DOCUMENTATION

By

Abubakar Ashrafi (2312353)

Raza Jawaid (2312384)

ACKNOWLEDGMENT

We are really grateful because we managed to complete our project within the time given by our teacher [Abid Ali and Usama Khalid]. This assignment cannot be completed without the effort and co-operation from our group members, Group members [Abubakar Ashrafi and Raza Jawaid].

We also sincerely thank our teacher [Abid Ali and Usama Khalid] for the guidance and encouragement in finishing this project and also for teaching us in this course.

Last but not the least, we would like to express our gratitude to our friends and respondents for the support and willingness directly or indirectly to spend some times with us to fill in the questionnaires.

CONTENT

Introduction .... **Error! Bookmark not defined.**

Objectives of the project.. 5

Topic of Our Project 6

Hardware/ Software Requirements.. 7

Er diagram…………………………………………………**9**

Code snippet…………………………………………………10

Introduction

Smart Shehri is a modern civic management system designed to improve complaint resolution and urban infrastructure services by connecting citizens, municipal departments, and field workers through a centralized digital platform. This project addresses key urban issues such as inefficient complaint handling, lack of communication between city departments, and delayed problem resolution.

The server module forms the **backend layer** of this system, offering a RESTful API built with **Node.js**, **Express.js**, and **PostgreSQL**. It ensures data validation, modular routing, and reliable communication with the database. Key features supported by this module include:

- Complaint registration and tracking
- Worker assignments and resolution logging
- Department and category management
- City and citizen data handling
- Dashboard and reporting endpoints

The Client module forms the frontend layer of this system, offering a minimalistic design and ui/ux built with **React** and **tailwind**. It ensures data validation, routing and design principles.

This documentation outlines the structure, functionality, and setup of the backend system as part of a final-year university project.

Objectives of the project

The primary objectives of the **Smart Shehri** project are:

- 📱💻 **Citizen Empowerment**: Allow residents to report urban issues (e.g., broken roads, garbage collection, streetlights) through an accessible digital platform.
- 🏢 **Efficient Administration**: Enable city departments to categorize, prioritize, and assign complaints systematically.
- 🔄 **Process Automation**: Automate the flow of information from complaint registration to resolution logging and closure.
- 📊 **Data-Driven Governance**: Provide dashboards and analytics to municipal authorities for tracking service performance and operational bottlenecks.
- 📍 **Location-Based Services**: Incorporate location data to facilitate worker dispatch and resolve issues faster.
- 🔐 **Secure & Validated Operations**: Implement input validation and route protection to ensure data integrity and system reliability.

Hardware/ Software Requirements

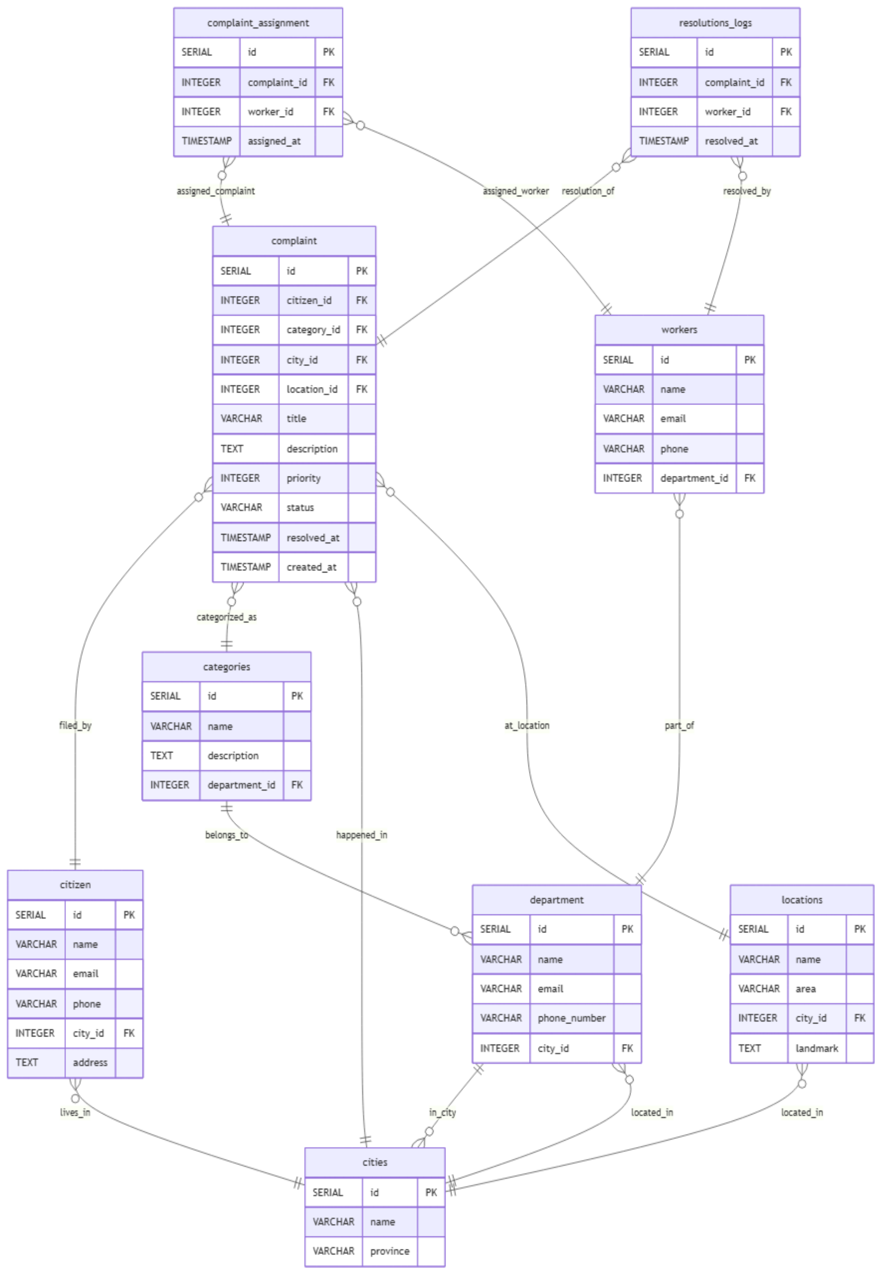
Backend Server:

Component	Technology Used
Language	JavaScript (ES6+)
Runtime	Node.js v18+
Framework	Express.js
Database	PostgreSQL v14+
Validation	Zod (for schema validation)
Environment Config	dotenv
Version Control	Git + GitHub
REST Testing Tool	Postman (for APIs)
Package Manager	npm

Frontend (Client)

Component	**Technology Used**
Language	JavaScript (ES6+)
Framework	React.js (with Vite)
Styling	Tailwind CSS
HTTP Client	Axios
Development Server	Vite.js
Routing	React Router
State Management	useState, useEffect
Code Quality	ESLint + Prettier

Er Diagram:


complaint_assignment
SERIAL id PK
INTEGER complaint_id FK
INTEGER worker_id FK
TIMESTAMP assigned_at
resolutions_logs
SERIAL id PK
INTEGER complaint_id FK
INTEGER worker_id FK
TIMESTAMP resolved_at
assigned_complaint
assigned_worker
resolution_of
resolved_by
complaint
SERIAL id PK
INTEGER citizen_id FK
INTEGER category_id FK
INTEGER city_id FK
INTEGER location_id FK
VARCHAR title
TEXT description
INTEGER priority
VARCHAR status
TIMESTAMP resolved_at
TIMESTAMP created_at
workers
SERIAL id PK
VARCHAR name
VARCHAR email
VARCHAR phone
INTEGER department_id FK
categorized_as
categories
SERIAL id PK
VARCHAR name
TEXT description
INTEGER department_id FK
filed_by
at_location
part_of
belongs_to
happened_in
citizen
SERIAL id PK
VARCHAR name
VARCHAR email
VARCHAR phone
INTEGER city_id FK
TEXT address
department
SERIAL id PK
VARCHAR name
VARCHAR email
VARCHAR phone_number
INTEGER city_id FK
locations
SERIAL id PK
VARCHAR name
VARCHAR area
INTEGER city_id FK
TEXT landmark
lives_in
in_city
located_in
located_in
cities
SERIAL id PK
VARCHAR name
VARCHAR province

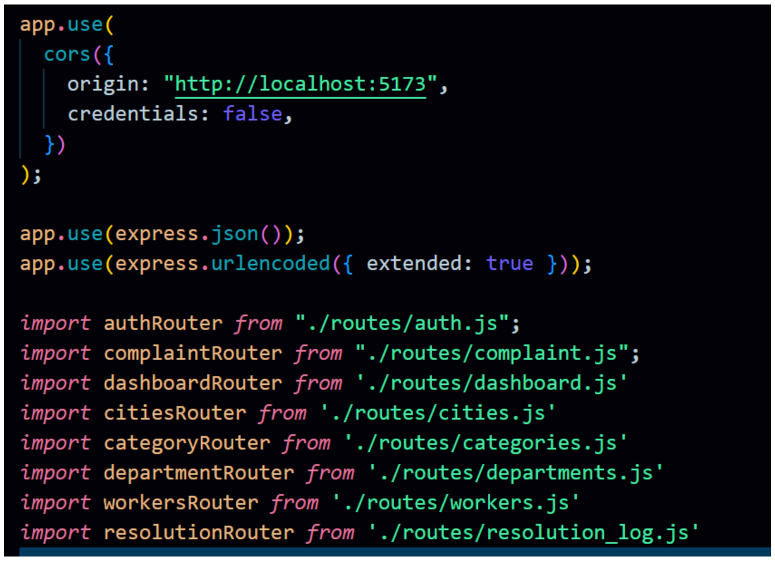
Code Snippets:

Server Side

1: Server Initialization (index.js)

Purpose

- Bootstraps Express application.
- Configures middleware (CORS, body-parsing).
- Imports and registers all route handlers.
- Defines health-check and 404 handlers.
- Starts the HTTP server.



```
app.use(
  cors({
    origin: "http://localhost:5173",
    credentials: false,
  })
);

app.use(express.json());
app.use(express.urlencoded({ extended: true }));

import authRouter from "./routes/auth.js";
import complaintRouter from "./routes/complaint.js";
import dashboardRouter from './routes/dashboard.js'
import citiesRouter from './routes/cities.js'
import categoryRouter from './routes/categories.js'
import departmentRouter from './routes/departments.js'
import workersRouter from './routes/workers.js'
import resolutionRouter from './routes/resolution_log.js'
```

