

# Supply Chain Transport Park Complaint Portal

# PROJECT REPORT

*Submitted in fulfilment for the J-Component of CSE3002–Internet and Web Programming*

# CAL COURSE

*in*

**B.Tech. (COMPUTER SCIENCE)**

*by*

**KULVIR SINGH (19BCE2074)**  
**ANITEJ SRIVASTAVA (19BCE0835)**

*Under the guidance of*

**Prof. Jayakumar Sadhasivam**  
**SCOPE**



VIT®

**Vellore Institute of Technology**

(Deemed to be University under section 3 of UGC Act, 1956)

**SCHOOL OF COMPUTER SCIENCE AND ENGINEERING**

Fall Semester 2021-2022

## TABLE OF CONTENTS

<b>CHAPTER NO.</b>	<b>TITLE</b>	<b>PAGE NO.</b>
<b>1</b>	Abstract	<b>3</b>
<b>2</b>	Introduction	<b>3</b>
<b>3</b>	Problem Statement	<b>4</b>
<b>4</b>	Process Flow Diagram	<b>4</b>
<b>5</b>	Modules	<b>6</b>
<b>6</b>	Database Design	<b>11</b>
<b>7</b>	Conclusion	<b>15</b>
<b>8</b>	References	<b>15</b>
<b>9</b>	Snapshots	<b>16</b>
<b>10</b>	Source Code	<b>21</b>

## **1.Abstract**

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.

## **2.Introduction**

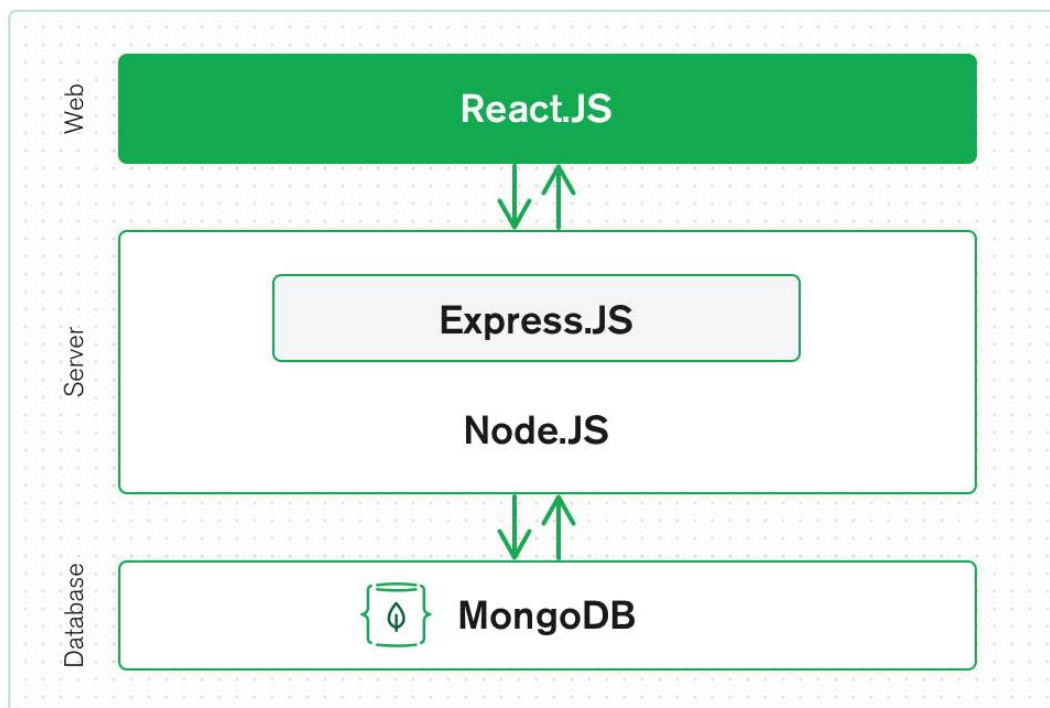
The project work deals with the implementation of a complaint registering system as a web application. The aim of the project is to develop a user friendly and easily interactable web based portal to log a complaint against any vehicle/transporter who is not abiding by the rules and guidelines of the 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.

### 3.Problem Statement

In any industry where logistics division is present, a network of transportation is imperative, especially road transport.

