**Team ID : SWTID1720075176**

**Indian citizen complaint box**

**API Documentation**

Overview

This document provides a comprehensive guide to the endpoints exposed by the backend of the Indian citizen complaint box project. It includes request methods, parameters, and example responses.

Technologies Used

Backend Framework: Node.js with Express.js

Database: MongoDB

File Upload: Multer

Project Structure

/controllers: Contains functions to handle requests and responses.

/models: Includes Mongoose schemas and models for MongoDB collections.

/routes: Defines the API endpoints and links them to controller functions.

/middlewares: Custom middleware functions for request processing.

/config: Configuration files for database connections, environment variables, etc.

API Endpoints

1. Complaints

POST `/api/complaints`

Description: Create a new complaint.

Request:

Headers: `ContentType: multipart/formdata`

Body:

```json

{

"category": "string",

"othercategory": "string",

"state": "string",

"district": "string",

"pincode": "string",

"comment": "string",

"phoneId": "string",

"date": "string"

}

```

Files: `photo`, `video`

Response:

Success (200): `{ "message": "Complaint registered successfully" }`

Error (400): `{ "error": "Error message" }`

DELETE `/api/complaints/:id`

Description: Delete a complaint by ID.

Request:

Params: `id` (Complaint ID)

Response:

Success (200): `{ "message": "Complaint deleted successfully", "deletedComplaint": { ... } }`

Error (404): `{ "message": "Complaint not found" }`

Error (500): `{ "message": "Internal server error" }`

PUT `/api/complaints/:id/likes`

Description: Increment likes for a complaint by ID.

Request:

Params: `id` (Complaint ID)

Response:

Success (200): `{ "message": "incremented like", "updatedLike": { ... } }`

Error (404): `{ "message": "Couldn't update/ID not found" }`

Error (500): `{ "message": "internal Error" }`

PUT `/api/complaints/:id/status`

Description: Update the status of a complaint by ID.

Request:

Params: `id` (Complaint ID)

Body:

```json

{

"status": "string"

}

```

Response:

Success (200): `{ "message": "Updation successful", "updatedStatus": { ... } }`

Error (404): `{ "message": "id not found" }`

Error (500): `{ "message": "Internal Error" }`

GET `/api/complaints`

Description: Fetch all complaints.

Response:

Success (200): `[ { ... }, { ... }, ... ]`

Error (400): `{ "error": "Error message" }`

2. User Management

POST `/api/signup`

Description: Register a new user.

Request:

Headers: `ContentType: application/json`

Body:

```json

{

"firstName": "string",

"lastName": "string",

"email": "string",

"phone": "string",

"dob": "string",

"password": "string"

}

```

Response:

Success (201): `{ "message": "User registered successfully" }`

Error (400): `{ "message": "Phone number already registered" }`

Error (500): `{ "message": "Server error" }`

POST `/api/login`

Description: Authenticate a user.

Request:

Headers: `ContentType: application/json`

Body:

```json

{

"phone": "string",

"password": "string"

}

```

Response:

Success (200): `{ "message": "Login successful" }`

Error (200): `{ "message": "User not found" }`

Error (401): `{ "message": "Invalid credentials" }`

Error (500): `{ "message": "Server error" }`

3. Feedback

POST `/api/feedback`

Description: Register feedback.

Request:

Headers: `ContentType: application/json`

Body:

```json

{

"name": "string",

"email": "string",

"stars": "number",

"comments": "string"

}

```

Response:

Success (201): `{ "message": "Feedback Registered" }`

Error (500): `{ "message": "Server error" }`

Integration with Frontend

The backend communicates with the frontend via RESTful APIs. Key points of integration include:

User Authentication: Tokens are passed between frontend and backend to handle authentication.

Data Fetching: Frontend components make API calls to fetch necessary data for display and interaction.

Error Handling and Validation

Error Handling: Centralized error handling using middleware.

Validation: Input validation using libraries like Joi or expressvalidator.

