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.

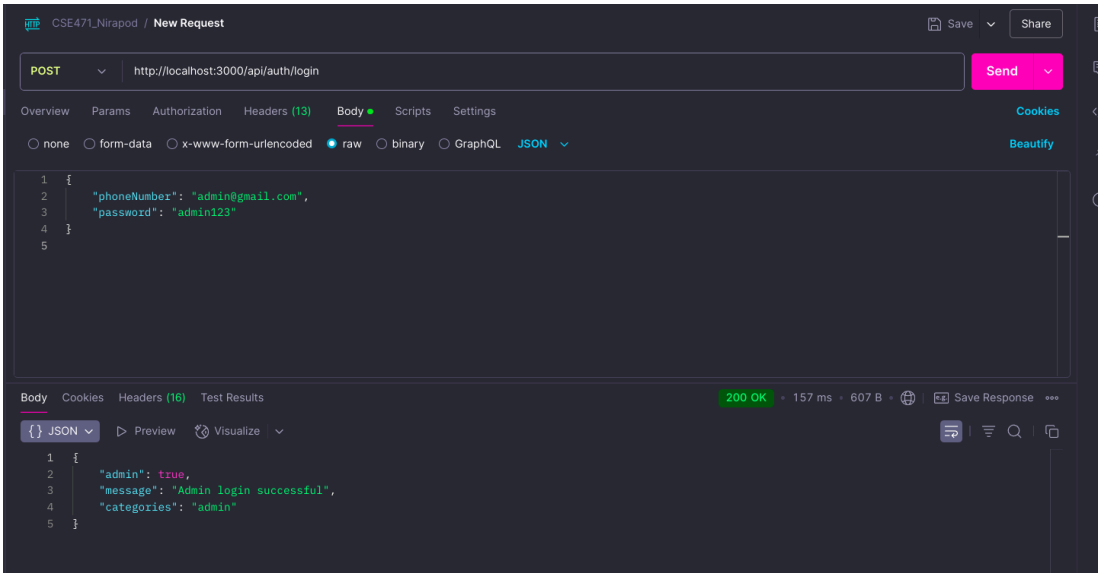
- **Special Issues or Constraints**
    - Must comply with data privacy laws and government regulations for handling personal ID information (NID, passport, etc.).
    - OTP delivery and live chat depend on third-party API reliability.
    - Needs robust moderation tools to manage abuse and fake reporting. 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)

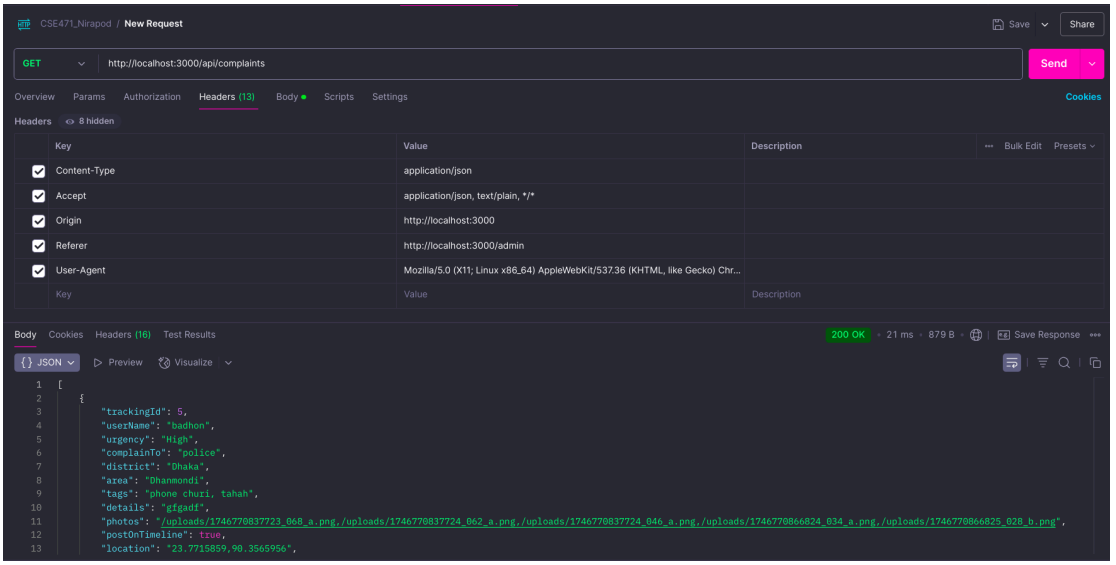
- # Backend Development

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.