If we take the case of steel producing industry like Tata Steel, there is a need to monitor and evaluate constantly the proper mechanism of parking and scheduling the road transport vehicles in order to reduce and nullify the issues that may arise due to uncoordinated transportation.

### 4. Process Flow Diagram



*Figure 1. Control Flow Diagram based on framework interaction*

As seen in Figure 1, the application has a context switching between the frameworks in the following manner. The frontend of the application on react server has various routes and services infused with a UI/UX for making the system application interactive. The routes can redirect the control to the backend hosted on a node and express based server. The backend is infused with a variety of components that provide functionality to the application. The routes, protected by a

middleware, along with the controllers and services form the core backend of the application. This server has the capability to perform CRUD (Create, Read, Update and Delete) operations on the database server. Hence the backend can interact with the mongodb atlas cloud database. The detailed explanation can be visualized using the process flow diagram Figure 2.

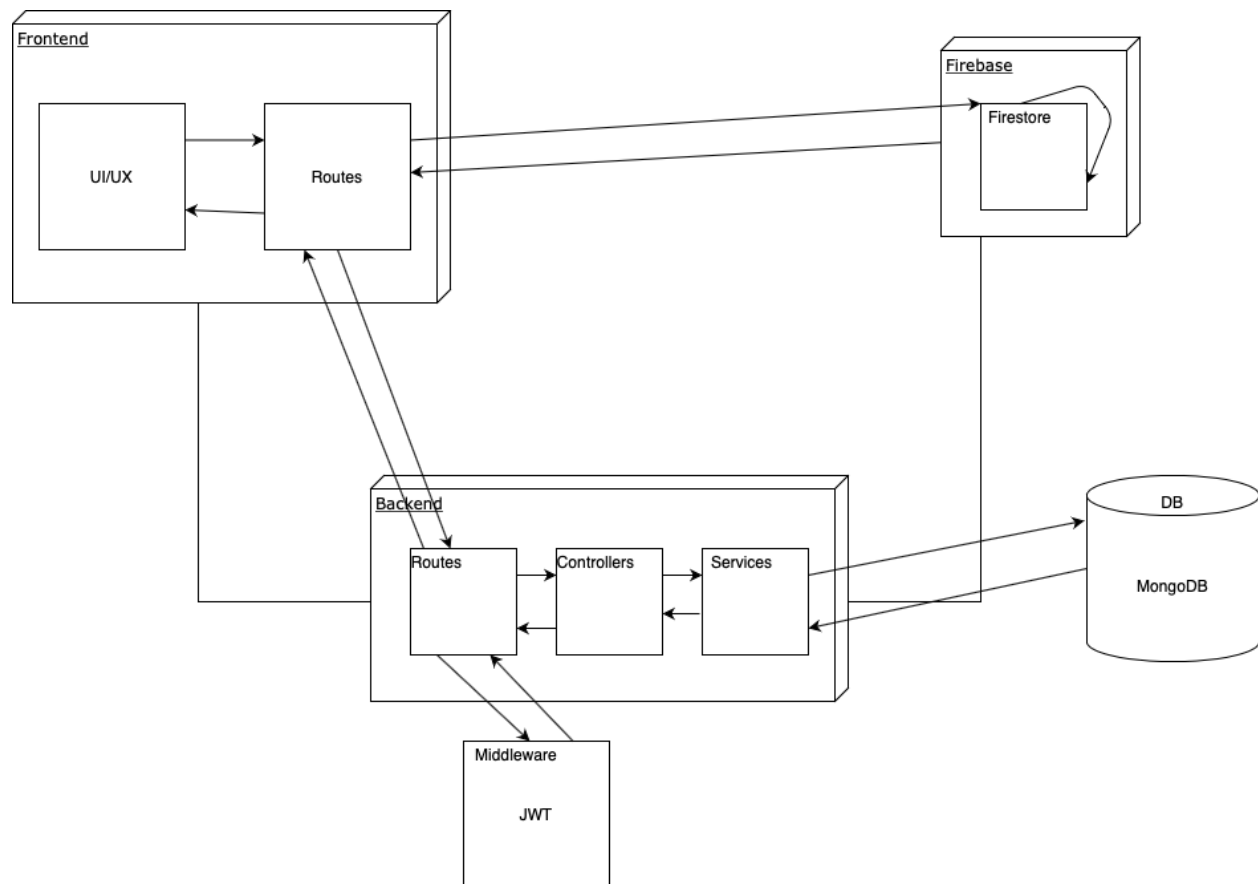


Figure 2. Process Flow Diagram

## 5. Modules

### 5.1 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

## **5.2 Firebase Implementation**

The Filestore feature of Firebase is only implemented at the frontend. The firebase 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

### **5.3 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 help 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.

