IN PARTNERSHIP WITH PLYMOUTH UNIVERSITY

Name: I U D Gunaratne
Student Reference Number: 10749830

Module Code: PUSL3120	Module Name: Full-Stack Development (22/AU/M)
Coursework Title: FULL-STACK DEVELOPMENT	
Deadline Date:1st Feb 2023	Member of staff responsible for coursework: Dr Mark Dixon
Programme: BSc (Hons) Computer Science	
Please note that University Academic Regulations are available under Rules and Regulations on the University website www.plymouth.ac.uk/studenthandbook.	
Individual assignment: ***I confirm that I have read and understood the Plymouth University regulations relating to Assessment Offences and that I am aware of the possible penalties for any breach of these regulations. I confirm that this is my own independent work.*** Signed : Ishani Gunarathne	
Use of translation software: failure to declare that translation software or a similar writing aid has been used will be treated as an assessment offence. I *have used/not used translation software. If used, please state name of software..	
Overall mark _____% Assessors Initials ______ Date_________	

Contents

Chapter 1-Introduction and Requirements

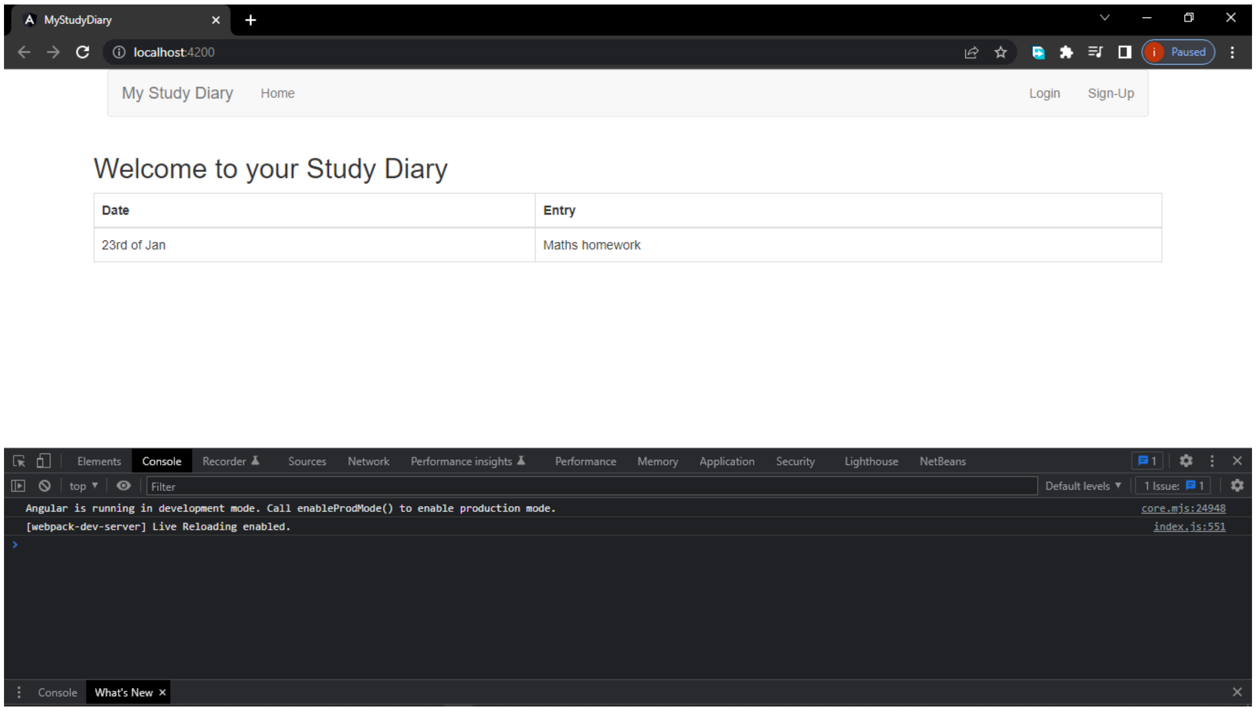
This is a MEAN stack single page web application to be used by students who wants to keep track of their daily study tasks. The entire code, from client to server-side is written in JavaScript. AngularJS is being used for developing the front-end of the application. AngularJS is an open source framework to build dynamic, single page web applications using JavaScript. As for the back-end, an Express.js is running on a Node.js server to power the server-side. NodeJS is a cross-platform runtime environment server to support executing JavaScript back-end whereas ExpressJS is a framework of NodeJS which supports to manage servers and routes while supporting the even-driven architecture of NodeJS. MongoDB Atlas which is the MongoDB Cloud is used for the storing of the data. The Express.js server can process JSON documents written in the Angular.js front end and store the user profiles and other content directly in MongoDB for subsequent retrieval.

Who is the application aimed at?

The application can be used for study progress tracking where it will act like a digital diary. This can be used as a study planner as well to improve the productivity. In most cases, students do not have a clear understanding on what they spent their time on while studying. Documenting what you do during a specific day of studying will give students a better grasp of their everyday study habits.

Features

Below is a screenshot of the homepage of the application.