2: Database Connection (database.js)

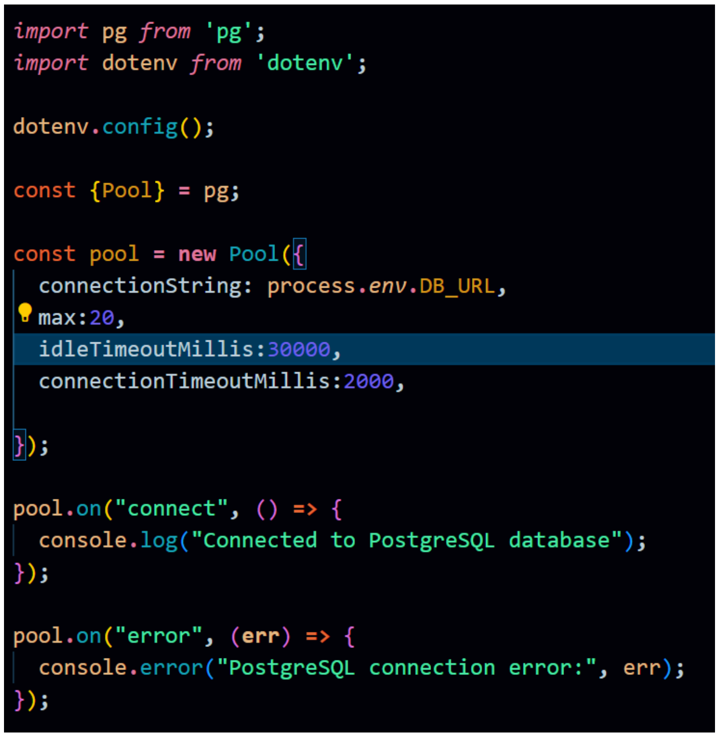
Purpose

Establishes a connection pool to PostgreSQL using environment configuration, to support concurrent queries.

Highlights

- Uses pg.Pool for efficient connection pooling.

- Config parameters include max, idleTimeoutMillis, and connectionTimeoutMillis.
- Monitors connection states:
    - Logs successful connections.
    - Captures and logs errors.



```
import pg from 'pg';
import dotenv from 'dotenv';

dotenv.config();

const {Pool} = pg;

const pool = new Pool({
  connectionString: process.env.DB_URL,
  max:20,
  idleTimeoutMillis:30000,
  connectionTimeoutMillis:2000,

});

pool.on("connect", () => {
  console.log("Connected to PostgreSQL database");
});

pool.on("error", (err) => {
  console.error("PostgreSQL connection error:", err);
});
```

3:Data Validation Schemas (zodschema.js)

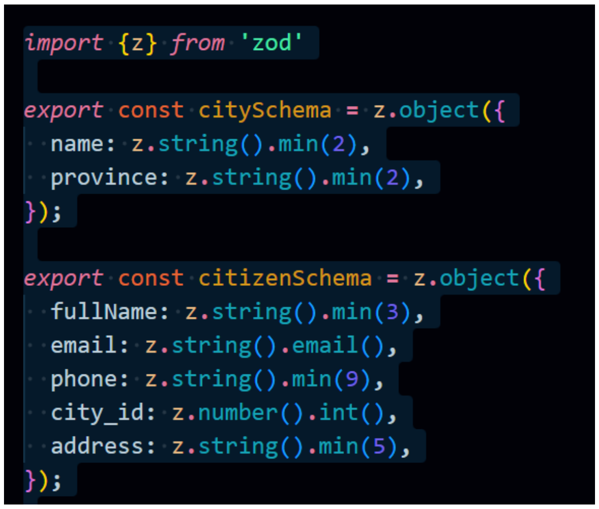
Purpose

Defines strict validation for incoming JSON payloads using the Zod library, ensuring integrity before database insertion or updates.

Schemas Overview

Schema	**Fields**
citySchema	name (min 2 chars), province (min 2 chars)
citizenSchema	fullName, email, phone, city_id, address
departmentSchema	name, email, phone_number, city_id

Schema	**Fields**
categorySchema	name (min 3), description (optional), department_id
workerSchema	name, email, phone, department_id — all optional
complaintSchema	citizen_id, category_id, city_id, location_id, title, description, etc.
resolutionSchema	complaint_id, category_id, optional resolved_at



```
import {z} from 'zod'

export const citySchema = z.object({
  name: z.string().min(2),
  province: z.string().min(2),
});

export const citizenSchema = z.object({
  fullName: z.string().min(3),
  email: z.string().email(),
  phone: z.string().min(9),
  city_id: z.number().int(),
  address: z.string().min(5),
});
```

6. Routing Modules

The routing modules in the Smart Shehri server handle various HTTP requests and interface with the PostgreSQL database to perform CRUD operations for users, cities, complaints, categories, departments, dashboard analytics, and more. Each route is responsible for specific modules of the system and maintains separation of concerns.

6.1 auth.js – Citizen Registration

Purpose:
Handles citizen onboarding by validating input and registering new users.

Key Logic:

- Validates incoming data using citizenSchema (Zod).

