RV UNIVERSITY

Go, change the world

an initiative of RV EDUCATIONAL INSTITUTIONS

CS3330 – Introduction to Full Stack Development

Complaint Management System

By

Kruthika S (1RVU23CSE228)

Nandisa Das (1RVU23CSE301)

Nishant Reddy (1RVU23CSE120)

Nikhita K Nagavar (1RVU23CSE309)

School of Computer Science and Engineering

RV University, Bangalore

2025-2026

RV UNIVERSITY

Go, change the world

an initiative of RV EDUCATIONAL INSTITUTIONS

School of Computer Science and Engineering

CERTIFICATE

Certified that the **CS3330 – Introduction to Full Stack Development** Project titled **Complaint Management System** is carried out by **Kruthika S (1RVU23CSE228), Nandisa Das (1RVU23CSE301), Nishant Reddy (1RVU23CSE120), Nikhita K Nagavar (1RVU23CSE309)**, who is a bonafide student of the School of Computer Science and Engineering, RV University, Bengaluru, during the year 2025–26. It is certified that all corrections/ suggestions from all the continuous internal evaluations have been incorporated in the project and in this report.

Faculty Guide	Program Director
Prof. Shankar M	Dr. Sudhakar K N

1. Introduction

Introduction and Context

This report presents the design, development and implementation of the Simple Complaint Management System (CMS), a full-stack web app that was created with the objective of providing organizations with a modern, efficient and transparent way to submit a complaint on behalf of their users. The proposed Simple Complaint Management System "CMS" replaces traditional fragmented interaction methods (email, paper forms) by providing an organized system for receiving and tracking complaints and ensuring better organizational accountability and user/customer satisfaction.

Problem Statement

The challenges of effectively managing an organization typically stem from an inefficient process of logging and resolving complaints. Most of the time there are complaints that simply get lost, take a long time to resolve, and have no transparency for the person making the complaint. Because of this, users become less confident in the organization's ability to respond and operate with a higher level of overhead to track complaints and prioritize effectively. The central problem that this project aims to address is the need of having one system that is secure and trackable to manage an entire complaint lifecycle from submission to final resolution.

Project Goals

The Simple Complaint Management System project's main goals were:

- To create a fully functional web application that enables the safe submission and retrieval of complaints using React and Supabase.
- To create a transparent workflow system that will allow administrators to effectively monitor, classify, and update the status of any complaints that are filed.
- To use contemporary techniques, such as Zod schema validation and a responsive interface designed with Tailwind CSS, to guarantee excellent data integrity and user experience.
- To verify system dependability through thorough functional testing and exacting Postman API testing.

2. Methodology

3.1 Project Lifecycle: The Phased Approach

The development of the Simple Complaint Management System was executed through a structured, multi-phase methodology, adapting principles of the Iterative Waterfall Model to ensure both control and flexibility.

Phase 1: Planning and Requirement Analysis

- **Goal:** Define the system scope and functional needs.
- **Activities:** Detailed analysis of user requirements (complainants and administrators), defining system actors, drafting the problem statement, and establishing key performance indicators (KPIs) like resolution time targets.
- **Deliverables:** Formal System Requirements Specification (SRS) document and a high-level system architecture blueprint.

Phase 2: Design and Architecture

- **Goal:** Translate requirements into a tangible design and select the optimal technology strategy.
- **Activities:** Development of the database schema (defining tables for Complaints, Users, and Statuses), drafting the component structure, and creating low-fidelity UI/UX mockups. Selection of the decoupled, serverless architecture was formalized here.
- **Deliverables:** Data Flow Diagrams, Entity-Relationship Diagrams (ERD), and documented technology stack rationale.

Phase 3: Implementation and Development

- **Goal:** Build the system based on the finalized design.
- **Activities:** Iterative coding of components, feature development (submission, listing, update functions), and continuous integration. Development was conducted in an environment optimized for rapid change and quick recompilation.
- **Deliverables:** Functional alpha and beta versions of the application.

