What was accomplished:

The prototype is based on my EDR which explored the implementation of extension of SSH cloud with a recipe suggestion function that, using the data about available ingredients from an SSH Camera, and a database of recipes, suggests possible recipes using those ingredients. The prototype aims to establish the feasibility of such feature integrated into the SSH cloud.

The prototype is developed using Express JS, which is a minimal and flexible Node.js web application framework that makes creating a robust API quick and easy by giving a plenty of HTTP utility methods and middleware at our disposal. The front end is rendered using EJS templates that make writing JavaScript alongside HTML seamless as any data passed by server to the templates, can be looped over or checked by if statements using JavaScript alongside html. A document based Mongo DB collections are used as the database. Jest is used to write unit and E2E integration tests to make sure that the code behaves as expected. Docker is used to build the image of application with all the dependencies so that the code can run in any environment

How it was accomplished:

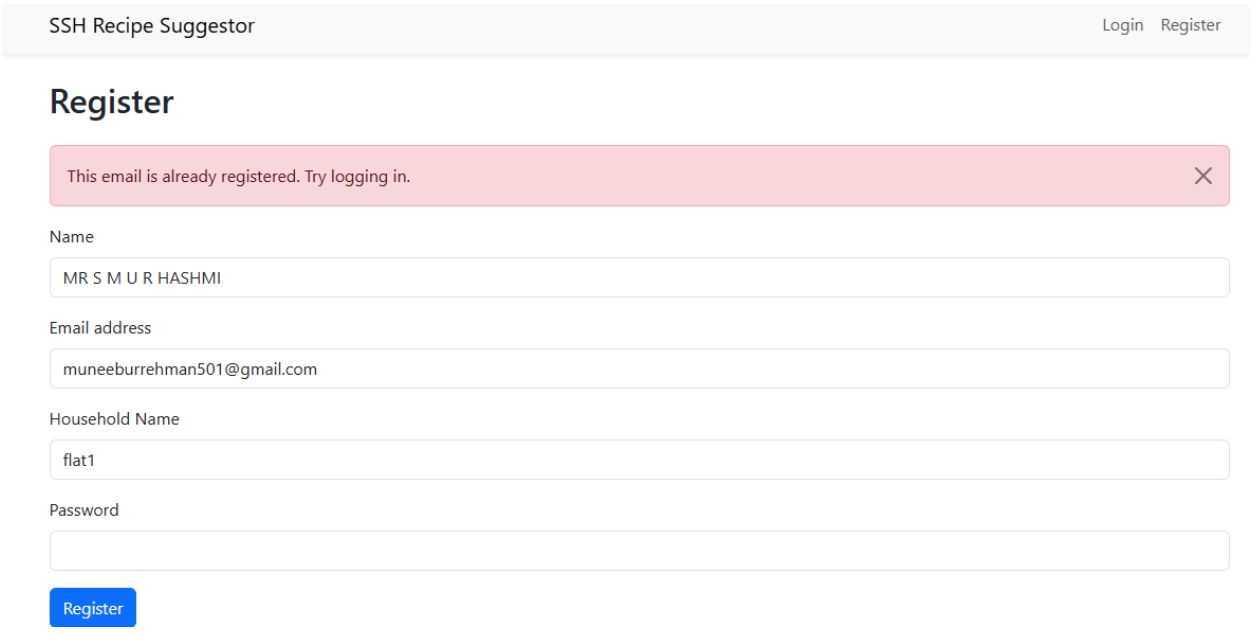
Firstly, there are four documents in the database. A User document stores name, email for our user alongside a reference to the object id in the House Hold document shedding light on which user belongs to which household. This is equivalent of primary and foreign key usage in SQL based databases. There is a Inventory document which stores contents of the fridge and each item has reference to the User and house hold document as well to make sure **when the fridge route is rendered, only items belonging to a particular user are displayed to him**. Lastly, there is Recipe document which stores content needed for each recipe and **has been seeded through seed.js.**

There are two authentication routes namely**, register and login.** The authentication and authorization is managed by **PassportJs** , a dependency which integrates **user data into session data** so that **user data persists across multiple https request made to different routes**. The **isauthenticated middleware** provided by PassportJs is used in fridge and recipes routes **to make them protected so that only a user logged in can visit these routes.**

