



**CSE471: System Analysis and Design
Project Report**
Project Title: Nirapod

Group No: 08, CSE471 Lab Section: 09, Fall 2024	
ID	Name
22101109	Saima Sobahan
22101182	Tawhid Chowdhury
22341082	Md Sakib Sadman Badhon

Submission Date: 11-05-25

Table of Contents

1. System Request.....	3
Business need:.....	3
Business requirements:.....	3
Business value:.....	4
Special issues or constraints:.....	4
2. Functional Requirements.....	5
3. Technology (Framework, Languages).....	5
4. Backend Development.....	6
5. User Interface Design.....	12
6. Frontend Development.....	15
7. User Manual.....	22
8. Performance and Network Analysis.....	39
9. Github Repo [Public] Link.....	40
10. Link of Deployed Project.....	40
11. Individual Contribution.....	41
12. References.....	42

● System Request

● Business Need

In Bangladesh, there is currently no unified digital platform where citizens can access essential emergency and civic services under one umbrella. When crimes, acts of vandalism, or other social issues occur, people often resort to posting about them on popular social media platforms. While some of these posts may go viral and attract public attention, they are quickly forgotten as new issues emerge. This cycle leads to unresolved cases and a growing backlog, with no structured follow-up or accountability.

This highlights the urgent need for a dedicated platform where the general public can directly connect with key authorities such as the police, fire service, city corporations, and animal welfare departments. A platform that ensures transparency, accountability, and timely communication, where the public can see how authorities are responding and track progress in real time.

Citizens currently face significant barriers in accessing fast and effective support from emergency and civic services. Our proposed system, **Nirapod**, aims to bridge this gap by offering a centralized digital solution for lodging complaints, sharing real-time updates, and fostering meaningful engagement between citizens and authorities. This platform aspires to enhance public trust, improve service response times, and ultimately contribute to a safer, more connected society.

● Business Requirements

- Secure login/signup with OTP and Google Auth.
- Citizens can lodge complaints with location, photos, and track case progress.
- Timeline for public posts, comments, follow/unfollow posts, and reporting to the admin.
- Live chat for real-time interaction with authorities.
- Privileged users (e.g., police) can update case statuses and search citizens using NID, passport, or driving license.
- Admin can review reported content and manage users/posts.

- **Business Value**

- Faster emergency response through real-time complaint tracking and location sharing.
- Increased transparency between citizens and authorities.
- Enhanced community engagement and trust via timeline and public discussions.
- Improved operational efficiency for authorities by automating complaint intake and updates.

- **Special Issues or Constraints**

- Must comply with data privacy laws and government regulations for handling personal ID information (NID, passport, etc.).
- OTP delivery and live chat depend on third-party API reliability.
- Needs robust moderation tools to manage abuse and fake reporting.
The application must be mobile-responsive and perform reliably under heavy usage in critical situations.
- Need government approval and government-appointed officials to successfully run the system

● Functional Requirements

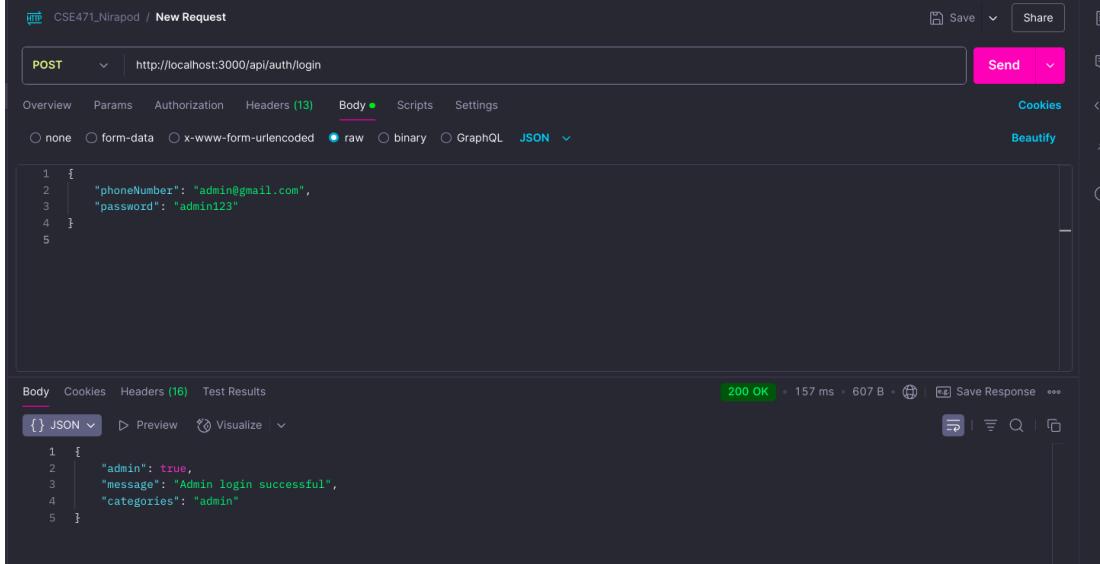
1. **User Authentication & Verification** (Signup/Login/Logout with multi-step verification)
2. **User Profile Management** (View, update profile details, change password)
3. **Community Service & Forum** (Discussion on social issues through comments and photos)
4. **Crime Prevention & Neighborhood Watch** (Crime and suspicious activity reporting)
5. **Emergency Service Requests** (Fire, police requests with live location sharing)
6. **Case Progress Tracking** (Assigned officer details, status updates)
7. **Public Complaints & Evidence Submission** (Crime reports with photo/video uploads)
8. **Alerts** (Notifications on comments and updates)
9. **Animal Shelter & Protection** (Reporting stray/injured animals, shelter integration)
10. **City Corporation Updates** (Communication with Local government in forum posts on civic matters)
11. **Community Chat for General Discussions** (Non-emergency discussions and help)
12. **External API Integration** (Maps API for live location tracking / Google Auth and OTP)
13. **Report** (Report the post to the admin)

● Technology (Framework, Languages)

- **Backend:** Spring Boot
- **Frontend:** React.js
- **Database:** PostgreSQL
- **ORM:** Hibernate (JPA)
- **Authentication:** Spring Security + JWT
- **Deployment:** Render (Backend), Netlify (Frontend), Neon (Database)
- **External API:** Leaflet (for live location tracking) / Google App (for Google Sign-in and OTP verification)

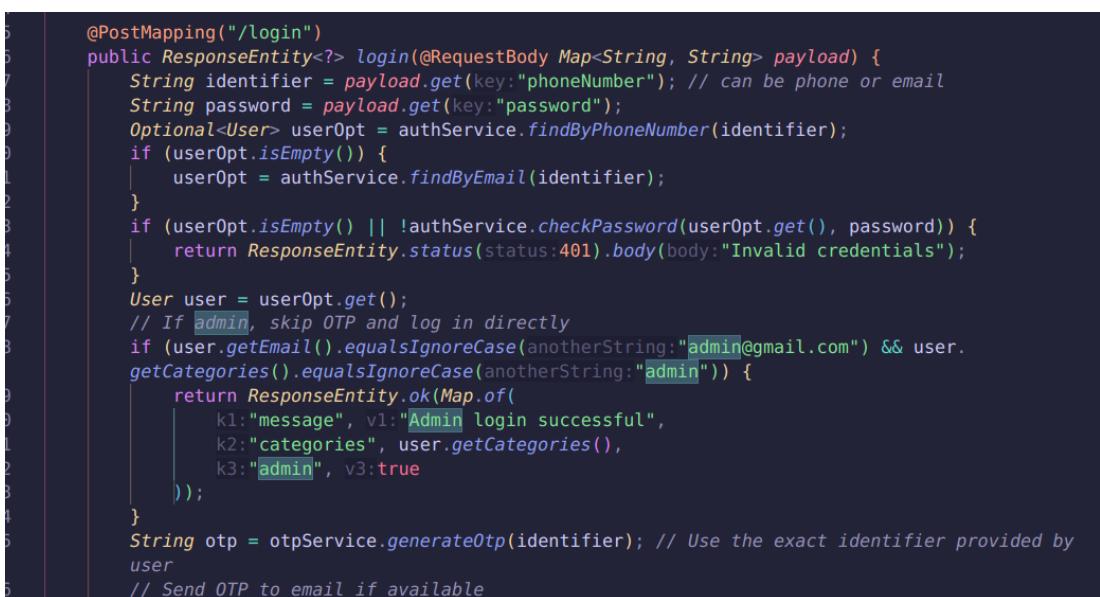
● Backend Development

1. **Login** - The Nirapod system defines three types of users: Normal User, Privileged User, and Administrator. All users, except the hardcoded administrator, require OTP (One-Time Password) verification during the sign-in process. For the administrator, a direct POST request is made to the admin sign-in endpoint. As the administrator credentials are hardcoded, no OTP verification is needed. Testing confirmed that the admin sign-in process functions as expected.



The screenshot shows a Postman request to `http://localhost:3000/api/auth/login`. The request method is `POST` and the body is a JSON object with `phoneNumber` and `password` fields. The response status is `200 OK` with a message indicating successful login for an admin user.

```
1 {  
2   "phoneNumber": "admin@gmail.com",  
3   "password": "admin123"  
4 }  
5  
  
1 {  
2   "admin": true,  
3   "message": "Admin login successful",  
4   "categories": "admin"  
5 }
```



```
5     @PostMapping("/login")  
6     public ResponseEntity<?> login(@RequestBody Map<String, String> payload) {  
7         String identifier = payload.get(key:"phoneNumber"); // can be phone or email  
8         String password = payload.get(key:"password");  
9         Optional<User> userOpt = authService.findByPhoneNumber(identifier);  
10        if (userOpt.isEmpty()) {  
11            userOpt = authService.findByEmail(identifier);  
12        }  
13        if (userOpt.isEmpty() || !authService.checkPassword(userOpt.get(), password)) {  
14            return ResponseEntity.status(status:401).body(body:"Invalid credentials");  
15        }  
16        User user = userOpt.get();  
17        // If admin, skip OTP and log in directly  
18        if (user.getEmail().equalsIgnoreCase(anotherString:"admin@gmail.com") && user.  
19            getCategories().equalsIgnoreCase(anotherString:"admin")) {  
20            return ResponseEntity.ok(Map.of(  
21                k1:"message", v1:"Admin login successful",  
22                k2:"categories", user.getCategories(),  
23                k3:"admin", v3:true  
24            ));  
25        }  
26        String otp = otpService.generateOtp(identifier); // Use the exact identifier provided by  
27        user  
28        // Send OTP to email if available
```

2. **Complaints** - After logging in, a user can fetch all available complaints and display them in a timeline view. Each complaint includes all stored details associated with it. The complaint feed may vary depending on the user's role and permissions. Some privileged users may have restricted access and may not be able to view certain users complaints. When logged in with an administrator account, full access is granted to all complaints and posts without any restrictions.

```

// Fetch all complaints
@GetMapping("/complaints")
public List<ComplaintResponse> getAllComplaints(
    @RequestHeader(value = "X-User-Category", required = false) String userCategory) {
    logger.info(msg:"Fetching all complaints");
    List<Complaint> complaints;
    if (userCategory != null && userCategory.equalsIgnoreCase("police")) {
        logger.info(msg:"User is police, returning only police complaints");
        complaints = repository.findByComplainToIgnoreCase(complainTo:"Police");
    }
    // ##Do not change plz (Faishal)
    else if (userCategory != null && userCategory.equalsIgnoreCase("fire")) {
        logger.info(msg:"User is fire, returning only fire complaints");
        complaints = repository.findByComplainToIgnoreCase(complainTo:"Fire");
    }

    else if (userCategory != null && userCategory.equalsIgnoreCase("city")) {
        logger.info(msg:"User is fire, returning only fire complaints");
        complaints = repository.findByComplainToIgnoreCase(complainTo:"City");
    }

    else if (userCategory != null && userCategory.equalsIgnoreCase("animal")) {
        logger.info(msg:"User is fire, returning only fire complaints");
        complaints = repository.findByComplainToIgnoreCase(complainTo:"Animal");
    }
}

```

3. **Reported** - When a user reports a complaint for violating community guidelines, the report is sent directly to the administrator for review. Only the administrator has access to these reports. Regular and privileged users cannot view reported complaints. It is the administrator's responsibility to verify each report and take appropriate action based on the severity and validity of the claim.

```

// Fetch all reported complaints
@GetMapping("/complaints/reported")
public List<ComplaintResponse> getReportedComplaints() {
    List<Complaint> complaints = repository.findAll();
    List<ComplaintResponse> responseList = new ArrayList<>();
    for (Complaint c : complaints) {
        if (c.getReport() != null && c.getReport().trim().isEmpty()) {
            String userName = userRepository.findById(c.getNid())
                .map(u -> u.getName())
                .orElse("");
            responseList.add(new ComplaintResponse(
                c.getTrackingId(),
                userName,
                c.getUrgency(),
                c.getComplainTo(),
                c.getDistrict(),
                c.getArea(),
                c.getTags(),
                c.getDetails(),
                c.getPhotos(),
                c.isPostOnTimeline(),
                c.getLocation(),
                c.getUpdateNote(),
                c.getStatus(),
                c.getFollow(),
                c.getComment(),
                c.getTime(),
                c.getReport() // Add report field for frontend
            ));
        }
    }
    return responseList;
}

```

4. **Investigate** - Using this feature, certain users (privileged users) are authorized to access information stored in the project's database. Privileged users can retrieve user details by searching with one of the following identification numbers: National ID (NID), Passport Number, Driving License Number. In the current scenario, the National ID (NID) is being used to fetch user information.

```

1  "utilityBillPhoto": "/uploads/1746778701770_038_b.png",
2  "userPhoto": "/uploads/1746778701772_065_a.png",
3  "passport": "aaaaaa",
4  "presentAddress": "adabor",
5  "phone": "01533024585",
6  "nidPhoto": "/uploads/1746778701773_098_a.png",
7  "drivingLicence": "",
8  "name": "badhon",
9  "name": "badhon",
10 "nid": "111111111",
11 "permanentAddress": "adabor",
12 "email": "simplelogin-newsletter.exndu@simplelogin.com",
13 "utilityBillId": "111111111"

```

```

@GetMapping("/search")
public ResponseEntity<?> searchUser(
    @RequestParam(required = false) String nid,
    @RequestParam(required = false) String drivingLicence,
    @RequestParam(required = false) String passport) {
    Optional<User> userOpt = Optional.empty();      You, yesterday * Almost fixed everything ...
    if (nid != null && !nid.isEmpty()) {
        userOpt = userRepository.findById(nid);
    } else if (drivingLicence != null && !drivingLicence.isEmpty()) {
        userOpt = userRepository.findByDrivingLicense(drivingLicence);
    } else if (passport != null && !passport.isEmpty()) {
        userOpt = userRepository.findByPassport(passport);
    }
    if (userOpt.isPresent()) {
        User u = userOpt.get();
        Map<String, Object> result = new HashMap<>();
        result.put(key:"name", u.getName());
        result.put(key:"nid", u.getNid());
        result.put(key:"passport", u.getPassport());
        result.put(key:"drivingLicence", u.getDrivingLicense());
        result.put(key:"presentAddress", u.getPresentAddress());
        result.put(key:"permanentAddress", u.getPermanentAddress());
        result.put(key:"email", u.getEmail());
        result.put(key:"phone", u.getPhone());
        result.put(key:"utilityBillID", u.getUtilityBillCustomerID());
        result.put(key:"utilityBillPhoto", u.getUtilityBillPhoto());
        result.put(key:"userPhoto", u.getUserPhoto());
        result.put(key:"nidPhoto", u.getNidPhoto());
        return ResponseEntity.ok(result);
    } else {
        return ResponseEntity.status(status:404).body(body:"User not found");
    }
}

```

5. **Update Profile** - Users have the ability to update their profile information. For example, in the first screenshot, both the current and permanent addresses are set to "Adabor." To update the permanent address, a request was made to change it to cahnged_add_postman. In the second screenshot, the update is reflected successfully, showing the new permanent address.

