Task:

Firstly create a folder mern-app and then create a client folder which has a frontend and server folder which has a backend code in it.

In client folder
Now go to cd client and do the following
Initially created the react app by using the following command

```
npx create-react-app <AppName>
```

Then the below command is used to run the app.

```
npm start
```

Did the following installations in order to get material ui which is used for front end, axios which is used for api calling and react-router-dom.

```
npm install @mui/material @emotion/react @emotion/styled
npm i axios
```

In Server folder
Initialize npm using `npm init`

Do the following installations in order to get express, mongoose dotenv and so on.

```
npm i express
npm i mongoose
npm i nodemon - no need to restart server
npm i dotenv- to store all configurations
npm i cors
npm i body-parser
npm i mongoose-auto-increment
```

The below command helps to start the backend application

```
nodemon start
```

Flow of the project

Register a user , then it redirects to the login page and then if successful it redirects to the dashboard.
From there you can access the website by clicking add user or All users on navbar in order to do crud operations.
The session logs out if the token expires.

When cloned from github
Cloning from GitHub:
Firstly, clone the repository to your local machine using the Git command.

Client-side setup:
Open a terminal and navigate to the "client" directory using cd client.
Run npm install to install the client-side dependencies.
Run npm start to start the development server for the React app.

Server-side setup:
Open another terminal and navigate to the "server" directory using cd server.
Run npm install --force to install the server-side dependencies. The --force flag is used to force a clean installation of packages.
Run nodemon start (or npm start, depending on how you configured the start script) to start the backend server.

When you are using without docker change the mongodb configuration according to the mongodb atlas created and also change IP address to localhost which is used in the client/service/api.js to call the API from the backend server.

Docker setup:
Before using Docker, ensure that Docker is installed on our machine. We can check this by running docker --version in the terminal.
Docker Compose:
In the root directory (where your docker-compose.yml file is located), run docker-compose up --build to build and start the Docker containers for the client, server, and other services like mongo in our case which is defined in the docker-compose.yml file.
It's important to note that if we have Docker files for the client and server already in the respective directories, then we can directly use docker-compose up --build without the need to install dependencies separately using npm install inside the client and server directories. Docker will handle the dependency installation and containerization process.

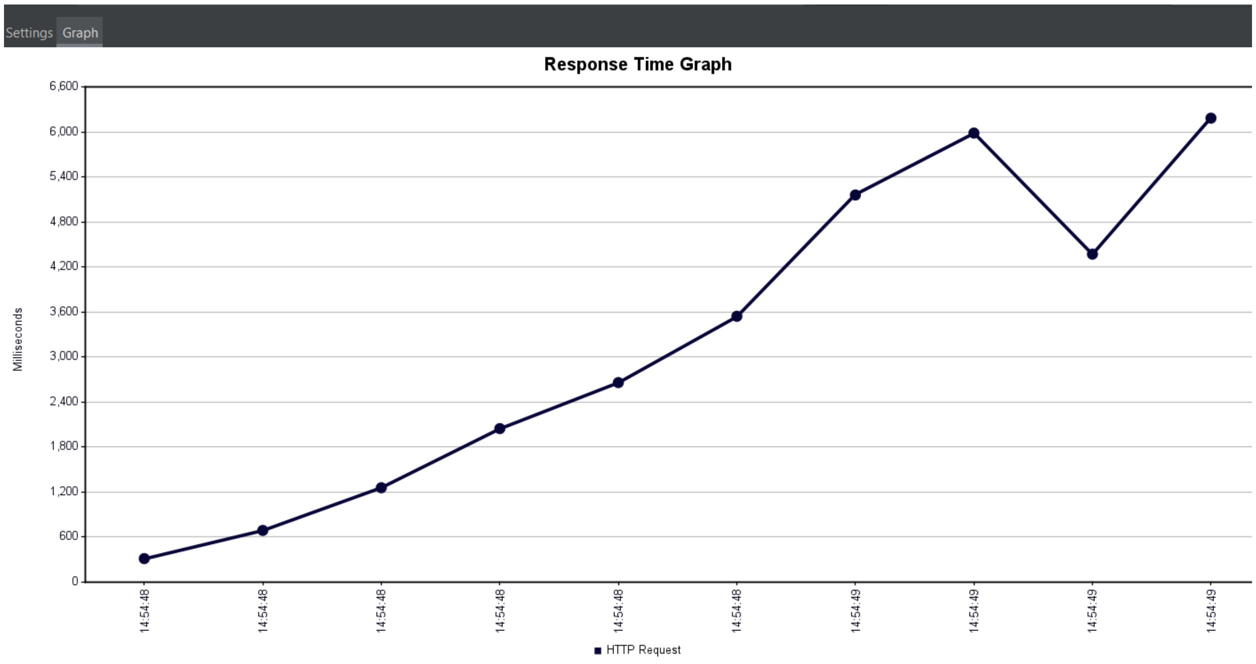
In order to run the backend server when docker is used, use the IP address of your local machine and change IPaddress in client/service/api.js accordingly.
When using Docker to containerize the MERN application, the backend server is running inside a Docker container. By default, the backend server inside the container is not accessible via localhost from the host machine (your local machine). Therefore, you need to use the IP address of your local machine to access the backend server running in the Docker container.
On Windows, open Command Prompt and type ipconfig. Look for the IPv4 address under your network adapter (e.g., Ethernet adapter or Wi-Fi adapter).