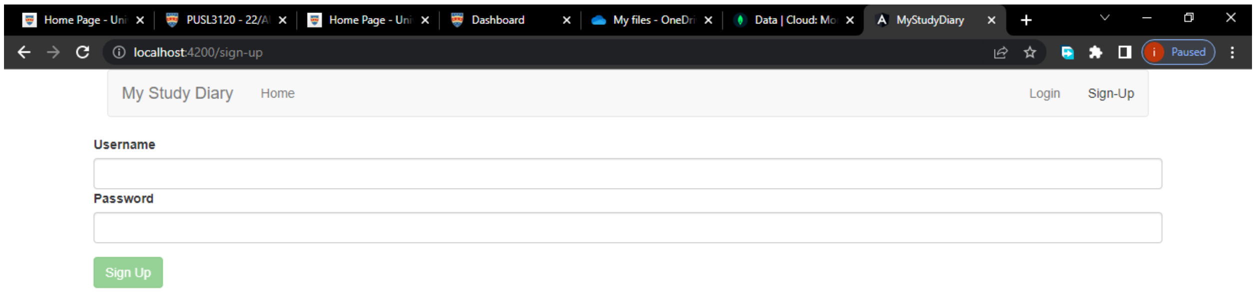
Welcome to your Study Diary

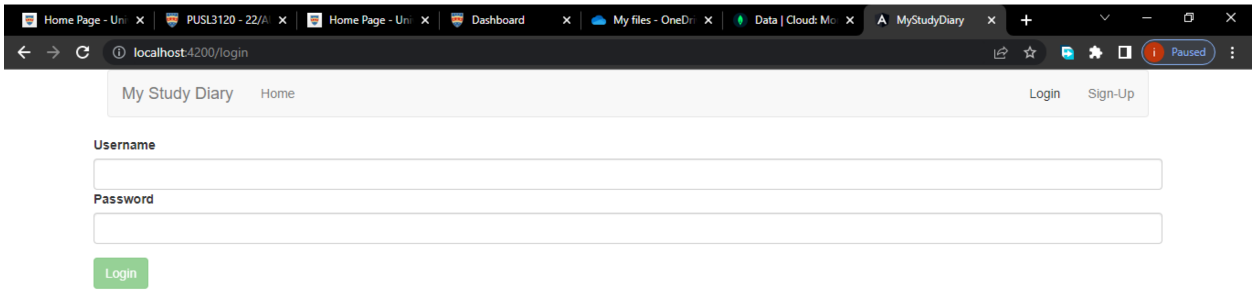
Date	Entry
23rd of Jan	Maths homework

```
Angular is running in development mode. Call enableProdMode() to enable production mode.    core.mjs:24948
[webpack-dev-server] Live Reloading enabled.                                                 index.js:551
```

Signup and Login pages

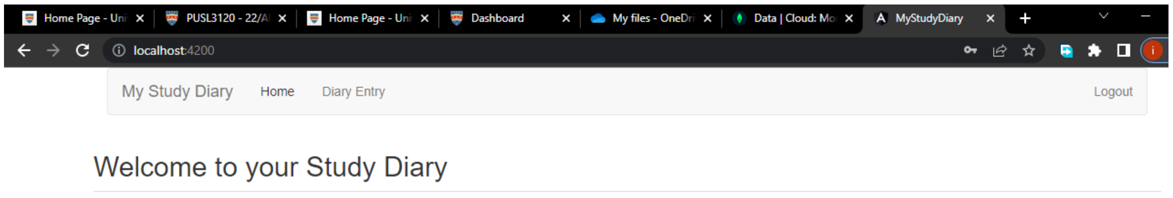
Users have to sign up to the system and then log in in order to be able to use the application. This feature is used for the purpose of submitting new users to the MongoDB when they sign in and authentication of the users with MongoDB when they log in. The users are being identified by the username and the password. The application cannot be accessed in the event where a user is incapable of providing the accurate credentials mentioned above. After signing up the users into the database, the access privileges of the signed up users are being elevated with the use of **web tokens and web sockets**.





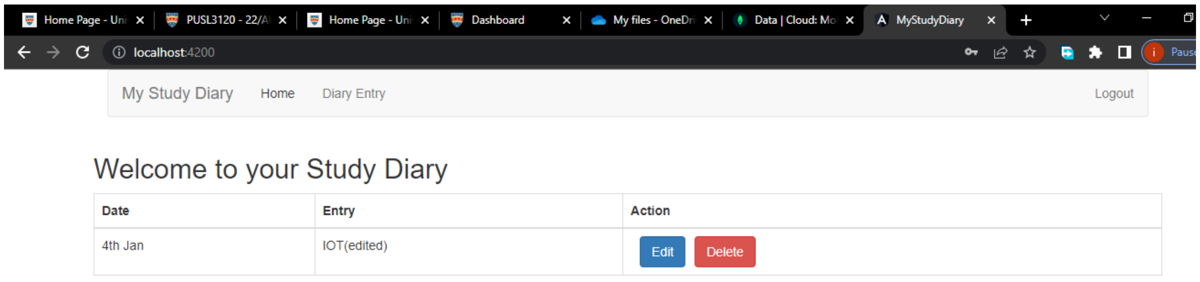
Header

Header component is to navigate the tabs.



Home Tab

Home tab contains all the entries that are being fed to the database by the user. This is where the users can see and check their progress of the work.

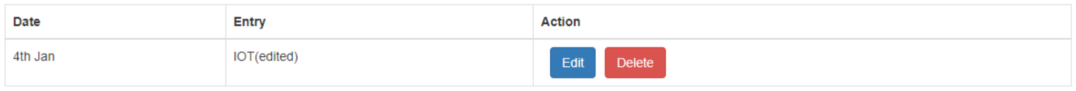


Welcome to your Study Diary

Date	Entry	Action
4th Jan	IOT(edited)	Edit Delete

Edit and Delete buttons

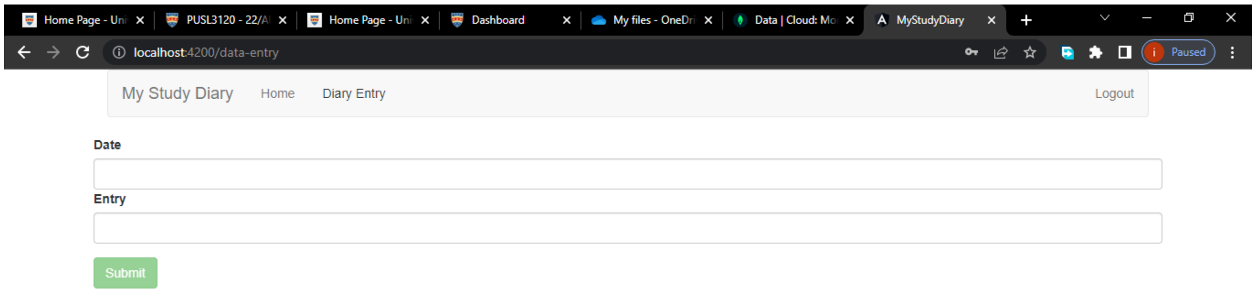
The ability to edit or delete the entries has been enabled by using two buttons.



