

Complaint Registering System

A Project Report

Compiled By :

Kulvir Singh

2nd Year Student

BTech – Computer Science and Engineering (Core)

School of Computer Science and Engineering



Under the guidance of

Mr. Rahul Nair

Tata Steel, Jamshedpur

TATA STEEL

Introduction

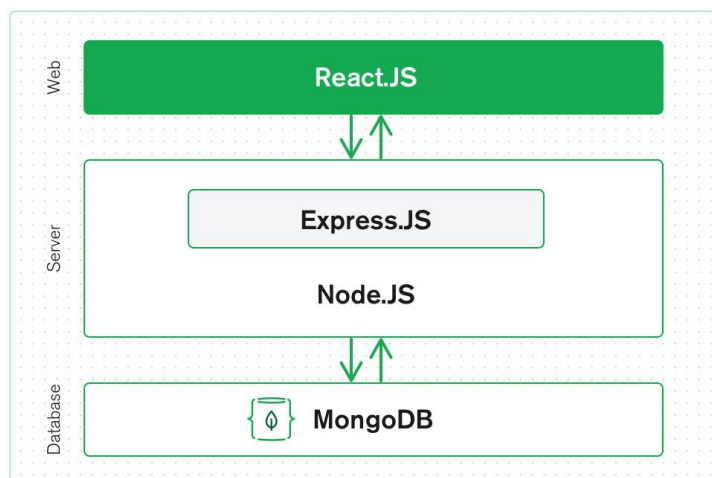
The project work deals with the implementation of a complaint registering system as a web application. The aim of the project is to develop a user friendly and easily interactable web based portal to log a complaint against any vehicle/transporter who is not abiding by the rules and guidelines of the company. It also includes the feature of viewing a complaint and its respective details along with its resolved status. The portal is supported by an admin feature which enables certain authorized users to interact and edit the status of the logged complaint. The main objective of the project was to develop a functioning website which has all the aforementioned features to make the complaint logging system digitalized and hassle free. This project also speeds up the complaint logging and complaint resolving process and also ensures a certain level of data and user security.

Development Method

The tech stack that has been used to develop this project is –

MERN Stack + Firebase

(MongoDB, ExpressJS, ReactJS, NodeJS)



MongoDB : MongoDB is an open-source, cross-platform and document-oriented database. It saves data using collections and documents and then stores them in binary JSON format. Also, it permits the fast exchange of data between client and server. The database is also used for the storage of massive data, that is highly scalable. This non-relational database or a document based database can be hosted online int cloud with the help of MongoDB ATLAS

ExpressJS : Express.js, or simply Express, is a back end web application framework for Node.js, released as free and open-source software under the MIT License. It is designed for building web applications and APIs. It has been called the de facto standard server framework for Node.js.

ReactJS : React (also known as React.js or ReactJS) is a free and open-source front-end JavaScript library for building user interfaces or UI components. It is maintained by Facebook and a community of individual developers and companies. React can be used as a base in the development of single-page or mobile applications. However, React is only concerned with state management and rendering that state to the DOM, so creating React applications usually requires the use of additional libraries for routing, as well as certain client-side functionality.

NodeJS : Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser.

Firebase : Firebase is a platform developed by Google for creating mobile and web applications. The application uses the Cloud Firestore feature of Firebase. It is an online cloud platform which can store any file and provide a URL for the same for viewing it externally and not from the Firebase console.

Why MERN stack?

The main reason behind choosing the MERN stack to develop this application is that it is one of the most commonly used tech stack for developing a website and hence makes the future scope and developments of this project relatively easy. Apart from that –

- It facilitates the MVC (Model View Controller) architecture that makes the web development process work smoothly.
- MERN framework facilitates open-source support, easily accessible and lower learning time, and development cost
- The framework comes with a pre-built extensive suite of testing tools
- The framework helps to develop robust and scalable web applications
- The four technologies provide smooth integration with cloud platforms

Project Setup

To kickstart the project on a local machine, one would need to follow the following steps:

- 1) Download the entire source-code in a directory
- 2) cd to that directory
- 3) cd to the server folder and enter the command **npm start**
- 4) In a new terminal, cd to the source code directory
- 5) Continuing the new terminal, cd to the server folder and enter the command **npm start**

```
C:\Users\kulvir\Desktop>cd tata-steel-project
C:\Users\kulvir\Desktop\tata-steel-project>cd server
C:\Users\kulvir\Desktop\tata-steel-project\server>npm start

> server@1.0.0 start C:\Users\kulvir\Desktop\tata-steel-project\server
> nodemon --exec babel-node app.js

[nodemon] 2.0.9
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `babel-node app.js`
Server listening on Port - 3001
http://localhost:3001
connected to db
```

Starting the backend server

```
C:\Users\kulvir\Desktop>cd tata-steel-project
C:\Users\kulvir\Desktop\tata-steel-project>cd client
C:\Users\kulvir\Desktop\tata-steel-project\client>npm start

> client@0.1.0 start C:\Users\kulvir\Desktop\tata-steel-project\client
> craco start

i [wds]: Project is running at http://192.168.80.1/
i [wds]: webpack output is served from
i [wds]: Content not from webpack is served from C:\Users\kulvir\Desktop\tata-steel-project\client\public
i [wds]: 404s will fallback to /
Starting the development server...
Compiled with warnings.
```

Starting the frontend server

Application Architecture

The application consists of a 3 tier server-client model. The frontend has been written in ReactJS while the backend has been written in NodeJS, ExpressJS which is supported by an online cloud database mongoDB's Atlas.

Logging In and Authentication

The user needs to login to the web portal to access the features. Each user has an id and a respective password. The user enters the id and password to the login form and is then sent to the backend for authentication. The user id is first validated and then the password is checked for the corresponding id. Since the password stored in the database is encrypted using **bcrypt**, the hash has to be decrypted to check for validity of password. On successful match of user id and password, a **JSON web token** is generated for the user and stored in the local storage. The token has certain validity time and hence creates a session. Once the token is past its validity time, it gets corrupted and the user's session will expire and will be logged out.

Frontend Routes

The frontend routes are divided across three major components – the public component whose accessibility is public and anyone can view the content encapsulated within it, the dashboard component and the admin component.

The Public Component

The public component only consists of the login page.

This route consists of a form which asks for the user id and password which is used for logging in and authentication. On entering the details required and clicking the login button, the frontend requests for the authentication route from the backend which validates and authenticates the user as discussed in the previous point. On successful authentication, the user enters the home page and is now able to interact with the dashboard components

The Dashboard Component

This is a custom component which is created to check if the user making request is authenticated or not. This is necessary to ensure security of the application and reduces the vulnerability of the application.

The dashboard component comprises of the following routes and pages on the frontend –

- **Dashboard page or Home page**
This page consists of a form which is used to register or log a complaint. It is here where firestore is implemented. The user can select and write various details of the complaint and also upload multiple images of the complaint.

- **Complaint View page**
This page displays all the complaints according to the search query and search parameters entered by the user. It displays the complaint that are filed by the user that is logged in. No other complaints are visible on this page.
- **Report page**
This page asks for the user to enter a specific date range and accordingly an excel file is generated which contains the dataset of complaints that are registered within the date range specified by the user. A download option is also available after the report has been generated
- **Admin page**
This page is secured and only allows those users whose type is of ADMIN kind to enter and view this page. it is safe guarded by the admin component discussed below.

Apart from this a logout button option is also present

The Admin Component

This custom component is created to check if the user making the request is authenticated as well as authorized. The authentication is similar to that of the dashboard component. The authorization check is dealt in the following way. The JSON Web Token associated to the user has **Payload** associated to it. The Payload data comprises of the user id as well as its type. The type property defines the role of the user. On decoding the Payload, if the type property is “ADMIN” only then the features of the component will be rendered.

The following pages fall under the Admin Component –

- **Admin page**
this page asks the admin type user to enter search parameters to look for a particular complaint to be edited
- **Edit Complaint page**
this page asks the admin type user to enter new status and new remarks for the selected complaint and also displays the existing details of the selected complaint.