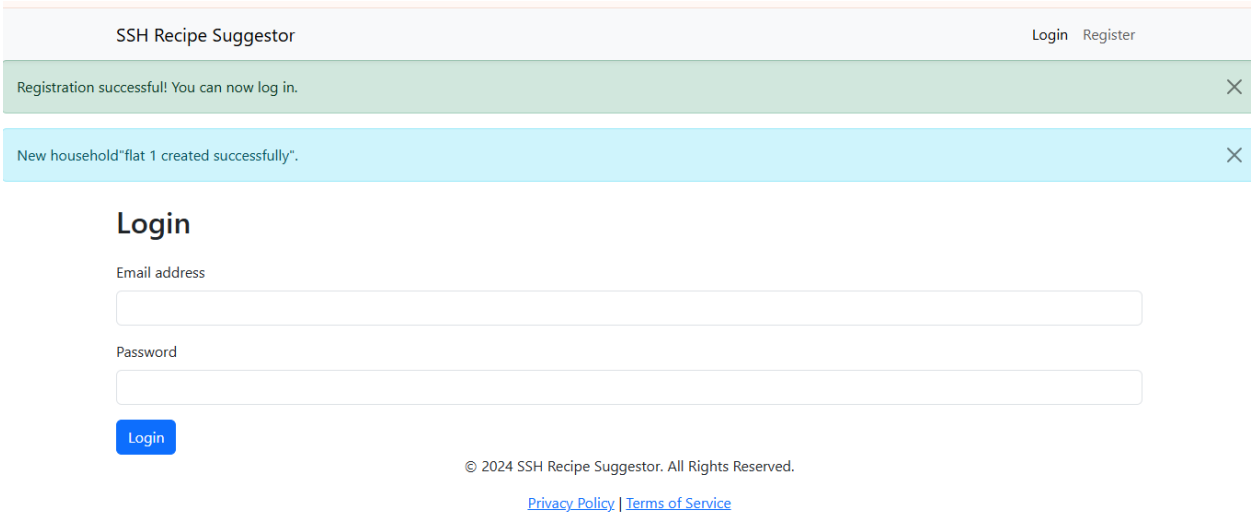
Visiting register route renders the following page:

.

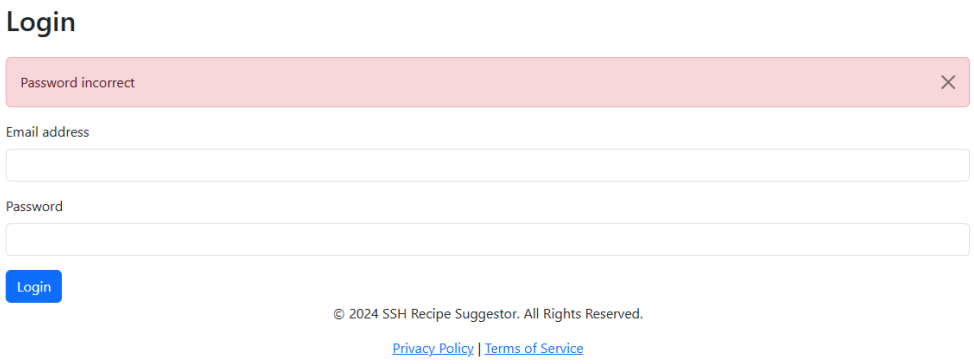
Register route creates a user into the User database with user name, email and household. If a user attempts to register with an email that is already in database records, the following feedback is rendered



Once the user is registered, he is redirected to the login page. Through the dependency of **connect-flash** , flash messages are displayed to the user to give them real time feedback of success or failure messages.

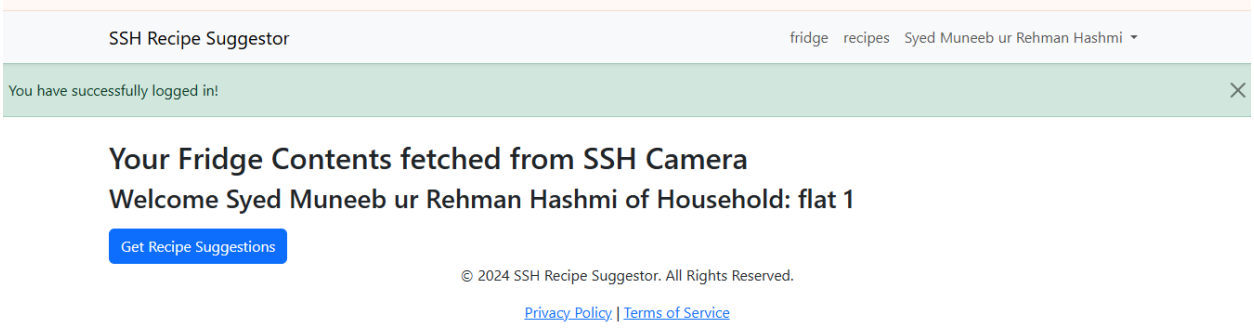


If a user registers with a new household name that isn't already present in database, a new household Is created in register route as well. If he registers with a household name that already exist in the database, a new user will be created in a pre-existing household.



Login route gives feed back when a user attempts to log in with a wrong password.