- Checks if the user already exists via name, email, or phone.
- If not found, inserts a new record into the citizens table.
- Returns user object upon success or existing record if duplicate.

Endpoint(s):

- POST /auth: Registers a new citizen or returns existing if duplicate.

6.2 categories.js – Complaint Categories

Purpose:
Handles category listing and filtering based on departments.

Key Logic:

- Joins with departments and complaints to enhance category metadata.
- Retrieves categories optionally filtered by department ID.

Endpoint(s):

- GET /categories: List all categories with complaint count.
- GET /categories/by-department/:departmentId: Filter categories by department.
- GET /categories/:id: Fetch a single category by ID.

6.3 cities.js – City and Location Management

Purpose:
Manage city records and their related complaint statistics and locations.

Key Logic:

- Validates and inserts city data.
- Fetches cities and their metadata (e.g., resolved complaint count).
- Fetches associated locations for a city.

Endpoint(s):

- GET /cities: Get all cities.
- GET /cities/:id: Get city details with complaint stats.

- POST /cities: Create a new city (validated via citySchema).
- GET /cities/:id/locations: Get locations under a city.

6.4 **complaint.js – Complaint Submission and Listing**

Purpose:
Facilitates complaint creation and paginated retrieval with filters.

Key Logic:

- Dynamic SQL where clause construction based on query params.
- Paginates complaints and joins with citizens, departments, locations, and categories.
- Complaint submission validated via complaintSchema.

Endpoint(s):

- GET /complaint: Retrieve complaints with filters (status, city, category, priority).
- POST /complaint: Submit a new complaint.

6.5 **dashboard.js – Analytics & Insights**

Purpose:
Provides summarized dashboard data to help administrators visualize system performance.

Key Logic:

- Fetches aggregate statistics: complaint count, ratings, department activity.
- Groups data by city, category, and time.
- Returns priority distributions and most recent complaints.

Endpoint(s):

- GET /dashboard/stats: Total counts and averages.
- GET /dashboard/complaints-by-city: Complaints per city.
- GET /dashboard/complaints-by-category: Complaints per category.

- GET /dashboard/department-ratings: Avg feedback per department.
- GET /dashboard/monthly-trends: Monthly complaint trends.
- GET /dashboard/priority-distribution: Priority label mapping.
- GET /dashboard/recent-complaints: Latest complaint entries.

6.6 **departments.js – Department Management**

Purpose:
Manages CRUD for departments and provides their associated metadata.

Key Logic:

- Joins with workers and categories to calculate performance.
- Computes department efficiency via resolution logs.
- Validates new department creation using departmentSchema.

Endpoint(s):

- GET /departments: List all departments with performance metrics.
- GET /departments/:id: Single department details.
- POST /departments: Create a new department.
- GET /departments/:id/workers: Active workers under a department.
- GET /departments/:id/categories: Active categories under a department.

6.7 resolution_log.js – Complaint Resolution

Purpose:
Handles resolution logging and randomly assigns a worker based on department.

Key Logic:

- Validates incoming data with resolutionSchema.
- Randomly selects a worker from the category's department.
- Inserts resolution log entry.

Endpoint(s):

- POST /resolution: Log a complaint resolution.

6.8 **workers.js – Worker Management**

Purpose:
Manage worker information and their assignments.

Key Logic:

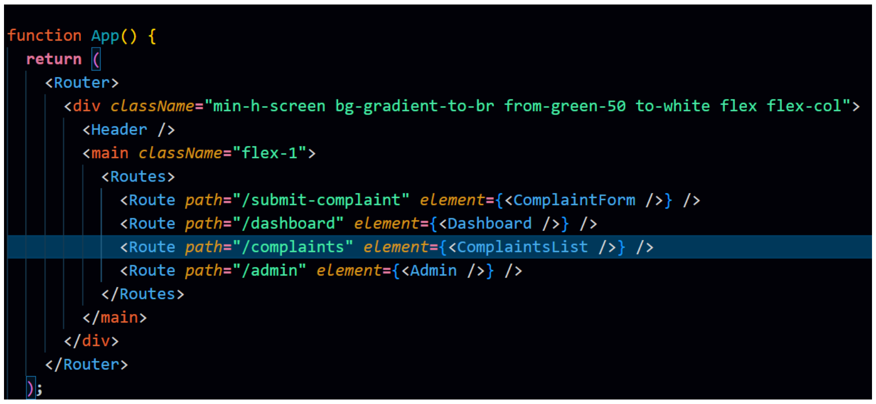
- Joins with assignments and resolution logs to report workload.
- Supports filtering available workers for assignment based on department.

Endpoint(s):

- GET /workers: List all workers with assignment count.
- GET /workers/:id: Worker profile with resolutions handled.
- POST /workers: Create a new worker (workerSchema validation).
- GET /workers/:id/assignments: Worker-specific complaint assignments.
- GET /workers/available/:departmentId: Filter available workers by department.

Client Side

1. Routing (App.jsx)



```
function App() {
  return (
    <Router>
      <div className="min-h-screen bg-gradient-to-br from-green-50 to-white flex flex-col">
        <Header />
        <main className="flex-1">
          <Routes>
            <Route path="/submit-complaint" element={<ComplaintForm />} />
            <Route path="/dashboard" element={<Dashboard />} />
            <Route path="/complaints" element={<ComplaintsList />} />
            <Route path="/admin" element={<Admin />} />
          </Routes>
        </main>
      </div>
    </Router>
  );
```

2. Admin Panel

The **AdminPanel** is a centralized interface for managing departments, workers, and cities within the Smart Shehri complaint management system. It also provides an overview dashboard displaying key metrics.

