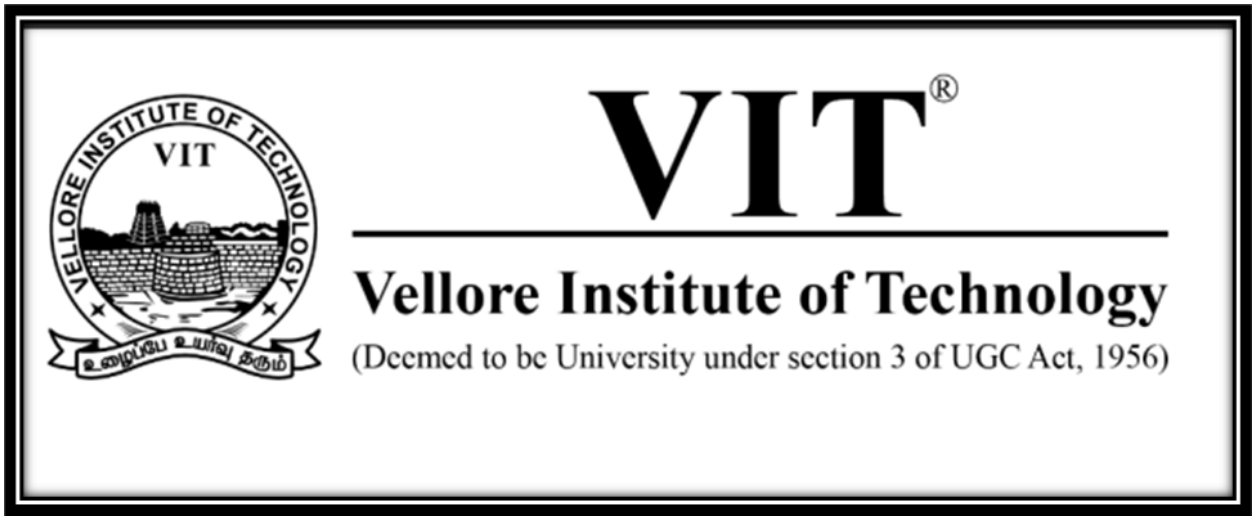

Digital Assignment - 3

CSE1901 - TECHNICAL ANSWERS FOR REAL WORLD PROBLEMS

Summer Semester Special 2021-2022

App Based Road Safety Violation Recording System

Name	**Reg No.**
Kulvir Singh	19BCE2074
Anitej Srivastava	19BCE0835
Vardhan Khara	19BCE0833

MODULES

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.

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

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.

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.

5. 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.

PSEUDO CODE

```
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 1. Complaint Schema Code

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

Figure 2. 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 3. 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 4. User Schema