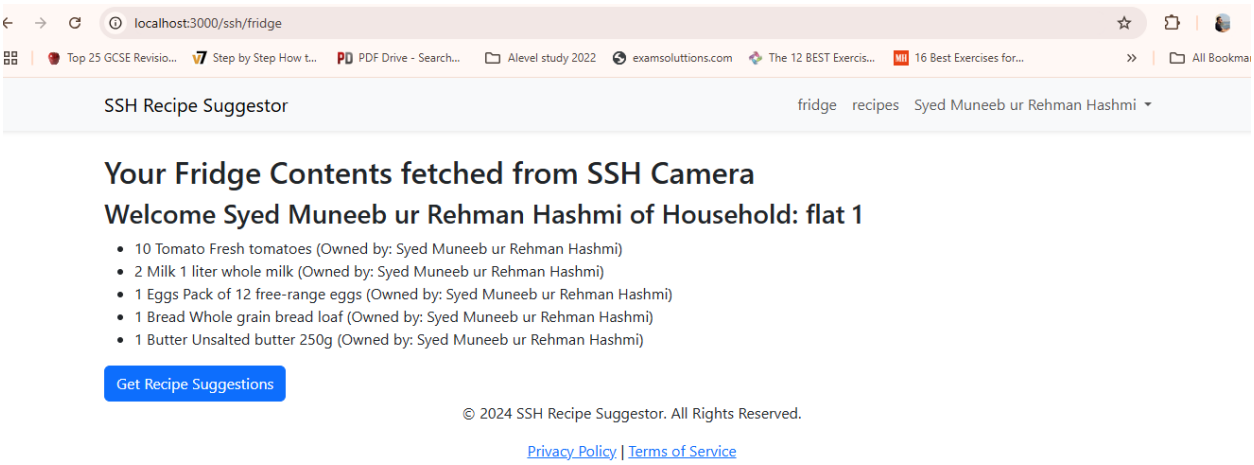
Once the user logs in, he is redirected to the fridge route where the following is rendered.



Currently the Inventory document is empty but it will be seeded through running “node seedfridge.js” in command line. This replicates the functionality that it is SSH camera which fetches the contents of the fridge of the user logged in.

```
C:\Users\Administrator\newprototype>node seedfridge
(node:2596) [MONGODB DRIVER] Warning: useNewUrlParser is a deprecated option: useNewUrlParser has no effect since Node.j
s Driver version 4.0.0 and will be removed in the next major version
(Use `node --trace-warnings ...` to show where the warning was created)
(node:2596) [MONGODB DRIVER] Warning: useUnifiedTopology is a deprecated option: useUnifiedTopology has no effect since
Node.js Driver version 4.0.0 and will be removed in the next major version
Connected to MongoDB for seeding fridge.
Fridge items seeded for the current user.
```

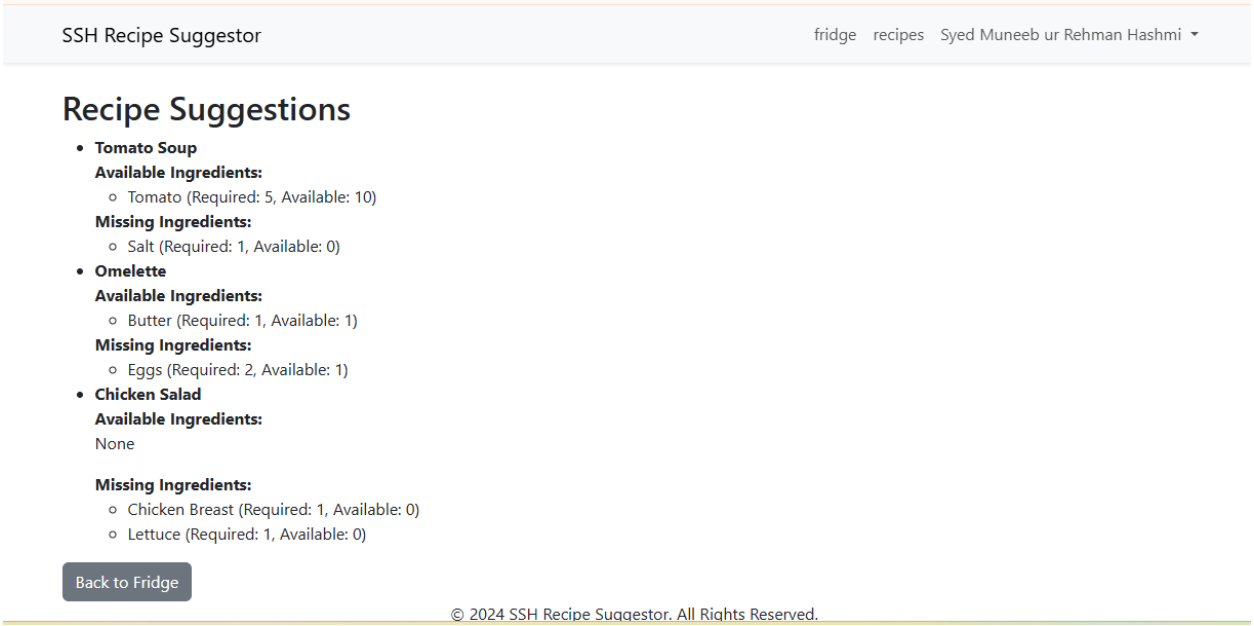
Once the database is seeded, we get to see the following in fridge route.