Firestore Implementation

The Firestore feature of Firebase is only implemented at the frontend. The firestore folder consists of the configuration of the Firebase console and the app related to the project. Similarly the config is exported and is used in the Services part of the frontend application

Services (Client)

The services at the client side comprise mainly of –

- **Authentication**
The file exports various functions to login, logout, send API requests all related to authentication and authorization. The functions are stored in an object which is then exported.

- Upload Service is used to interact with firestore and upload images to the cloud
- Edit Complaint service is used make API request to bring about changes in the complaint
- Various Get Data services are used to make API calls to request and fetch data from the backend regarding transporters, locations and complaints to be displayed on the webpage

Middleware and Authorization

Each request that is made from the frontend to the backend is first authorized and checked before the entering the controllers part of the backend route. Authorization is carried out using the **JSON Web Token middleware**. The token which is generated only on login is associated for a particular time period. If the token is corrupted or not valid, the middleware catches the error and sends the message to the frontend part. This disallows the flow of control to reach the data modification or database interaction part of the application and prevents any unauthenticated or unauthorized user to access functionalities beyond his/her privileges.

Backend Routes

Every backend route is associated with a middleware for verification and then the control shifts to a particular controller which in turn accesses a required service to performs the task. The controller then returns the result received from the service to the frontend for completion of request.

All routes in the backend are “POST” method calls.

The following routes and its functions are discussed below –

- **User Dashboard Route**
This route checks the validity of the token and returns an appropriate message to the frontend of the user token is corrupted or not.
- **Upload Complaint**
This route is used to upload the complaint details entered by the user. It creates a new document in the complaint database and creates a new complaint in the database.
- **Get Location Data**
This route is used to get the names of all the locations that are present in the locations collection and send them as an object array to the frontend
- **Get Transporter Data**
This route is used to send the documents present in the transporters collection as an array of objects to the frontend.
- **Get Complaint Data**
This route is used to send the documents present in the complaints collection as an array of objects to the frontend where the ‘reportedBy’ property is equal to id stored as a payload of the JSON Web Token associated to the request.
- **Get Admin Complaint Data**
This route is used to send the documents present in the complaints collection as an array of objects to the frontend

- **Report**
This route is used to send the documents present in the complaints collection as an array of objects to the frontend where the createdOn property is within the date range specified as the query parameter.
- **Update Complaint Data**
This route is used to update any change made by the admin user to a particular complaint. The changes are directly reflected in the database.

API Specifications

- **‘/login_user’**
This POST method checks if the user id and password present in json data sent as a body of the request is valid or not and returns message accordingly
- **‘/user_dashboard’**
This POST method checks if the access-token is present as a header of the request and returns a json accordingly
- **‘/upload_complaint’**
This POST method creates a new document in the complaints collection and fills the fields of the document by extracting information from the json sent as body of the request.
- **‘/get_location_data’**
This POST method returns the documents present in the locations collection as an array of objects
- **‘/get_transporter_data’**
This POST method returns the documents present in the transporters collection as an array of objects
- **‘/get_complaint_data’**
This POST method returns the documents present in the complaints collection as an array of objects of the logged in user only
- **‘/get_complaint_data_admin’**
This POST method searches for the documents according to the search parameters given in the body as JSON object and returns those documents present in the complaints collection as an array of objects
- **‘/update_complaint_data’**
This POST method updates the specified document of the complaints collection according to the data sent as the body of the request.
- **‘/report’**
This POST method returns the documents present in the complaints collection as an array of objects according to the date range specified in the body of the request.

Database Design

The entire database has been hosted on the mongoDB cloud service – Atlas. A total of 4 tables or collections have been made to interact with. The schema for each collection has been written in 4 separate files kept in the *models* folder of the server directory.

Collection 1 – locations

This collection has been made to enter all the possible locations inside the works where a fault or a complaint can be registered. The schema comprises of :

1. location – type: object
This object has a sub property called coordinates which is an array of type Point. It stores the longitude and latitude of the location. It also checks if the longitude and latitude stored is a valid one or not
2. name – type: string
This stores the name of the location

Collection 2 – transporters

This collection has been made to enter all the transporters that are registered to the company. The schema comprises of :

1. name – type: string
This stores the name of the transporter
2. id – type: string
This stores the unique id that has been given to the transporter in the master database of the company

Collection 3 – users

This collection has been made to store all the users that are permitted to access specified or all features of the application. The schema comprises of :

1. id – type: string
This stores the unique id given to every employee of the company which can be used to login and identify the user of the application.
2. password – type: string
This stores the password associated to the respective user id in encrypted form.
3. type – type: string [“VIEWER”, “ADMIN”]
This stores the accessibility or role of the user. It enables the website to figure out if a user has ADMIN privileges or VIEWER privileges.