Key Features

Tabs (Navigation)

- **Overview**: Displays total departments, workers, cities, and complaints.
- **Departments**: View and create departments.
- **Workers**: View and add new workers.
- **Cities**: View and add new cities.

Modal System

- Dynamic form modal for creating:
  - Departments
  - Workers
  - Cities

- Validation:
  - Name, phone, and email checks
  - Regex for email/phone
  - Required dropdown selection for cities and departments

▯ Logic & Components

useApi (hook)

Used for fetching:

- Departments (departmentsAPI.getAll)
- Workers (workersAPI.getAll)
- Cities (citiesAPI.getAll)
- Complaints (complaintsAPI.getAll)

Custom Components

- **LoadingSpinner**: Shows loading indicators
- **EfficiencyBar**: Displays department efficiency visually

Dynamic Rendering

renderContent() switches between tab content based on activeTab.

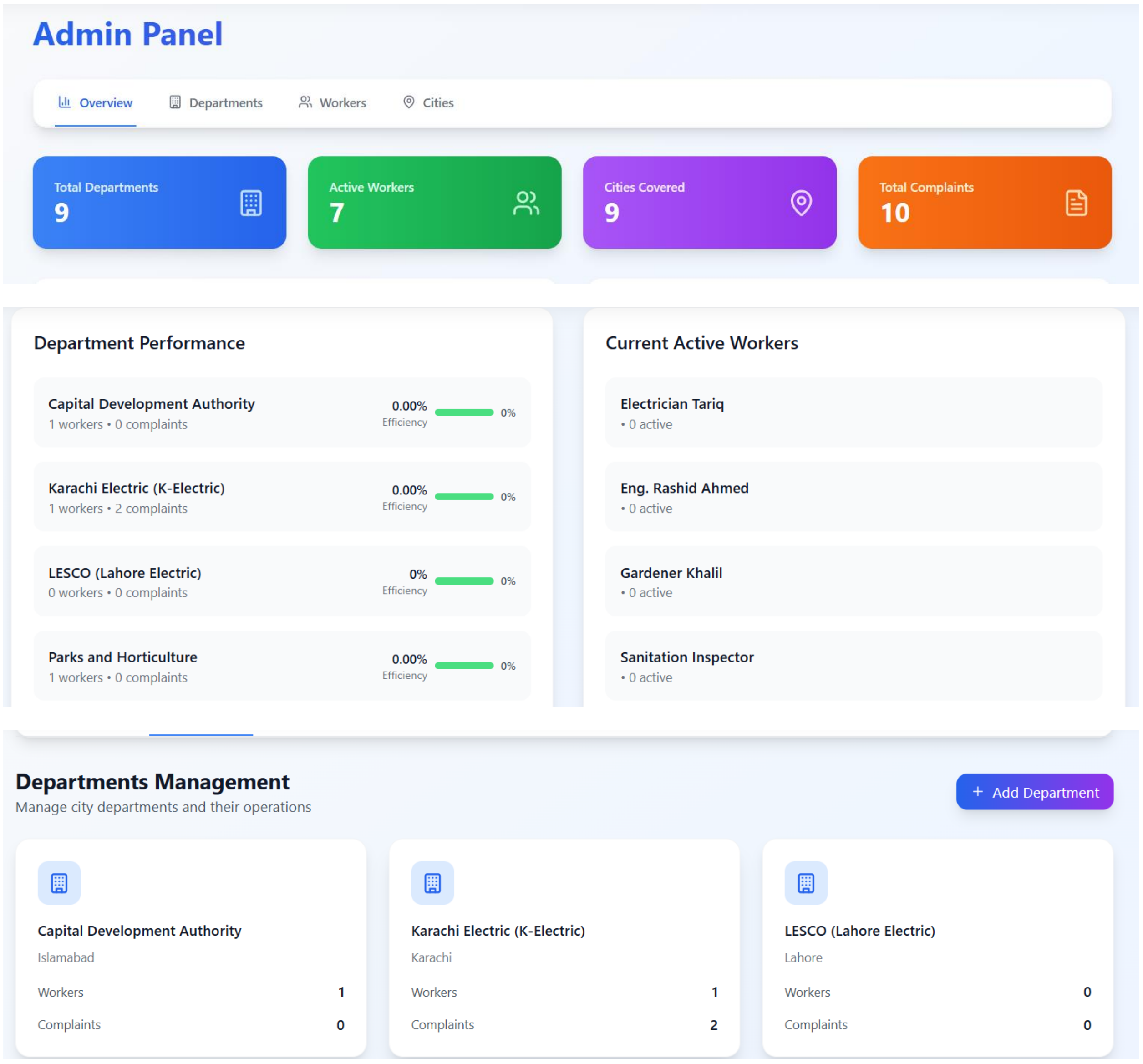
🔄 Reusable Functions

- **handleAdd(type)**: Opens the modal for adding a new entry
- **handleChange(e)**: Updates form state
- **handleSubmit(e)**: Validates and sends POST request based on the modal type