### **5.4 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.

## 6. 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.

```
const complaintSchema = new mongoose.Schema ({
  // id: { type: String, required: true },
  reportedBy: { type: String, required: true },
  reportedLocation: { type: String, required: true },
  description: { type: String },
  imageUrl: [String],
  vehicleNumber: { type: String, required: true},
  transporterName: { type: String },
  transporterCode: { type: String },
  comments: { type: String },
  createdOn: { type: Date, default: Date.now },
  status: { type: String, enum: ["OPEN","CLOSED"], default: "OPEN" },
  remarks: { type: String, default: "None" }
});
```

Figure 3. Complaint Schema Code

```
const transporterSchema = new mongoose.Schema({
  name: { type: String, required: true },
  id: { type: String, required: true}
});
```

Figure 4. Transporter Schema Code

```

const locationSchema = new mongoose.Schema({
  name: { type: String, required: true },
  location: {
    type: {
      type: String,
      enum: ["Point"],
      required: true
    },
    coordinates: {
      type: [Number],
      required: true
    }
  }
});

```

Figure 5. Location Schema Code

```

const userSchema = new mongoose.Schema ({
  id: { type: String, required: true } ,
  password: { type: String, required: true } ,
  type: { type: String, enum: ["ADMIN", "VIEWER"], required: true } ,
  lastLogin: { type: Date, default: Date.now },
  createdOn: { type: Date, default: Date.now }
  // currentToken: String,
  // currentTokenTimestamp: Date
  // modifiedOn: Date
});

```

Figure 6. User Schema

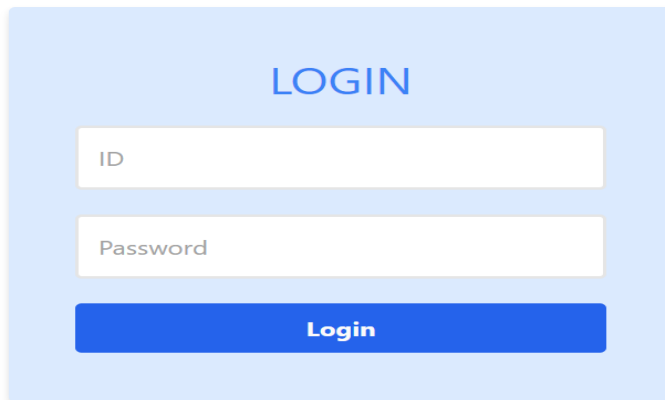
## 7.Conclusion

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.