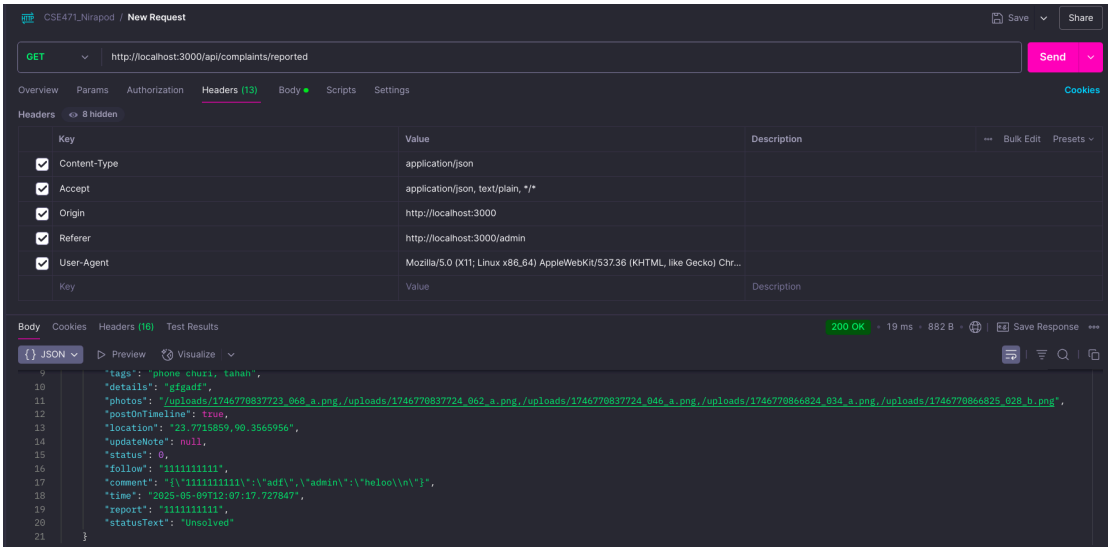

```java
@PostMapping("/login")
public ResponseEntity<?> login(@RequestBody Map<String, String> payload) {
    String identifier = payload.get(key:"phoneNumber"); // can be phone or email
    String password = payload.get(key:"password");
    Optional<User> userOpt = authService.findByPhoneNumber(identifier);
    if (userOpt.isEmpty()) {
        userOpt = authService.findByEmail(identifier);
    }
    if (userOpt.isEmpty() || !authService.checkPassword(userOpt.get(), password)) {
        return ResponseEntity.status(status:401).body(body:"Invalid credentials");
    }
    User user = userOpt.get();
    // If admin, skip OTP and log in directly
    if (user.getEmail().equalsIgnoreCase(anotherString:"admin@gmail.com") && user.
    getCategories().equalsIgnoreCase(anotherString:"admin")) {
        return ResponseEntity.ok(Map.of(
            k1:"message", v1:"Admin login successful",
            k2:"categories", user.getCategories(),
            k3:"admin", v3:true
        ));
    }
    String otp = otpService.generateOtp(identifier); // Use the exact identifier provided by
    user
    // 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.



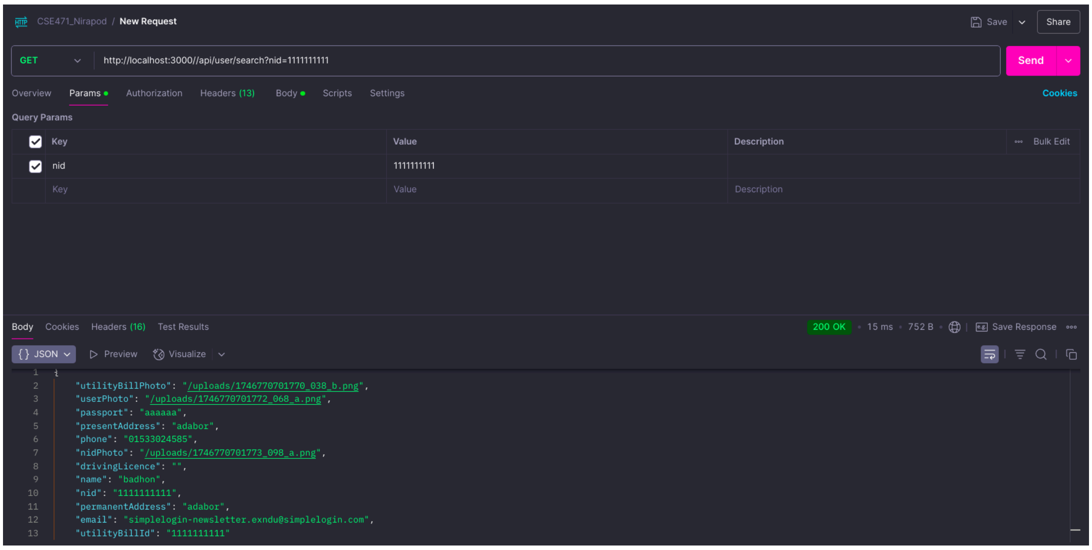

```java
    // Fetch all complaints
    @GetMapping("/complaints")
    public List<ComplaintResponse> getAllComplaints(
            @RequestHeader(value = "X-User-Category", required = false) String userCategory) {
        logger.info(msg:"Fetching all complaints");
        List<Complaint> complaints;
        if (userCategory != null && userCategory.equalsIgnoreCase(anotherString:"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(anotherString:"fire")) {
            logger.info(msg:"User is fire, returning only fire complaints");
            complaints = repository.findByComplainToIgnoreCase(complainTo:"Fire");
        }

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

        else if (userCategory != null && userCategory.equalsIgnoreCase(anotherString:"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.




```java
// Fetch all reported complaints
@GetMapping("/complaints/reported")      You, yesterday • Almost fixed everything …
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.findByNid(c.getNid())
                    .map(u -> u.getName())
                    .orElse(other:"");
            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;
}
```

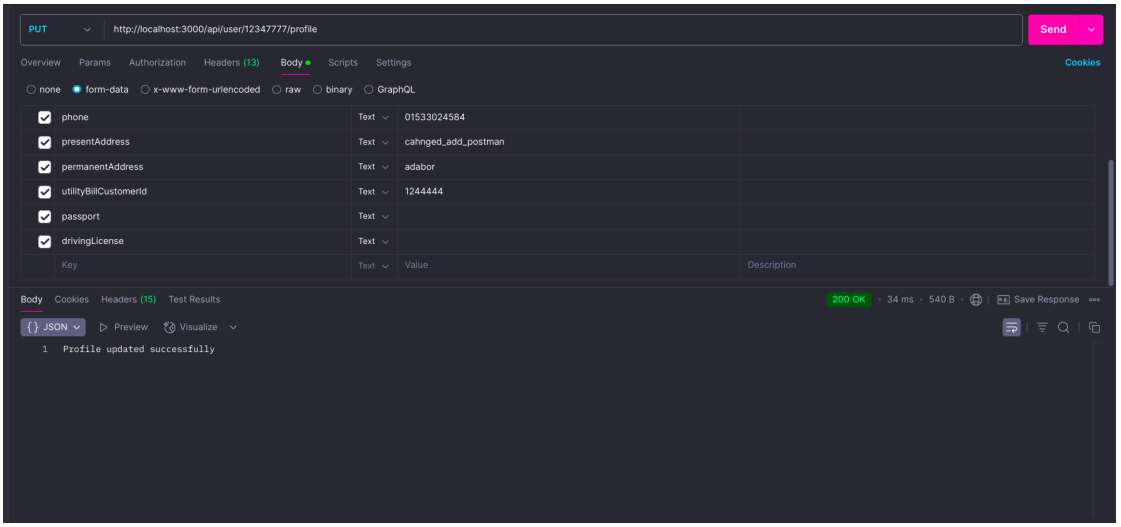
4. **Investigate -** Using this feature, certain users (privileged users) are authorized to access information stored in the project's database. Privileged users can retrieve user details by searching with one of the following identification numbers: National ID (NID), Passport Number, Driving License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




```java
@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.findByNid(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                          | $2a$10
$RVOs1DqXq4ic79RCAGgJL.eP7NeMH1h8GDcH.TADFgQVGnmcuaA1C | Md Sakib Sadman Badhon
| 01533024584 | adabor          | adabor          |                        |
              |                 | 1244444                  | /uploads/1746
925151865_046_a.png | /uploads/1746925151868_034_a.png | /uploads/1746925151869_
:
```