---

✅ Best Practices Used

- Controlled forms with validation
- Clean UI with meaningful color schemes and gradients
- Responsive layout with grid-based cards
- Clear separation of concerns using reusable components and hooks



Admin Panel

Total Departments	Active Workers	Cities Covered	Total Complaints
9	7	9	10

Department Performance

Capital Development Authority 1 workers • 0 complaints	0.00% Efficiency	0%
Karachi Electric (K-Electric) 1 workers • 2 complaints	0.00% Efficiency	0%
LESCO (Lahore Electric) 0 workers • 0 complaints	0% Efficiency	0%
Parks and Horticulture 1 workers • 0 complaints	0.00% Efficiency	0%

Current Active Workers

Electrician Tariq
• 0 active

Eng. Rashid Ahmed
• 0 active

Gardener Khalil
• 0 active

Sanitation Inspector
• 0 active

Departments Management
Manage city departments and their operations

	Capital Development Authority	Karachi Electric (K-Electric)	LESCO (Lahore Electric)
	Islamabad	Karachi	Lahore
Workers	1	1	0
Complaints	0	2	0

3. ComplaintForm.jsx — Multi-Step Complaint Submission Form

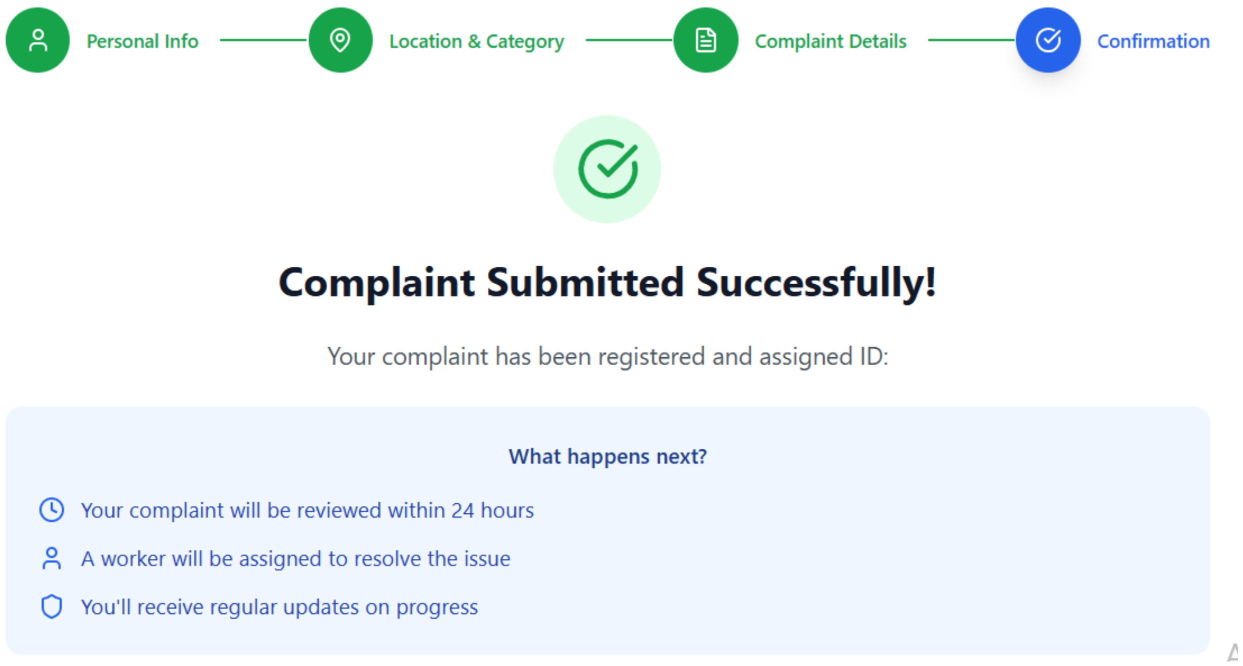
🔄 **Steps:**

1. **Personal Info**: Collects citizen name, email, and phone.
2. **Location & Category**: Lets users select city, area, and issue category.
3. **Complaint Details**: Gathers complaint title, description, and priority level.

4. **Confirmation**: Shows success or failure message with submission feedback.

 Features:

- **Stateful Step Navigation** (useState)
- **Real-time Field Validation** with meaningful feedback
- **Dynamic Dropdowns** for locations based on selected city
- **Category Cards** with icons and colors
- **Priority Selection** using styled radio cards
- **Submit Flow**:
    1. Creates citizen
    2. Submits complaint with linked citizen_id
- **Loading & Error Handling** with fallback messages and spinners



Complaint Submitted Successfully!

Your complaint has been registered and assigned ID:

What happens next?