**Routes**

AdminRoutes

const express = require("express");

const router = express.Router();

const Admin = require("../models/Admin");

router.post("/Adminlogin", async (req, res) => {

try {

const { email, phone, password } = req.body;

const admin = await Admin.findOne({ phone });

if (!admin) {

console.log("Phone number not found");

return res

.status(404)

.json({ message: "Could not find the phone number" });

}

const isPassword = admin.password === password;

if (!isPassword) {

return res.status(401).json({ message: "Invalid Credentials" });

}

const isEmail = admin.email === email;

if (!isEmail) {

return res.status(401).json({ message: "Invalid Email" });

}

res.json({ message: "Login successful" });

} catch (error) {

res.status(500).json({ message: "Server error" });

console.error("Couldn't find the user", error);

}

});

module.exports = router;

feedback routes

const express = require("express");

const router = express.Router();

const Feedback = require("../models/Feedback");

router.post("/", async (req, res) => {

try {

const { name, email, stars, comments } = req.body;

const newfeedback = new Feedback({

name,

email,

stars,

comments,

});

await newfeedback.save();

res.status(201).json({ message: "Feedback Reegistered" });

} catch (error) {

console.log("Couldn't save feedback", error);

res.status(500).json({ message: "Server error" });

}

});

module.exports = router;

const express = require("express");

const User = require("../models/UserModel");

const bcrypt = require("bcryptjs");

const router = express.Router();

// Login Route

router.post("/", async (req, res) => {

const { phone, password } = req.body;

try {

const user = await User.findOne({ phone });

if (!user) {

return res.status(200).json({ message: "User not found" });

}

const isPasswordValid = await bcrypt.compare(password, user.password);

if (!isPasswordValid) {

return res.status(401).json({ message: "Invalid credentials" });

}

// Respond with a simple success message

res.json({ message: "Login successful" });

} catch (error) {

console.error("Error logging in:", error);

res.status(500).json({ message: "Server error" });

}

});

module.exports = router;

login route

// routes/LoginRoute.js

const express = require("express");

const User = require("../models/UserModel");

const bcrypt = require("bcryptjs");

const router = express.Router();

// Login Route

router.post("/", async (req, res) => {

const { phone, password } = req.body;

try {

const user = await User.findOne({ phone });

if (!user) {

return res.status(200).json({ message: "User not found" });

}

const isPasswordValid = await bcrypt.compare(password, user.password);

if (!isPasswordValid) {

return res.status(401).json({ message: "Invalid credentials" });

}

// Respond with a simple success message

res.json({ message: "Login successful" });

} catch (error) {

console.error("Error logging in:", error);

res.status(500).json({ message: "Server error" });

}

});

module.exports = router;

// signup.js route

const express = require("express");

const router = express.Router();

const User = require("../models/UserModel");

// POST route to create a new user

router.post("/", async (req, res) => {

const { firstName, lastName, email, phone, dob, password } = req.body;

try {

// Check if user with the given phone number already exists

const existingUser = await User.findOne({ phone });

if (existingUser) {

return res

.status(400)

.json({ message: "Phone number already registered" });

}

// Create a new user instance

const newUser = new User({

firstName,

lastName,

email,

phone,

dob,

password,

});

// Save the user to the database

await newUser.save();

res.status(201).json({ message: "User registered successfully" });

} catch (error) {

console.error("Error saving user:", error);

res.status(500).json({ message: "Server error" });

}

});

module.exports = router;

complain.js

const express = require("express");

const multer = require("multer");

const router = express.Router();

const Complaint = require("../models/Complaint");

// Multer setup for file storage

const storage = multer.diskStorage({

destination: function (req, file, cb) {

cb(null, "uploads/");

},

filename: function (req, file, cb) {

cb(null, `${Date.now()}-${file.originalname}`);

},

});

const upload = multer({ storage: storage });

// POST route to create a complaint

router.post(

"/",

upload.fields([{ name: "photo" }, { name: "video" }]),

(req, res) => {

// console.log("Request Body:", req.body);

// console.log("Request Files:", req.files);

const newComplaint = new Complaint({

category: req.body.category,

othercategory: req.body.othercategory,

state: req.body.state,

district: req.body.district,

pincode: req.body.pincode,

comment: req.body.comment,

photo: req.files["photo"] ? req.files["photo"][0].path : "",

video: req.files["video"] ? req.files["video"][0].path : "",

phoneId: req.body.phoneId,

date: req.body.date,

});

newComplaint

.save()

.then(() => res.json({ message: "Complaint registered successfully" }))

.catch((err) => {

console.error("Error saving complaint:", err);

res.status(400).json({ error: err.message });

});

}

);

//delete

router.delete("/:id", async (req, res) => {

try {

const deletedComplaint = await Complaint.findByIdAndDelete(req.params.id);

if (!deletedComplaint) {

return res.status(404).json({ message: "Complaint not found" });

}

res.json({ message: "Complaint deleted successfully", deletedComplaint });

} catch (error) {

console.error("Error deleting complaint:", error);

res.status(500).json({ message: "Internal server error" });

}

});

router.put("/:id/likes", async (req, res) => {

try {

const updateLike = await Complaint.findByIdAndUpdate(

req.params.id,

{

$inc: { likes: 1 },

},

{ new: true }

);

if (!updateLike) {

return res.status(404).json({ message: "Couldn't update/ID not found" });