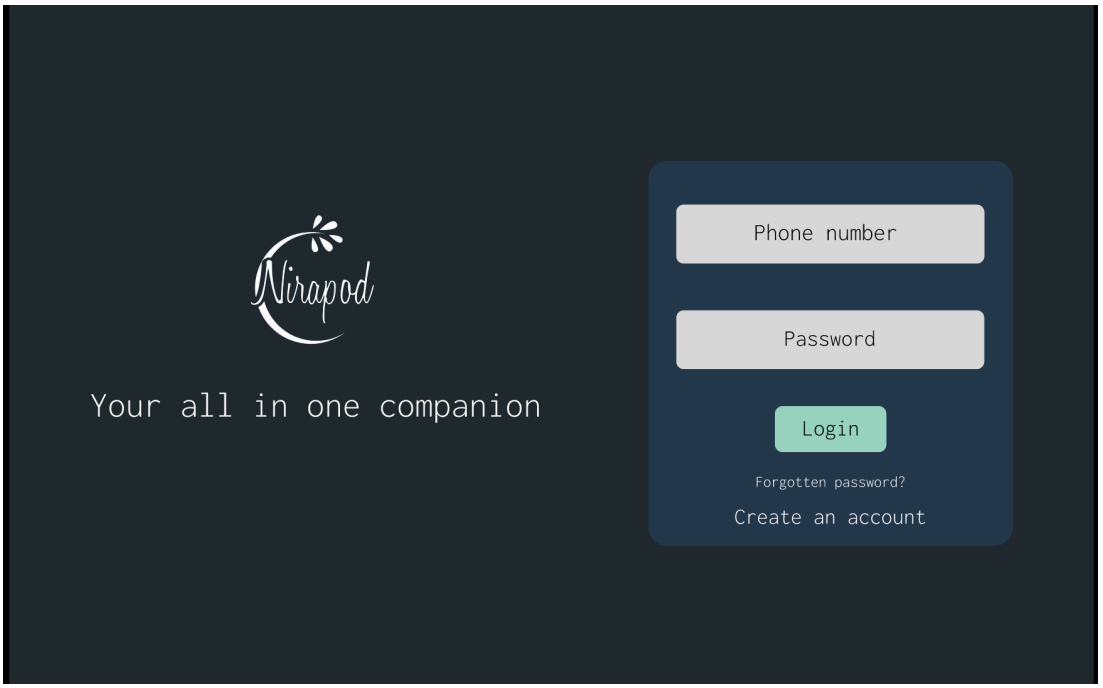

PUT
http://localhost:3000/api/user/12347777/profile
Send
Overview
Params
Authorization
Headers (13)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
phone
Text
01533024584
presentAddress
Text
cahnged_add_postman
permanentAddress
Text
adabor
utilityBillCustomerId
Text
1244444
passport
Text
drivingLicense
Text
Key
Text
Value
Description
Body
Cookies
Headers (15)
Test Results
200 OK
34 ms
540 B
Save Response
{ } JSON
Preview
Visualize
1 Profile updated successfully


12347777 | normal | badhon495@gmail.com | $2a$10
$RVOs1DqXq4ic79RCAGgJL.eP7NeMH1h8GDcH.TADFgQVGnmcuaA1C | Md Sakib Sadman Badhon
| 01533024584 | cahnged_add_postman | adabor | |
| | | | 1244444 | /uploads/
1746925151865_046_a.png | /uploads/1746925151868_034_a.png | /uploads/1746925151

```java
@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(required = false) MultipartFile photo,
        @RequestParam(required = false) MultipartFile utilityBillPhoto,
        @RequestParam(required = false) MultipartFile passportImg,
        @RequestParam(required = false) MultipartFile drivingLicenseImg
) throws IOException {
    Optional<User> userOpt = userRepository.findByNid(nid);
    if (userOpt.isEmpty()) return ResponseEntity.status(status:404).body(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();
```

```java
    // 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");
}
```

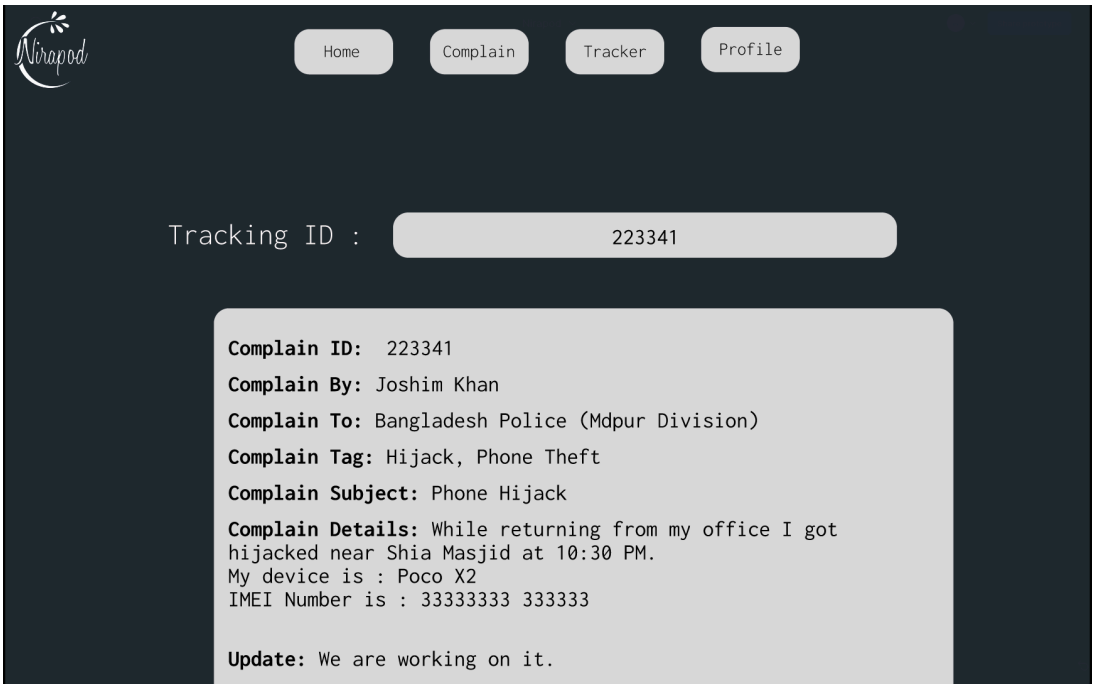
- # User Interface Design
    1. **Figma Project Link -** [Nirapod_Figma]
    2. **Login**