12347777	normal	badhon495@gmail.com	Text adabor	\$2a\$10
\$RV0s1DqXq4ic79RCAGgJL.eP7NeMH1h8GDcH.TADFgQVGnmcuaA1C	Md Sakib Sadman Badhon			
01533024584	adabor	adabor	Text 1244444	/uploads/1746

PUT http://localhost:3000/api/user/12347777/profile

Overview Params Authorization Headers (13) Body Scripts Settings

Body (form-data)

phone	Text	01533024584
presentAddress	Text	cahngeed_add_postman
permanentAddress	Text	adabor
utilityBillCustomerId	Text	1244444
passport	Text	
drivingLicense	Text	

Body Cookies Headers (15) Test Results

{ JSON Preview Visualize }

Profile updated successfully

200 OK 34 ms 540 B Save Response

Send

```
12347777 | normal | badhon495@gmail.com | $2a$10
$RV0s1DqXq4ic79RCAGgJL.eP7NeMH1h8GDch.TADFgQVGnmcuaA1C | Md Sakib Sadman Badhon
| 01533024584 | cahngeed_add_postman | adabor | |
| | | 1244444 Profile | |
1746925151865_046_a.png | /uploads/1746925151868_034_a.png | /uploads/1746925151
:|
```

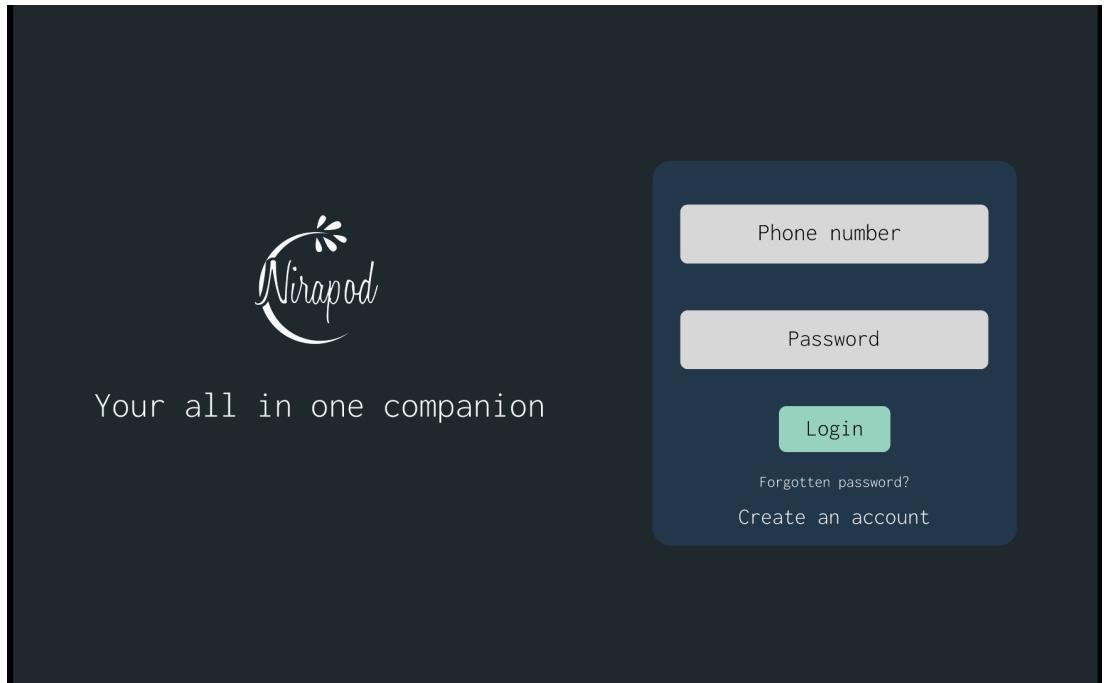
```

    @PutMapping("/{nid}/profile")
    public ResponseEntity<?> updateProfile(
        @PathVariable String nid,
        @RequestParam(required = false) String name,
        @RequestParam(required = false) String email,
        @RequestParam(required = false) String phone,
        @RequestParam(required = false) String presentAddress,
        @RequestParam(required = false) String permanentAddress,
        @RequestParam(required = false) String utilityBillCustomerId,
        @RequestParam(required = false) String passport,
        @RequestParam(required = false) String drivingLicense,
        @RequestParam MultipartFile photo,
        @RequestParam MultipartFile utilityBillPhoto,
        @RequestParam MultipartFile passportImg,
        @RequestParam MultipartFile drivingLicenseImg
    ) throws IOException {
        Optional<User> userOpt = userRepository.findByNid(nid);
        if (userOpt.isEmpty()) return ResponseEntity.status(HttpStatus.NOT_FOUND).body("User not found");
        User user = userOpt.get();
        if (name != null) user.setName(name);
        if (email != null) user.setEmail(email);
        if (phone != null) user.setPhone(phone);
        if (presentAddress != null) user.setPresentAddress(presentAddress);
        if (permanentAddress != null) user.setPermanentAddress(permanentAddress);
        if (utilityBillCustomerId != null) user.setUtilityBillCustomerId(utilityBillCustomerId);
        if (passport != null) user.setPassport(passport);
        if (drivingLicense != null) user.setDrivingLicense(drivingLicense);
        // Save files if present
        Path dirPath = Paths.get(uploadDir);
        if (!Files.exists(dirPath)) Files.createDirectories(dirPath);
        if (photo != null && !photo.isEmpty()) {
            String filename = System.currentTimeMillis() + "_photo_" + photo.getOriginalFilename();
            Path filePath = dirPath.resolve(filename);
            photo.transferTo(filePath);
            user.setUserPhoto("/uploads/" + filename);
        }
        if (utilityBillPhoto != null && !utilityBillPhoto.isEmpty()) {
            String filename = System.currentTimeMillis() + "_utilitybill_" + utilityBillPhoto.getOriginalFilename();
        }
        // Save files if present
        Path dirPath = Paths.get(uploadDir);
        if (!Files.exists(dirPath)) Files.createDirectories(dirPath);
        if (photo != null && !photo.isEmpty()) {
            String filename = System.currentTimeMillis() + "_photo_" + photo.getOriginalFilename();
            Path filePath = dirPath.resolve(filename);
            photo.transferTo(filePath);
            user.setUserPhoto("/uploads/" + filename);
        }
        if (utilityBillPhoto != null && !utilityBillPhoto.isEmpty()) {
            String filename = System.currentTimeMillis() + "_utilitybill_" + utilityBillPhoto.getOriginalFilename();
            Path filePath = dirPath.resolve(filename);
            utilityBillPhoto.transferTo(filePath);
            user.setUtilityBillPhoto("/uploads/" + filename);
        }
        if (passportImg != null && !passportImg.isEmpty()) {
            String filename = System.currentTimeMillis() + "_passport_" + passportImg.getOriginalFilename();
            Path filePath = dirPath.resolve(filename);
            passportImg.transferTo(filePath);
            user.setPassportImg("/uploads/" + filename);
        }
        if (drivingLicenseImg != null && !drivingLicenseImg.isEmpty()) {
            String filename = System.currentTimeMillis() + "_dl_" + drivingLicenseImg.getOriginalFilename();
            Path filePath = dirPath.resolve(filename);
            drivingLicenseImg.transferTo(filePath);
            user.setDrivingLicenseImg("/uploads/" + filename);
        }
        userRepository.save(user);
        return ResponseEntity.ok().body("Profile updated successfully");
    }
}

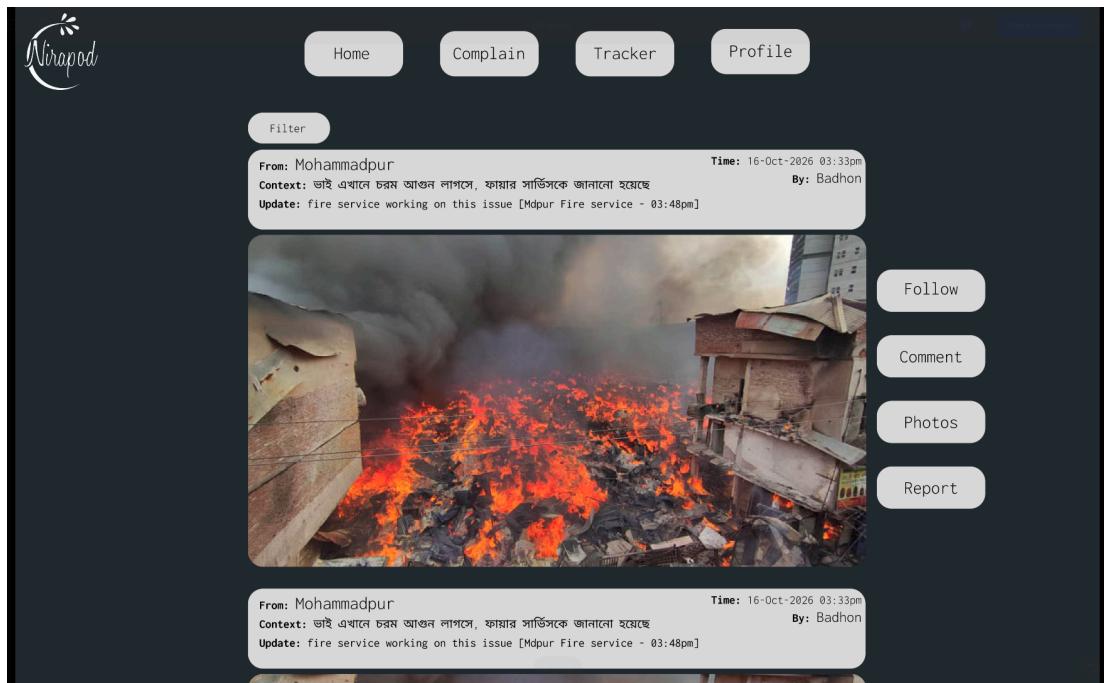
```

● User Interface Design

1. Figma Project Link - [Nirapod Figma](#)
2. Login



3. Homepage



4. Create Complaint

The screenshot shows the 'Nirapod' app interface for creating a new complaint. At the top, there is a navigation bar with four buttons: 'Home', 'Complain', 'Tracker', and 'Profile'. The 'Complain' button is highlighted. Below the navigation bar, there are several input fields for entering complaint details:

- Complain by :** Joshim Khan (NID Number)
- Complain to :** Bangladesh Police (Mdpur Zone)
- Urgency :** Low *See Level
- Tag :** Phone Theft, Hijack
- Details :** While returning from my office I got hijacked near Shia Masjid at 10:30 PM. My device is : Poco X2 IMEI Number is : 33333333 333333

At the bottom of the form, there are two buttons: 'Photos : Add photo' and 'Post it on timeline'.

5. Tracker

The screenshot shows the 'Nirapod' app interface for tracking a complaint. At the top, there is a navigation bar with four buttons: 'Home', 'Complain', 'Tracker', and 'Profile'. The 'Tracker' button is highlighted. Below the navigation bar, there is a tracking ID field:

Tracking ID : 223341

Below the tracking ID, there is a detailed view of the complaint information:

- Complain ID:** 223341
- Complain By:** Joshim Khan
- Complain To:** Bangladesh Police (Mdpur Division)
- Complain Tag:** Hijack, Phone Theft
- Complain Subject:** Phone Hijack
- Complain Details:** While returning from my office I got hijacked near Shia Masjid at 10:30 PM. My device is : Poco X2 IMEI Number is : 33333333 333333
- Update:** We are working on it.

6. Complain List

The screenshot shows a list of three complaints on the Nirapod platform. Each entry includes the Tracking ID, Complain Subject, Complain Tag, Complain Time, Urgency level, and a status indicator (Unsolved or Solved) in a red or green rounded rectangle. A 'Details' button is also present in each entry.

- Tracking ID :** 223341
Complain Subject : Phone Hijack
Complain Tag : Hijack, Phone Theft
Complain Time : 28-10-2025
Urgency : Medium
Status: Unsolved
- Tracking ID :** 22323
Complain Subject : Someone harassed a woman
Complain Tag : Harassment
Complain Time : 8-10-2025
Urgency : High
Status: Solved
- Tracking ID :** 223331
Complain Subject : Phone
Complain Tag : Hijack
Complain Time : 2-10-2025
Urgency : Medium
Status: Unsolved

7. Investigate

The screenshot shows the investigation interface for a specific individual. It features a search bar with 'Shahrukh Khan' and '23545555555555' entered, and a 'Search' button. Below the search results, a card displays the individual's details, including a profile picture, name, address, phone number, NID, and e-mail.

Looking for : Shahrukh Khan

NID : 23545555555555

Search

Details :

Shahrukh Khan	
Address :	Banani
Phone :	01900000000
NID :	23545555555555
E-mail :	shahrukh@gmail.com

● Frontend Development

- **Home.js** - The Home component manages the core logic for displaying and interacting with posts on the homepage. It uses various useState hooks to handle different aspects of the component's behavior. The posts state stores the list of displayed posts, while filters hold user-defined criteria for filtering posts, such as area, urgency, district, tags, and date ranges. The page and hasMore states are used to support pagination and infinite scrolling, while loading prevents multiple simultaneous fetches. Popups for commenting, viewing photos, and reporting are managed using openComment, openPhotos, and openReport, respectively. User interactions are tracked with states like commentInput, photoFiles for photo uploads, and followed for followed posts. commentUserNames stores user data for comments. The sortOpen state controls the visibility of sorting options. React's useRef is used to create an observer for infinite scrolling, as well as filterBtnRef and filterDropdownRef to manage dropdown visibility. The user's identifier is retrieved from local storage (user Nid), and the useLocation hook from React Router is used to detect route changes.

```
13
14  function Home() {
15    const [posts, setPosts] = useState([]);
16    const [filters, setFilters] = useState({ area: '', urgency: '', district: '', tags: '', fromDate: '', toDate: '' });
17    const [page, setPage] = useState(0);
18    const [hasMore, setHasMore] = useState(true);
19    const [loading, setLoading] = useState(false);
20    const [openComment, setOpenComment] = useState(null);
21    const [commentInput, setCommentInput] = useState('');
22    const [commentUserNames, setCommentUserNames] = useState({});
23    const [openPhotos, setOpenPhotos] = useState(null);
24    const [photoFiles, setPhotoFiles] = useState([]);
25    const [openReport, setOpenReport] = useState(null);
26    const [followed, setFollowed] = useState([]);
27    const [sortOpen, setSortOpen] = useState(false);
28    const observer = useRef();
29    const userNid = localStorage.getItem('nirapod_identifier');
30    const filterBtnRef = useRef(null);
31    const filterDropdownRef = useRef(null);
32    const location = useLocation();
```

- **Navbar.js** - In Navbar.js, the users are categorized for different layouts. The categories are checked as ‘police’, ‘fire’, ‘animal’, and ‘city’. They respectively indicate the Police, Fire Service, Animal Protection & Welfare, and City Corporation. For all the users, there are common buttons like home, complaints, and a special button for police, “Investigate”. Upon clicking the buttons, the pages are redirected to the specific pages. There is a dropdown menu named ‘profile’ which has the following menus: Update Profile, Your

Complaints, Live Chat, and Logout. Upon clicking the pages are displayed.

```
77 if (categories === 'police') {
78   return (
79     <nav className="navbar-custom">
80       <div className="navbar-logo-box" onClick={() => navigate('/home')} style={{ cursor: 'pointer' }}>
81         <img src={logo} alt="Mirapod Logo" className="navbar-logo-img" />
82       </div>
83       <div className="navbar-btn-group">
84         <a href="/home" className="navbar-btn">Home</a>
85         <a href="/complains" className="navbar-btn">Complains</a>
86         <a href="/investigate" className="navbar-btn">Investigate</a>
87         <div className="navbar-profile-dropdown" ref={dropdownRef}>
88           <button className="navbar-btn" onClick={() => setDropdownOpen(v => !v)}>
89             Profile {unreadCount > 0 && <span className="navbar-notification-dot"></span>} <span style={{marginLeft: 6}}>▼</span>
90           </button>
91           {dropdownOpen && (
92             <div className="navbar-dropdown-menu">
93               <Link to="/profile" className="navbar-dropdown-item" onClick={() => setDropdownOpen(false)}>Update Profile</Link>
94               <Link to="/my-complains" className="navbar-dropdown-item" onClick={() => setDropdownOpen(false)}>Your Complaints</Link>
95               <NotificationLink />
96               <Link to="/livechat" className="navbar-dropdown-item" onClick={() => setDropdownOpen(false)}>Live Chat</Link>
97               <button className="navbar-dropdown-item" onClick={handleLogout}>Logout</button>
98             </div>
99           )
100         </div>
101       </div>
102       <div className="navbar-welcome-msg">Welcome Officer, {userName}</div>
103     </nav>
104   );
105 }
106
107
108 }
```