## 8.References

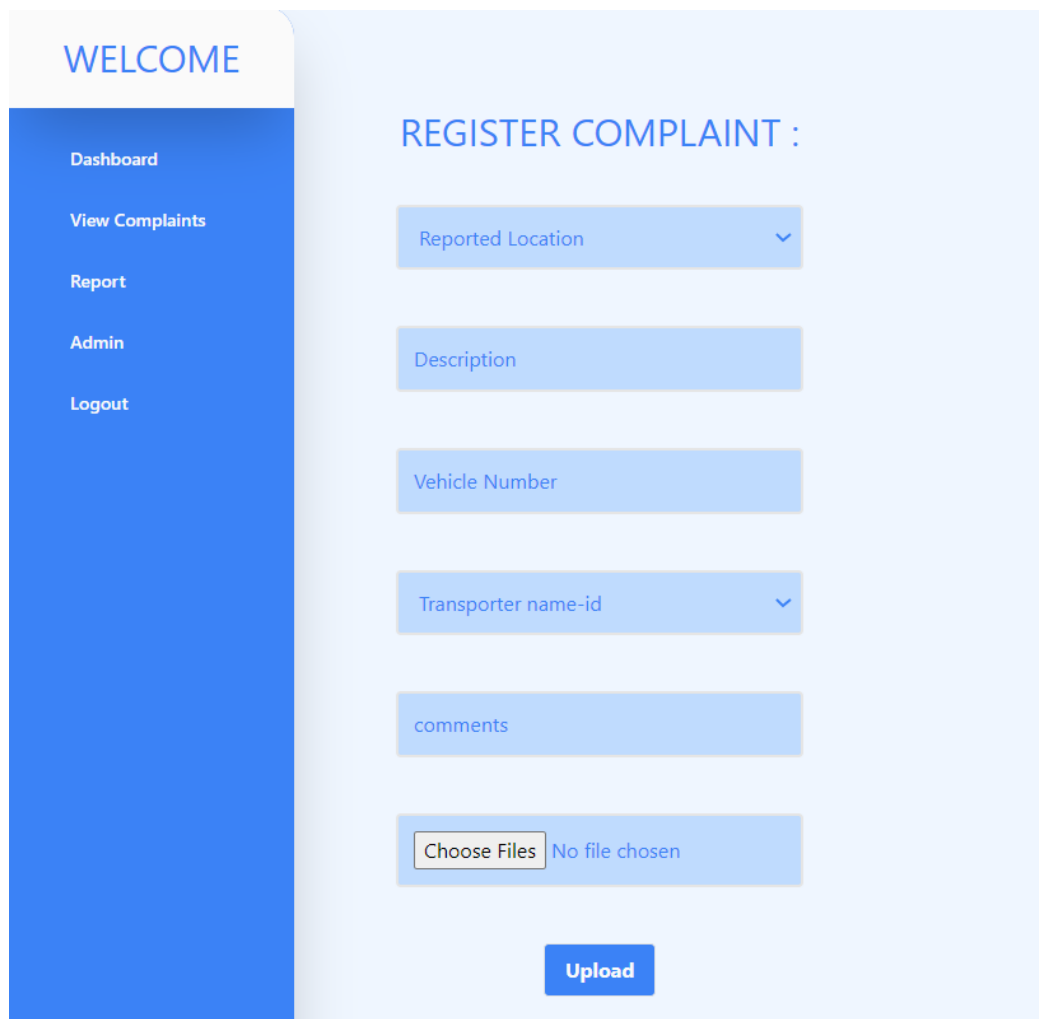
<https://docs.atlas.mongodb.com/>  
<https://mongoosejs.com/docs/guide.html>  
<https://firebase.google.com/docs/web/setup>  
<https://firebase.google.com/docs/firestore>  
<https://nodejs.org/en/docs/es6/>  
<https://babeljs.io/docs/en/>  
<https://expressjs.com/en/5x/api.html>  
<https://reactjs.org/docs/getting-started.html>  
<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://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.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>

## 9.Snapshots



A login form with a light blue background. At the top, the word "LOGIN" is centered in blue. Below it are two white input fields: the first is labeled "ID" and the second is labeled "Password". At the bottom is a blue button with the text "Login" in white.