3. **Homepage**

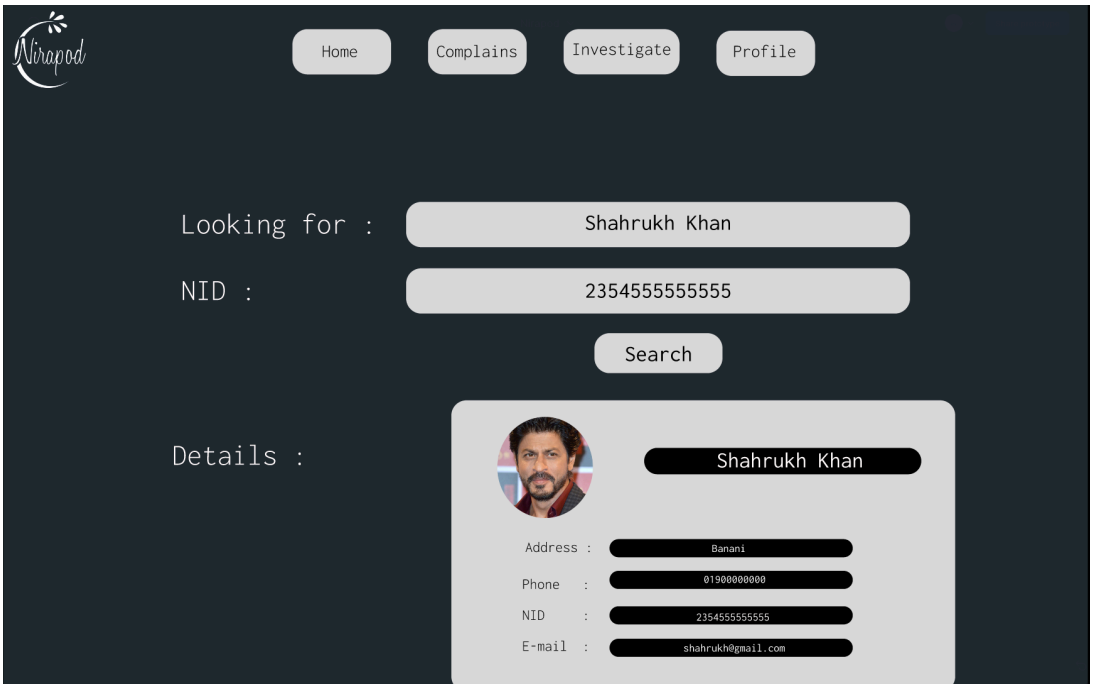
4. **Create Complaint**

5. **Tracker**



6. **Complain List**

7. **Investigate**



- # 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.

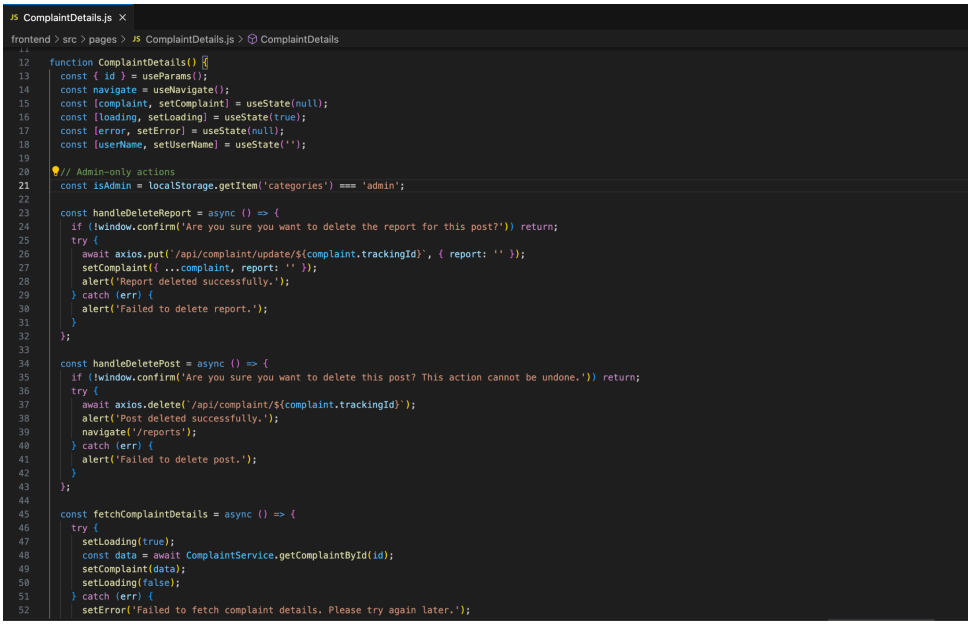
```
13
14  function Home() {
15    const [posts, setPosts] = useState([]);
16    const [filters, setFilters] = useState({ area: '', urgency: '', district: '', tags: '', fromDate: '', toDate: '' });
17    const [page, setPage] = useState(0);
18    const [hasMore, setHasMore] = useState(true);
19    const [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();
33
```

    - **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

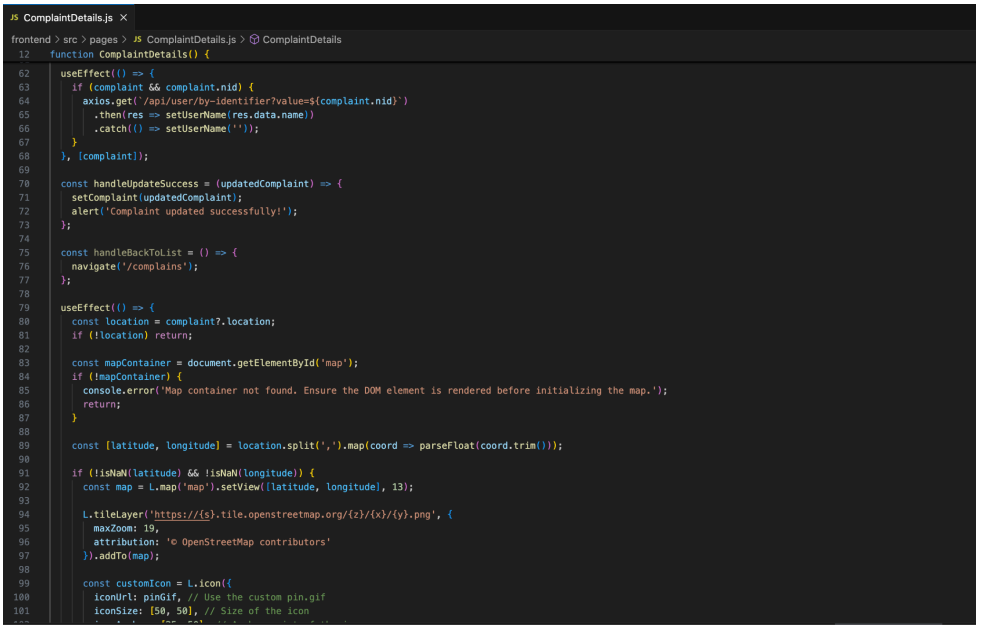
Complaints, Live Chat, and Logout. Upon clicking the pages are displayed.

```
if (categories === 'police') {
  return (
    <nav className="navbar-custom">
      <div className="navbar-logo-box" onClick={() => navigate('/home')} style={{ cursor: 'pointer' }}>
        <img src={logo} alt="Nirapod Logo" className="navbar-logo-img" />
      </div>
      <div className="navbar-btn-group">
        <a href="/home" className="navbar-btn">Home</a>
        <a href="/complains" className="navbar-btn">Complains</a>
        <a href="/investigate" className="navbar-btn">Investigate</a>
        <div className="navbar-profile-dropdown" ref={dropdownRef}>
          <button className="navbar-btn" onClick={() => setDropdownOpen(v => !v)}>
            Profile {unreadCount > 0 && <span className="navbar-notification-dot"></span>} <span style={{marginLeft: 6}}>▼</span>
          </button>
          {dropdownOpen && (
            <div className="navbar-dropdown-menu">
              <Link to="/profile" className="navbar-dropdown-item" onClick={() => setDropdownOpen(false)}>Update Profile</Link>
              <Link to="/my-complains" className="navbar-dropdown-item" onClick={() => setDropdownOpen(false)}>Your Complains</Link>
              <NotificationLink />
              <Link to="/livechat" className="navbar-dropdown-item" onClick={() => setDropdownOpen(false)}>Live Chat</Link>
              <button className="navbar-dropdown-item" onClick={handleLogout}>Logout</button>
            </div>
          )}
        </div>
      </div>
      <div className="navbar-welcome-msg">Welcome Officer, {userName}</div>
    </nav>
  );

}
```