Date	Entry	Action
4th Jan	IOT(edited)	Edit Delete

Daily entry

This is where the users can keep record of their study progress entry. The progress entries have two components to it which are the date and the description of the entry.



Chapter 02-Design

System Architecture

MEAN technology’s basic architecture can be described as when client make a request, the request is handled by AngularJS. The client side requests then processed to enter the server which is NodeJS. This stage is known as parsing the client-side request. NodeJS and ExpressJS work together to make the request to the MongoDB database through Mongoose. MongoDB will retrieve the relevant data and return it to ExpressJS and then to NodeJS. NodeJS will return the values to AngularJS to be displayed as the result. Model View Controller Architecture (MCV) is supported by MEAN stack technology.

There are three tiers to this application which are front-end, back-end and data tier.

Front-end

Angular applications are typically built with hard-coded JSON file for angular to know when a component is created. The process when a user starts to interact with the application is, angular uses the routing modules to checks the necessary paths of what the user enters. Then the respective component for the user request will be called. Then, service packages are called to perform the task according to the request. The request will be sent to the server-side in the form of JSON format.
When Client-side server gets the response from the server-side, service.ts will get that data and send it back to the controller and then to the view components. View/template is rendered with necessary components and user can see the response in the user interface.

RESTful API

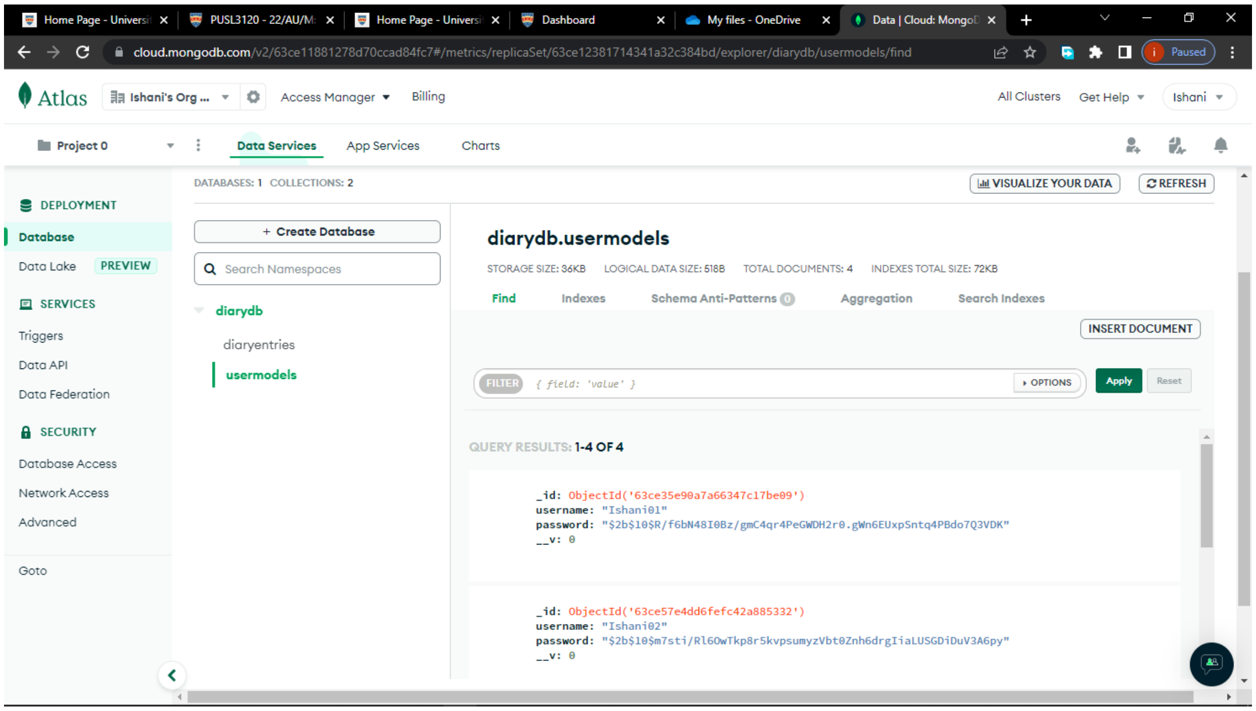
Representational state transfer is what was being used to act as the mediator between the front-end server and the back-end server. RESTful API transfers the users request to the end-point which is NodeJS server in the form of HTTP: JSON.

Back-end

When the request make to the back-end, the router which is an Express code which includes all the URL paths and router methods. When the back-end knows which path that was entered (RESTful endpoint), server will locate the controllers and execute the callback functions to get the data from the database. Now the back-end has all the necessary data hence node server can send back a response using JSON format.

Data Tier

MongoDB Atlas which is the MongoDB run in the cloud is what has been used as the data tier. Using Atlas with just few clicks, you can quickly deploy and scale a MongoDB cluster in the cloud.
Below are screenshots of the database.