- **ComplaintDetails.js** - The ComplaintDetails component displays detailed information about a specific complaint based on a tracking ID obtained from the URL parameters. It fetches complaint data using a service (ComplaintService.getComplaintById) and conditionally displays user information, complaint attributes (like tags, urgency, district, area, photos, and status), and a map using Leaflet that marks the reported location with a custom icon. If the user is an admin (determined via localStorage), they are given options to delete either the complaint post or its report. The component also integrates the UpdateComplaint form for modifying complaint details. Images and user photos are rendered with fallback handling, and timestamps are formatted into a readable local string. The map is dynamically loaded when the component detects valid GPS coordinates in the complaint's location field. Overall, this component serves as a comprehensive view and management interface for individual complaint records in the system.

```

JS ComplaintDetails.js X
frontend > src > pages > JS ComplaintDetails.js > ComplaintDetails
11
12 function ComplaintDetails() {
13   const { id } = useParams();
14   const navigate = useNavigate();
15   const [complaint, setComplaint] = useState(null);
16   const [loading, setLoading] = useState(true);
17   const [error, setError] = useState(null);
18   const [userName, setUserName] = useState('');
19
20 // Admin-only actions
21   const isAdmin = localStorage.getItem('categories') === 'admin';
22
23   const handleDeleteReport = async () => {
24     if (!window.confirm('Are you sure you want to delete the report for this post?')) return;
25     try {
26       await axios.put('/api/complaint/update/${complaint.trackingId}', { report: '' });
27       setComplaint({ ...complaint, report: '' });
28       alert('Report deleted successfully.');
29     } catch (err) {
30       alert('Failed to delete report.');
31     }
32   };
33
34   const handleDeletePost = async () => {
35     if (!window.confirm('Are you sure you want to delete this post? This action cannot be undone.')) return;
36     try {
37       await axios.delete('/api/complaint/by-id/${complaint.trackingId}');
38       alert('Post deleted successfully.');
39       navigate('/reports');
40     } catch (err) {
41       alert('Failed to delete post.');
42     }
43   };
44
45   const fetchComplaintDetails = async () => {
46     try {
47       setLoading(true);
48       const data = await ComplaintService.getComplaintById(id);
49       setComplaint(data);
50       setLoading(false);
51     } catch (err) {
52       setError('Failed to fetch complaint details. Please try again later.');
53     }
54   };
55
56   const handleUpdateSuccess = (updatedComplaint) => {
57     setComplaint(updatedComplaint);
58     alert('Complaint updated successfully!');
59   };
60
61   const handleBackToList = () => {
62     navigate('/complaints');
63   };
64
65   useEffect(() => {
66     if (complaint && complaint.nid) {
67       axios.get(`api/user/by-identifier?values=${complaint.nid}`)
68         .then(res => setUserName(res.data.name))
69         .catch(() => setUserName(''));
70     }
71   }, [complaint]);
72
73   const handleUpdateSuccess = (updatedComplaint) => {
74     setComplaint(updatedComplaint);
75     alert('Complaint updated successfully!');
76   };
77
78   const handleBackToList = () => {
79     navigate('/complaints');
80   };
81
82   useEffect(() => {
83     const location = complaint?.location;
84     if (!location) return;
85
86     const mapContainer = document.getElementById('map');
87     if (!mapContainer) {
88       console.error('Map container not found. Ensure the DOM element is rendered before initializing the map.');
89       return;
90     }
91
92     const [latitude, longitude] = location.split(',');
93     const map = L.map('map').setView([latitude, longitude], 13);
94
95     L.tileLayer('https://s.tile.openstreetmap.org/{z}/{x}/{y}.png', {
96       maxZoom: 19,
97       attribution: '© OpenStreetMap contributors'
98     }).addTo(map);
99
100    const customIcon = L.icon({
101      iconUrl: pinGif, // Use the custom pin.gif
102      iconSize: [50, 50], // Size of the icon
103    });
104
105    const marker = L.marker([latitude, longitude], { icon: customIcon });
106    marker.addTo(map);
107  }, [complaint]);
108
109  const handlePasswordChange = (password, confirmPassword) => {
110    if (password !== confirmPassword) {
111      setError('Passwords do not match');
112      return;
113    }
114
115    const formData = new FormData();
116    formData.append('password', password);
117    formData.append('confirmPassword', confirmPassword);
118
119    axios.put('/api/complaint/password', formData)
120      .then(res => {
121        setComplaint({ ...complaint, password: password });
122        alert('Password changed successfully!');
123      })
124      .catch(err => {
125        setError('Failed to change password');
126      });
127  };
128
129  const handleFileUpload = (file) => {
130    const formData = new FormData();
131    formData.append('file', file);
132
133    axios.put('/api/complaint/upload', formData)
134      .then(res => {
135        setComplaint({ ...complaint, file: file });
136        alert('File uploaded successfully!');
137      })
138      .catch(err => {
139        setError('Failed to upload file');
140      });
141  };
142
143  const handleImageUpload = (image) => {
144    const formData = new FormData();
145    formData.append('image', image);
146
147    axios.put('/api/complaint/image', formData)
148      .then(res => {
149        setComplaint({ ...complaint, image: image });
150        alert('Image uploaded successfully!');
151      })
152      .catch(err => {
153        setError('Failed to upload image');
154      });
155  };
156
157  const handleDocumentUpload = (document) => {
158    const formData = new FormData();
159    formData.append('document', document);
160
161    axios.put('/api/complaint/document', formData)
162      .then(res => {
163        setComplaint({ ...complaint, document: document });
164        alert('Document uploaded successfully!');
165      })
166      .catch(err => {
167        setError('Failed to upload document');
168      });
169  };
170
171  const handleAddressUpdate = (address) => {
172    const formData = new FormData();
173    formData.append('address', address);
174
175    axios.put('/api/complaint/address', formData)
176      .then(res => {
177        setComplaint({ ...complaint, address: address });
178        alert('Address updated successfully!');
179      })
180      .catch(err => {
181        setError('Failed to update address');
182      });
183  };
184
185  const handlePhoneUpdate = (phone) => {
186    const formData = new FormData();
187    formData.append('phone', phone);
188
189    axios.put('/api/complaint/phone', formData)
190      .then(res => {
191        setComplaint({ ...complaint, phone: phone });
192        alert('Phone updated successfully!');
193      })
194      .catch(err => {
195        setError('Failed to update phone');
196      });
197  };
198
199  const handleEmailUpdate = (email) => {
200    const formData = new FormData();
201    formData.append('email', email);
202
203    axios.put('/api/complaint/email', formData)
204      .then(res => {
205        setComplaint({ ...complaint, email: email });
206        alert('Email updated successfully!');
207      })
208      .catch(err => {
209        setError('Failed to update email');
210      });
211  };
212
213  const handlePassportUpdate = (passport) => {
214    const formData = new FormData();
215    formData.append('passport', passport);
216
217    axios.put('/api/complaint/passport', formData)
218      .then(res => {
219        setComplaint({ ...complaint, passport: passport });
220        alert('Passport updated successfully!');
221      })
222      .catch(err => {
223        setError('Failed to update passport');
224      });
225  };
226
227  const handleDrivingLicenseUpdate = (drivingLicense) => {
228    const formData = new FormData();
229    formData.append('drivingLicense', drivingLicense);
230
231    axios.put('/api/complaint/drivingLicense', formData)
232      .then(res => {
233        setComplaint({ ...complaint, drivingLicense: drivingLicense });
234        alert('Driving license updated successfully!');
235      })
236      .catch(err => {
237        setError('Failed to update driving license');
238      });
239  };
240
241  const handleVerificationDocumentUpdate = (verificationDocument) => {
242    const formData = new FormData();
243    formData.append('verificationDocument', verificationDocument);
244
245    axios.put('/api/complaint/verificationDocument', formData)
246      .then(res => {
247        setComplaint({ ...complaint, verificationDocument: verificationDocument });
248        alert('Verification document updated successfully!');
249      })
250      .catch(err => {
251        setError('Failed to update verification document');
252      });
253  };
254
255  const handleCustomIconUpdate = (customIcon) => {
256    const formData = new FormData();
257    formData.append('customIcon', customIcon);
258
259    axios.put('/api/complaint/customIcon', formData)
260      .then(res => {
261        setComplaint({ ...complaint, customIcon: customIcon });
262        alert('Custom icon updated successfully!');
263      })
264      .catch(err => {
265        setError('Failed to update custom icon');
266      });
267  };
268
269  const handlePinGifUpdate = (pinGif) => {
270    const formData = new FormData();
271    formData.append('pinGif', pinGif);
272
273    axios.put('/api/complaint/pinGif', formData)
274      .then(res => {
275        setComplaint({ ...complaint, pinGif: pinGif });
276        alert('Pin gif updated successfully!');
277      })
278      .catch(err => {
279        setError('Failed to update pin gif');
280      });
281  };
282
283  const handleImageUpdate = (image) => {
284    const formData = new FormData();
285    formData.append('image', image);
286
287    axios.put('/api/complaint/image', formData)
288      .then(res => {
289        setComplaint({ ...complaint, image: image });
290        alert('Image updated successfully!');
291      })
292      .catch(err => {
293        setError('Failed to update image');
294      });
295  };
296
297  const handleDocumentUpdate = (document) => {
298    const formData = new FormData();
299    formData.append('document', document);
300
301    axios.put('/api/complaint/document', formData)
302      .then(res => {
303        setComplaint({ ...complaint, document: document });
304        alert('Document updated successfully!');
305      })
306      .catch(err => {
307        setError('Failed to update document');
308      });
309  };
310
311  const handleAddressUpdate = (address) => {
312    const formData = new FormData();
313    formData.append('address', address);
314
315    axios.put('/api/complaint/address', formData)
316      .then(res => {
317        setComplaint({ ...complaint, address: address });
318        alert('Address updated successfully!');
319      })
320      .catch(err => {
321        setError('Failed to update address');
322      });
323  };
324
325  const handlePhoneUpdate = (phone) => {
326    const formData = new FormData();
327    formData.append('phone', phone);
328
329    axios.put('/api/complaint/phone', formData)
330      .then(res => {
331        setComplaint({ ...complaint, phone: phone });
332        alert('Phone updated successfully!');
333      })
334      .catch(err => {
335        setError('Failed to update phone');
336      });
337  };
338
339  const handleEmailUpdate = (email) => {
340    const formData = new FormData();
341    formData.append('email', email);
342
343    axios.put('/api/complaint/email', formData)
344      .then(res => {
345        setComplaint({ ...complaint, email: email });
346        alert('Email updated successfully!');
347      })
348      .catch(err => {
349        setError('Failed to update email');
350      });
351  };
352
353  const handlePassportUpdate = (passport) => {
354    const formData = new FormData();
355    formData.append('passport', passport);
356
357    axios.put('/api/complaint/passport', formData)
358      .then(res => {
359        setComplaint({ ...complaint, passport: passport });
360        alert('Passport updated successfully!');
361      })
362      .catch(err => {
363        setError('Failed to update passport');
364      });
365  };
366
367  const handleDrivingLicenseUpdate = (drivingLicense) => {
368    const formData = new FormData();
369    formData.append('drivingLicense', drivingLicense);
370
371    axios.put('/api/complaint/drivingLicense', formData)
372      .then(res => {
373        setComplaint({ ...complaint, drivingLicense: drivingLicense });
374        alert('Driving license updated successfully!');
375      })
376      .catch(err => {
377        setError('Failed to update driving license');
378      });
379  };
380
381  const handleVerificationDocumentUpdate = (verificationDocument) => {
382    const formData = new FormData();
383    formData.append('verificationDocument', verificationDocument);
384
385    axios.put('/api/complaint/verificationDocument', formData)
386      .then(res => {
387        setComplaint({ ...complaint, verificationDocument: verificationDocument });
388        alert('Verification document updated successfully!');
389      })
390      .catch(err => {
391        setError('Failed to update verification document');
392      });
393  };
394
395  const handleCustomIconUpdate = (customIcon) => {
396    const formData = new FormData();
397    formData.append('customIcon', customIcon);
398
399    axios.put('/api/complaint/customIcon', formData)
400      .then(res => {
401        setComplaint({ ...complaint, customIcon: customIcon });
402        alert('Custom icon updated successfully!');
403      })
404      .catch(err => {
405        setError('Failed to update custom icon');
406      });
407  };
408
409  const handlePinGifUpdate = (pinGif) => {
410    const formData = new FormData();
411    formData.append('pinGif', pinGif);
412
413    axios.put('/api/complaint/pinGif', formData)
414      .then(res => {
415        setComplaint({ ...complaint, pinGif: pinGif });
416        alert('Pin gif updated successfully!');
417      })
418      .catch(err => {
419        setError('Failed to update pin gif');
420      });
421  };
422
423  const handleImageUpdate = (image) => {
424    const formData = new FormData();
425    formData.append('image', image);
426
427    axios.put('/api/complaint/image', formData)
428      .then(res => {
429        setComplaint({ ...complaint, image: image });
430        alert('Image updated successfully!');
431      })
432      .catch(err => {
433        setError('Failed to update image');
434      });
435  };
436
437  const handleDocumentUpdate = (document) => {
438    const formData = new FormData();
439    formData.append('document', document);
440
441    axios.put('/api/complaint/document', formData)
442      .then(res => {
443        setComplaint({ ...complaint, document: document });
444        alert('Document updated successfully!');
445      })
446      .catch(err => {
447        setError('Failed to update document');
448      });
449  };
450
451  const handleAddressUpdate = (address) => {
452    const formData = new FormData();
453    formData.append('address', address);
454
455    axios.put('/api/complaint/address', formData)
456      .then(res => {
457        setComplaint({ ...complaint, address: address });
458        alert('Address updated successfully!');
459      })
460      .catch(err => {
461        setError('Failed to update address');
462      });
463  };
464
465  const handlePhoneUpdate = (phone) => {
466    const formData = new FormData();
467    formData.append('phone', phone);
468
469    axios.put('/api/complaint/phone', formData)
470      .then(res => {
471        setComplaint({ ...complaint, phone: phone });
472        alert('Phone updated successfully!');
473      })
474      .catch(err => {
475        setError('Failed to update phone');
476      });
477  };
478
479  const handleEmailUpdate = (email) => {
480    const formData = new FormData();
481    formData.append('email', email);
482
483    axios.put('/api/complaint/email', formData)
484      .then(res => {
485        setComplaint({ ...complaint, email: email });
486        alert('Email updated successfully!');
487      })
488      .catch(err => {
489        setError('Failed to update email');
490      });
491  };
492
493  const handlePassportUpdate = (passport) => {
494    const formData = new FormData();
495    formData.append('passport', passport);
496
497    axios.put('/api/complaint/passport', formData)
498      .then(res => {
499        setComplaint({ ...complaint, passport: passport });
500        alert('Passport updated successfully!');
501      })
502      .catch(err => {
503        setError('Failed to update passport');
504      });
505  };
506
507  const handleDrivingLicenseUpdate = (drivingLicense) => {
508    const formData = new FormData();
509    formData.append('drivingLicense', drivingLicense);
510
511    axios.put('/api/complaint/drivingLicense', formData)
512      .then(res => {
513        setComplaint({ ...complaint, drivingLicense: drivingLicense });
514        alert('Driving license updated successfully!');
515      })
516      .catch(err => {
517        setError('Failed to update driving license');
518      });
519  };
520
521  const handleVerificationDocumentUpdate = (verificationDocument) => {
522    const formData = new FormData();
523    formData.append('verificationDocument', verificationDocument);
524
525    axios.put('/api/complaint/verificationDocument', formData)
526      .then(res => {
527        setComplaint({ ...complaint, verificationDocument: verificationDocument });
528        alert('Verification document updated successfully!');
529      })
530      .catch(err => {
531        setError('Failed to update verification document');
532      });
533  };
534
535  const handleCustomIconUpdate = (customIcon) => {
536    const formData = new FormData();
537    formData.append('customIcon', customIcon);
538
539    axios.put('/api/complaint/customIcon', formData)
540      .then(res => {
541        setComplaint({ ...complaint, customIcon: customIcon });
542        alert('Custom icon updated successfully!');
543      })
544      .catch(err => {
545        setError('Failed to update custom icon');
546      });
547  };
548
549  const handlePinGifUpdate = (pinGif) => {
550    const formData = new FormData();
551    formData.append('pinGif', pinGif);
552
553    axios.put('/api/complaint/pinGif', formData)
554      .then(res => {
555        setComplaint({ ...complaint, pinGif: pinGif });
556        alert('Pin gif updated successfully!');
557      })
558      .catch(err => {
559        setError('Failed to update pin gif');
560      });
561  };
562
563  const handleImageUpdate = (image) => {
564    const formData = new FormData();
565    formData.append('image', image);
566
567    axios.put('/api/complaint/image', formData)
568      .then(res => {
569        setComplaint({ ...complaint, image: image });
570        alert('Image updated successfully!');
571      })
572      .catch(err => {
573        setError('Failed to update image');
574      });
575  };
576
577  const handleDocumentUpdate = (document) => {
578    const formData = new FormData();
579    formData.append('document', document);
580
581    axios.put('/api/complaint/document', formData)
582      .then(res => {
583        setComplaint({ ...complaint, document: document });
584        alert('Document updated successfully!');
585      })
586      .catch(err => {
587        setError('Failed to update document');
588      });
589  };
590
591  const handleAddressUpdate = (address) => {
592    const formData = new FormData();
593    formData.append('address', address);
594
595    axios.put('/api/complaint/address', formData)
596      .then(res => {
597        setComplaint({ ...complaint, address: address });
598        alert('Address updated successfully!');
599      })
600      .catch(err => {
601        setError('Failed to update address');
602      });
603  };
604
605  const handlePhoneUpdate = (phone) => {
606    const formData = new FormData();
607    formData.append('phone', phone);
608
609    axios.put('/api/complaint/phone', formData)
610      .then(res => {
611        setComplaint({ ...complaint, phone: phone });
612        alert('Phone updated successfully!');
613      })
614      .catch(err => {
615        setError('Failed to update phone');
616      });
617  };
618
619  const handleEmailUpdate = (email) => {
620    const formData = new FormData();
621    formData.append('email', email);
622
623    axios.put('/api/complaint/email', formData)
624      .then(res => {
625        setComplaint({ ...complaint, email: email });
626        alert('Email updated successfully!');
627      })
628      .catch(err => {
629        setError('Failed to update email');
630      });
631  };
632
633  const handlePassportUpdate = (passport) => {
634    const formData = new FormData();
635    formData.append('passport', passport);
636
637    axios.put('/api/complaint/passport', formData)
638      .then(res => {
639        setComplaint({ ...complaint, passport: passport });
640        alert('Passport updated successfully!');
641      })
642      .catch(err => {
643        setError('Failed to update passport');
644      });
645  };
646
647  const handleDrivingLicenseUpdate = (drivingLicense) => {
648    const formData = new FormData();
649    formData.append('drivingLicense', drivingLicense);
650
651    axios.put('/api/complaint/drivingLicense', formData)
652      .then(res => {
653        setComplaint({ ...complaint, drivingLicense: drivingLicense });
654        alert('Driving license updated successfully!');
655      })
656      .catch(err => {
657        setError('Failed to update driving license');
658      });
659  };
660
661  const handleVerificationDocumentUpdate = (verificationDocument) => {
662    const formData = new FormData();
663    formData.append('verificationDocument', verificationDocument);
664
665    axios.put('/api/complaint/verificationDocument', formData)
666      .then(res => {
667        setComplaint({ ...complaint, verificationDocument: verificationDocument });
668        alert('Verification document updated successfully!');
669      })
670      .catch(err => {
671        setError('Failed to update verification document');
672      });
673  };
674
675  const handleCustomIconUpdate = (customIcon) => {
676    const formData = new FormData();
677    formData.append('customIcon', customIcon);
678
679    axios.put('/api/complaint/customIcon', formData)
680      .then(res => {
681        setComplaint({ ...complaint, customIcon: customIcon });
682        alert('Custom icon updated successfully!');
683      })
684      .catch(err => {
685        setError('Failed to update custom icon');
686      });
687  };
688
689  const handlePinGifUpdate = (pinGif) => {
690    const formData = new FormData();
691    formData.append('pinGif', pinGif);
692
693    axios.put('/api/complaint/pinGif', formData)
694      .then(res => {
695        setComplaint({ ...complaint, pinGif: pinGif });
696        alert('Pin gif updated successfully!');
697      })
698      .catch(err => {
699        setError('Failed to update pin gif');
700      });
701  };
702
703  const handleImageUpdate = (image) => {
704    const formData = new FormData();
705    formData.append('image', image);
706
707    axios.put('/api/complaint/image', formData)
708      .then(res => {
709        setComplaint({ ...complaint, image: image });
710        alert('Image updated successfully!');
711      })
712      .catch(err => {
713        setError('Failed to update image');
714      });
715  };
716
717  const handleDocumentUpdate = (document) => {
718    const formData = new FormData();
719    formData.append('document', document);
720
721    axios.put('/api/complaint/document', formData)
722      .then(res => {
723        setComplaint({ ...complaint, document: document });
724        alert('Document updated successfully!');
725      })
726      .catch(err => {
727        setError('Failed to update document');
728      });
729  };
730
731  const handleAddressUpdate = (address) => {
732    const formData = new FormData();
733    formData.append('address', address);
734
735    axios.put('/api/complaint/address', formData)
736      .then(res => {
737        setComplaint({ ...complaint, address: address });
738        alert('Address updated successfully!');
739      })
740      .catch(err => {
741        setError('Failed to update address');
742      });
743  };
744
745  const handlePhoneUpdate = (phone) => {
746    const formData = new FormData();
747    formData.append('phone', phone);
748
749    axios.put('/api/complaint/phone', formData)
750      .then(res => {
751        setComplaint({ ...complaint, phone: phone });
752        alert('Phone updated successfully!');
753      })
754      .catch(err => {
755        setError('Failed to update phone');
756      });
757  };
758
759  const handleEmailUpdate = (email) => {
760    const formData = new FormData();
761    formData.append('email', email);
762
763    axios.put('/api/complaint/email', formData)
764      .then(res => {
765        setComplaint({ ...complaint, email: email });
766        alert('Email updated successfully!');
767      })
768      .catch(err => {
769        setError('Failed to update email');
770      });
771  };
772
773  const handlePassportUpdate = (passport) => {
774    const formData = new FormData();
775    formData.append('passport', passport);
776
777    axios.put('/api/complaint/passport', formData)
778      .then(res => {
779        setComplaint({ ...complaint, passport: passport });
780        alert('Passport updated successfully!');
781      })
782      .catch(err => {
783        setError('Failed to update passport');
784      });
785  };
786
787  const handleDrivingLicenseUpdate = (drivingLicense) => {
788    const formData = new FormData();
789    formData.append('drivingLicense', drivingLicense);
790
791    axios.put('/api/complaint/drivingLicense', formData)
792      .then(res => {
793        setComplaint({ ...complaint, drivingLicense: drivingLicense });
794        alert('Driving license updated successfully!');
795      })
796      .catch(err => {
797        setError('Failed to update driving license');
798      });
799  };
800
801  const handleVerificationDocumentUpdate = (verificationDocument) => {
802    const formData = new FormData();
803    formData.append('verificationDocument', verificationDocument);
804
805    axios.put('/api/complaint/verificationDocument', formData)
806      .then(res => {
807        setComplaint({ ...complaint, verificationDocument: verificationDocument });
808        alert('Verification document updated successfully!');
809      })
810      .catch(err => {
811        setError('Failed to update verification document');
812      });
813  };
814
815  const handleCustomIconUpdate = (customIcon) => {
816    const formData = new FormData();
817    formData.append('customIcon', customIcon);
818
819    axios.put('/api/complaint/customIcon', formData)
820      .then(res => {
821        setComplaint({ ...complaint, customIcon: customIcon });
822        alert('Custom icon updated successfully!');
823      })
824      .catch(err => {
825        setError('Failed to update custom icon');
826      });
827  };
828
829  const handlePinGifUpdate = (pinGif) => {
830    const formData = new FormData();
831    formData.append('pinGif', pinGif);
832
833    axios.put('/api/complaint/pinGif', formData)
834      .then(res => {
835        setComplaint({ ...complaint, pinGif: pinGif });
836        alert('Pin gif updated successfully!');
837      })
838      .catch(err => {
839        setError('Failed to update pin gif');
840      });
841  };
842
843  const handleImageUpdate = (image) => {
844    const formData = new FormData();
845    formData.append('image', image);
846
847    axios.put('/api/complaint/image', formData)
848      .then(res => {
849        setComplaint({ ...complaint, image: image });
850        alert('Image updated successfully!');
851      })
852      .catch(err => {
853        setError('Failed to update image');
854      });
855  };
856
857  const handleDocumentUpdate = (document) => {
858    const formData = new FormData();
859    formData.append('document', document);
860
861    axios.put('/api/complaint/document', formData)
862      .then(res => {
863        setComplaint({ ...complaint, document: document });
864        alert('Document updated successfully!');
865      })
866      .catch(err => {
867        setError('Failed to update document');
868      });
869  };
870
871  const handleAddressUpdate = (address) => {
872    const formData = new FormData();
873    formData.append('address', address);
874
875    axios.put('/api/complaint/address', formData)
876      .then(res => {
877        setComplaint({ ...complaint, address: address });
878        alert('Address updated successfully!');
879      })
880      .catch(err => {
881        setError('Failed to update address');
882      });
883  };
884
885  const handlePhoneUpdate = (phone) => {
886    const formData = new FormData();
887    formData.append('phone', phone);
888
889    axios.put('/api/complaint/phone', formData)
890      .then(res => {
891        setComplaint({ ...complaint, phone: phone });
892        alert('Phone updated successfully!');
893      })
894      .catch(err => {
895        setError('Failed to update phone');
896      });
897  };
898
899  const handleEmailUpdate = (email) => {
900    const formData = new FormData();
901    formData.append('email', email);
902
903    axios.put('/api/complaint/email', formData)
904      .then(res => {
905        setComplaint({ ...complaint, email: email });
906        alert('Email updated successfully!');
907      })
908      .catch(err => {
909        setError('Failed to update email');
910      });
911  };
912
913  const handlePassportUpdate = (passport) => {
914    const formData = new FormData();
915    formData.append('passport', passport);
916
917    axios.put('/api/complaint/passport', formData)
918      .then(res => {
919        setComplaint({ ...complaint, passport: passport });
920        alert('Passport updated successfully!');
921      })
922      .catch(err => {
923        setError('Failed to update passport');
924      });
925  };
926
927  const handleDrivingLicenseUpdate = (drivingLicense) => {
928    const formData = new FormData();
929    formData.append('drivingLicense', drivingLicense);
930
931    axios.put('/api/complaint/drivingLicense', formData)
932      .then(res => {
933        setComplaint({ ...complaint, drivingLicense: drivingLicense });
934        alert('Driving license updated successfully!');
935      })
936      .catch(err => {
937        setError('Failed to update driving license');
938      });
939  };
940
941  const handleVerificationDocumentUpdate = (verificationDocument) => {
942    const formData = new FormData();
943    formData.append('verificationDocument', verificationDocument);
944
945    axios.put('/api/complaint/verificationDocument', formData)
946      .then(res => {
947        setComplaint({ ...complaint, verificationDocument: verificationDocument });
948        alert('Verification document updated successfully!');
949      })
950      .catch(err => {
951        setError('Failed to update verification document');
952      });
953  };
954
955  const handleCustomIconUpdate = (customIcon) => {
956    const formData = new FormData();
957    formData.append('customIcon', customIcon);
958
959    axios.put('/api/complaint/customIcon', formData)
960      .then(res => {
961        setComplaint({ ...complaint, customIcon: customIcon });
962        alert('Custom icon updated successfully!');
963      })
964      .catch(err => {
965        setError('Failed to update custom icon');
966      });
967  };
968
969  const handlePinGifUpdate = (pinGif) => {
970    const formData = new FormData();
971    formData.append('pinGif', pinGif);
972
973    axios.put('/api/complaint/pinGif', formData)
974      .then(res => {
975        setComplaint({ ...complaint, pinGif: pinGif });
976        alert('Pin gif updated successfully!');
977      })
978      .catch(err => {
979        setError('Failed to update pin gif');
980      });
981  };
982
983  const handleImageUpdate = (image) => {
984    const formData = new FormData();
985    formData.append('image', image);
986
987    axios.put('/api/complaint/image', formData)
988      .then(res => {
989        setComplaint({ ...complaint, image: image });
990        alert('Image updated successfully!');
991      })
992      .catch(err => {
993        setError('Failed to update image');
994      });
995  };
996
997  const handleDocumentUpdate = (document) => {
998    const formData = new FormData();
999    formData.append('document', document);
1000
1001    axios.put('/api/complaint/document', formData)
1002      .then(res => {
1003        setComplaint({ ...complaint, document: document });
1004        alert('Document updated successfully!');
1005      })
1006      .catch(err => {
1007        setError('Failed to update document');
1008      });
1009  };
1010
1011  const handleAddressUpdate = (address) => {
1012    const formData = new FormData();
1013    formData.append('address', address);
1014
1015    axios.put('/api/complaint/address', formData)
1016      .then(res => {
1017        setComplaint({ ...complaint, address: address });
1018        alert('Address updated successfully!');
1019      })
1020      .catch(err => {
1021        setError('Failed to update address');
1022      });
1023  };
1024
1025  const handlePhoneUpdate = (phone) => {
1026    const formData = new FormData();
1027    formData.append('phone', phone);
1028
1029    axios.put('/api/complaint/phone', formData)
1030      .then(res => {
1031        setComplaint({ ...complaint, phone: phone });
1032        alert('Phone updated successfully!');
1033      })
1034      .catch(err => {
1035        setError('Failed to update phone');
1036      });
1037  };
1038
1039  const handleEmailUpdate = (email) => {
1040    const formData = new FormData();
1041    formData.append('email', email);
1042
1043    axios.put('/api/complaint/email', formData)
1044      .then(res => {
1045        setComplaint({ ...complaint, email: email });
1046        alert('Email updated successfully!');
1047      })
1048      .catch(err => {
1049        setError('Failed to update email');
1050      });
1051  };
1052
1053  const handlePassportUpdate = (passport) => {
1054    const formData = new FormData();
1055    formData.append('passport', passport);
1056
1057    axios.put('/api/complaint/passport', formData)
1058      .then(res => {
1059        setComplaint({ ...complaint, passport: passport });
1060        alert('Passport updated successfully!');
1061      })
1062      .catch(err => {
1063        setError('Failed to update passport');
1064      });
1065  };
1066
1067  const handleDrivingLicenseUpdate = (drivingLicense) => {
1068    const formData = new FormData();
1069    formData.append('drivingLicense', drivingLicense);
1070
1071    axios.put('/api/complaint/drivingLicense', formData)
1072      .then(res => {
1073        setComplaint({ ...complaint, drivingLicense: drivingLicense });
1074        alert('Driving license updated successfully!');
1075      })
1076      .catch(err => {
1077        setError('Failed to update driving license');
1078      });
1079  };
1080
1081  const handleVerificationDocumentUpdate = (verificationDocument) => {
1082    const formData = new FormData();
1083    formData.append('verificationDocument', verificationDocument);
1084
1085    axios.put('/api/complaint/verificationDocument', formData)
1086      .then(res => {
1087        setComplaint({ ...complaint, verificationDocument: verificationDocument });
1088        alert('Verification document updated successfully!');
1089      })
1090      .catch(err => {
1091        setError('Failed to update verification document');
1092      });
1093  };
1094
1095  const handleCustomIconUpdate = (customIcon) => {
1096    const formData = new FormData();
1097    formData.append('customIcon', customIcon);
1098
1099    axios.put('/api/complaint/customIcon', formData)
1100      .then(res => {
1101        setComplaint({ ...complaint, customIcon: customIcon });
1102        alert('Custom icon updated successfully!');
1103      })
1104      .catch(err => {
1105        setError('Failed to update custom icon');
1106      });
1107  };
1108
1109  const handlePinGifUpdate = (pinGif) => {
1110    const formData = new FormData();
1111    formData.append('pinGif', pinGif);
1112
1113    axios.put('/api/complaint/pinGif', formData)
1114      .then(res => {
1115        setComplaint({ ...complaint, pinGif: pinGif });
1116        alert('Pin gif updated successfully!');
1117      })
1118      .catch(err => {
1119        setError('Failed to update pin gif');
1120      });
1121  };
1122
1123  const handleImageUpdate = (image) => {
1124    const formData = new FormData();
1125    formData.append('image', image);
1126
1127    axios.put('/api/complaint/image', formData)
1128
```

```
JS ComplaintDetails.js  JS Profile.js  X
frontend > src > pages > JS Profile.js > ⚡ Profile > ✨ useEffect() callback > ✨ then() callback
  3
  4   function Profile() {
  5     const userId = localStorage.getItem('nirapod_identifier'); // Use NID from localStorage
  6     const [user, setUser] = useState(null);
  7     const [form, setForm] = useState({
  8       name: '',
  9       email: '',
10       phone: '',
11       presentAddress: '',
12       permanentAddress: '',
13       utilityBillCustomerId: '',
14       passport: '',
15       drivingLicense: '',
16     });
17     const [files, setFiles] = useState({
18       photo: null,
19       utilityBillPhoto: null,
20       passportImg: null,
21       drivingLicenseImg: null,
22     });
23     const [message, setMessage] = useState('');
24     const [pwForm, setPwForm] = useState({ oldPassword: '', newPassword: '', confirmPassword: '' });
25     const [pwMsg, setPwMsg] = useState('');
26
27     useEffect(() => {
28       axios.get(`/api/user/${userId}`).then(res => {
29         setUser(res.data);
30         setForm({
31           name: res.data.name || '',
32           email: res.data.email || '',
33           phone: res.data.phone || '',
34           presentAddress: res.data.presentAddress || '',
35           permanentAddress: res.data.permanentAddress || '',
36           utilityBillCustomerId: res.data.utilityBillCustomerId || res.data.utilityBillCustomerID || res.data.utilityBillId || '',
37           passport: res.data.passport || '',
38           drivingLicense: res.data.drivingLicense || '',
39         });
40       });
41     }, [userId]);
42
43     const handleChange = e => {
44       const { name, value, files: fileInput } = e.target;
```

```

js ComplaintDetails.js | js Profile.js ×
frontend > src > pages > js Profile.js > ⚡ Profile > useEffect() callback > then() callback
4  function Profile() {
41
42    const handleChange = e => {
43      const { name, value, files: fileInput } = e.target;
44      if (fileInput) {
45        setFiles(f => ({ ...f, [name]: fileInput[0] }));
46      } else {
47        setForm(f => ({ ...f, [name]: value }));
48      }
49    };
50
51
52    const handleProfileUpdate = async e => {
53      e.preventDefault();
54      setMessage('');
55      const formData = new FormData();
56      Object.entries(form).forEach(([key, value]) => formData.append(key, value));
57      Object.entries(files).forEach(([key, value]) => { if (value) formData.append(key, value); });
58      try {
59        await axios.put('/api/user/${userId}/profile', formData, { headers: { 'Content-Type': 'multipart/form-data' } });
56      setMessage('Profile updated successfully.');
60    } catch {
61      setMessage('Failed to update profile.');
62    }
63  };
64
65
66    const handlePwChange = e => setPwForm(f => ({ ...f, [e.target.name]: e.target.value }));
67    const handleChangePassword = async e => {
68      e.preventDefault();
69      setPwMsg('');
70      if (pwForm.newPassword !== pwForm.confirmPassword) {
71        setPwMsg('Passwords do not match.');
72        return;
73      }
74      try {
75        await axios.post('/api/user/${userId}/change-password', pwForm);
76        setPwMsg('Password changed successfully.');
77        setPwForm({ oldPassword: '', newPassword: '', confirmPassword: '' });
78      } catch {
79        setPwMsg('Failed to change password.');
80      }
81    };
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122