- **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.

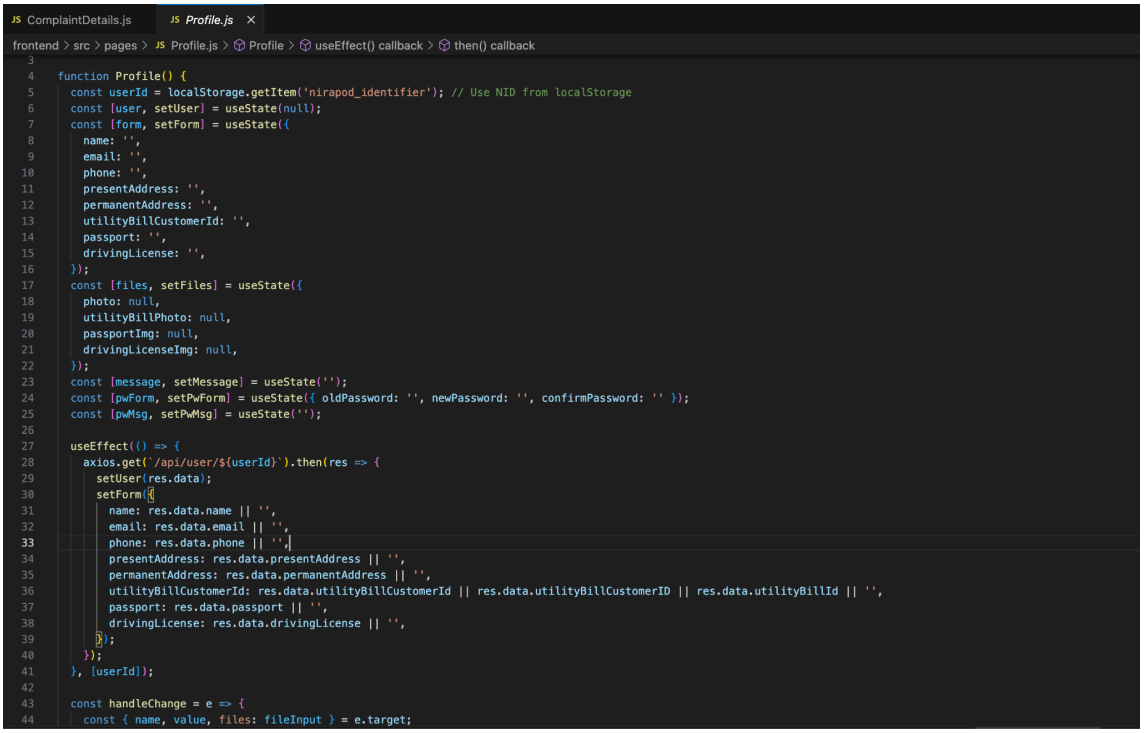


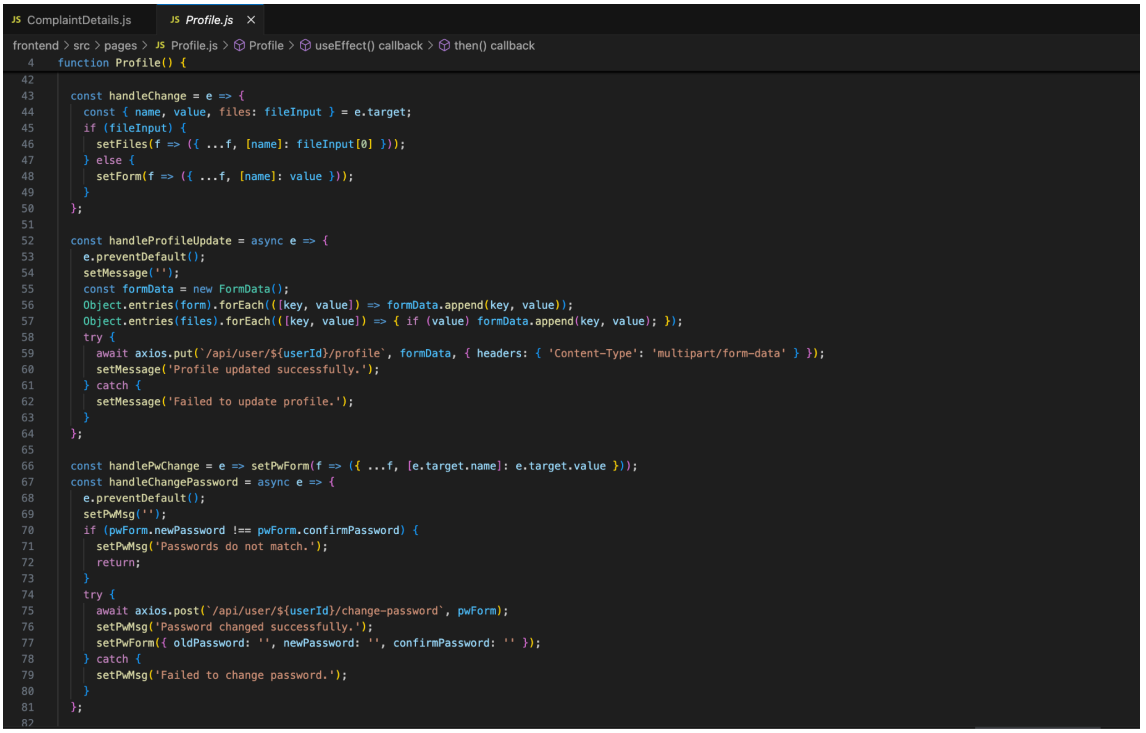
```
function ComplaintDetails() {
  const { id } = useParams();
  const navigate = useNavigate();
  const [complaint, setComplaint] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);
  const [userName, setUserName] = useState('');

  // Admin-only actions
  const isAdmin = localStorage.getItem('categories') === 'admin';

  const handleDeleteReport = async () => {
    if (!window.confirm('Are you sure you want to delete the report for this post?')) return;
    try {
      await axios.put(`/api/complaint/update/${complaint.trackingId}`, { report: '' });
      setComplaint({ ...complaint, report: '' });
      alert('Report deleted successfully.');
    } catch (err) {
      alert('Failed to delete report.');
    }
  };

  const handleDeletePost = async () => {
    if (!window.confirm('Are you sure you want to delete this post? This action cannot be undone.')) return;
    try {
      await axios.delete(`/api/complaint/${complaint.trackingId}`);
      alert('Post deleted successfully.');
      navigate('/reports');
    } catch (err) {
      alert('Failed to delete post.');
    }
  };

  const fetchComplaintDetails = async () => {
    try {
      setLoading(true);
      const data = await ComplaintService.getComplaintById(id);
      setComplaint(data);
      setLoading(false);
    } catch (err) {
      setError('Failed to fetch complaint details. Please try again later.');
```