Hence this page represents the contents of the fridge displayed to the user through the backend job which runs 3 times a day as mentioned in the EDR. But what is happening in the prototype is that in this fridge route, **through session data, it is checked who is logged in and using the user id and household saved in session data by Passport JS , we fetch items from inventory table.**

```
_id: ObjectId('675a2fd9c4eebe6d804c6145')
name : "Syed Muneeb ur Rehman Hashmi"
email : "muneeburrehman501@gmail.com"
password_hash : "$2a$10$argXq8nemaBAGDLmISeCLuKeZDhnoikc8sTyDNvMQEpw/pw5JfVlS"
household : ObjectId('675a2fd9c4eebe6d804c6143')
__v : 0
```


Moreover this screenshot shows that when user is saved in the database, his password is hashed through bycrypt dependency so that if there are any malicious attempt to hack the database of SSH company , the users' sensitive personal information are not threatened.



The following is rendered in the recipes route. Here the user is shown the ingredients he has for each recipe and what items are needed to be purchased. The actual recipes will be fetched from an external API which is out from the scope of this prototype.

Unit test is implemented using **Jest** which checks the functionality implemented inside recipes route is correct. This functionality is separated out in sshservices.js inside services folder so that it could be tested using unit test and what it does is to calculate that for each recipe, how many items are required and missing. The unit test ensures that the function returns the correct expected value. Since unit test is focused on a particular function only that is to check whether it returns expected output or not, so

rather than making a call to mongo db database which isn't related to the working of the function we want to test, the data is mocked.

This is the output of jest unit test.

```
C:\Users\Administrator\newprototype>npm run test

> newprototype@1.0.0 test
> jest _tests_

 PASS  __tests__/ssh.spec.js
  calculateSuggestions
    ✓ should correctly determine available and missing ingredients (3 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.528 s, estimated 1 s
Ran all test suites matching /_tests_/i.
```

E2E tests are written as well to "end to end" test whether whole login and register routes return the expected status so that we know they successfully login and register a user. This is the result of running e2e test script.

```
C:\Users\Administrator\newprototype>npm run test:e2e

> newprototype@1.0.0 test:e2e
> jest e2e

(node:900) [MONGODB DRIVER] Warning: useNewUrlParser is a deprecated option: useNewUrlParser has no effect since Node.js
Driver version 4.0.0 and will be removed in the next major version
(Use `node --trace-warnings ...` to show where the warning was created)
(node:900) [MONGODB DRIVER] Warning: useUnifiedTopology is a deprecated option: useUnifiedTopology has no effect since N
ode.js Driver version 4.0.0 and will be removed in the next major version
  console.log
    Connected to Test Database

      at Object.log (e2e/auth.e2e.spec.js:13:13)

 PASS  e2e/auth.e2e.spec.js
  User Registration and Login
    ✓ should register a user successfully (99 ms)
    ✓ should log in the user successfully (15 ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        2.072 s
```

A CI pipeline is implemented to check whether the code written is in node 16 and whether the unit test pass whenever there is pull request on main branch. A docker image is generated so that the code can be run in any environment using the container.

Project Management and reflections on team :

The initial plan was that the server-side logic and database design will be written by me and my second colleague will handle the responsibility of working on front end using EJS templates as well as styling it using bootstrap. Another colleague would have been responsible to write unit and E2E tests and implementing CI pipeline.

But my colleagues didn't respond to my emails and thus my attempts to reach out to them proved to be futile. As a good colleague of theirs, I tried whatever was in my capacity to give my colleagues a chance to contribute to this project in whatever way possible. Therefore, in order to be a good colleague I assumed the responsibility of the project.

Group projects can pose all kinds of unexpected situations. In my case it was the absence of my colleagues in the project we were supposed to work on together. In face of this uncertain situation I found myself in, rather than giving up, I displayed the necessary headship and responsibility skills to take this prototype to its final form.

To conclude, this prototype made an attempt that the features proposed in the EDR are viable and hence the risks associated with the full design are minimal and SSH should proceed with it.