Phase 4: Testing and Quality Assurance (QA)

- **Goal:** Verify system functionality, integrity, and performance.
- **Activities:** Comprehensive, multi-tiered testing (detailed in Section 3.4). Focus on bug identification, performance bottlenecks, and validation against the SRS.
- **Deliverables:** Formal Test Reports and a stable, production-ready release candidate.

3.2 Architectural and Strategic Choices

The system utilizes a decoupled architecture to achieve maximum flexibility and maintainability.

- **Decoupled Architecture:** The system maintains a clear division between the user-facing presentation layer and the managed, external data/API layer. This prevents server downtime from affecting the UI and allows for independent scaling of both components.
- **Focus on Serverless:** Leveraging a cloud-based managed relational database and API service significantly reduced the time spent on infrastructure management, allowing the project to focus 90% of effort on user-facing features and business logic.
- **Accessibility and UI/UX Focus:** Design decisions prioritized user accessibility standards and ensured a responsive, modern interface capable of adapting across desktop and mobile devices.

3.3 Data Integrity Procedures

Robust measures were implemented to ensure the reliability and security of stored data.

- **Schema Enforcement:** Every data entry point is protected by **strict schema validation**. This procedure verifies the format, type, and completeness of submitted data, preventing malformed entries from reaching the core database.
- **Optimized Data Retrieval:** A dedicated caching and synchronization layer was implemented to manage all server-side data traffic. This minimizes load on the database and provides near-instant updates to the user interface.

3.4 Rigorous Quality Management

Beyond basic functional checks, quality management included verification of both front-end and back-end logic.

- **Systematic API Testing:** All database interaction endpoints were subjected to systematic testing using a specialized platform to confirm their stability, data transaction integrity, and adherence to security policies.
- **Boundary and Edge Case Testing:** Specific test cases were developed to challenge the system at its limits, including inputting maximum character lengths, zero-data scenarios, and concurrent status updates.
- **Deployment Quality Gates:** Before pushing updates to the live environment, an automated process ensured all code passed continuous integration checks and was free of critical build errors.

3. Technology Stack

The Simple Complaint Management System is built on a modern, high-performance architecture utilizing the following key technologies:

4.1 Frontend and UI Layer

- **React (JavaScript)**
  - **What it is:** A popular, open-source JavaScript library for building user interfaces using components.
  - **Role in Project:** Used for building all dynamic and reusable components, including the complaint submission forms and the core tracking dashboard.
- **Vite**
  - **What it is:** A modern build tool that significantly improves the frontend development server and build processes.
  - **Role in Project:** Provides the lightning-fast development experience (Hot Module Replacement) and optimizes the final application bundle for production.
- **Tailwind CSS**
  - **What it is:** A utility-first CSS framework that enables developers to build custom designs rapidly.
  - **Role in Project:** Used for all low-level styling, ensuring a consistent theme, and implementing responsive design across the application.
- **shadcn/ui**
  - **What it is:** A collection of highly accessible and customizable components built on Radix UI and styled with Tailwind CSS.
  - **Role in Project:** Provides the professional, polished UI components (tables, dialogs, inputs) that give the system its clean, modern appearance.

4.2 Backend and Data Layer

- **Supabase (BaaS)**
  - **What it is:** An open-source Backend as a Service providing a suite of backend tools centered around a PostgreSQL database.

    - **Role in Project:** Functions as the complete backend, providing managed database hosting, instant auto-generated APIs, and built-in user authentication.
- **PostgreSQL**
    - **What it is:** A powerful, highly reliable open-source relational database management system.
    - **Role in Project:** The core database engine used to securely store all structured data, including complaints, user records, and status change logs.
- **Vercel**
    - **What it is:** A cloud platform focused on performance, enabling fast deployment and hosting for frontend applications.
    - **Role in Project:** Used as the hosting and CI/CD platform, automating deployment and ensuring global availability and low latency for the application.

4.3 Data Handling and Quality

- **TanStack React Query**
    - **What it is:** A library for managing, caching, and synchronizing server state within React applications.
    - **Role in Project:** Optimized data fetching, reduced unnecessary network load, and ensured the UI remains synchronized with the database in real-time.
- **Zod**
    - **What it is:** A TypeScript-first schema declaration and validation library.
    - **Role in Project:** Integrated into form handling to enforce strict validation rules, guaranteeing the integrity and correct format of all data sent to the backend.