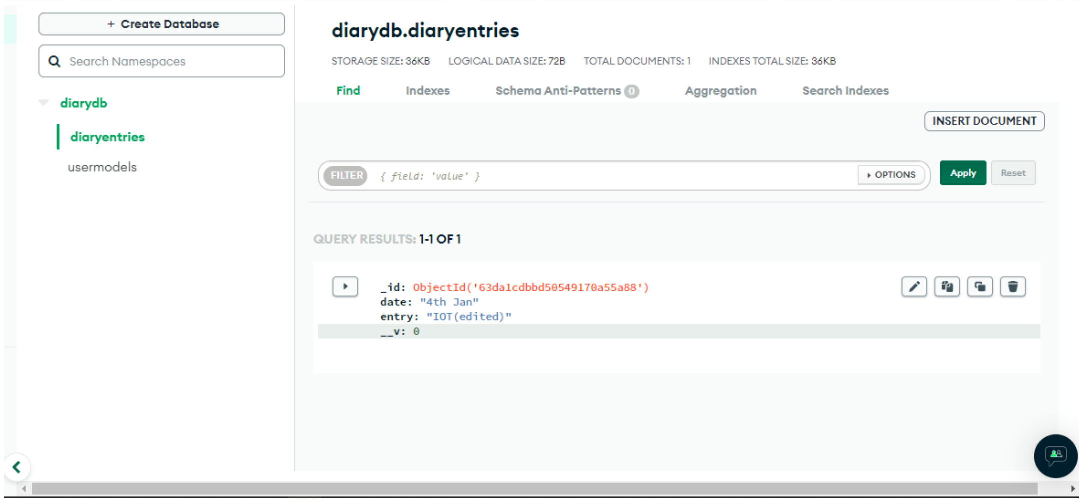


diarydb.usermodels

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 518B TOTAL DOCUMENTS: 4 INDEXES TOTAL SIZE: 72KB

QUERY RESULTS: 1-4 OF 4

```
_id: ObjectId('63ce35e90a7a66347c17be09')
username: "Ishani01"
password: "$2b$10$R/f6bN48I0Bz/gmC4qr4PeGWDH2r0.gWn6EUxpSntq4PBdo7Q3VDK"
__v: 0

_id: ObjectId('63ce57e4dd6fefc42a885332')
username: "Ishani02"
password: "$2b$10$m7sti/Rl6OwTkp8r5kvpsumyzVbt0Znh6drgIiaLUSGDiDuV3A6py"
__v: 0
```



diarydb.diaryentries

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 72B TOTAL DOCUMENTS: 1 INDEXES TOTAL SIZE: 36KB

QUERY RESULTS: 1-1 OF 1

```
_id: ObjectId('63da1cdbbd50549170a55a88')
date: "4th Jan"
entry: "IOT(edited)"
__v: 0
```

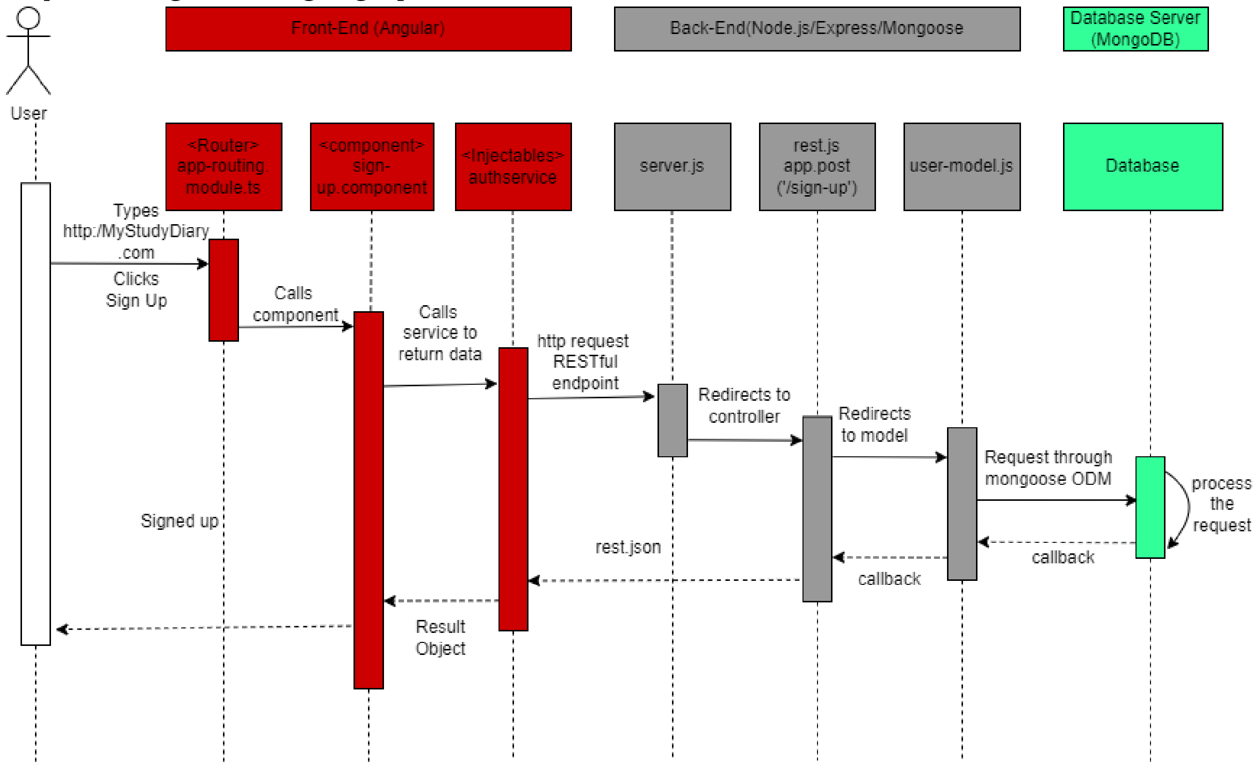
WebSockets

WebSockets are typically used for standard real-time communication. In the sign-up and login functionalities, the WebSockets and JSON web tokens are being used to authenticate users. To enable this, a WebSocket server was created and HTTP/HTTPS server with two endpoints was created. When the login credentials are submitted, frontend will fetch web token and initiate the WebSocket connection.