}

res.json({ message: "incrementd like", updatedLike: updateLike });

} catch (error) {

console.log("couldnt increament");

res.status(500).json({ message: "internal Error " });

}

});

router.put("/:id/status", async (req, res) => {

try {

const updateStatus = await Complaint.findByIdAndUpdate(

req.params.id,

{ status: req.body.status },

{ new: true }

);

if (!updateStatus) {

return res.status(404).json({ message: "id not found" });

}

res

.status(200)

.json({ message: "Updation successful", updatedStatus: updateStatus });

} catch (error) {

console.log("Internal server error", error);

res.status(500).json({ message: "Internal Error " });

}

});

// GET route to fetch complaints

router.get("/", async (req, res) => {

try {

const query = {};

if (req.query.state && req.query.state !== "All") {

query.state = req.query.state;

}

if (req.query.district && req.query.district !== "All") {

query.district = req.query.district;

}

if (req.query.pincode && req.query.pincode !== "All") {

query.pincode = req.query.pincode;

}

const complaints = await Complaint.find(query);

res.json(complaints);

} catch (err) {

console.error("Error fetching complaints:", err);

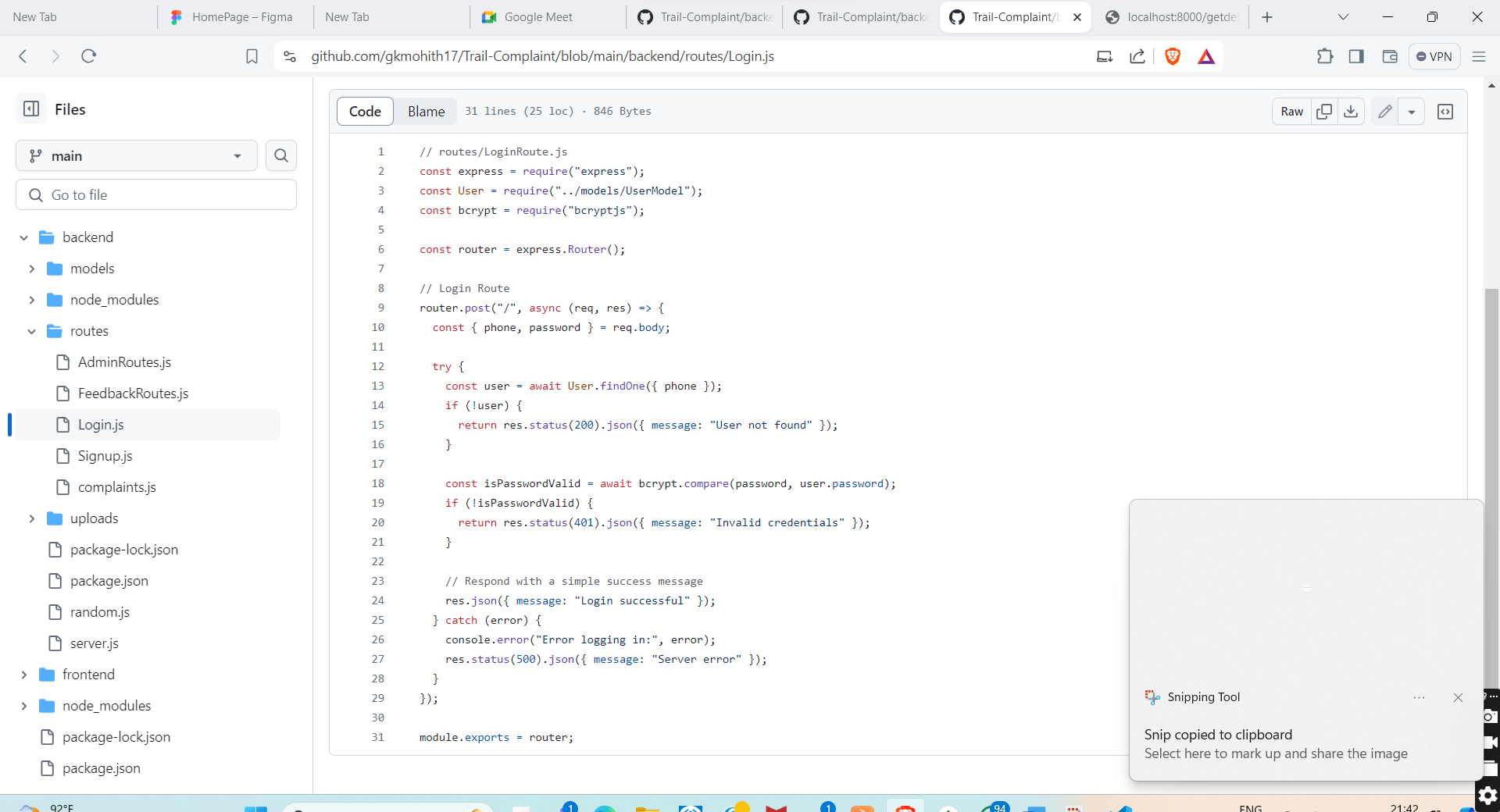
res.status(400).json({ error: err.message });