```

```

js ComplaintDetails.js | js Profile.js ×
frontend > src > pages > js Profile.js > ⚡ Profile > (eo) handleChangePassword
4  function Profile() {
41
42    if (!user) return <div>Loading...</div>;
43
44    return (
45      <div className="profile-container" style={{ background: '#232B31', minHeight: '100vh', display: 'flex', flexDirection: 'column', alignItems: 'center', justifyContent: 'center' }}>
46        <form onSubmit={handleProfileUpdate} style={{ background: 'none', display: 'flex', flexDirection: 'column', alignItems: 'center', gap: 24, width: 480 }}>
47          <label htmlFor="photo" style={{ display: 'flex', flexDirection: 'column', alignItems: 'center', marginBottom: 12, minHeight: 130 }}>
48            {user.userPhoto && user.userPhoto !== 'null' && user.userPhoto !== '' ? (
49              <img
50                src={user.userPhoto.startsWith('/uploads/') ? `http://localhost:8080/${user.userPhoto.replace('/uploads/', '')}` : `http://localhost:8080/${user.userPhoto}`}
51                alt="Profile"
52                style={{ width: 120, height: 120, borderRadius: '50%', objectFit: 'cover', marginBottom: 12, border: '2px solid #232B36' }}
53              />
54            ) : (
55              <div style={{ width: 120, height: 120, borderRadius: '50%', background: '#ffff', display: 'flex', alignItems: 'center', justifyContent: 'center', fontSize: 24, color: '#232B36' }}>
56                No Photo
57              </div>
58            )
59            <input type="file" name="photo" id="photo" style={{ display: 'none' }} onChange={handleChange} accept="image/*" />
60            <button type="button" onClick={() => document.getElementById('photo').click()} style={{ background: '#232B36', color: '#ffff', border: 'none', borderRadius: 8, padding: 12, width: 120 }}>
61              Select Photo
62            </button>
63          </label>
64          <div style={profileRowStyle}>
65            <span style={profileLabelStyle}>Name :</span><input name="name" placeholder="Name" value={form.name} onChange={handleChange} style={inputStyle}>
66          <div style={profileRowStyle}>
67            <span style={profileLabelStyle}>Phone :</span><input name="phone" placeholder="Phone" value={form.phone} onChange={handleChange} style={inputStyle}>
68          <div style={profileRowStyle}>
69            <span style={profileLabelStyle}>Email :</span><input name="email" placeholder="Email" value={form.email} onChange={handleChange} style={inputStyle}>
70          <div style={profileRowStyle}>
71            <span style={profileLabelStyle}>NID :</span><input name="nid" placeholder="NID" value={user.nid} || '' readOnly style={...profileInputStyle}>
72          <div style={profileRowStyle}>
73            <span style={profileLabelStyle}>Present Address :</span><input name="presentAddress" placeholder="Present Address" value={form.presentAddress} onChange={handleChange} style={inputStyle}>
74          <div style={profileRowStyle}>
75            <span style={profileLabelStyle}>Permanent Address :</span><input name="permanentAddress" placeholder="Permanent Address" value={form.permanentAddress} onChange={handleChange} style={inputStyle}>
76          <div style={profileRowStyle}>
77            <span style={profileLabelStyle}>Utility Bill Customer ID :</span><input name="utilityBillCustomerId" placeholder="Utility Bill Customer ID" value={form.utilityBillCustomerId} onChange={handleChange} style={inputStyle}>
78          <div style={profileRowStyle}>
79            <span style={profileLabelStyle}>Utility Bill Photo :</span><input type="file" name="utilityBillPhoto" onChange={handleChange} style={inputStyle}>
80          <div style={profileRowStyle}>
81            <span style={profileLabelStyle}>Passport :</span><input name="passport" placeholder="Passport" value={form.passport} onChange={handleChange} style={inputStyle}>
82          <div style={profileRowStyle}>
83            <span style={profileLabelStyle}>Driving License Image :</span><input type="file" name="drivingLicenseImg" onChange={handleChange} style={inputStyle}>
84          <div style={profileRowStyle}>
85            <span style={profileLabelStyle}>Driving License :</span><input name="drivingLicense" placeholder="Driving License" value={form.drivingLicense} onChange={handleChange} style={inputStyle}>
86          <div style={profileRowStyle}>
87            <span style={profileLabelStyle}>Driving License Image :</span><input type="file" name="drivingLicenseImg" onChange={handleChange} style={inputStyle}>
88          <div style={profileRowStyle}>
89            <span style={profileLabelStyle}>Submit :</span><button type="submit" style={{ background: 'ffff', color: '#232B36', border: 'none', borderRadius: 12, padding: '10px 38px', fontWeight: 600, fontSize: 18, margin: 12 }}>
90              Submit
91            </button>
92          </div>
93        </div>
94        <form onSubmit={handleChangePassword} style={{ background: 'none', display: 'flex', flexDirection: 'column', alignItems: 'center', gap: 12, width: 420, marginTop: 12 }}>
95          <h3 style={{ color: '#ffff', marginBottom: 8 }}>Change Password</h3>
96          <input name="oldPassword" type="password" placeholder="Old Password" value={pwForm.oldPassword} onChange={handlePwChange} style={inputStyle} required />
97          <input name="newPassword" type="password" placeholder="New Password" value={pwForm.newPassword} onChange={handlePwChange} style={inputStyle} required />
98          <input name="confirmPassword" type="password" placeholder="Confirm New Password" value={pwForm.confirmPassword} onChange={handlePwChange} style={inputStyle} required />
99        </form>
100      </div>
101    );
102  );
103
```