login page



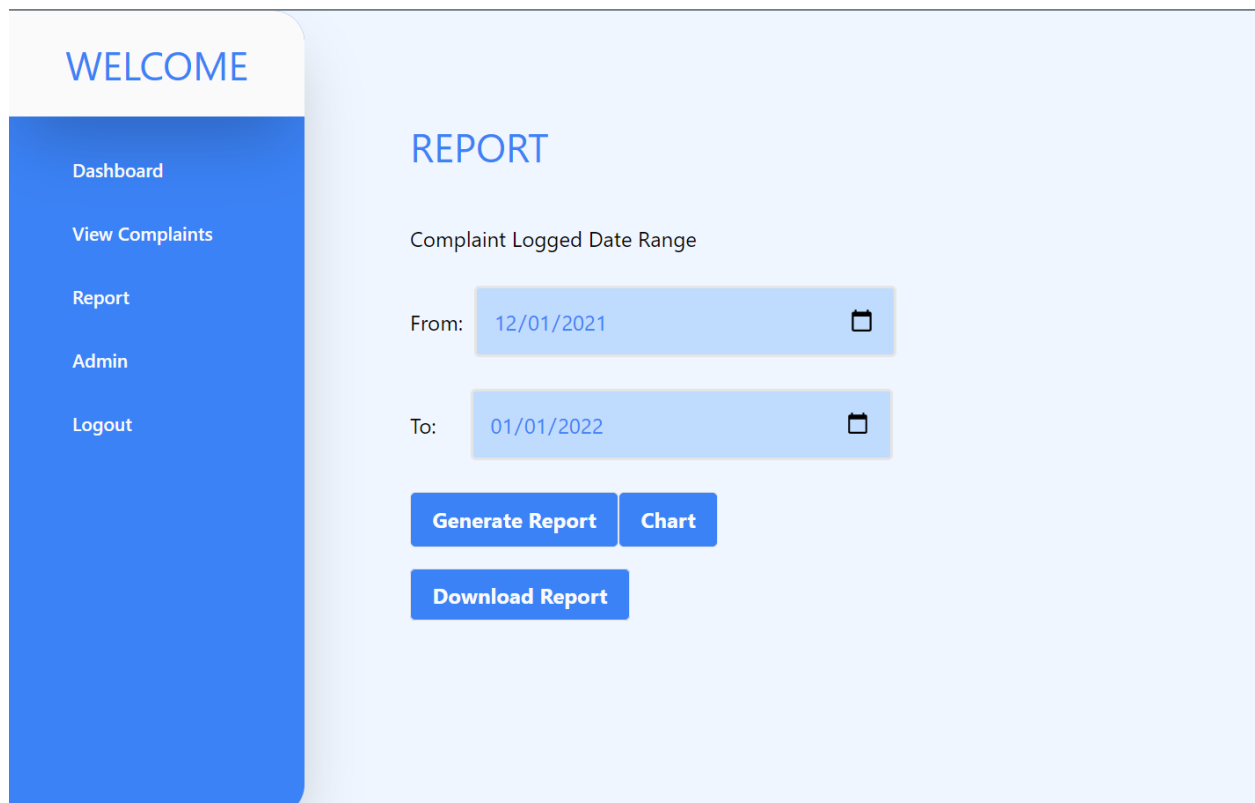
A dashboard page with a light blue background. On the left is a blue sidebar with the word "WELCOME" in white at the top. Below it are links: "Dashboard", "View Complaints", "Report", "Admin", and "Logout". The main content area has the title "REGISTER COMPLAINT :" in blue. It contains several input fields: "Reported Location" (a dropdown menu), "Description", "Vehicle Number", "Transporter name-id" (a dropdown menu), and "comments". At the bottom of the form is a file upload section with a "Choose Files" button and the text "No file chosen". A blue "Upload" button is at the very bottom.