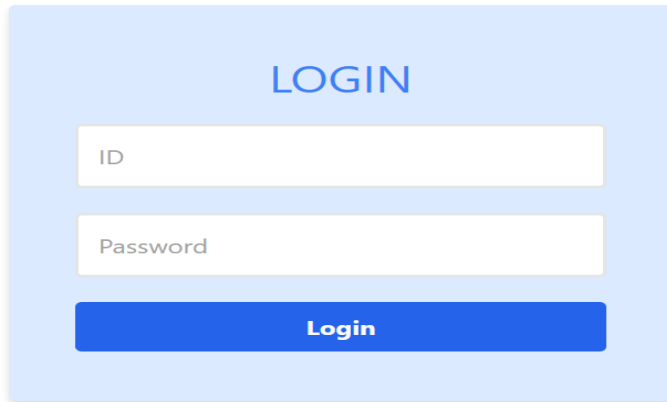
4. lastLogin – type: date
This stores the date and time of the user’s most recent login to the application.
5. createdOn – type: date
this stores the date and time when the document was first added to the collection

Collection 4 – complaints

This collection has been made to store any complaint that is uploaded by the user. The schema comprises of :

1. imageUrl – type: array
This stores all the URLs of the images uploaded by the user while logging a complaint. The file is stored in the firestore cloud storage service of Google.
2. Status – type: string [“OPEN”, “CLOSED”]
This stores the status of the complaint. OPEN is the complaint has not been resolved and CLOSED if the complaint has been resolved.
3. Remarks – type: string
This stores the remarks added by the admin about the complaint.
4. reportedBy – type: string
This stores the id of the user who has reported the complaint.
5. reportedLocation – type: string
This stores the name of the location where the fault has occurred.
6. description – type: string
this stores the description of the fault and describes the complaint raised.
7. vehicleNumber – type: string
This stores the vehicle number of the vehicle which has not followed the protocol of the company
8. transporterName – type: string , optional
This stores the name of the transporter to which the vehicle belongs
9. transporterCode – type: string , optional
This stores the id of the transporter to which the vehicle belongs
10. comments – type: string
This stores the comments of the user while registering the complaint
11. createdOn – type: date
This stores the date and time when the complaint was created by the user.

Screenshots



LOGIN

ID

Password

Login

login webpage

WELCOME

Dashboard

View Complaints

Report

Admin

Logout

REGISTER COMPLAINT :

Reported Location



Description

Vehicle Number

Transporter name-id



comments

Choose Files

No file chosen

Upload

dashboard/ home page

WELCOME

Dashboard

View Complaints

Report

Admin

Logout

VIEW COMPLAINTS

List All



display

COMPLAINT 1:

Description : wrong parking
Reported Location : G Blast Furnace
Vehicle Number : DL-211-111
Transporter Name *(optional)* : Total Transports
Transporter Code *(optional)* : A233
Created On : 2021-07-15T15:53:44.012Z
Comments : bad
Image URLs :
Link 1

Status : OPEN

Remarks : ghasdsdg

COMPLAINT 2:

Description : fff
Reported Location : H Blast Furnace
Vehicle Number : sss
Transporter Name *(optional)* :
Transporter Code *(optional)* :
Created On : 2021-07-15T16:32:51.415Z
Comments : qqq
Image URLs :