- **Notification.js** - The Notifications component displays real-time user notifications and allows interaction with each item. It retrieves the logged-in user's identifier from localStorage and fetches their notifications from the backend using Axios. The component displays a loading indicator during fetch, shows error messages if the request fails, and sorts notifications by creation time in descending order. Notifications are refreshed every 30 seconds using setInterval. When a notification is clicked, it is marked as read via an API call, and if linked to a related post, the user is navigated to the homepage with the post highlighted. The component also includes logic to

verify user authentication before redirecting. Notifications are visually distinguished based on whether they are read or unread, and timestamps are converted into human-readable relative time (e.g., “2 hours ago”). This provides users with an interactive and dynamic way to manage and view their updates or alerts.

```

JS Notifications.js ×
frontend > src > pages > JS Notifications.js > ...
1 import React, { useState, useEffect } from 'react';
2 import axios from 'axios';
3 import { useNavigate } from 'react-router-dom';
4 import './ComplaintList.css';
5
6 function Notifications() {
7   const [notifications, setNotifications] = useState([]);
8   const [loading, setLoading] = useState(true);
9   const [error, setError] = useState(null);
10  const userId = localStorage.getItem("nirapod_identifier");
11  const navigate = useNavigate();
12
13  const fetchNotifications = async () => {
14    if (!userId) {
15      setError('Please log in to view notifications');
16      setLoading(false);
17      return;
18    }
19
20    try {
21      console.log('Fetching notifications for user:', userId);
22      const res = await axios.get(`/api/notifications/user/${userId}`);
23      console.log('Notifications response:', res.data);
24
25      if (!Array.isArray(res.data)) {
26        console.error('Expected array of notifications but got:', res.data);
27        setError('Invalid response format from server');
28        setLoading(false);
29        return;
30      }
31
32      // Sort notifications by createdAt in descending order (newest first)
33      const sortedNotifications = res.data.sort((a, b) =>
34        new Date(b.createdAt) - new Date(a.createdAt)
35      );
36
37      setNotifications(sortedNotifications);
38      setLoading(false);
39      setError(null);
40    } catch (err) {
41      console.error('Failed to fetch notifications:', err.response?.data || err.message);
42      setError('Failed to fetch notifications. Please try again later.');
43    }
44  }
45
46  useEffect(() => {
47    fetchNotifications();
48  }, []);
49
50  const handleClick = async (notification) => {
51    if (!userId) {
52      navigate('/login');
53      return;
54    }
55
56    try {
57      if (!notification.read) {
58        await axios.put(`/api/notifications/${notification.id}/read`);
59        setNotifications(prevNotifications =>
60          prevNotifications.map(n =>
61            n.id === notification.id ? { ...n, read: true } : n
62          )
63        );
64
65        if (notification.relatedPostId) {
66          // First verify that the user is still authenticated
67          try {
68            await axios.get(`/api/user/by-identifier?value=${userId}`);
69            // User is authenticated, proceed with navigation
70            navigate('/home', {
71              state: {
72                openPost: notification.relatedPostId,
73                fromNotification: true
74              }
75            });
76          } catch (authErr) {
77            // Authentication failed, redirect to login
78            localStorage.removeItem('nirapod_identifier');
79            navigate('/login');
80          }
81        } catch (err) {
82          console.error('Error handling notification:', err);
83        }
84      };
85      useEffect(() => {
86        fetchNotifications();
87      }, []);
88    }
89  };
90
91  return (
92    <div>
93      <h2>Notifications</h2>
94      <ul>
95        {notifications.map(notification => (
96          <li key={notification.id}>
97            {notification.title}
98            {notification.read ? <span>Read</span> : <span>Unread</span>}
99            {notification.createdAt}
100           <button onClick={()=>handleClick(notification)}>View</button>
101          </li>
102        )}
103      </ul>
104    </div>
105  );
106}

```

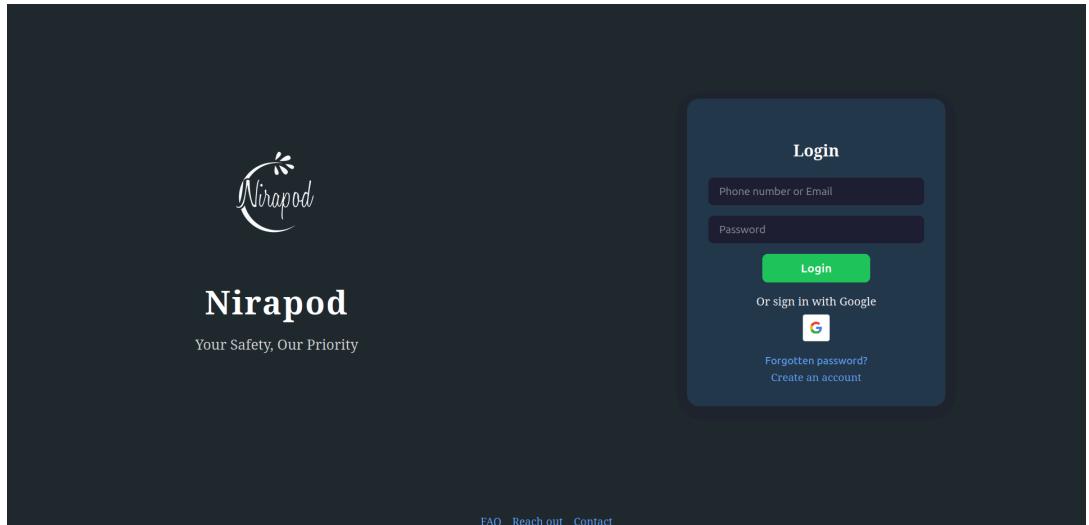
```

JS Notifications.js ×
frontend > src > pages > JS Notifications.js > ⚡ Notifications > 🕒 formatTime
6   function Notifications() {
90
91     const getNotificationStyle = (notification) => {
92       let className = 'notification-item';
93       if (!notification.read) className += ' notification-unread';
94       return className;
95     };
96
97     const formatTime = (dateString) => {
98       if (!dateString) return '';
99
100      const date = new Date(dateString);
101      if (isNaN(date.getTime())) return dateString; // Return original if invalid
102
103      const now = new Date();
104      const diff = now - date;
105
106      if (diff < 60000) { // Less than 1 minute
107        return 'Just now';
108      }
109
110      if (diff < 24 * 60 * 60 * 1000) {
111        const hours = Math.floor(diff / (60 * 60 * 1000));
112        if (hours < 1) {
113          const minutes = Math.floor(diff / (60 * 1000));
114          return `${minutes} minutes${minutes !== 1 ? 's' : ''} ago`;
115        }
116        return `${hours} hour${hours !== 1 ? 's' : ''} ago`;
117      }
118
119      if (diff < 7 * 24 * 60 * 60 * 1000) {
120        const days = Math.floor(diff / (24 * 60 * 60 * 1000));
121        return `${days} day${days !== 1 ? 's' : ''} ago`;
122      }
123
124      return date.toLocaleDateString();
125    };
126
127    if (loading) return (
128      <div className="notifications-wrapper">
129        <h2 className="page-title">Notifications</h2>
130        <div className="loading">Loading notifications...</div>
131      </div>
132    );
133
134    if (error) return (
135      <div className="notifications-wrapper">
136        <h2 className="page-title">Notifications</h2>
137        <div className="error">{error}</div>
138      </div>
139    );
140
141    return (
142      <div className="notifications-wrapper">
143        <h2 className="page-title">Notifications</h2>
144        {notifications.length === 0 ? (
145          <div className="empty-state">
146            | No notifications yet
147          </div>
148        ) : (
149          <div className="notifications-list">
150            {notifications.map(notification => (
151              <div
152                key={notification.id}
153                className={getNotificationStyle(notification)}
154                onClick={() => handleClick(notification)}
155              >
156                <div className="notification-content">
157                  <div className="notification-message">{notification.message}</div>
158                  <div className="notification-time">{formatTime(notification.createdAt)}</div>
159                </div>
160              </div>
161            )));
162          </div>
163        )
164      </div>
165    );
166  }
167
168  export default Notifications;
169

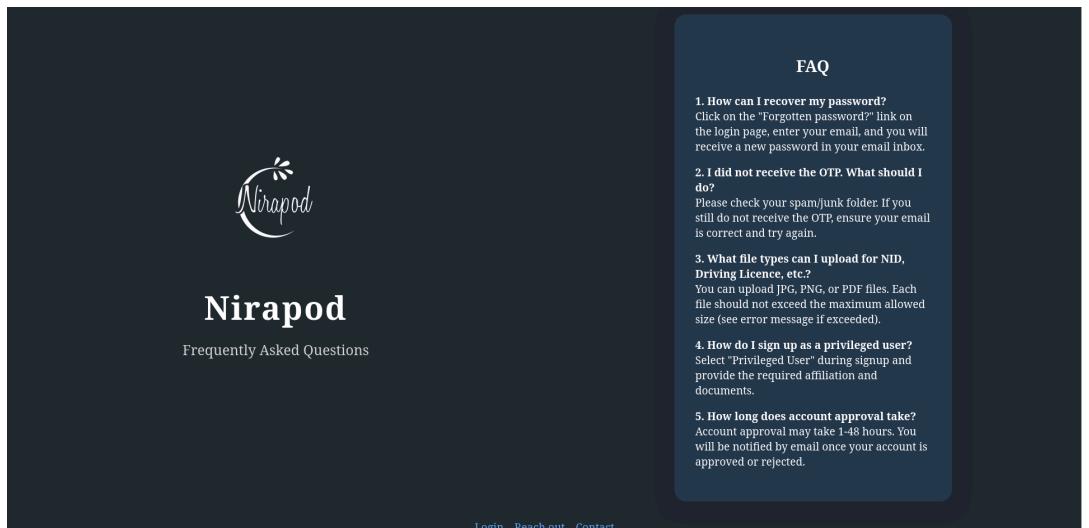
```

● User Manual

1. **Landing Page** - The landing page is the first page a user encounters upon visiting the website. Users cannot access any website content without first logging in. The layout is designed to be clean, simple, and free of unnecessary clutter, ensuring an easy and intuitive user experience. Users can log in manually using their email and password or sign in via Google authentication. Additionally, three guideline pages are accessible from the footer to assist users in understanding the platform's policies and usage rules.

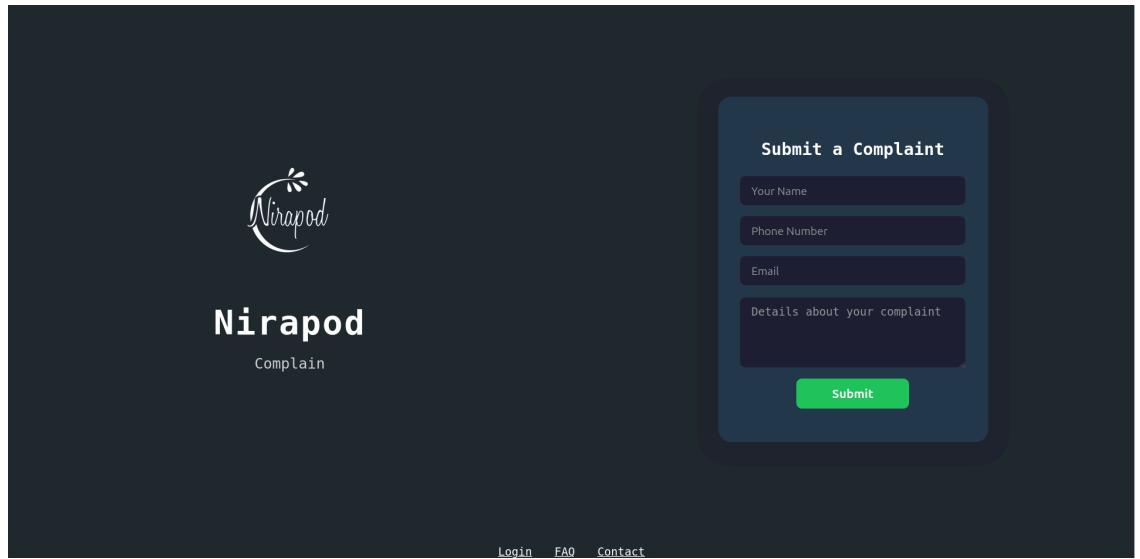


2. **FAQ page** - If a user encounters difficulties or feels lost while navigating the website, they can visit the FAQ (Frequently Asked Questions) page. The FAQ page provides answers to common questions and helps users resolve issues quickly without needing additional support.

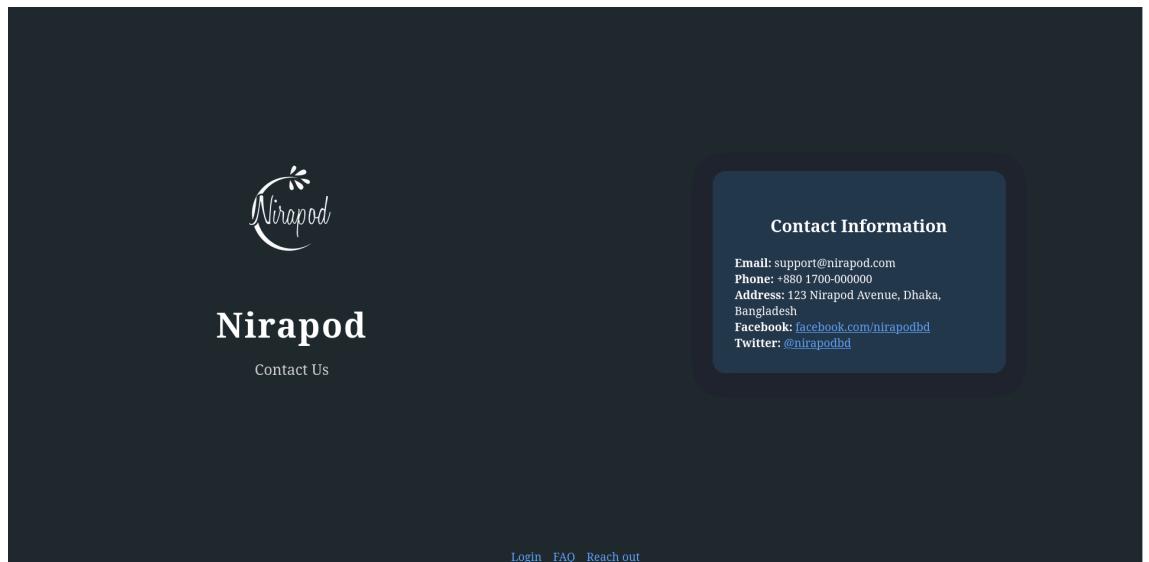


3. **ReachOut** - Users can submit complaints to the administrator regarding issues they encounter, such as login problems or other technical difficulties. When a complaint is submitted, the administrator receives an email notification containing the details of the issue. The administrator can then contact the user through the contact information provided in the complaint form to assist with

resolving the problem.

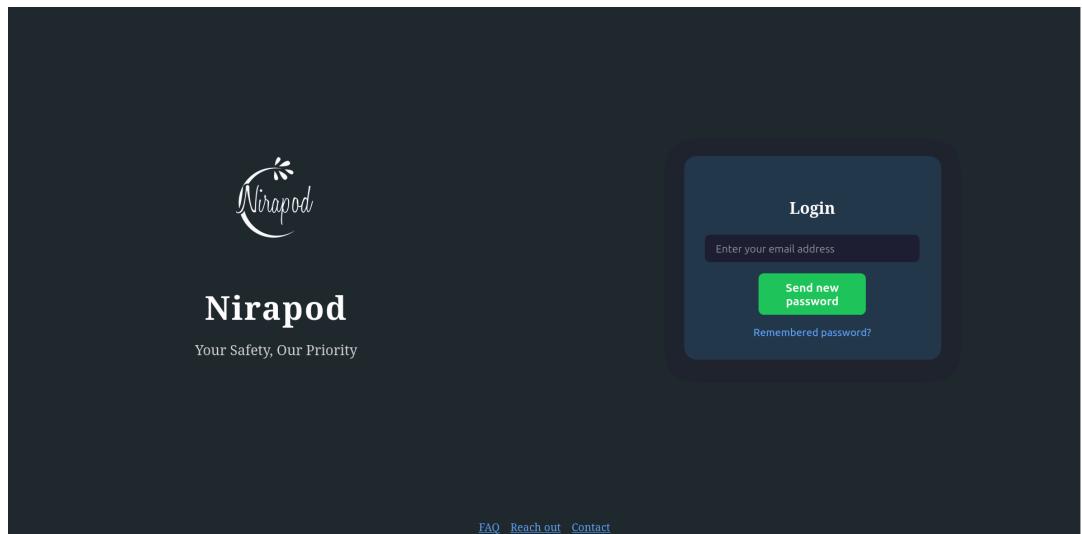


4. **Contract** - This page contains the official contact details of the organization, allowing users to reach out for support, inquiries, or further communication.