The MEAN stack offers increased scalability and user management, making it the ideal solution for creating cloud-native apps. Creating single-page applications (SPAs) that display all information and features on a single page would be another excellent use of the AngularJS front-end framework. When it comes to transferring code to different frameworks, MEAN Stack is incredibly adaptable. This can be used to create a strong framework that can sustain daily needs. MEAN stack provides a structured framework that is compatible with MVC architecture. The MVC design is supported by MERN, however only MEAN makes it simpler to handle the coding and upgrade.

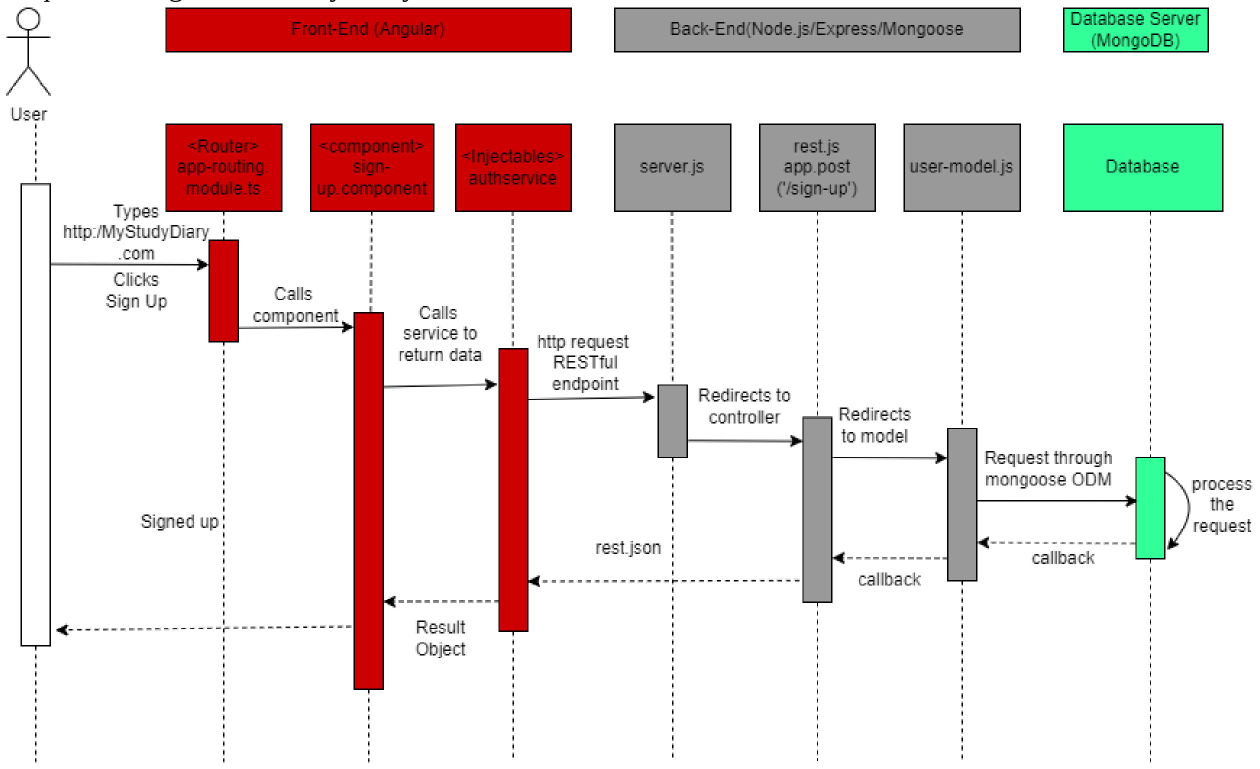
UML Diagrams

Sequence Diagram for Signing Up



User
Front-End (Angular)
Back-End(Node.js/Express/Mongoose
Database Server (MongoDB)
<Router> app-routing.module.ts
<component> sign-up.component
<Injectables> authservice
server.js
rest.js app.post ('/sign-up')
user-model.js
Database
Types http:/MyStudyDiary.com
Clicks Sign Up
Calls component
Calls service to return data
http request RESTful endpoint
Redirects to controller
Redirects to model
Request through mongoose ODM
process the request
callback
callback
rest.json
Signed up
Result Object

Sequence Diagram for Diary Entry


Front-End (Angular)
Back-End(Node.js/Express/Mongoose
Database Server (MongoDB)
User
<Router> app-routing.module.ts
<component> sign-up.component
<Injectables> authservice
server.js
rest.js app.post ('/sign-up')
user-model.js
Database
Types http:/MyStudyDiary.com
Clicks Sign Up
Calls component
Calls service to return data
http request RESTful endpoint
Redirects to controller
Redirects to model
Request through mongoose ODM
process the request
callback
callback
rest.json
Signed up
Result Object

Chapter 3-Testing

Front-end testing

Unit tests (Angular TestBed and Fixtures)

Unit tests focus on each component's capability and behavior without considering the behavior of the application as a whole. The angular unit testing is done by using the Angular testing utility called the **TestBed** which allows to test functionalities depends on the Angular framework. This automatically generate a testing environment for the classes that needed to be unit tested by creating Angular testing modules for those classes. The process creates classes to mock with its dependencies.

The reason why Angular TestBed and Fixtures were used in the testing process is that this is an integral part of Angular development. The feature ensures to test the application function by function almost effortlessly.

Jasmine testing framework

The angular CLI comes with the Jasmine testing framework. The test runner **Karma** was used to run tests on browser. This is a fully automated task runner for testing. Karma creates the tests, open browser to run them and return results to the command line interface.

