data “myFirstDatabase” and “shop.” They both store products, sessions and users. The difference is “myFirstDatabase” is a test database we used for our testing process whilst “shop” is our production database. Products as the name suggests, stores the name of the product, “Converse” for example, the price, a description of the product, a unique id (for selection/querying) and an image.

We used validation throughout our project to ensure that products meet our specifications. This was done primarily through the package mongoose as well as Joi. For our users and products model we used mongoose to declare the types, as well as set requirements for each aspect.

```
//Creates the product schema which will be referred to the in MongoDB database.
const productSchema = new mongoose.Schema({
  name:{
    type: String,
    required: true,
    min: 3,
    max: 255,
  },
  description:{
    type: String,
    required: true,
    max: 1024,
    min: 3
  },
  price:{
    type: Number,
    required: true,
    min: 0.01,
  },
},
```

Joi was used for validation (middleware/dataValidation.js) to ensure that requests did meet the specifications when doing POST requests.

```

//validation package
const Joi = require("@hapi/joi");

//register validation function checks that the schema meets the requirements
const registerValidation = (req, res, next) => {
  const schema = Joi.object({
    name: Joi.string().min(6).required(),
    email: Joi.string().min(6).required().email(),
    password: Joi.string().min(6).required(),
  });
  const { error } = schema.validate(req.body);
  if (error) {
    req.session.message = error.details[0].message;
    return res.redirect('./register');
  }
  next(); // move to next middleware
};

```

To connect to our database you must be authenticated, the DB connection code is

```

const port = process.env.PORT || 3000;

database.connect().then(() => {
  app.listen(port, () => console.log(`Server Listening at Port: ${port}`));
})

```

In regards to how to access the database, the DB is connected here and the driver for the database is in startup/database.js.

This is where we pass in the connection code which is stored in our .env file as well as connect to the database. Upon a successful connection a user will see a print statement in the terminal saying "Database connection successful."

```

    } else {
      mongoose.connect(process.env.DB_CONNECT,
        { useNewUrlParser: true})
        .then((res, err) => {
          if (err) return reject(err);
          console.log('Database connection successful');
          resolve();
        })
    }
  }
}

```

User Model Database schema

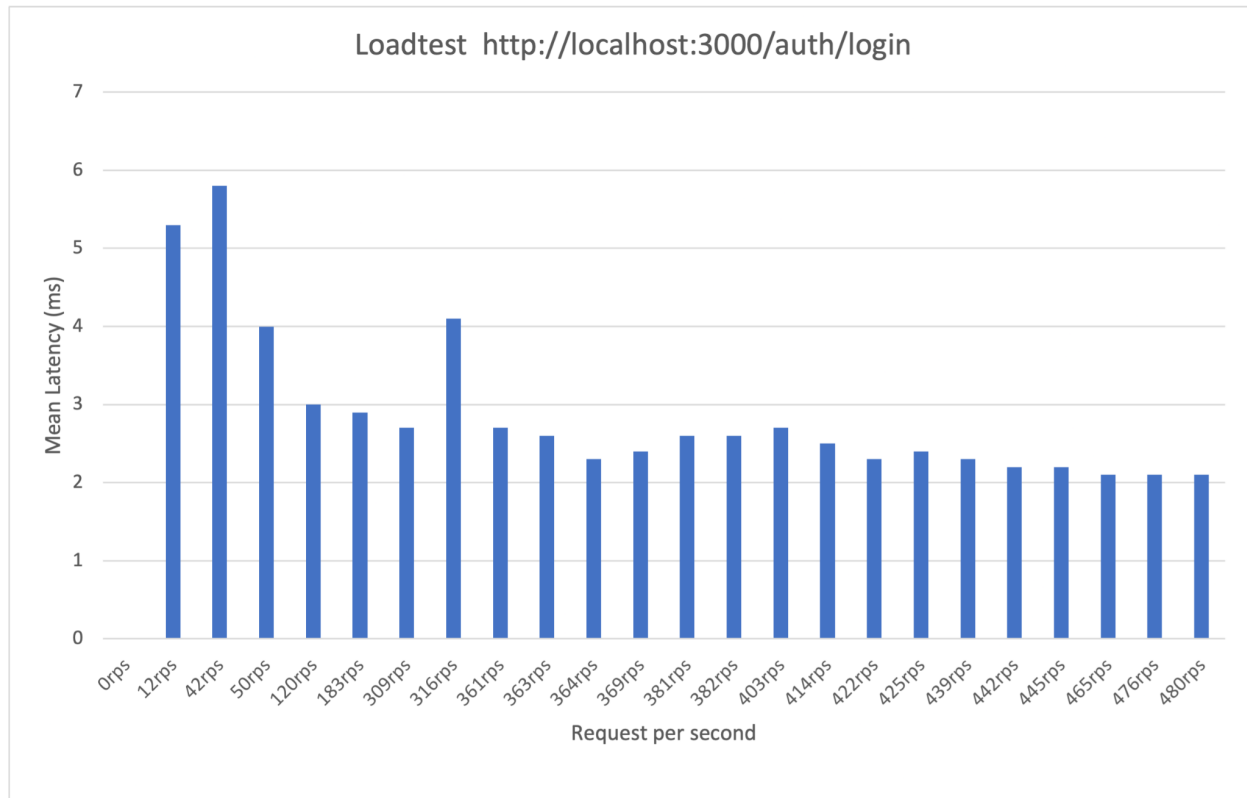
The user model contains the following properties:

- Name:
 - This property is just a string which is used to Identify the user by their name. Helps customize the web page and target each specific user.
- Email:
 - This property is used as the username as email tends to be unique, proves to be useful when trying to find the user from the database and serves as a better alternative compared to using the object ID.
- Password:
 - This property is used to authenticate a user and check if the account/information they are trying to access is actually theirs. Passwords are hashed using bcrypt before being saved in the database. For the required field a function is used which makes the password required if a local user is trying to create an account.
- type:
 - This property lets the server determine which type of user they are. Currently there can be two types of users: "local" and "google". This property is used as we don't have access to the passwords of google users. Hence every time a user logs in using google they will be authenticated against google users.
- Cart:
 - This property is an object array, where each object contains a reference to a product model and a quantity. This array is used to keep track of the products in each user's cart. Everytime a query is made for the user's cart, reference IDs get populated with the products in the database.
- date:
 - This property simply means the date at which the user registered into the server.
- isAdmin:
 - This property is used to distinguish admin users from normal customers, the property is a boolean value which is set to false by default on creation. All admin users will start off as normal users and in order to turn a user into an admin this property must be manually changed in from the database.

Load Tests Example

Testing GET requests to Login Page

```loadtest <http://localhost:3000/auth/login>``` (with variations in RPS)



GET REQUEST for localhost/3000 example

```
Database connection successful
Server Listening at Port: 3000
[nodemon] restarting due to changes...
[nodemon] starting 'node server.js'
Database connection successful
Server Listening at Port: 3000
GET / [STARTED]
GET / [FINISHED] 127.671 ms
GET / [CLOSED] 147.567 ms
GET /images/2021-10-16T07:43:39.063Z599627_00.png.png [STARTED]
GET /images/2021-10-16T07:43:39.063Z599627_00.png.png [CLOSED] 141.748 ms
GET /images/2021-10-16T07:45:09.118Z567948_00.png.png [STARTED]
GET /images/2021-10-16T07:45:09.118Z567948_00.png.png [CLOSED] 3.496 ms
GET /images/2021-10-16T07:46:18.960Z702994_00.png.png [STARTED]
GET /images/2021-10-16T07:46:18.960Z702994_00.png.png [CLOSED] 6.408 ms
GET /images/2021-10-16T07:47:18.190Z641289_00.png.png [STARTED]
GET /images/2021-10-16T07:51:38.384Z105568_00.png.png [STARTED]
GET /images/2021-10-16T07:47:18.190Z641289_00.png.png [CLOSED] 21.511 ms
GET /images/2021-10-16T07:51:38.384Z105568_00.png.png [CLOSED] 17.762 ms
GET /images/2021-10-16T07:49:56.140Z488879_00.png.png [STARTED]
GET /images/2021-10-16T07:53:36.668Z551106_00.png.png [STARTED]
GET /images/2021-10-16T07:49:56.140Z488879_00.png.png [CLOSED] 4.643 ms
GET /images/2021-10-16T07:53:36.668Z551106_00.png.png [CLOSED] 5.104 ms
```

## Packages used

- **@hapi/joi** - for validation incoming request data
- **bcryptjs** - For hashing passwords
- **connect-mongo** -For saving users sessions on the database instead of server memory
- **dotenv** - Configuring/setting environment variables
- **Ejs** -Template engine and rendering web pages
- **Express-session** - Used to track user sessions
- **Express** - Server side framework to build api routes
- **Mongoose** - Object data modeling library for MongoDB
- **mongoose-findOrCreate** -A plugin function used during google oauth
- **Multer** - To parse multipart form data and handle incoming files on requests
- **passport** - Library used to build the authentication and authorization logic
- **passport-google-Oauth-20** - Implement the google strategy and allow google logins
- **Passport-local** - Implement the local passport strategy and allow users to login in locally
- **Chai** - Assertion library used for testing
- **Chai-http** - To test api requests
- **Mocha** - Javascript test runner
- **Mockgoose** - Used to initialize Mock database
- **Nodemon** - Used for development to automatically restart server