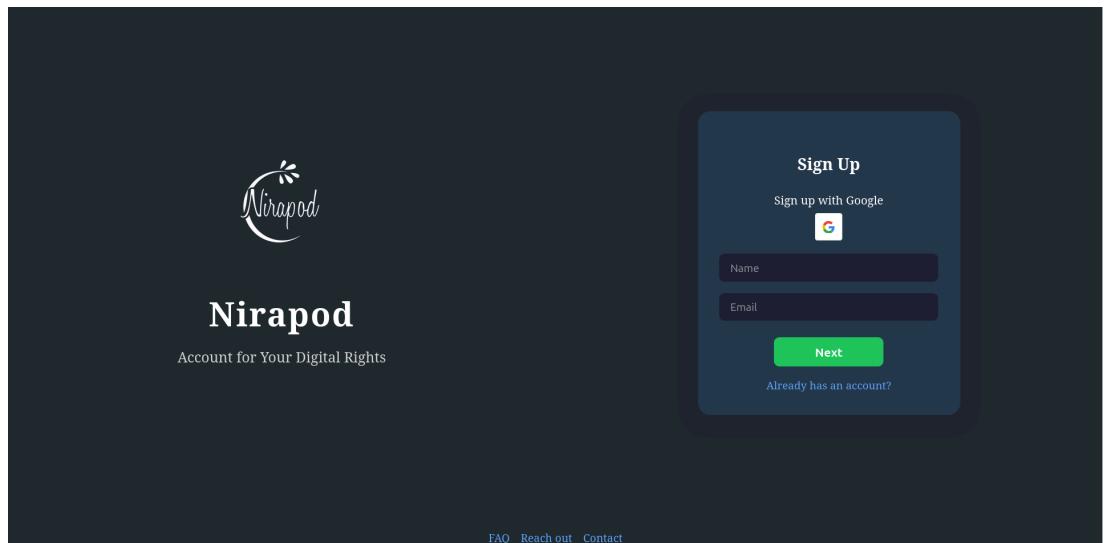


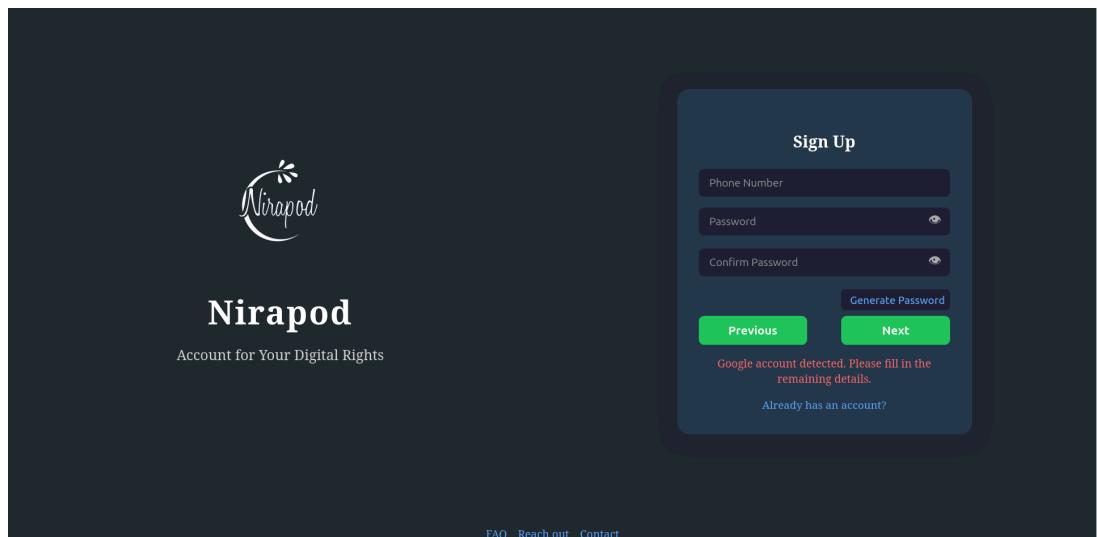
5. **Forget Password** - Users have the ability to reset their account password if needed. Upon requesting a password reset, the system will generate a new password and send it instantly to the user's registered email address. Users can

then use the new password to log in and update it after logging in.

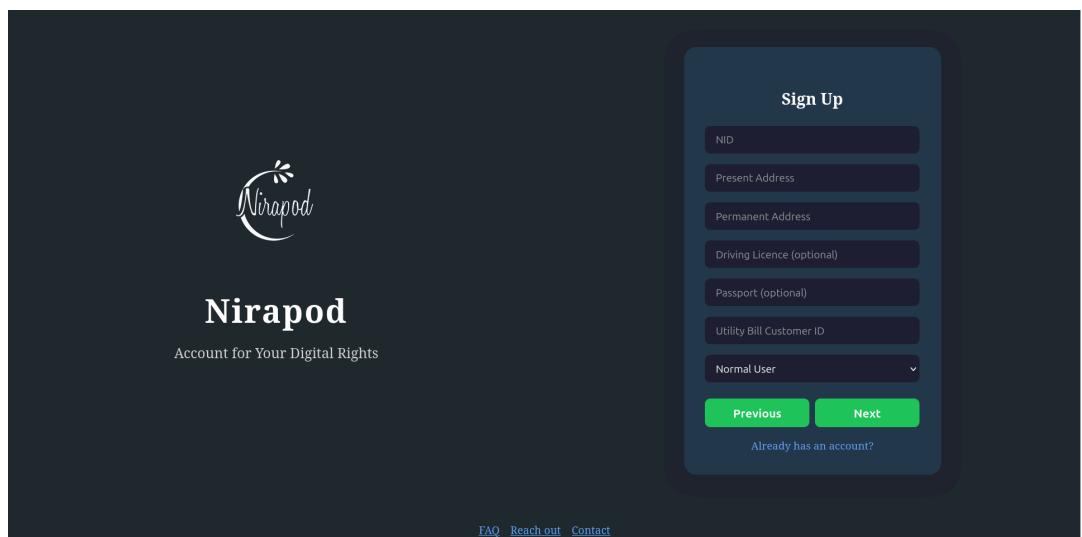


6. **Signup** - If a user does not have an account on the platform, they can easily create one through the signup process. During registration, an OTP (One-Time Password) will be sent to the user's provided email address for verification. Therefore, providing a valid and active email is mandatory. To confirm their identity, users must also submit personal details, such as their National ID (NID) and a utility bill. Additionally, users can select their user type during signup. By default, the system selects "Normal User." However, if the registrant is a police officer or holds a similar role, they can choose "Privileged User" and specify their type (e.g., "Police").

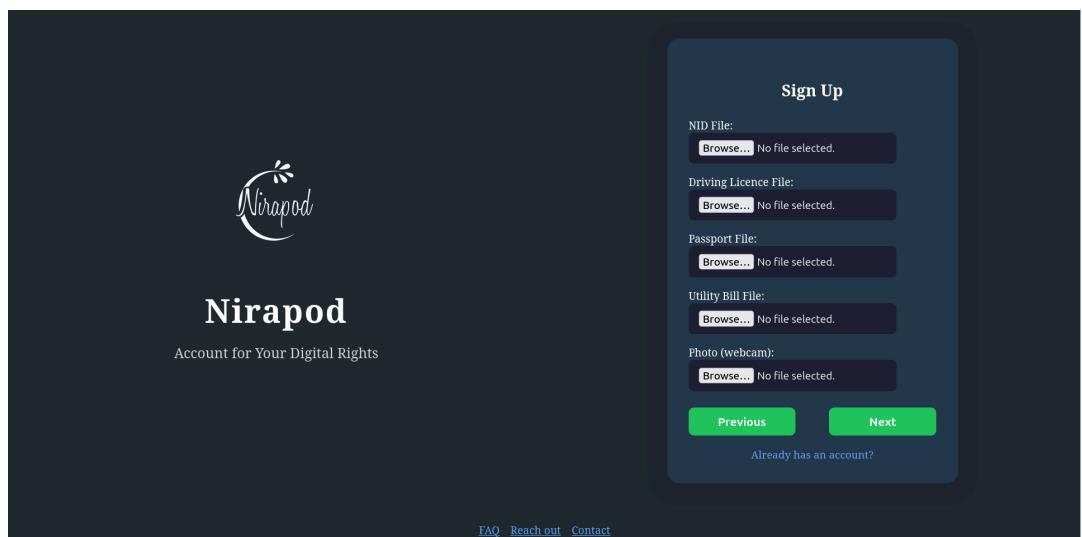




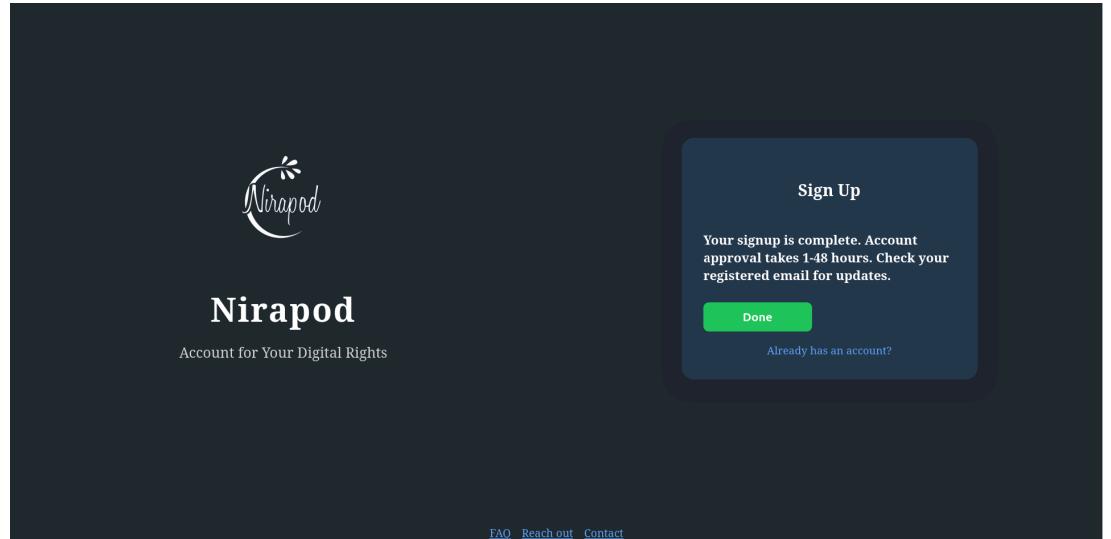
FAQ Reach out Contact



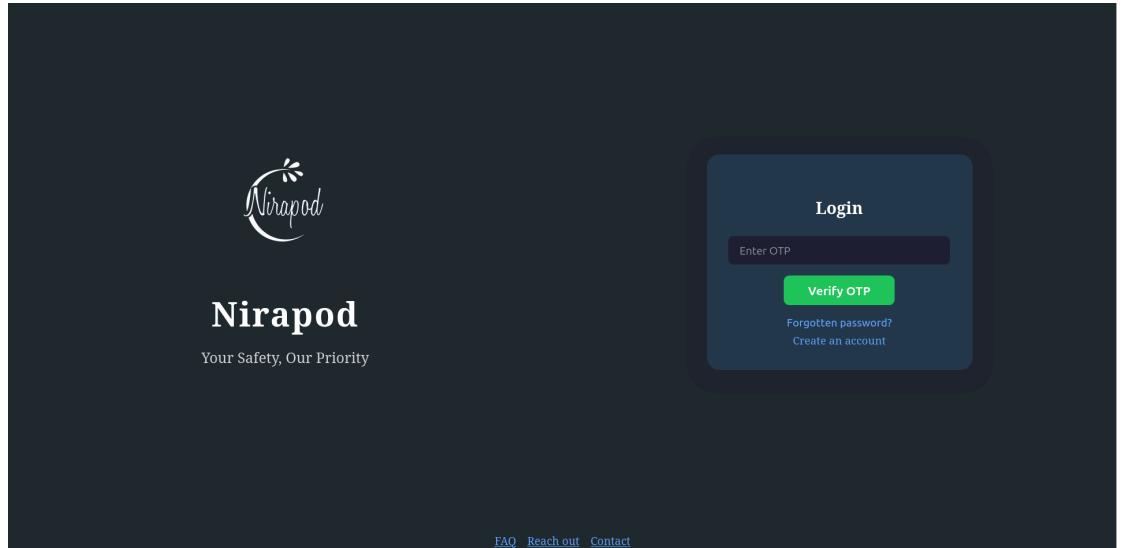
FAQ Reach out Contact



FAQ Reach out Contact



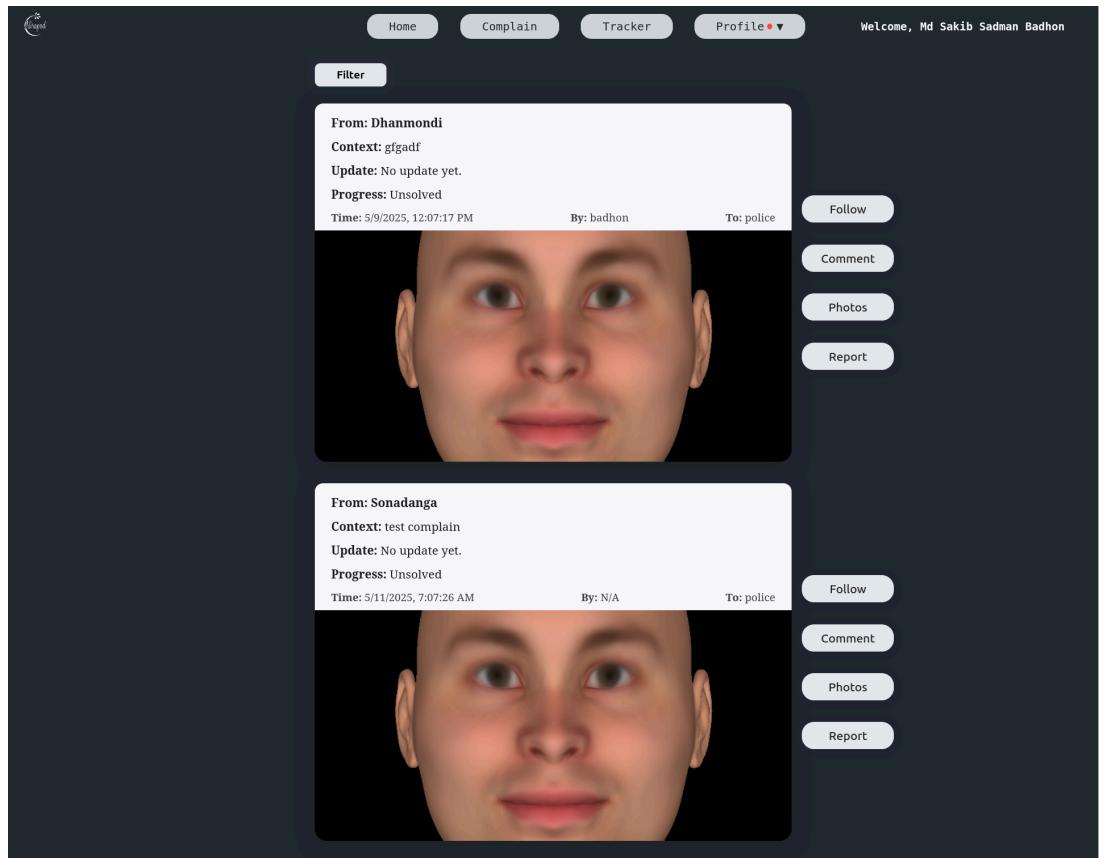
7. **OTP Verification-** To ensure that the actual user is attempting to log in, the platform includes an OTP (One-Time Password) verification step. When logging in with email and password, users will receive an OTP via their registered email, which they must enter to complete the login process. However, if a user chooses to log in using the Google Sign-In method, OTP verification is not required, as Google's authentication process is considered secure.



 sadmansakib022@gmail.com
to me ▾
Your OTP code is: 920815

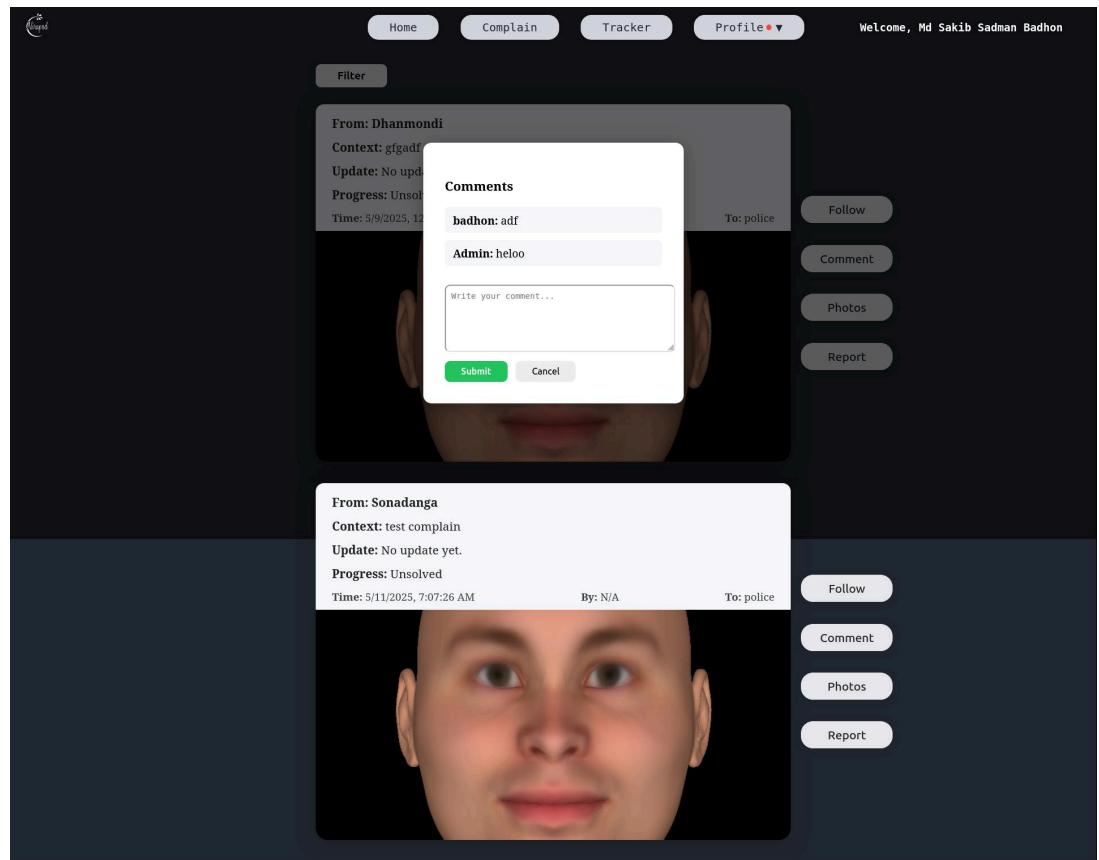
08:43 (1 minute ago) ⭐ ⓘ ↵ :

8. **Homepage** - After successfully logging in, users are redirected to the homepage. On the homepage, users can view all complaints — both those they have personally submitted and complaints made by other users, depending on their access permissions.