4.4 Quality Assurance

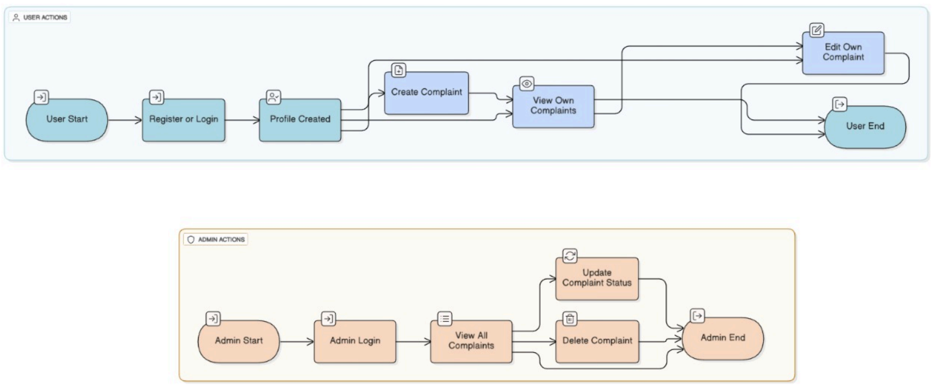
- **Postman**
    - **What it is:** A popular platform used for building, testing, and documenting APIs.
    - **Role in Project:** Utilized to systematically test all backend API endpoints (CRUD operations) out-of-browser, verifying data security policies and backend logic prior to UI integration.

4. UML Diagrams

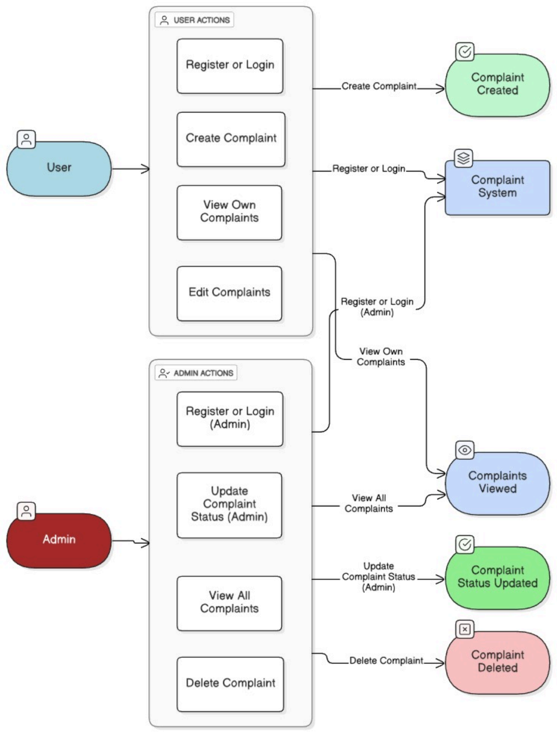
A UML diagram in the context of your Simple Complaint Management System is a vital set of visual tools used to define, design, and document the system before and during the development process.