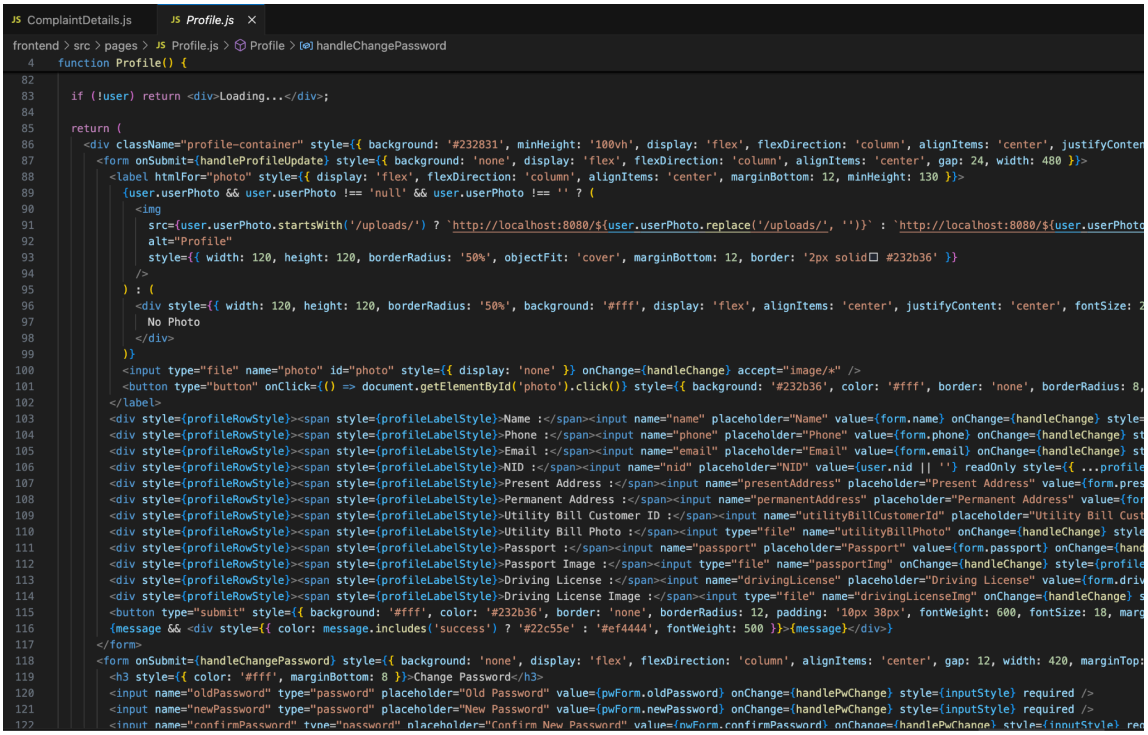
```
function ComplaintDetails() {
  useEffect(() => {
    if (complaint && complaint.nid) {
      axios.get(`/api/user/by-identifier?value=${complaint.nid}`)
        .then(res => setUserName(res.data.name))
        .catch(() => setUserName(''));
    }
  }, [complaint]);

  const handleUpdateSuccess = (updatedComplaint) => {
    setComplaint(updatedComplaint);
    alert('Complaint updated successfully!');
  };

  const handleBackToList = () => {
    navigate('/complains');
  };

  useEffect(() => {
    const location = complaint?.location;
    if (!location) return;

    const mapContainer = document.getElementById('map');
    if (!mapContainer) {
      console.error('Map container not found. Ensure the DOM element is rendered before initializing the map.');
      return;
    }

    const [latitude, longitude] = location.split(',').map(coord => parseFloat(coord.trim()));

    if (!isNaN(latitude) && !isNaN(longitude)) {
      const map = L.map('map').setView([latitude, longitude], 13);

      L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
        maxZoom: 19,
        attribution: '© OpenStreetMap contributors'
      }).addTo(map);

      const customIcon = L.icon({
        iconUrl: pinGif, // Use the custom pin.gif
        iconSize: [50, 50], // Size of the icon
```

- **Profile.js -** The Profile component is a user interface that allows authenticated users to view, update, and manage their personal profile and change their password. It fetches user data based on a unique identifier stored in localStorage and populates a form with fields like name, email, phone, addresses, passport, and driving license details. It also supports uploading profile and verification documents (e.g., photo, utility bill, passport image). Users can modify their information, and upon submission, the data—including any uploaded files—is sent to the backend via a PUT request using FormData. The component includes feedback messages for success or failure. Additionally, it provides a separate form to securely change the user's password, verifying that the new password and confirmation match before sending a request to the server. It uses inline styling and basic form validation, ensuring a responsive, styled, and user-friendly experience.