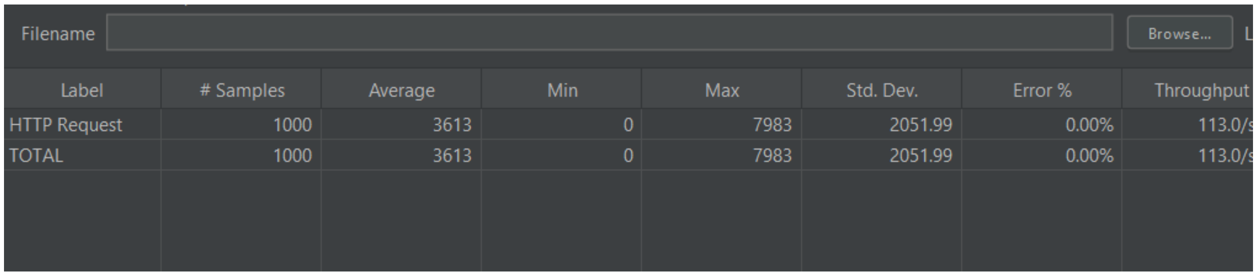
Bonus:
As JWT provides a stateless and secure way to handle user authentication in web applications it is implemented for authentication in this application.

Docker is used to containerize the above application which can be further used for deployment and scaling.

Load testing is done using JMeter for the developed Docker image in the development environment: As the Load testing with JMeter helps evaluate how well your application performs under various levels of concurrent user requests and transactions. By conducting the load testing on the Dockerized application in the development environment, we can gain insights into its performance characteristics, identify potential performance bottlenecks, and optimize our application accordingly.

If the application gets hit with 1000 users within 1 sec then the response time graph and summary report are as follows in Jmeter





Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput
HTTP Request	1000	3613	0	7983	2051.99	0.00%	113.0/s
TOTAL	1000	3613	0	7983	2051.99	0.00%	113.0/s

This indicates that average response time is 3613ms. So we can balance the load by using 3 or 4 replicas of the server, so that the response time would be reduced. As we can see that when the number of requests are low for a server then the response time is low.

As dockerization for the mern app is done, we can further move on to docker swarm or kubernates in order to do load balancing by creating replicas which will help to improve the response time and handle high traffic more efficiently.

Future steps:

- To handle large amounts of data we can use scalable databases like mongo db which has shared clusters or distributed NOSQL.
- We can also implement caching using reddis to reduce the load on the database and improve response time.
- Implementation of load balancing and horizontal scaling for backend servers to distribute incoming requests using docker swarm or kubernates.
- We can also use CDN’s (content delivery networks) to serve static assets this reduces load on servers.
- Continuous monitoring and testing the performance of the app to identify any issues or response time and optimize the app’s performance.