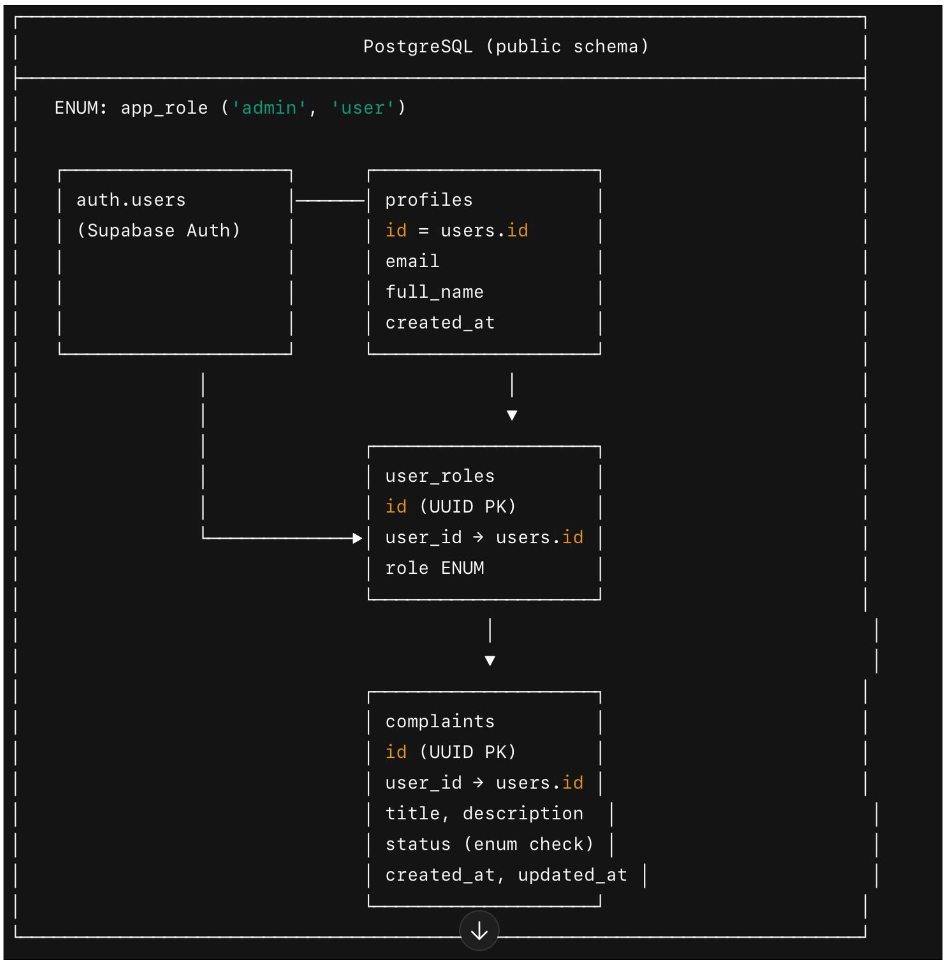
For this type of project, the UML diagrams serve two main purposes: describing the structure and illustrating the behavior of the application.

UML Diagram-1



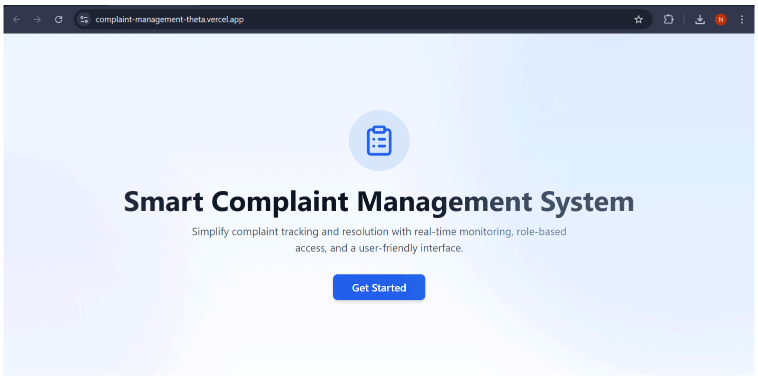
UML Diagram-2





```
┌─────────────────────────────────────────────────────────────────────────────┐
│                         PostgreSQL (public schema)                          │
├─────────────────────────────────────────────────────────────────────────────┤
│   ENUM: app_role ('admin', 'user')                                          │
│                                                                             │
│   ┌────────────────────┐      ┌────────────────────┐                        │
│   │ auth.users         │──────│ profiles           │                        │
│   │ (Supabase Auth)    │      │ id = users.id      │                        │
│   │                    │      │ email              │                        │
│   │                    │      │ full_name          │                        │
│   │                    │      │ created_at         │                        │
│   └────────────────────┘      └────────────────────┘                        │
│                │                           │                                │
│                │                           ▼                                │
│                │              ┌────────────────────┐                        │
│                │              │ user_roles         │                        │
│                │              │ id (UUID PK)       │                        │
│                └─────────────▶│ user_id → users.id │                        │
│                               │ role ENUM          │                        │
│                               └────────────────────┘                        │
│                                         │                                    │
│                                         ▼                                    │
│                               ┌────────────────────┐                        │
│                               │ complaints         │                        │
│                               │ id (UUID PK)       │                        │
│                               │ user_id → users.id │                        │
│                               │ title, description  │                        │
│                               │ status (enum check) │                        │
│                               │ created_at, updated_at │                      │
│                               └────────────────────┘                        │
└─────────────────────────────────────────────────────────────────────────────┘
```