Your complaint will be reviewed within 24 hours

A worker will be assigned to resolve the issue

You'll receive regular updates on progress

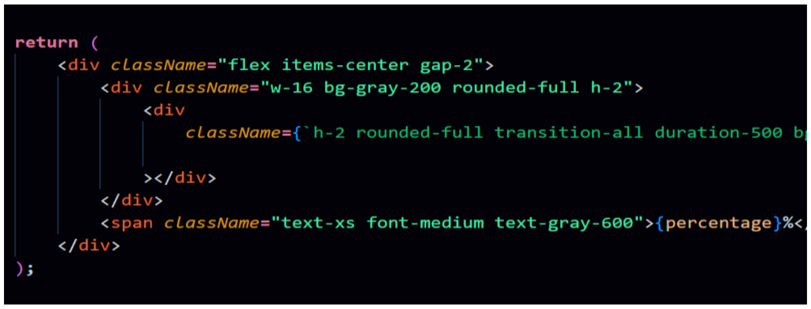
4. Components

Header.jsx

- Sticky top navigation
- Icons with route highlighting
- Navigation items: Dashboard, Submit Complaint, Complaints, Admin

EfficiencyBar.jsx

- Visual progress bar for efficiency %.

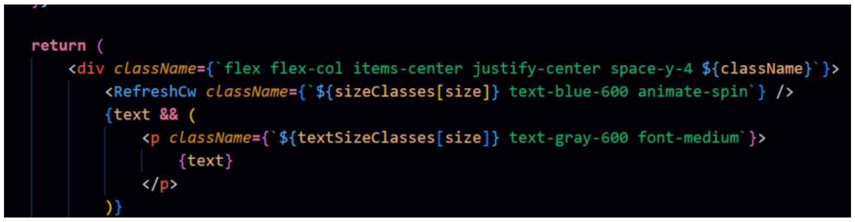


```
return (
    <div className="flex items-center gap-2">
        <div className="w-16 bg-gray-200 rounded-full h-2">
            <div
                className={`h-2 rounded-full transition-all duration-500 b

            ></div>
        </div>
        <span className="text-xs font-medium text-gray-600">{percentage}%<
    </div>
);
```

LoadingSpinner.jsx

- Configurable spinner with sizes and text.



```
  return (
    <div className={`flex flex-col items-center justify-center space-y-4 ${className}`}>
      <RefreshCw className={`${sizeClasses[size]} text-blue-600 animate-spin`} />
      {text && (
        <p className={`${textSizeClasses[size]} text-gray-600 font-medium`}>
          {text}
        </p>
      )}
```

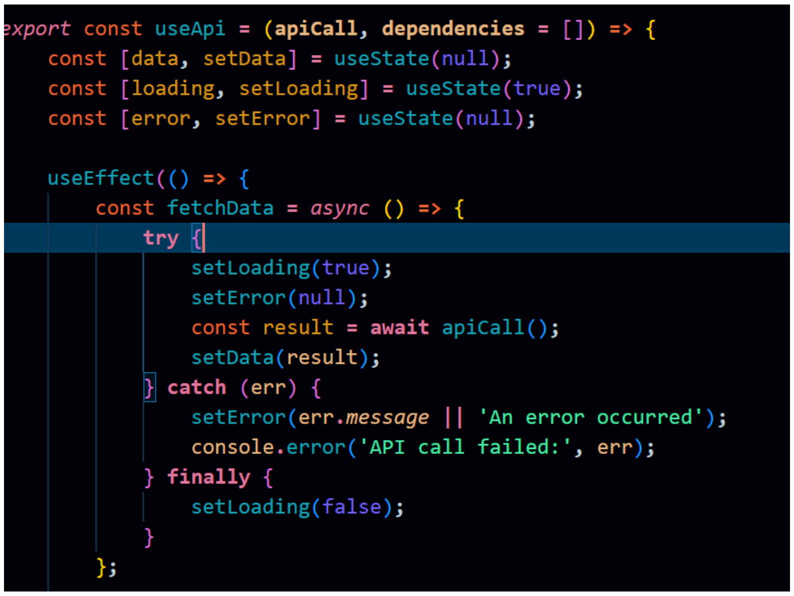
---

2. Reusable Hooks

useApi(apiCall, dependencies)

Fetches data on mount or when dependencies change.

- **Returns**: { data, loading, error, refetch }



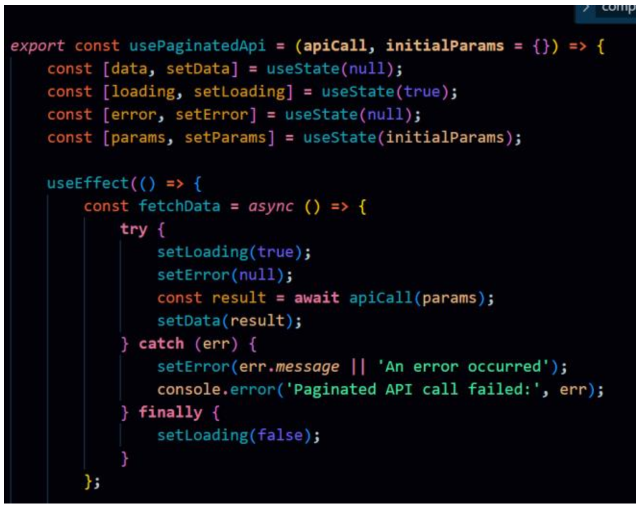
```
xport const useApi = (apiCall, dependencies = []) => {
    const [data, setData] = useState(null);
    const [loading, setLoading] = useState(true);
    const [error, setError] = useState(null);

    useEffect(() => {
        const fetchData = async () => {
            try {
                setLoading(true);
                setError(null);
                const result = await apiCall();
                setData(result);
            } catch (err) {
                setError(err.message || 'An error occurred');
                console.error('API call failed:', err);
            } finally {
                setLoading(false);
            }
        };
```

usePaginatedApi(apiCall, initialParams)

For paginated endpoints with param tracking.