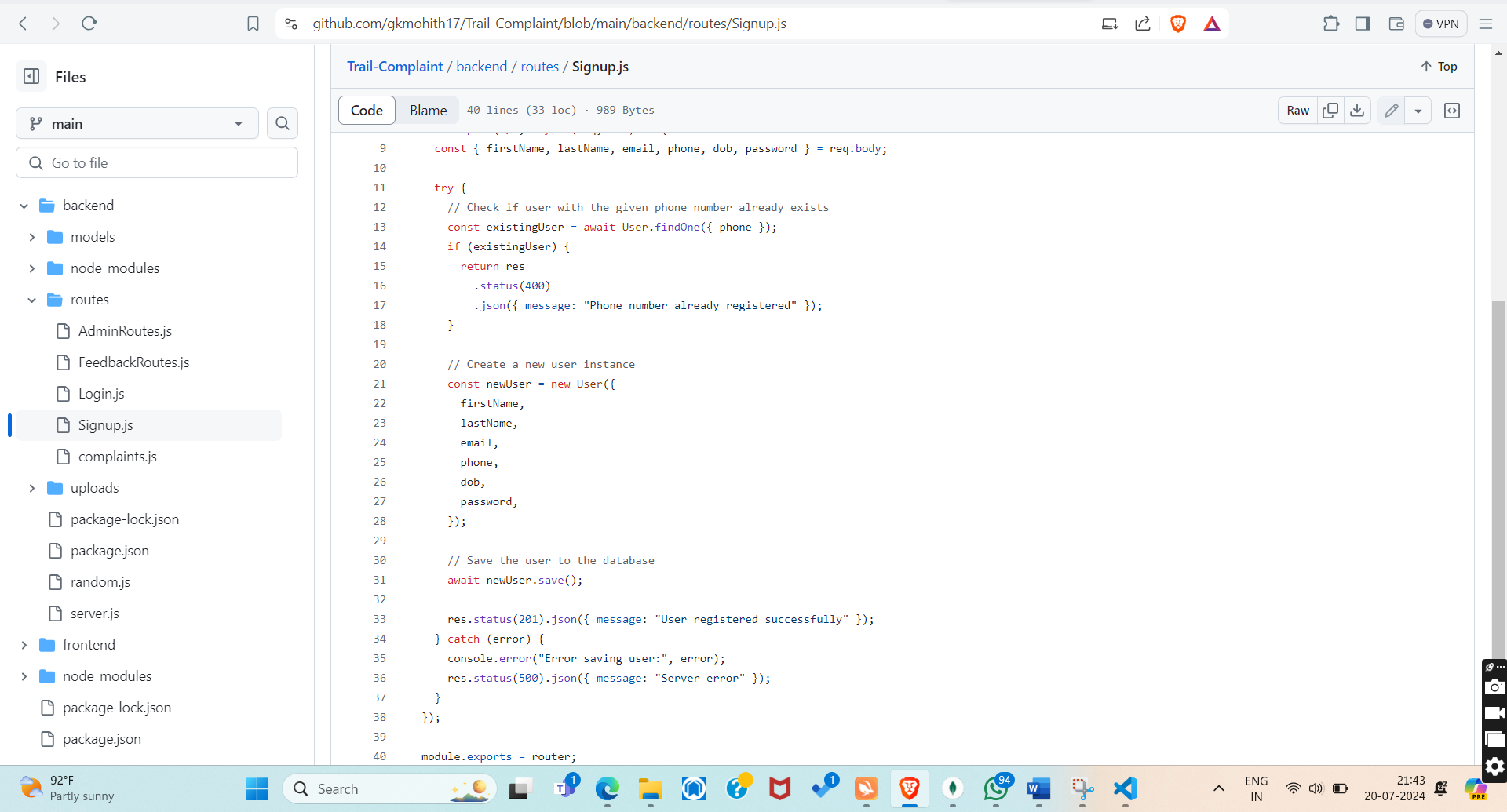
}

});

module.exports = router;

**Screenshots**

****

****