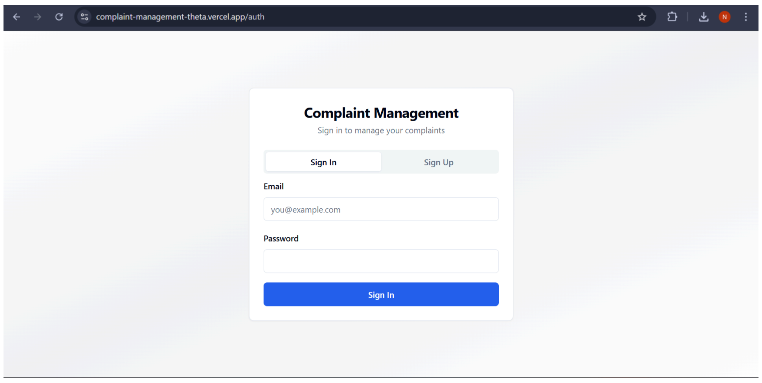
5. Results and Output


complaint-management-theta.vercel.app

Smart Complaint Management System

Simplify complaint tracking and resolution with real-time monitoring, role-based access, and a user-friendly interface.

Get Started


complaint-management-theta.vercel.app/auth

Complaint Management

Sign in to manage your complaints

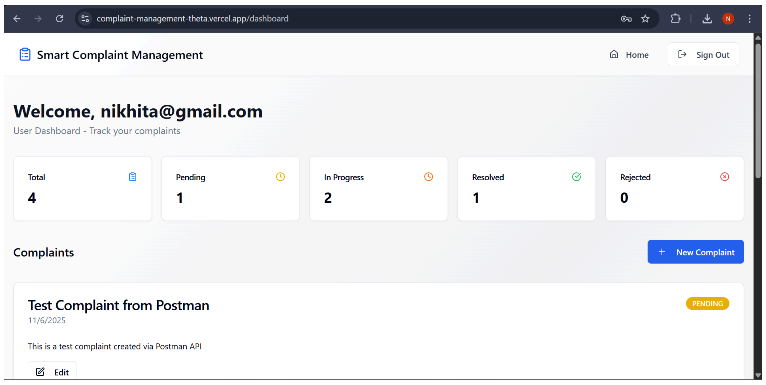
Sign In | Sign Up

Email

you@example.com

Password

Sign In


complaint-management-theta.vercel.app/dashboard

Smart Complaint Management — Home — Sign Out

Welcome, nikhita@gmail.com

User Dashboard - Track your complaints

Total	Pending	In Progress	Resolved	Rejected
4	1	2	1	0

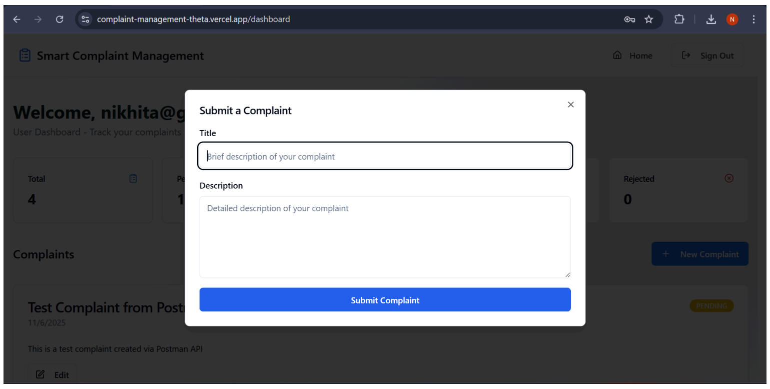
Complaints — + New Complaint

Test Complaint from Postman — PENDING

11/6/2025

This is a test complaint created via Postman API

Edit


complaint-management-theta.vercel.app/dashboard

Submit a Complaint

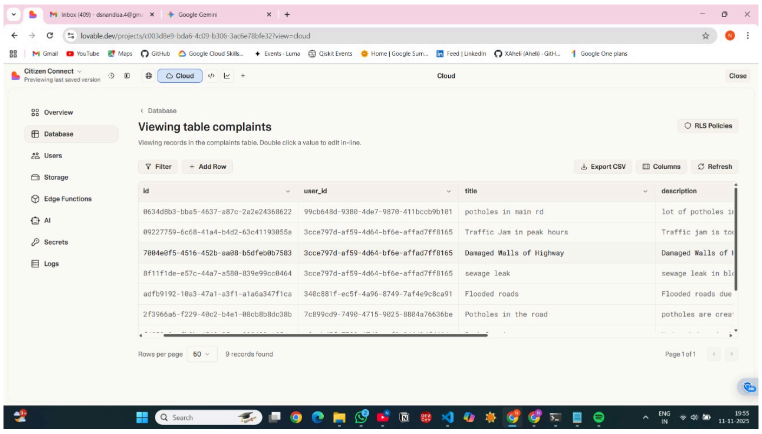