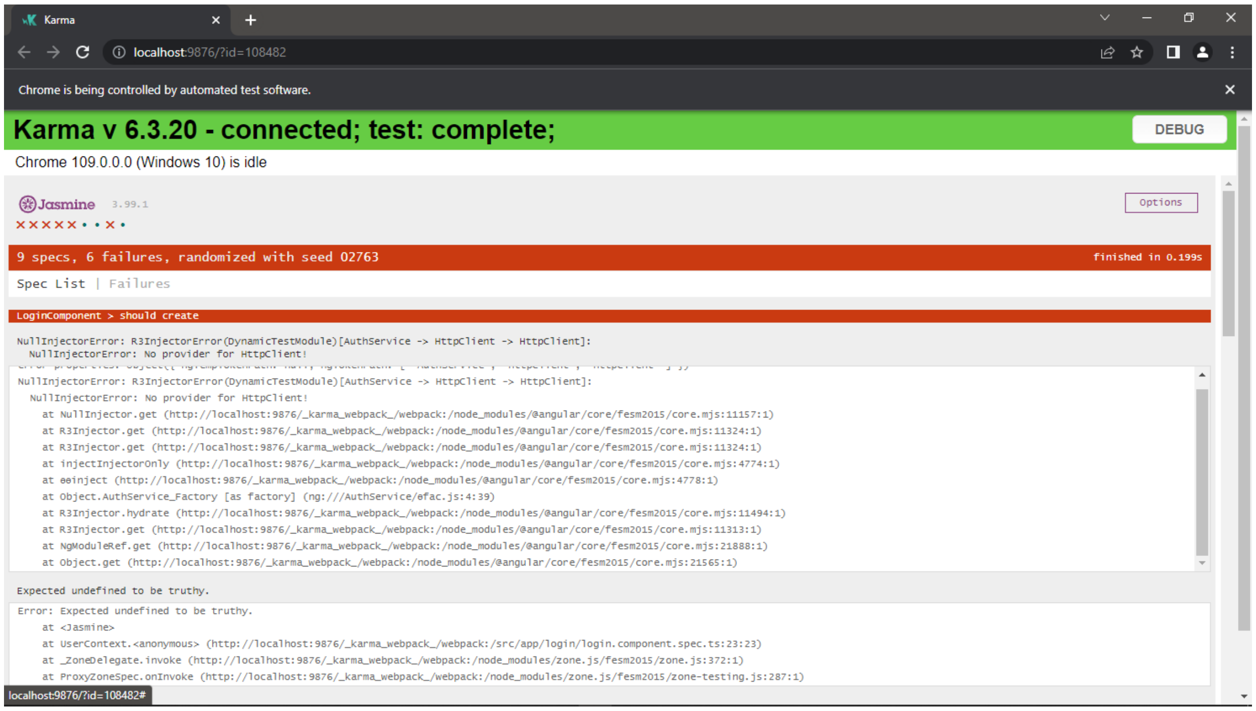
Karma can be run in a browser which is the same exact environment and conditions that when a client executes the application. This is the main reason why Karma is used for the testing purposes.

Back-end

Mocha and Chai

Mocha.js is a testing framework to be used to test the server-side of the application. The framework runs on the node.js and the browser as well. The expected output of a method and the output given by running the tests can be compared by using the Chai library.

Mocha is a framework that is highly customizable. The developers has the freedom to decide which libraries they are going to use. Chai is an assertion library that can be used for including assertion into the tests to be run against the code.