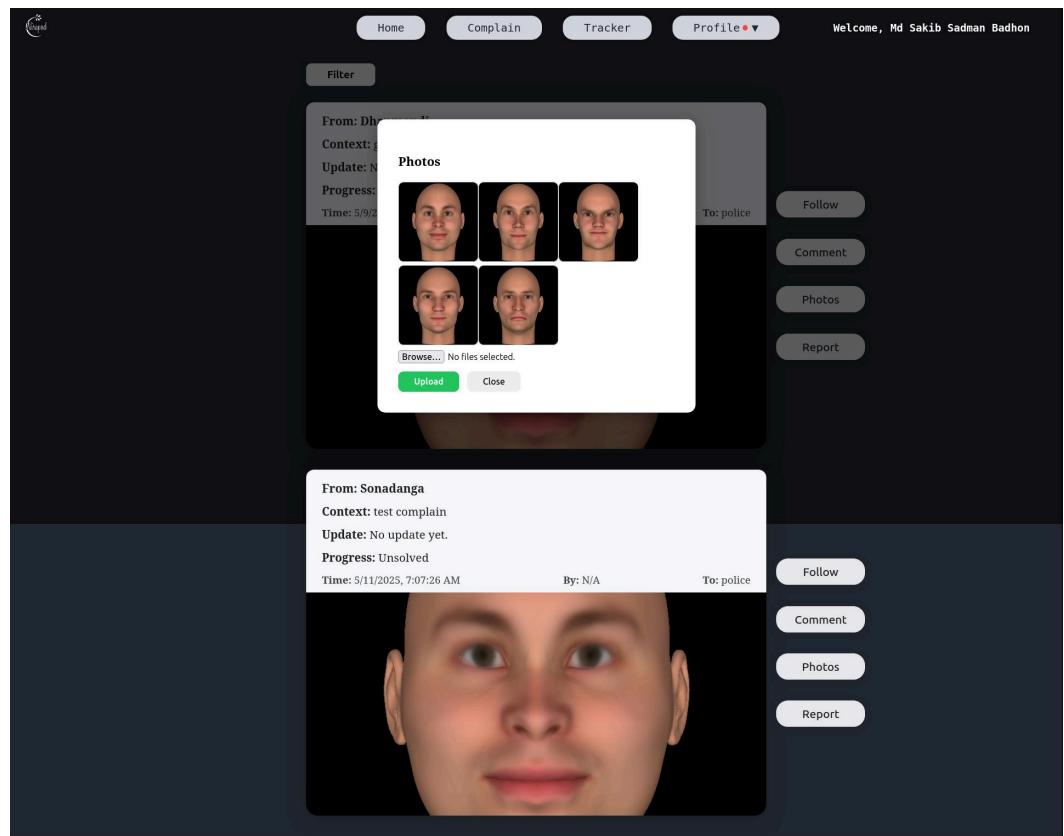


From the homepage, users also have the option to follow complaints. By following a post, users will receive notifications whenever a privileged user provides an update or when someone comments on that complaint. This

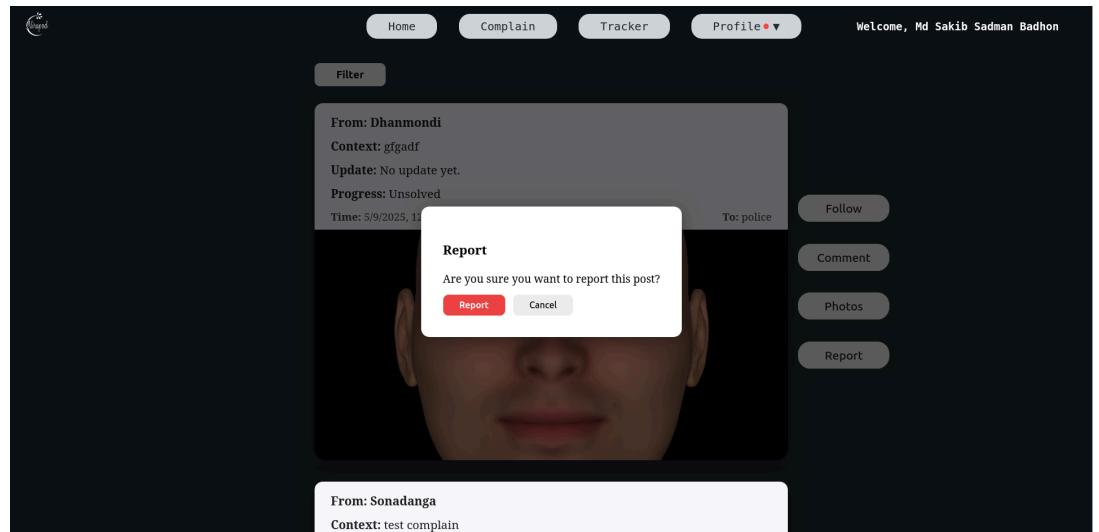
feature helps users stay informed about issues they are interested in.



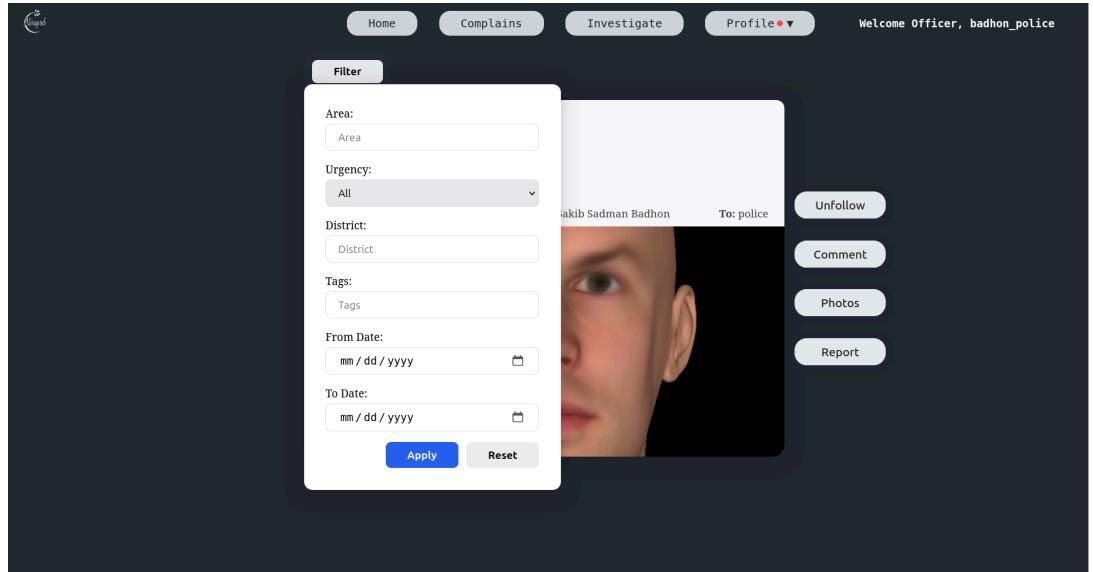
In the comment section of each complaint, users can view all comments made by other users. Users also have the ability to add their own comments to any post, allowing for discussion, feedback, or additional information sharing related to the complaint.



In the photo section, users can view all photos related to a specific incident or complaint. If the user wishes to contribute additional visuals, they also have the ability to upload and add their own photos to the post.



Users can report any post they find inappropriate or in violation of community guidelines by pressing the Report button. Once a report is submitted, the administrator gains access to all reports. The admin can then review the reports and take appropriate actions, such as deleting the report or removing the post entirely if necessary.

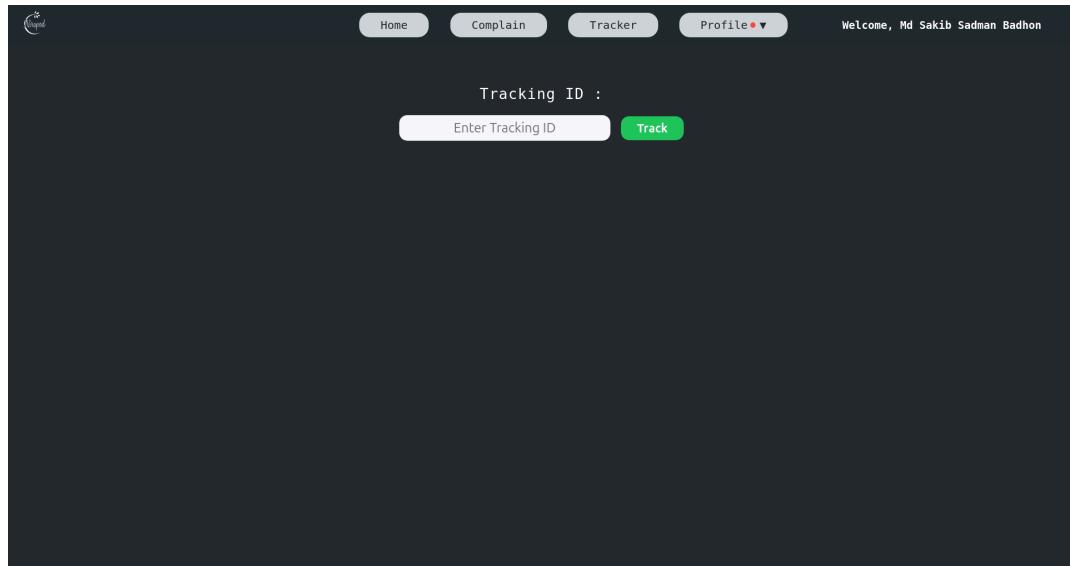


Users can use the Filter option to refine and sort posts based on various criteria. This allows users to view complaints that meet specific conditions, such as date, category, or status, making it easier to find relevant posts.

9. **Complain** - Users can submit complaints to higher authorities using this dedicated page. This page is also linked to the homepage, allowing users to directly post complaints to the timeline. If a user selects the "Post it on Timeline" option while submitting their complaint, the complaint will be publicly visible on the timeline for all users to see. Users can also opt to "Get Location" by granting location access through a browser popup. After providing the necessary permissions, the system will automatically capture and attach the user's current location to the complaint. Additionally, users have the option to upload multiple photos related to the complaint, enhancing the information provided.

Name :	Md Sakib Sadman Badhon
Phone :	01533024584
Email :	badhon495@gmail.com
NID :	12347777
Complain to :	Select
Urgency :	Select <small>*See Level</small>
District :	Select
Area :	Select
Location :	Full address <input type="button" value="Get Location"/>
Tag :	Ex: Phone Theft, Hijack, Fire
Details :	Write all the details about your complain
Photos :	<input type="button" value="Add photos"/> <input checked="" type="checkbox"/> Post it on timeline
<input type="button" value="Submit"/>	

10. **Tracker** - After submitting a complaint, the user will receive a tracking number for their complaint. Using this tracking number, the user can check the status and receive updates regarding their complaint's progress at any time.



11. **Update profile** - The Update Profile option is available in the dropdown menu of the user's profile. Using this feature, users can update their profile information, such as their name, email, and address. However, certain details, such as the National ID (NID), cannot be updated by the user for security and

verification purposes.

The screenshot shows a profile update form on a dark-themed website. At the top right, it says "Welcome, Md Sakib Sadman Badhon". Below is a circular placeholder for a profile photo with the text "Choose Photo". The form fields include:

- Name : Md Sakib Sadman
- Phone : 01533024584
- Email : badhon495@gmail.com
- NID : 12347777
- Present Address : cahnged_add_pos
- Permanent Address : adabor
- Utility Bill Customer ID : 1244444
- Utility Bill Photo : [Browse... No...d.]
- Passport : Passport
- Passport Image : [Browse... No...d.]
- Driving License : Driving License
- Driving License Image : [Browse... No...d.]

A large "Update" button is centered below the fields. At the bottom, there's a "Change Password" section with three input fields: "Old Password", "New Password", and "Confirm New Password", followed by a "Change Password" button.

12. **Your Complaints** - In this section, users can view all the complaints they have made. This allows them to easily track and manage their submitted complaints.

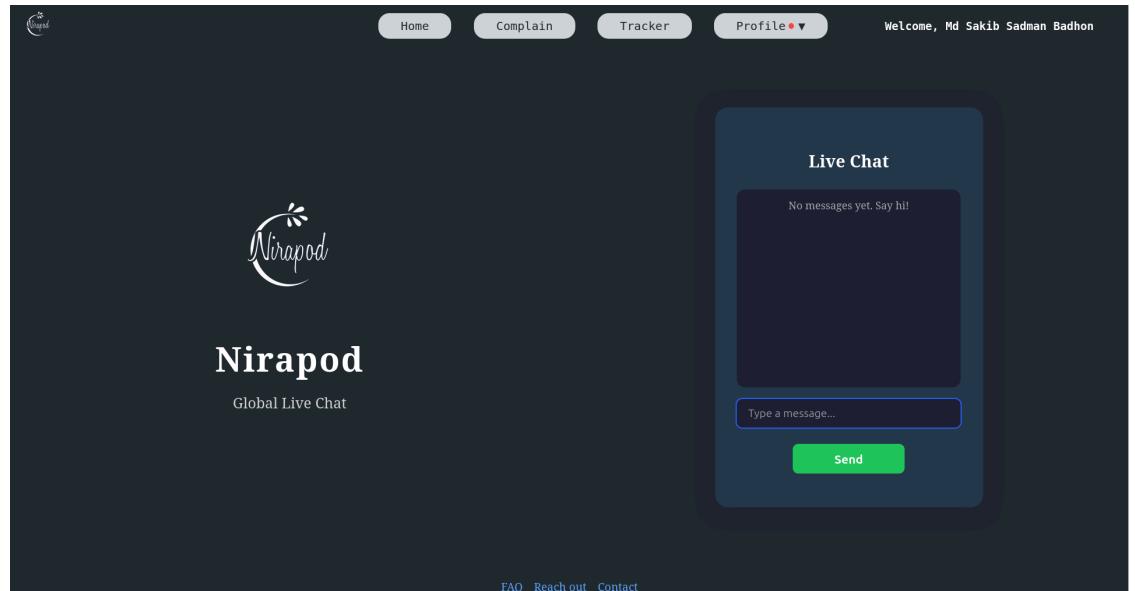
The screenshot shows a complaint tracking interface. At the top right, it says "Welcome, Md Sakib Sadman Badhon" and has a red "Unsolved" badge. Below is a table with one row of data:

Tracking ID : 1
Name : Md Sakib Sadman Badhon
Complained Time : 5/12/2025, 3:35:09 AM
Tag : phone theft
Urgency : High

13. **Notifications** - When a user follows a post, they will receive notifications for any updates made to the post, such as changes by privileged users or new comments. Even if the user is not actively browsing the platform, they will receive notifications about the updates. Once the user signs in, they will also

be notified of any new changes or activities related to the posts they are following.

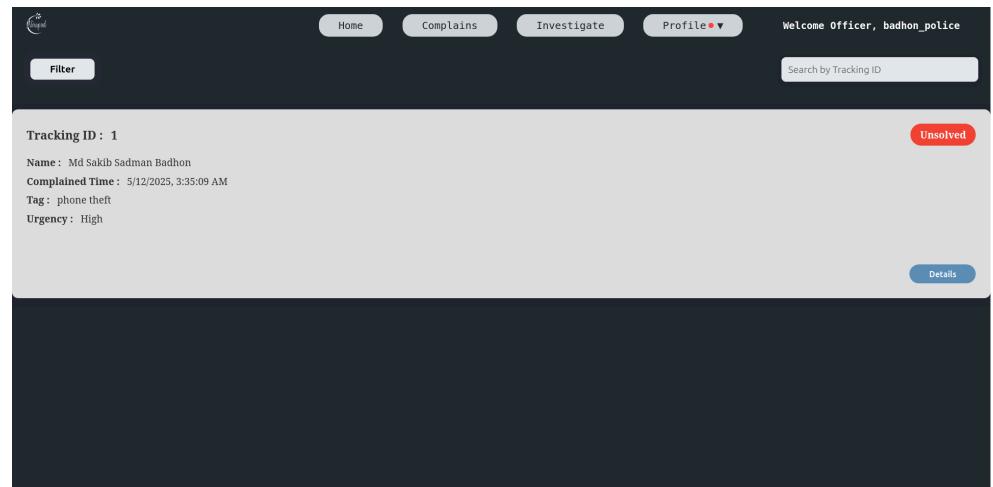
14. **Live chat** - Users have the ability to chat with other users directly on the platform. These chats are saved in the local storage of the user's browser. As a result, if the user reloads the page or logs out, previous messages will no longer be accessible. This chat feature is intended for non-emergency discussions between users, such as general inquiries or casual communication.