ComplaintDetails.js
Profile.js
frontend > src > pages > Profile.js > Profile > useEffect() callback > then() callback
function Profile() {
const userId = localStorage.getItem('nirapod_identifier'); // Use NID from localStorage
const [user, setUser] = useState(null);
const [form, setForm] = useState({
name: '',
email: '',
phone: '',
presentAddress: '',
permanentAddress: '',
utilityBillCustomerId: '',
passport: '',
drivingLicense: '',
});
const [files, setFiles] = useState({
photo: null,
utilityBillPhoto: null,
passportImg: null,
drivingLicenseImg: null,
});
const [message, setMessage] = useState('');
const [pwForm, setPwForm] = useState({ oldPassword: '', newPassword: '', confirmPassword: '' });
const [pwMsg, setPwMsg] = useState('');
useEffect(() => {
axios.get(`/api/user/${userId}`).then(res => {
setUser(res.data);
setForm({
name: res.data.name || '',
email: res.data.email || '',
phone: res.data.phone || '',
presentAddress: res.data.presentAddress || '',
permanentAddress: res.data.permanentAddress || '',
utilityBillCustomerId: res.data.utilityBillCustomerId || res.data.utilityBillCustomerID || res.data.utilityBillId || '',
passport: res.data.passport || '',
drivingLicense: res.data.drivingLicense || '',
});
});
}, [userId]);
const handleChange = e => {
const { name, value, files: fileInput } = e.target;



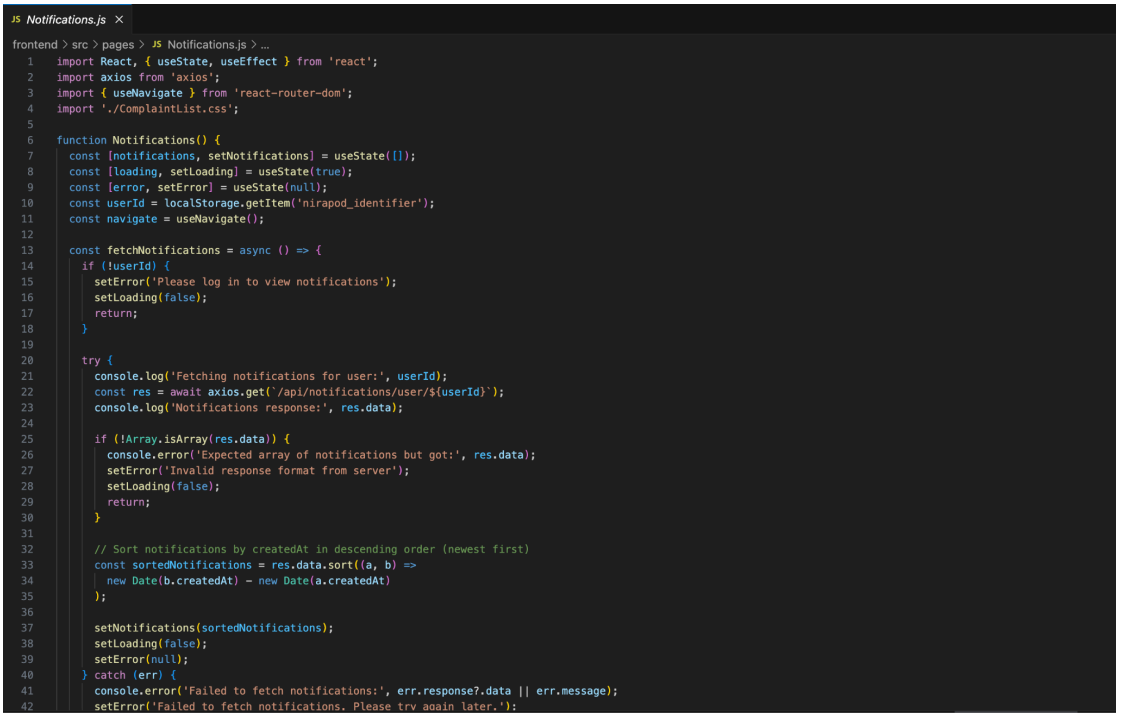
```
JS ComplaintDetails.js   JS Profile.js ×
frontend > src > pages > JS Profile.js > Profile > useEffect() callback > then() callback
4  function Profile() {
42
43   const handleChange = e => {
44     const { name, value, files: fileInput } = e.target;
45     if (fileInput) {
46       setFiles(f => ({ ...f, [name]: fileInput[0] }));
47     } else {
48       setForm(f => ({ ...f, [name]: value }));
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/${userId}/profile`, formData, { headers: { 'Content-Type': 'multipart/form-data' } });
60       setMessage('Profile updated successfully.');
61     } catch {
62       setMessage('Failed to update profile.');
63     }
64   };
65
66   const handlePwChange = e => setPwForm(f => ({ ...f, [e.target.name]: e.target.value }));
67   const handleChangePassword = async e => {
68     e.preventDefault();
69     setPwMsg('');
70     if (pwForm.newPassword !== pwForm.confirmPassword) {
71       setPwMsg('Passwords do not match.');
72       return;
73     }
74     try {
75       await axios.post(`/api/user/${userId}/change-password`, pwForm);
76       setPwMsg('Password changed successfully.');
77       setPwForm({ oldPassword: '', newPassword: '', confirmPassword: '' });
78     } catch {
79       setPwMsg('Failed to change password.');
80     }
81   };
```