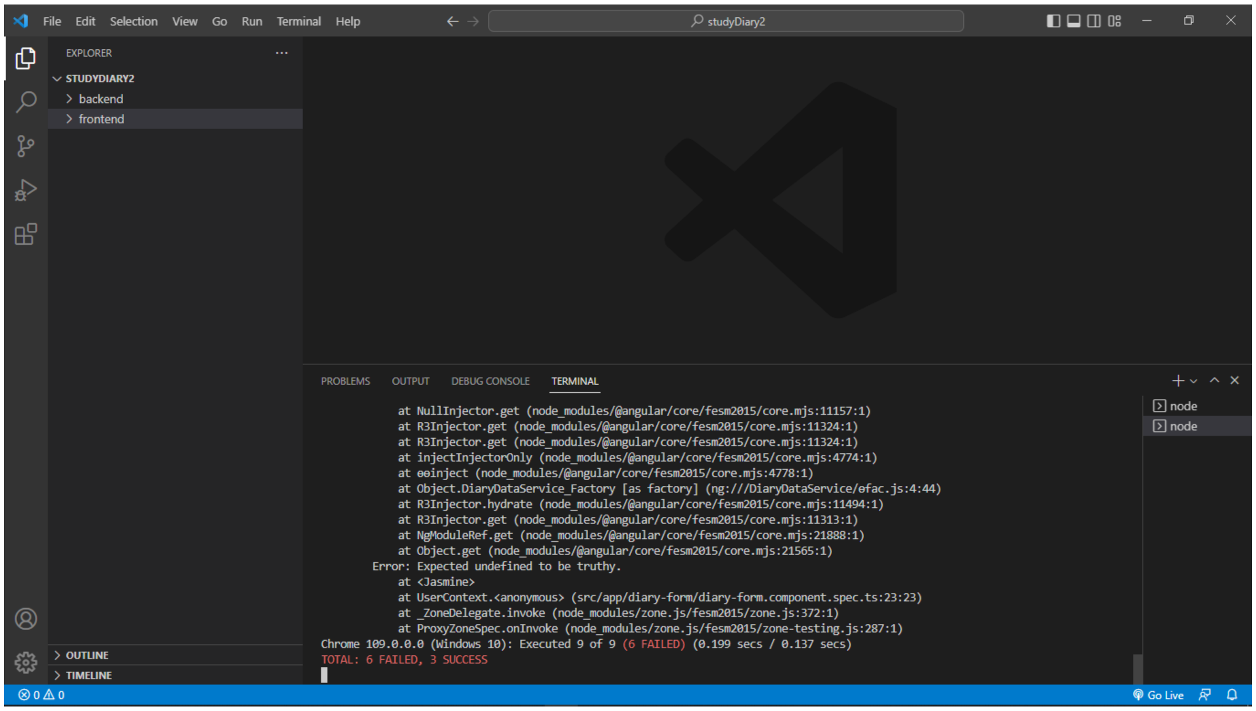
Karma

localhost:9876/?id=108482

Chrome is being controlled by automated test software.

```
Karma v 6.3.20 - connected; test: complete;
Chrome 109.0.0.0 (Windows 10) is idle

Jasmine 3.99.1
×××××••×•

9 specs, 6 failures, randomized with seed 02763                    finished in 0.199s
Spec List | Failures

LoginComponent > should create

NullInjectorError: R3InjectorError(DynamicTestModule)[AuthService -> HttpClient -> HttpClient]:
  NullInjectorError: No provider for HttpClient!

NullInjectorError: R3InjectorError(DynamicTestModule)[AuthService -> HttpClient -> HttpClient]:
  NullInjectorError: No provider for HttpClient!
    at NullInjector.get (http://localhost:9876/_karma_webpack_/webpack:/node_modules/@angular/core/fesm2015/core.mjs:11157:1)
    at R3Injector.get (http://localhost:9876/_karma_webpack_/webpack:/node_modules/@angular/core/fesm2015/core.mjs:11324:1)
    at R3Injector.get (http://localhost:9876/_karma_webpack_/webpack:/node_modules/@angular/core/fesm2015/core.mjs:11324:1)
    at injectInjectorOnly (http://localhost:9876/_karma_webpack_/webpack:/node_modules/@angular/core/fesm2015/core.mjs:4774:1)
    at ɵɵinject (http://localhost:9876/_karma_webpack_/webpack:/node_modules/@angular/core/fesm2015/core.mjs:4778:1)
    at Object.AuthService_Factory [as factory] (ng:///AuthService/ɵfac.js:4:39)
    at R3Injector.hydrate (http://localhost:9876/_karma_webpack_/webpack:/node_modules/@angular/core/fesm2015/core.mjs:11494:1)
    at R3Injector.get (http://localhost:9876/_karma_webpack_/webpack:/node_modules/@angular/core/fesm2015/core.mjs:11313:1)
    at NgModuleRef.get (http://localhost:9876/_karma_webpack_/webpack:/node_modules/@angular/core/fesm2015/core.mjs:21888:1)
    at Object.get (http://localhost:9876/_karma_webpack_/webpack:/node_modules/@angular/core/fesm2015/core.mjs:21565:1)

Expected undefined to be truthy.
Error: Expected undefined to be truthy.
    at <Jasmine>
    at UserContext.<anonymous> (http://localhost:9876/_karma_webpack_/webpack:/src/app/login/login.component.spec.ts:23:23)
    at _ZoneDelegate.invoke (http://localhost:9876/_karma_webpack_/webpack:/node_modules/zone.js/fesm2015/zone.js:372:1)
    at ProxyZoneSpec.onInvoke (http://localhost:9876/_karma_webpack_/webpack:/node_modules/zone.js/fesm2015/zone-testing.js:287:1)
```

DEBUG
Options
localhost:9876/?id=108482#



File Edit Selection View Go Run Terminal Help
studyDiary2
EXPLORER
STUDYDIARY2
backend
frontend
OUTLINE
TIMELINE
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
node
node
0 0
Go Live

```
            at NullInjector.get (node_modules/@angular/core/fesm2015/core.mjs:11157:1)
            at R3Injector.get (node_modules/@angular/core/fesm2015/core.mjs:11324:1)
            at R3Injector.get (node_modules/@angular/core/fesm2015/core.mjs:11324:1)
            at injectInjectorOnly (node_modules/@angular/core/fesm2015/core.mjs:4774:1)
            at ɵɵinject (node_modules/@angular/core/fesm2015/core.mjs:4778:1)
            at Object.DiaryDataService_Factory [as factory] (ng:///DiaryDataService/ɵfac.js:4:44)
            at R3Injector.hydrate (node_modules/@angular/core/fesm2015/core.mjs:11494:1)
            at R3Injector.get (node_modules/@angular/core/fesm2015/core.mjs:11313:1)
            at NgModuleRef.get (node_modules/@angular/core/fesm2015/core.mjs:21888:1)
            at Object.get (node_modules/@angular/core/fesm2015/core.mjs:21565:1)
        Error: Expected undefined to be truthy.
            at <Jasmine>
            at UserContext.<anonymous> (src/app/diary-form/diary-form.component.spec.ts:23:23)
            at _ZoneDelegate.invoke (node_modules/zone.js/fesm2015/zone.js:372:1)
            at ProxyZoneSpec.onInvoke (node_modules/zone.js/fesm2015/zone-testing.js:287:1)
Chrome 109.0.0.0 (Windows 10): Executed 9 of 9 (6 FAILED) (0.199 secs / 0.137 secs)
TOTAL: 6 FAILED, 3 SUCCESS
```