Title

Brief description of your complaint

Description

Detailed description of your complaint

Submit Complaint

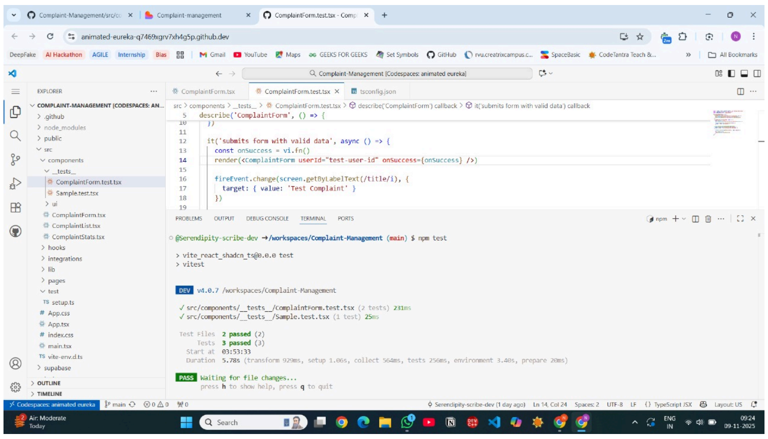


Viewing table complaints

Viewing records in the complaints table. Double click a value to edit in-line.

id	user_id	title	description
0634d8b3-bba5-4637-a87c-2a2e24368622	99cb648d-9380-4de7-9870-411bccb9b101	potholes in main rd	lot of potholes i
09227759-6c68-41a4-b4d2-63c41193055a	3cce797d-af59-4d64-bf6e-affad7ff8165	Traffic Jam in peak hours	Traffic jam is to
7004e0f5-4516-452b-aa08-b5dfeb0b7583	3cce797d-af59-4d64-bf6e-affad7ff8165	Damaged Walls of Highway	Damaged Walls of
8f11f1de-e57c-44a7-a580-839e99cc0464	3cce797d-af59-4d64-bf6e-affad7ff8165	sewage leak	sewage leak in bl
adfb9192-10a3-47a1-a3f1-a1a6a347f1ca	340c881f-ec5f-4a96-8749-7af4e9c8ca91	Flooded roads	Flooded roads due
2f3966a6-f229-40c2-b4e1-08cb8b8dc38b	7c899cd9-7490-4715-9025-8804a76636be	Potholes in the road	potholes are crea