view complaints page

WELCOME

Dashboard

View Complaints

Report

Admin

Logout

REPORT

Complaint Logged Date Range

From: 07/03/2021



To: 07/30/2021



Generate Report

Download Report



Report (1).csv

Report Page

	A	B	C	D	E	F	G	H	I	J	K
1	Created On	Status	Remarks	Reported	Reported	Description	Vehicle No	Transport	Transport	Comments	
2	2021-07-1	OPEN	ghasdsdg	asd	G Blast Fu	wrong par	DL-211-11	Total Tran	A233	bad	
3	2021-07-1	OPEN	asdqewra	asd	H Blast Fu	fff	sss			qqq	
4	2021-07-2	OPEN	None	abc123	CRM	jjj	kkk	MAZDA &	K033	ooo	
5	2021-07-2	OPEN	hgfgsdadf	asd	CRM	oooo	pppp	MAZDA &	K033	qqqq	
6	2021-07-2	OPEN	remark ne	asd	TSPDL	zzzz	xxxxx	Kailash Ro	34112	cccc	
7	2021-07-2	OPEN	None	abc123	CDC	wfghjgf	sgfhfdghj	Magadh A	12322	ghmgjhjg	
8	2021-07-2	OPEN	None	abc123	Safety Dep	jhfdsdrfv	yugjdfgda	Konkan Er	W042	ghfdfgasdrf	
9											
10											

The downloaded excel file

WELCOME

Dashboard

View Complaints

Report

Admin

Logout

ADMIN

Search and Edit Complaints

asd

G Blast Furnace

ANY

Go

COMPLAINT 1:

Complaint ID : 60f05a080509c04c40194e51

Reported By : asd

Description : wrong parking

Reported Location : G Blast Furnace

Vehicle Number : DL-211-111

Transporter Name *(optional)* : Total Transports

Transporter Code *(optional)* : A233

Created On : 2021-07-15T15:53:44.012Z

Comments : bad

Image URLs :

Link 1

Status : OPEN

Remarks : ghasdsdg

Edit Complaint

Admin page

EDIT COMPLAINT

COMPLAINT ID = 60F97A4771F1AD329CE38F6E

Reported By : abc123

Description : jjj

Reported Location : CRM

Vehicle Number : kkk

Transporter Name (optional) : MAZDA & YADAV ASSOCIATES

Transporter Code (optional) : K033

Created On : 2021-07-22T14:01:43.757Z

Comments : ooo

Image URLs :

Link 1

Status : OPEN

Remarks : None

ADD REMARKS

Choose a status

Edit Complaint

edit complaint page

Conclusion

This project was a great learning opportunity for me. I came across various technologies and frameworks that are of great importance in the field of web development. I learnt about how real world problems are dealt with and tried to create a model solution for the same. I gained knowledge in cloud database along with its integration to a backend server hosted on NodeJS. The most important gain for me through this project was learning ReactJS which helped me design the frontend. Learning about Firebase console was also very interesting and useful. To conclude, this project helped me become better at web development in totality and helped me understand what it's like to deal with a real world problem in an industry.

References

<https://docs.atlas.mongodb.com/>
<https://mongoosejs.com/docs/guide.html>
<https://firebase.google.com/docs/web/setup>
<https://firebase.google.com/docs/firestore>
<https://nodejs.org/en/docs/es6/>
<https://babeljs.io/docs/en/>
<https://expressjs.com/en/5x/api.html>
<https://reactjs.org/docs/getting-started.html>
<https://jwt.io/>
<https://www.npmjs.com/package/jsonwebtoken>
<https://www.npmjs.com/package/react>
<https://www.npmjs.com/package/bcrypt>
<https://www.npmjs.com/package/jwt>
<https://www.npmjs.com/package/jwt-decode>
<https://tailwindcss.com/docs>
<https://www.ordinarycoders.com/blog/article/reactjs-tailwindcss>
<https://bezkoder.com/react-jwt-auth/>
<https://www.pragimtech.com/blog/reactjs/protected-routes-in-react/>
https://www.youtube.com/watch?v=kByiWTWdpww&ab_channel=uidotdev
<https://tailwindcomponents.com/component/sidebar-navigation>
<https://www.techomoro.com/how-to-create-a-multi-page-website-with-react-in-5-minutes/>
<https://rapidapi.com/blog/create-react-app-express/>
https://www.youtube.com/watch?v=8r1Pb6Ja90o&ab_channel=HongLy
<https://www.digitalocean.com/community/tutorials/nodejs-crud-operations-mongoose-mongodb-atlas>
<https://codeburst.io/to-handle-authentication-with-node-js-express-mongo-jwt-7e55f5818181>