Dashboard





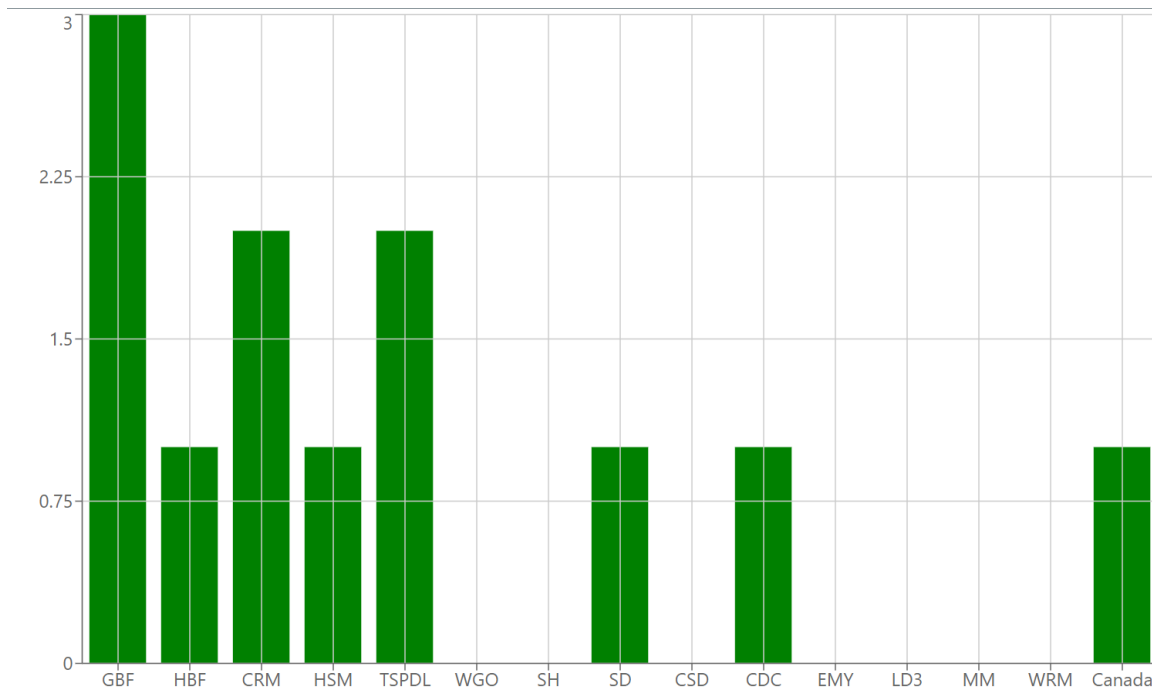
View Complaints



Report Page

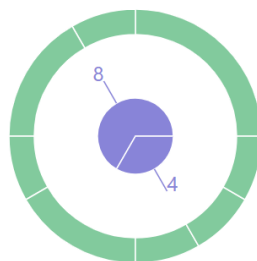
	A	B	C	D	E	F	G	H	I	J	K
1	Created On	Status	Remarks	Reported	Reported	Description	Vehicle No	Transport	Transport	Comments	
2	2021-07-1	OPEN	ghasdsdg	asd	G Blast Fu	wrong par	DL-211-11	Total Tran	A233	bad	
3	2021-07-1	OPEN	asdqewra	asd	H Blast Fu	fff	sss			qqq	
4	2021-07-2	OPEN	None	abc123	CRM	jjj	kkk	MAZDA &	K033	ooo	
5	2021-07-2	OPEN	hgfgsdaf	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	jhfdasdrfv	yugjdfgda	Konkan Er	W042	ghfdfgasdrf	
9											
10											

## Excel Report



### Overall Analysis

1. The green ring suggests the number of complaints distributed across various departments.
2. The inner ring gives the count for the number of open complaints to the closed one.



## Various graphical and chart report

## EDIT COMPLAINT

**COMPLAINT ID = 60F97A4771F1AD329CE3BF6E**

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 interface

```
> 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
```