Rows per page 50 · 9 records found · Page 1 of 1

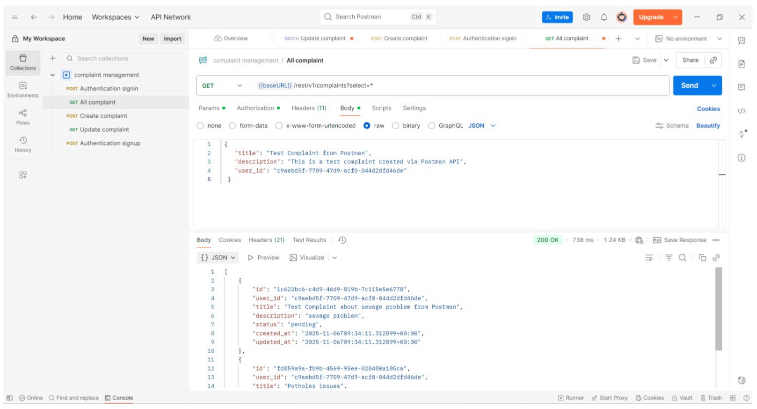


src > components > __tests__ > ComplaintForm.test.tsx > describe('ComplaintForm') callback > it('submits form with valid data') callback

```
describe('ComplaintForm', () => {
  })

  it('submits form with valid data', async () => {
    const onSuccess = vi.fn()
    render(<ComplaintForm userId="test-user-id" onSuccess={onSuccess} />)

    fireEvent.change(screen.getByLabelText(/title/i), {
      target: { value: 'Test Complaint' }
    })
```

```
@Serendipity-scribe-dev →/workspaces/Complaint-Management (main) $ npm test

> vite_react_shadcn_ts@0.0.0 test
> vitest


 DEV  v4.0.7 /workspaces/Complaint-Management

 ✓ src/components/__tests__/ComplaintForm.test.tsx (2 tests) 231ms
 ✓ src/components/__tests__/Sample.test.tsx (1 test) 25ms

 Test Files  2 passed (2)
      Tests  3 passed (3)
   Start at  03:53:33
   Duration  5.78s (transform 929ms, setup 1.06s, collect 564ms, tests 256ms, environment 3.40s, prepare 20ms)

 PASS  Waiting for file changes...
       press h to show help, press q to quit
```

Serendipity-scribe-dev (1 day ago) Ln 14, Col 24 Spaces: 2 UTF-8 LF TypeScript JSX Layout: US



complaint management / All complaint

GET {{baseURL}} /rest/v1/complaints?select=*

none · form-data · x-www-form-urlencoded · [✓ raw] · binary · GraphQL · JSON

```
{
    "title": "Test Complaint from Postman",
    "description": "This is a test complaint created via Postman API",
    "user_id": "c9aebd5f-7709-47d9-acf0-044d2dfd46de"
}
```

Body · Cookies · Headers (21) · Test Results · 200 OK · 738 ms · 1.24 KB · Save Response

```
[
    {
        "id": "1c622bc6-c4d9-46d9-819b-7c115e5e6778",
        "user_id": "c9aebd5f-7709-47d9-acf0-044d2dfd46de",
        "title": "Test Complaint about sewage problem from Postman",
        "description": "sewage problem",
        "status": "pending",
        "created_at": "2025-11-06T09:34:11.312899+00:00",
        "updated_at": "2025-11-06T09:34:11.312899+00:00"
    },
    {
        "id": "fd859a9a-fb9b-4569-95ee-020480a185ca",
        "user_id": "c9aebd5f-7709-47d9-acf0-044d2dfd46de",
        "title": "Potholes issues",
```

6. Conclusion and Future Scope

Conclusion

- **System Success:** Successfully delivered a modern, full-stack Complaint Management System that centralizes feedback and ensures process transparency.
- **Technical Achievement:** Demonstrated skill in using high-performance technologies (React, Supabase, etc.) and robust quality practices.
- **Key Impact:** Solved the problem of inefficiency by dramatically improving complaint tracking, accountability, and user satisfaction.

Future Scope

- **Advanced Features:** Implement Role-Based Access Control (RBAC) to manage user permissions.
- **Real-time Communication:** Add instant email or in-app Real-Time Notifications for status updates.
- **File Handling**: Integrate cloud storage to allow users to attach files (images/documents) to complaints.
- **Analytics:** Develop a dedicated dashboard for administrators to track complaint trends and resolution metrics.