15. Privileged user

- a. **Login and signup** - Privileged users will follow the same login process as mentioned earlier. During the signup process, privileged users must select their user type as Privileged User and provide their specific verification ID (such as Police ID, etc.) to confirm their role. This ensures that only authorized individuals can access privileged user features.
- b. **Timeline** - Privileged users have access to the same timeline as other users. However, their view is restricted to only the complaints that are specifically directed to them. This ensures that privileged users can focus on the complaints relevant to their role while maintaining a streamlined experience. All the privileged user will have the same basic features like update profiles, live chat etc available.
- c. **Complaints** - From here a police officer or other privileged user will be able to see all the complaints that are filed to him. He can also see the details about a complaint by clicking the Details button, not only that, by filter and search option, he can easily filter the complaints by searching the tracking id or filtering based on the level and other

things.



- d. **Complain Details** - From here a privileged user can view the details of complaints directed to them. In addition to viewing, they have the ability to update the complaint status and provide information regarding the progress of resolving the issue. This feature enables privileged users to keep complainants informed about the steps taken to address their concerns.

The screenshot displays a web-based police investigation system. At the top right, it says "Welcome Officer, badhon_police". Below the header are several input fields for a complaint:

- Name: Md Sakib Sadman Badhon
- Tracking ID: 1
- NID: 1111111111
- Tags: phone theft
- Urgency: High
- District: Dhaka
- Area: Dhanmondi
- Location: 23.7716702,90.3565688

A map shows the location of the reported problem in Dhanmondi, Dhaka.

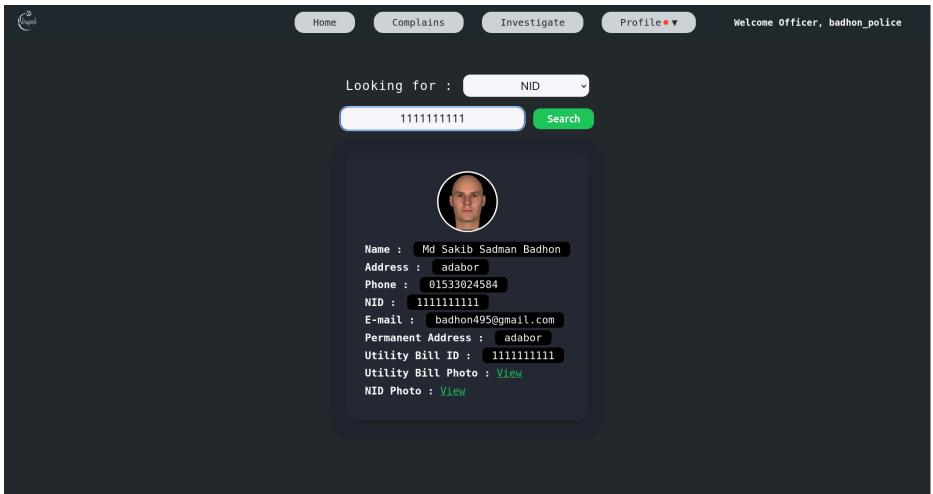
Below the map are more fields:

- Timestamp: 5/12/2025, 3:35:09 AM
- Details: details
- Photos: complaint-photo-0
- Current Status: Unsolved

At the bottom, there's an "Update Complaint Status" section with a dropdown for "Status" set to "Unsolved" and a note field for "Update Note" containing "Write your update here...". A green "Update Status" button is at the bottom right.

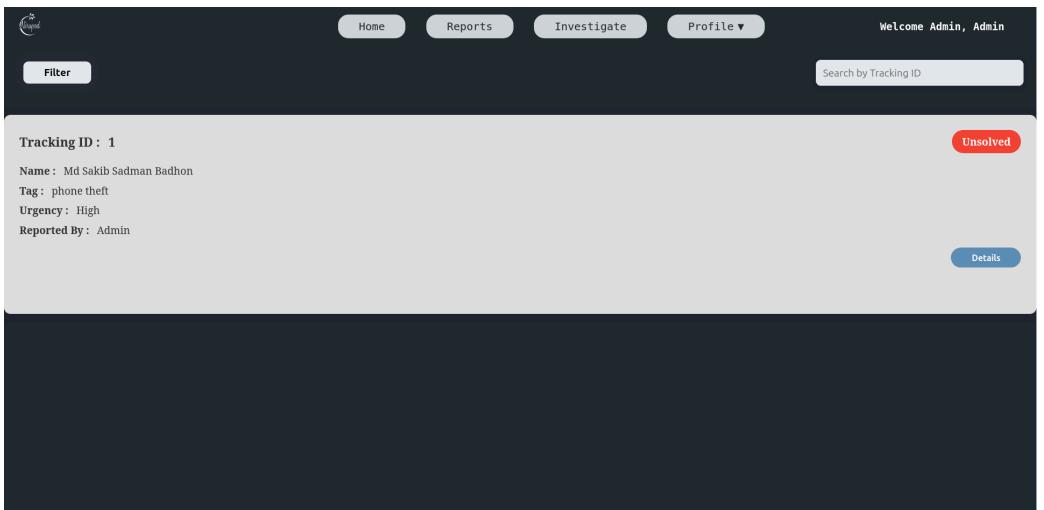
- e. **Investigation** - The Investigation feature is exclusive to police officers, as they may require access to user details for various investigative purposes. Through this feature, police officers can search for a user and retrieve their details using identifiers such as National ID (NID), Driving License, or Passport. This ensures that law enforcement can access necessary information to carry out investigations efficiently

and securely.



16. **Admin** - The Admin is the primary controller of the website. Admins have full access to all complaints, including the ability to delete any complaint submitted by users. Additionally, admins can view all reports made by users and take appropriate actions based on the nature of the report. This includes investigating, resolving, or removing content that violates community guidelines.

- a. **Reports** - This page is similar to the complaint page available to regular users. However, instead of viewing complaints, the Admin will see all user reports against posts. Admins can review these reports to assess whether the posts violate community guidelines and take the appropriate actions, such as deleting the report or removing the post.



- b. **Report Details** - The Report Details page is similar to the complaint details page, but with two additional features specifically for admins.
 - i. Delete Post: This feature allows the admin to delete a user's post if it violates community guidelines or is deemed inappropriate.
 - ii. Delete Report: This button enables the admin to remove a user's report if it is determined to be invalid or unjustified.

These actions help the admin effectively manage and moderate content on the platform.

The screenshot displays a web-based administrative interface for managing complaints. At the top, there is a navigation bar with links for Home, Reports, Investigate, Profile, and Welcome Admin, Admin. Below the navigation bar, the main content area shows a detailed view of a complaint report and an update status form.

Complaint Report Details:

- Name: Md Sakib Sadman Badhon
- Tracking ID: 1
- NID: 1111111111
- Tags: phone theft
- Urgency: High
- District: Dhaka
- Area: Dhanmondi
- Location: 23.7716702,90.3565688

Map: A map of the Dhanmondi area in Dhaka, showing the location of the reported problem. A blue marker indicates the exact location, and a tooltip says "The problem was reported here".

Timestamp: 5/12/2025, 3:35:09 AM

Details: details

Photos: complaint-photo-0

Current Status: Unsolved

Update Complaint Status Form:

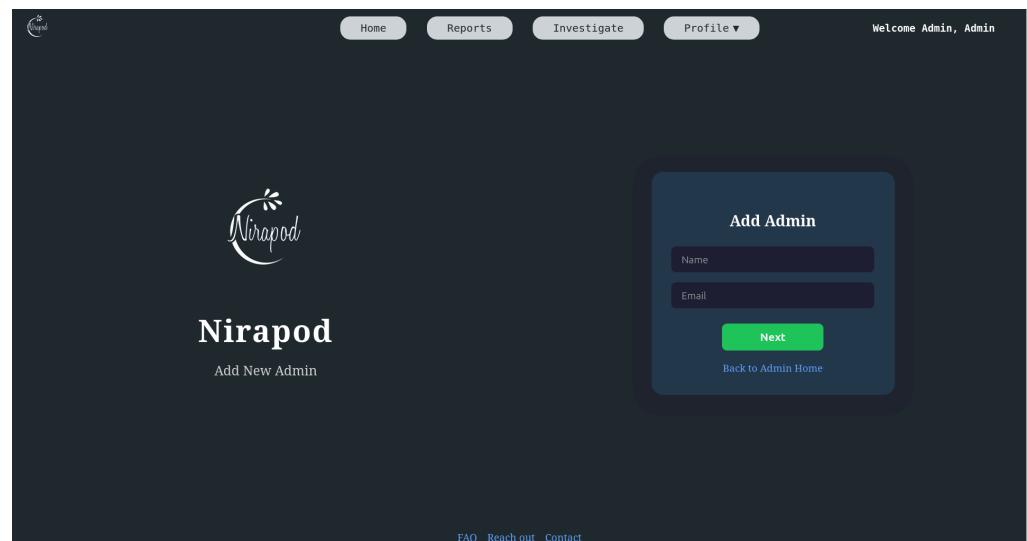
Status: Unsolved

Update Note: Write your update here...

Buttons at the bottom of the update form: Update Status (green), Delete Report (orange), and Delete Post (red).

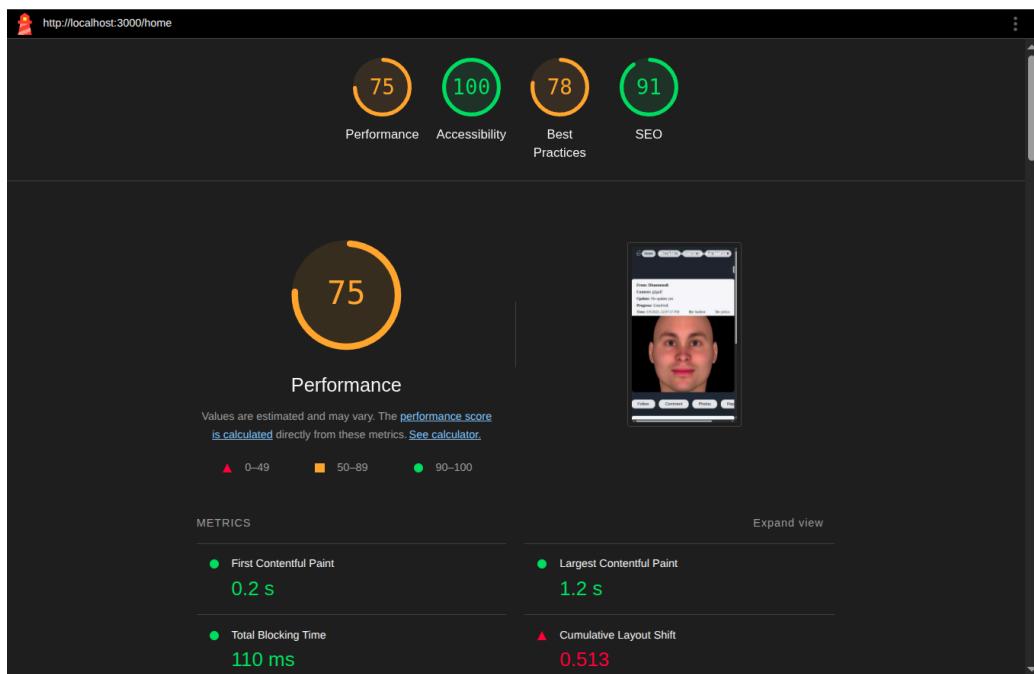
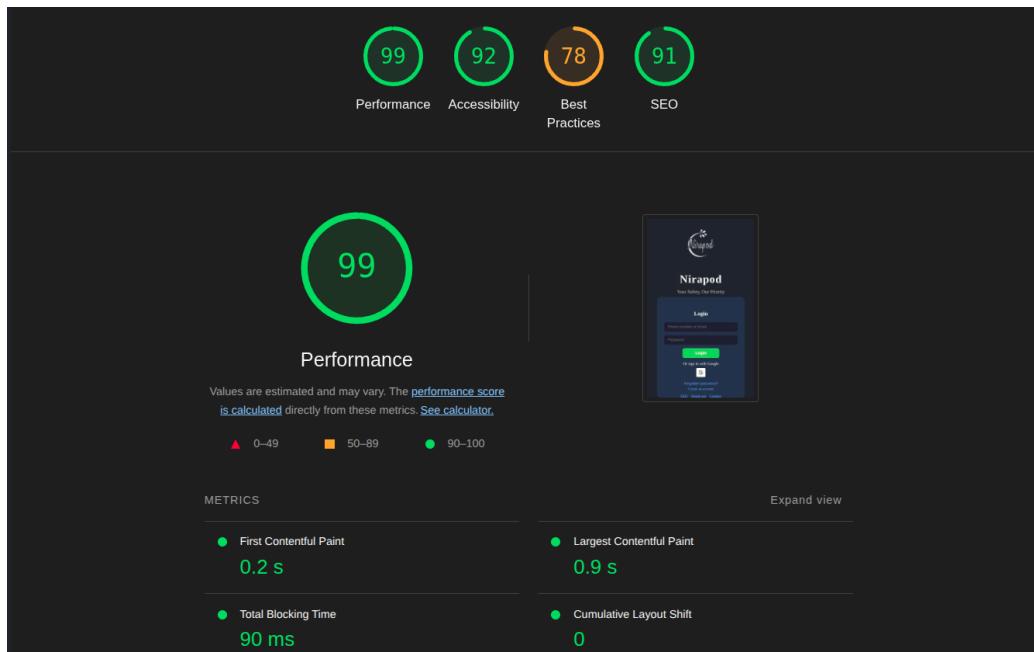
- c. **Add Admin** - The System Admin has the ability to appoint a new admin. To add a new admin, the system admin must provide the Name and Email of the individual being appointed. This process is similar to the signup process for normal or privileged users, with the key difference being the role assigned to the new user. Once added, the new admin will have the appropriate permissions and access levels

based on their role.

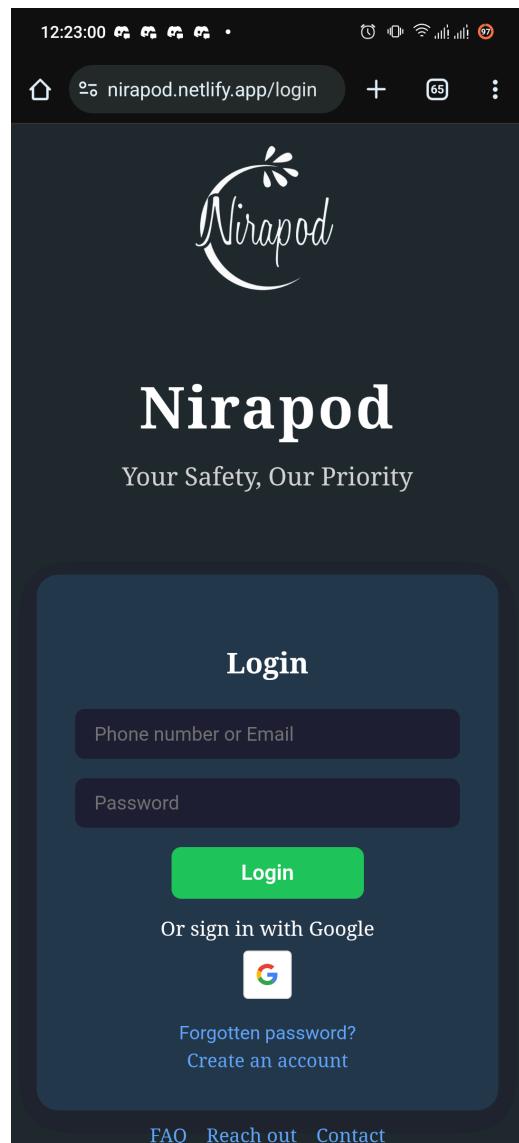


● Performance and Network Analysis

1. Lighthouse Report



2. Mobile Site -



- **Github Repo [Public] Link - [Nirapod Code](#)**
- **Link to Deployed Project - [Nirapod](#)**

● Individual Contribution

Group member - 01	
Name: Saima Sobahan	Student ID: 22101109
Functional Requirements, which are developed by this member:	
1. Discussion using Comment and Photo Upload	
2. Timeline posts and filtering to find and spread criminal activity awareness	
3. Notifications and alerts	
4. City Corporation Privileged User's dashboard (view case, update status, update details)	

Group member - 02	
Name: Tawhid Chowdhury	Student ID: 22101182
Functional Requirements, which are developed by this member:	
1. User Profile management (view and update profile details, change password)	
2. Emergency Service Requests (request to Police, Fireservice ,City Corporation, Animal Welfare etc. with live location sharing)	
3. Case Progress Tracking (Assigned officer details, status updates) and All Complaint list	
4. Animal Shelter & Protection Privileged Users' dashboard (view case, update status, update details)	

Group member - 03	
Name: Md Sakib Sadman Badhon	Student ID: 22341182
Functional Requirements, which are developed by this member:	
1. User Authentication and Verification (Sign Up, Login)	
2. Evidence Submission (Multiple photos and files), Filter Option	
3. Community Live Chat for non-emergency discussion	
4. External API integration	

● References

1. [Postgresql Documentation](#)
2. [Spring Documentation](#)
3. [React Quickstart](#)
4. [tutorialspoint](#)
5. [!\[\]\(b918af89b3d29b3fd1d13eb4ddf791b5_img.jpg\) Spring Boot Tutorial for Beginners \[2025\]](#)
6. [!\[\]\(da9ce1d6b0f2a0fcf0d0a74eb777d743_img.jpg\) PostgreSQL Tutorial for Beginners](#)
7. [!\[\]\(9b41209e48913d157f03a61d958bbddd_img.jpg\) React Tutorial for Beginners](#)