Backend server

```

> client@0.1.0 start C:\Users\kulvir\Desktop\internship\TataSteel\PROJECT\client
> craco start
[wds] : Project is running at http://192.168.80.1/
[wds] : webpack output is served from
[wds] : Content not from webpack is served from C:\Users\kulvir\Desktop\interns
hip\TataSteel\PROJECT\client\public
[wds] : 404s will fallback to /
Starting the development server...

Compiled with warnings.

src\App.js
  Line 3:34: 'Link' is defined but never used      no-unused-vars
  Line 3:40: 'NavLink' is defined but never used      no-unused-vars
  Line 3:64: 'Redirect' is defined but never used      no-unused-vars

src\pages\Complaint-View-Dashboard.jsx
  Line 4:8: 'ReactDOM' is defined but never used      no-unused-vars

src\pages\Edit-Complaint.jsx
  Line 17:7: React Hook React.useEffect has missing dependencies: 'complaintId'
and 'getComplaint'. Either include them or remove the dependency array react-h
ooks/exhaustive-deps

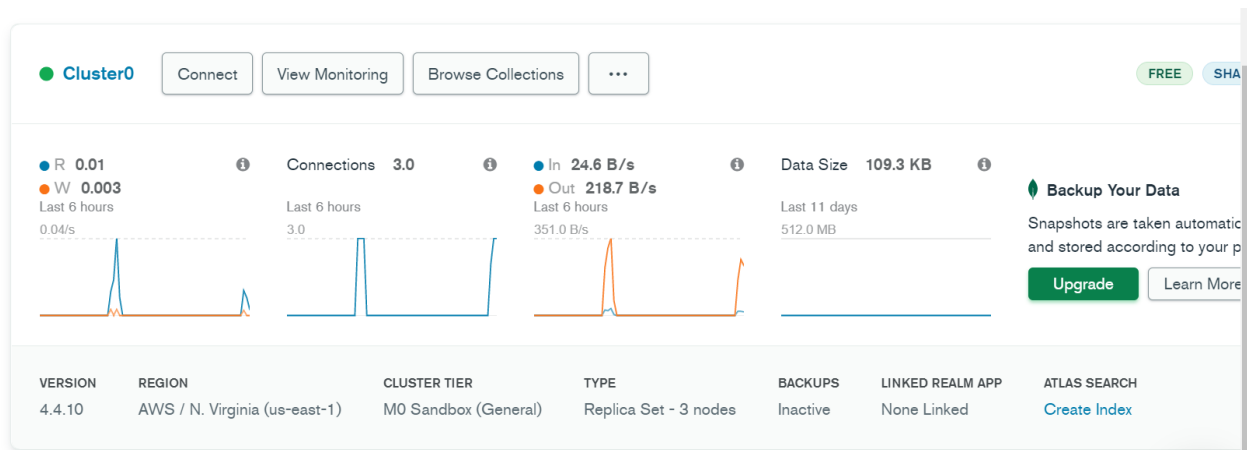
src\pages\Login.jsx
  Line 1:27: 'Component' is defined but never used      no-unused-vars
  Line 2:26: 'Router' is defined but never used      no-unused-vars
  Line 2:34: 'Link' is defined but never used      no-unused-vars
  Line 2:40: 'NavLink' is defined but never used      no-unused-vars
  Line 2:49: 'Switch' is defined but never used      no-unused-vars
  Line 2:57: 'Route' is defined but never used      no-unused-vars
  Line 2:64: 'Redirect' is defined but never used      no-unused-vars

src\pages\Register.jsx
  Line 1:27: 'Component' is defined but never used      no-unused-vars

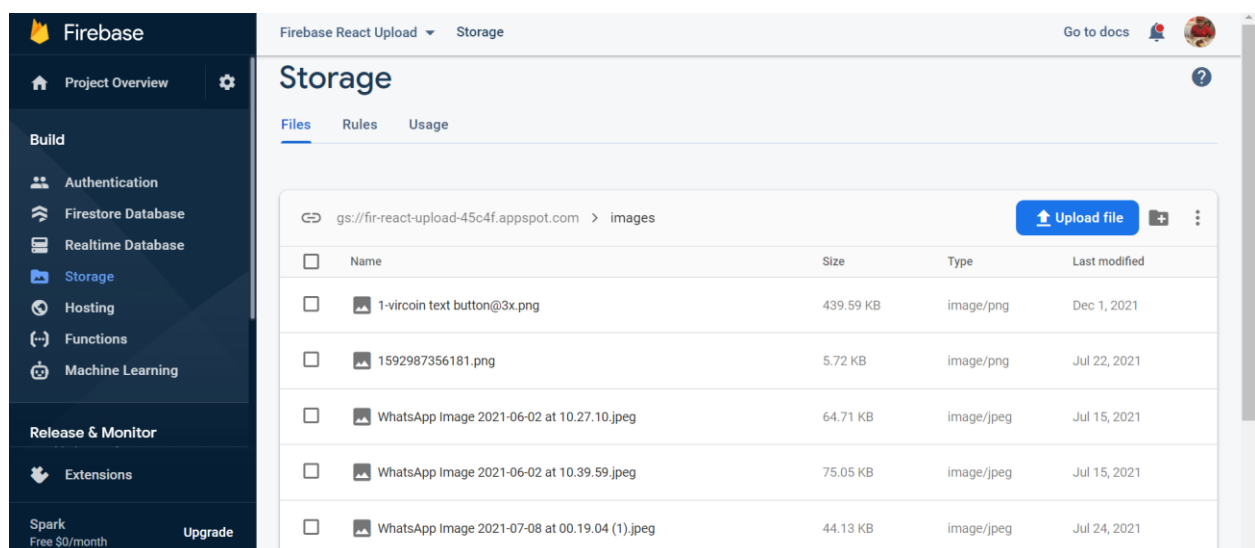
Search for the keywords to learn more about each warning.
To ignore, add // eslint-disable-next-line to the line before.

```

## Frontend server



## Cloud database (mongodb atlas)



## Cloud file storage (Firestore)

## 10. Source Code

[https://github.com/JayakumarClassroom/fs\\_21\\_22\\_cse3002\\_eth\\_g2\\_iwp-kulvir06/tree/main/J-component](https://github.com/JayakumarClassroom/fs_21_22_cse3002_eth_g2_iwp-kulvir06/tree/main/J-component)