Chapter 04-DevOps pipeline

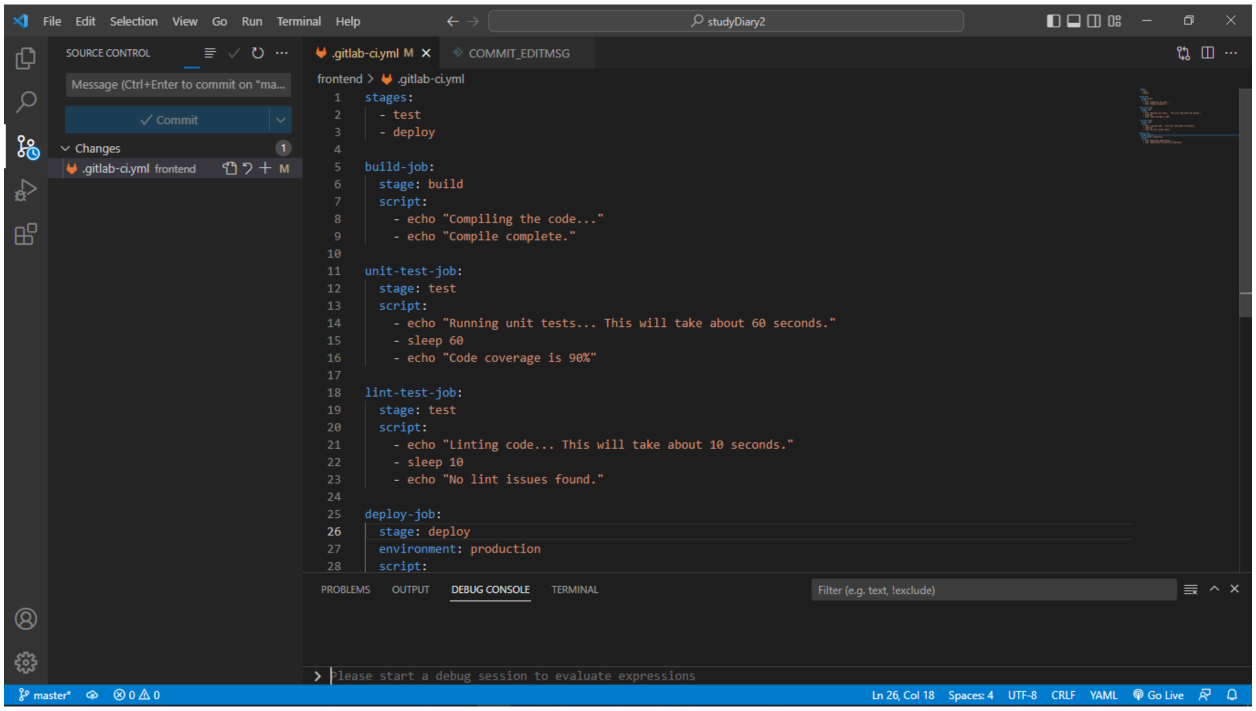
CI/CD (continuous integration and continuous deployment) pipelines supposed to make it possible for developer teams to provide smaller releases in less time by reducing manual errors and improving feedback loops across the SDLC. A successful CI/CD strategy is an effective way to hasten deployment and increase the value of each release for users.
Github action is used for automating developer workflow. CI/CD is one of the workflows which can be used in automating the workflow. You can automate your build, test, and deployment workflow with GitHub Actions, a platform for continuous integration and delivery (CI/CD).
Basic steps for working of the CI/CD pipeline is committing the code, building, testing, building the tested code, pushing and deploying.

How the CI/CD pipeline with GitHub Actions was used

As the first step, a github repository was created and setup and configured the workflow. YAML file for the workflow was created. The events that triggers the workflow were created. The required jobs and actions were created.

gitlab.yml file

gitlab.yml file is the file wherein you specify GitLab CI/CD instructions including the organization and sequencing of the jobs the runner should run and the choices that should be made by the runner under various circumstances. The jobs you specified in the gitlab-ci.yml file are started and executed by the CI/CD pipeline.


File Edit Selection View Go Run Terminal Help
studyDiary2
SOURCE CONTROL
Message (Ctrl+Enter to commit on "ma...
Commit
Changes 1
.gitlab-ci.yml frontend M
.gitlab-ci.yml M
COMMIT_EDITMSG
frontend > .gitlab-ci.yml
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Filter (e.g. text, !exclude)
master* 0 0
Ln 26, Col 18 Spaces: 4 UTF-8 CRLF YAML Go Live

```yaml
stages:
  - test
  - deploy

build-job:
  stage: build
  script:
    - echo "Compiling the code..."
    - echo "Compile complete."

unit-test-job:
  stage: test
  script:
    - echo "Running unit tests... This will take about 60 seconds."
    - sleep 60
    - echo "Code coverage is 90%"

lint-test-job:
  stage: test
  script:
    - echo "Linting code... This will take about 10 seconds."
    - sleep 10
    - echo "No lint issues found."

deploy-job:
  stage: deploy
  environment: production
  script:
```

```
> Please start a debug session to evaluate expressions
```

Chapter 05-Personal reflection

The biggest issue I faced during the implementation is that the CI/CD pipeline failure. The issue raised when CI pipelines were run on the master branch but some tests were not passed. The project's home page displayed a red cross, indicating unstable source code and undermining the trust of users. This caused an error in continuous deployment/delivery stream line.
After doing some research, failure was seemed to be a security vulnerability or most likely a merge conflict. I have tried several methods of troubleshooting. Testing and debugging every section instead of rerunning the full build was one of them and by doing this, I was convinced that the issue was not in my code, but the pipeline itself.

Other biggest challenge I had to face during the time of developing the project was lack of community support and tutorials on internet when it comes to the newer, latest versions of tools I choose to work with. Most of the community support I found was old and was not updated. It was hard to find relatable questions and answers for some particular issues I faced.

Choosing the correct npm modules for the task was another challenging work. There are a lot of npm modules out there to choose from. I always tried to go for the packages with larger developer communities.

After building the whole project and after struggling and failing to understand what caused all the errors, I have understood that one must have a thorough understanding of the inner workings of the application to swiftly and effectively debug any mistake. Before starting a new MEAN stack project, I will learn more about HTTP requests and the back-and-forth transfer of data to get a clear understanding of how clients and servers interact with each other. I have to learn more about how to stop pipelines for merged results from having broken masters. Finding the best tool for the work will be easier if I can be aware of the benefits and drawbacks of each npm modules I think of using.

Chapter 06-References

Aggarwal, S. and Verma, J., 2018. Comparative analysis of MEAN stack and MERN stack. International Journal of Recent Research Aspects, 5(1), pp.127-132.

W3schools.com. (2019). What is Full Stack. [online] Available at: https://www.w3schools.com/whatis/whatis_fullstack.asp.

www.javatpoint.com. (n.d.). Mean Stack Tutorial | Mean.js Tutorial - javatpoint. [online] Available at: https://www.javatpoint.com/mean-stack-tutorial.

Link to the YouTube video

https://youtu.be/balzzRGuewk