- **Returns**: { data, loading, error, params, updateParams, refetch }



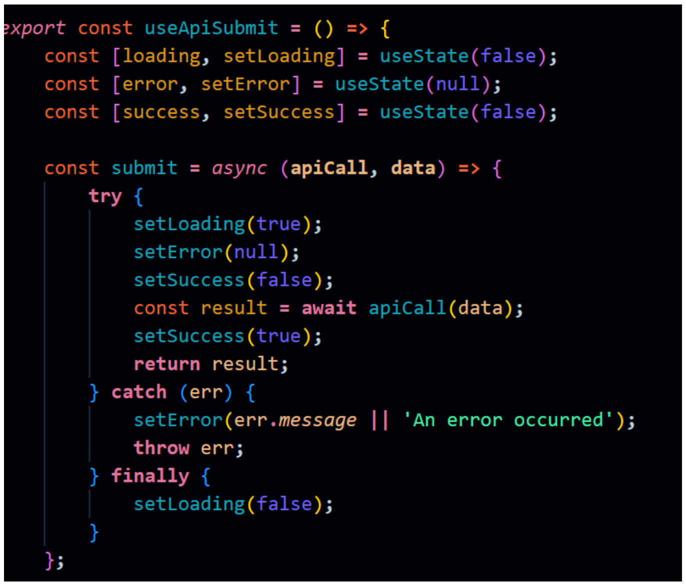
```
export const usePaginatedApi = (apiCall, initialParams = {}) => {
    const [data, setData] = useState(null);
    const [loading, setLoading] = useState(true);
    const [error, setError] = useState(null);
    const [params, setParams] = useState(initialParams);

    useEffect(() => {
        const fetchData = async () => {
            try {
                setLoading(true);
                setError(null);
                const result = await apiCall(params);
                setData(result);
            } catch (err) {
                setError(err.message || 'An error occurred');
                console.error('Paginated API call failed:', err);
            } finally {
                setLoading(false);
            }
        };
```

useApiSubmit()

Handles POST with submission feedback.

- **Returns**: { submit, loading, error, success, reset }



```
export const useApiSubmit = () => {
    const [loading, setLoading] = useState(false);
    const [error, setError] = useState(null);
    const [success, setSuccess] = useState(false);

    const submit = async (apiCall, data) => {
        try {
            setLoading(true);
            setError(null);
            setSuccess(false);
            const result = await apiCall(data);
            setSuccess(true);
            return result;
        } catch (err) {
            setError(err.message || 'An error occurred');
            throw err;
        } finally {
            setLoading(false);
        }
    };
```

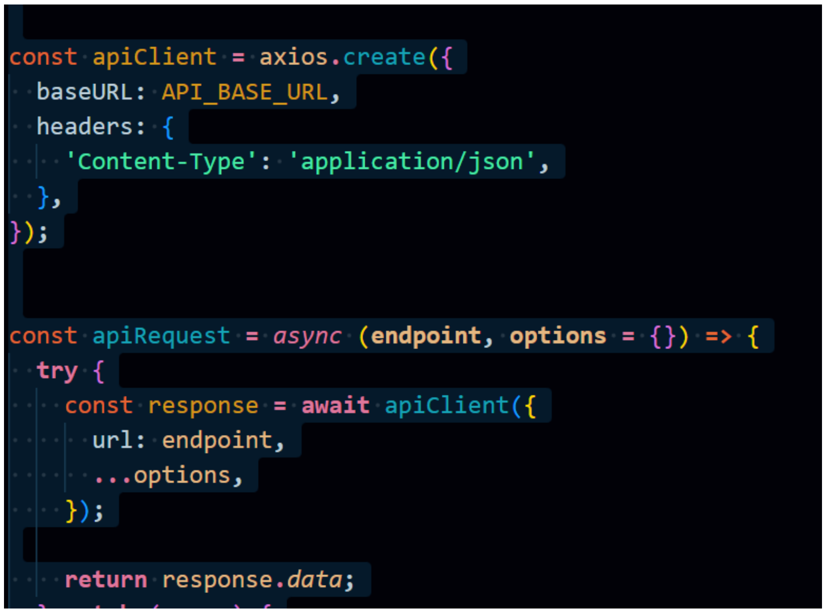
🔗 API Services

/services/api.js

- **Dashboard APIs**
    - getStats()
    - getComplaintsByCity()
    - getDepartmentRatings() etc.
- **Complaint APIs**

- getAll(params)
- getById(id)
- create(data)

- **Worker APIs**
  - getAll()
  - getAssignments(workerId)
  - create(data)
- **Feedback APIs**
  - getAll(params)
  - getByComplaint(id)
  - create(data)
- **City/Department/Category APIs** for CRUD and nested queries.



```
const apiClient = axios.create({
  baseURL: API_BASE_URL,
  headers: {
    'Content-Type': 'application/json',
  },
});


const apiRequest = async (endpoint, options = {}) => {
  try {
    const response = await apiClient({
      url: endpoint,
      ...options,
    });

    return response.data;
```

THANK YOU