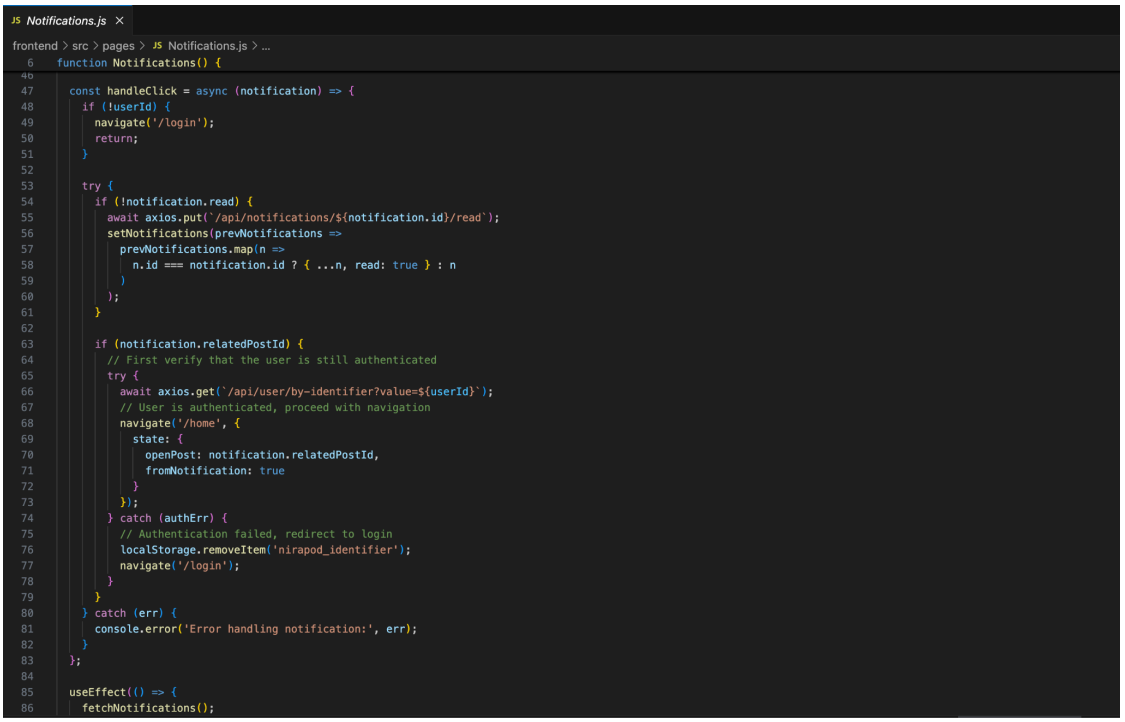
```
JS ComplaintDetails.js   JS Profile.js ×
frontend > src > pages > JS Profile.js > Profile > [∞] handleChangePassword
4  function Profile() {
82
83   if (!user) return <div>Loading...</div>;
84
85   return (
86     <div className="profile-container" style={{ background: '#232831', minHeight: '100vh', display: 'flex', flexDirection: 'column', alignItems: 'center', justifyConten
87       <form onSubmit={handleProfileUpdate} style={{ background: 'none', display: 'flex', flexDirection: 'column', alignItems: 'center', gap: 24, width: 480 }}>
88         <label htmlFor="photo" style={{ display: 'flex', flexDirection: 'column', alignItems: 'center', marginBottom: 12, minHeight: 130 }}>
89           {user.userPhoto && user.userPhoto !== 'null' && user.userPhoto !== '' ? (
90             <img
91               src={user.userPhoto.startsWith('/uploads/') ? `http://localhost:8080/${user.userPhoto.replace('/uploads/', '')}` : `http://localhost:8080/${user.userPhoto
92               alt="Profile"
93               style={{ width: 120, height: 120, borderRadius: '50%', objectFit: 'cover', marginBottom: 12, border: '2px solid #232b36' }}
94             />
95           ) : (
96             <div style={{ width: 120, height: 120, borderRadius: '50%', background: '#fff', display: 'flex', alignItems: 'center', justifyContent: 'center', fontSize: 2
97               No Photo
98             </div>
99           )}
100          <input type="file" name="photo" id="photo" style={{ display: 'none' }} onChange={handleChange} accept="image/*" />
101          <button type="button" onClick={() => document.getElementById('photo').click()} style={{ background: '#232b36', color: '#fff', border: 'none', borderRadius: 8,
102        </label>
103        <div style={profileRowStyle}><span style={profileLabelStyle}>Name :</span><input name="name" placeholder="Name" value={form.name} onChange={handleChange} style=
104        <div style={profileRowStyle}><span style={profileLabelStyle}>Phone :</span><input name="phone" placeholder="Phone" value={form.phone} onChange={handleChange} st
105        <div style={profileRowStyle}><span style={profileLabelStyle}>Email :</span><input name="email" placeholder="Email" value={form.email} onChange={handleChange} st
106        <div style={profileRowStyle}><span style={profileLabelStyle}>NID :</span><input name="nid" placeholder="NID" value={user.nid || ''} readOnly style={{ ...profile
107        <div style={profileRowStyle}><span style={profileLabelStyle}>Present Address :</span><input name="presentAddress" placeholder="Present Address" value={form.pres
108        <div style={profileRowStyle}><span style={profileLabelStyle}>Permanent Address :</span><input name="permanentAddress" placeholder="Permanent Address" value={for
109        <div style={profileRowStyle}><span style={profileLabelStyle}>Utility Bill Customer ID :</span><input name="utilityBillCustomerId" placeholder="Utility Bill Cust
110        <div style={profileRowStyle}><span style={profileLabelStyle}>Utility Bill Photo :</span><input type="file" name="utilityBillPhoto" onChange={handleChange} style
111        <div style={profileRowStyle}><span style={profileLabelStyle}>Passport :</span><input name="passport" placeholder="Passport" value={form.passport} onChange={hand
112        <div style={profileRowStyle}><span style={profileLabelStyle}>Passport Image :</span><input type="file" name="passportImg" onChange={handleChange} style={profile
113        <div style={profileRowStyle}><span style={profileLabelStyle}>Driving License :</span><input name="drivingLicense" placeholder="Driving License" value={form.driv
114        <div style={profileRowStyle}><span style={profileLabelStyle}>Driving License Image :</span><input type="file" name="drivingLicenseImg" onChange={handleChange} s
115        <button type="submit" style={{ background: '#fff', color: '#232b36', border: 'none', borderRadius: 12, padding: '10px 38px', fontWeight: 600, fontSize: 18, marg
116        {message && <div style={{ color: message.includes('success') ? '#22c55e' : '#ef4444', fontWeight: 500 }}>{message}</div>}
117      </form>
118      <form onSubmit={handleChangePassword} style={{ background: 'none', display: 'flex', flexDirection: 'column', alignItems: 'center', gap: 12, width: 420, marginTop:
119        <h3 style={{ color: '#fff', marginBottom: 8 }}>Change Password</h3>
120        <input name="oldPassword" type="password" placeholder="Old Password" value={pwForm.oldPassword} onChange={handlePwChange} style={inputStyle} required />
121        <input name="newPassword" type="password" placeholder="New Password" value={pwForm.newPassword} onChange={handlePwChange} style={inputStyle} required />
122        <input name="confirmPassword" type="password" placeholder="Confirm New Password" value={pwForm.confirmPassword} onChange={handlePwChange} style={inputStyle} req
```

- **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.



```
Notifications.js
frontend > src > pages > Notifications.js > ...
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';
import './ComplaintList.css';

function Notifications() {
  const [notifications, setNotifications] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);
  const userId = localStorage.getItem('nirapod_identifier');
  const navigate = useNavigate();

  const fetchNotifications = async () => {
    if (!userId) {
      setError('Please log in to view notifications');
      setLoading(false);
      return;
    }

    try {
      console.log('Fetching notifications for user:', userId);
      const res = await axios.get(`/api/notifications/user/${userId}`);
      console.log('Notifications response:', res.data);

      if (!Array.isArray(res.data)) {
        console.error('Expected array of notifications but got:', res.data);
        setError('Invalid response format from server');
        setLoading(false);
        return;
      }

      // Sort notifications by createdAt in descending order (newest first)
      const sortedNotifications = res.data.sort((a, b) =>
        new Date(b.createdAt) - new Date(a.createdAt)
      );

      setNotifications(sortedNotifications);
      setLoading(false);
      setError(null);
    } catch (err) {
      console.error('Failed to fetch notifications:', err.response?.data || err.message);
      setError('Failed to fetch notifications. Please try again later.');
```



```
Notifications.js
frontend > src > pages > Notifications.js > ...
function Notifications() {

  const handleClick = async (notification) => {
    if (!userId) {
      navigate('/login');
      return;
    }

    try {
      if (!notification.read) {
        await axios.put(`/api/notifications/${notification.id}/read`);
        setNotifications(prevNotifications =>
          prevNotifications.map(n =>
            n.id === notification.id ? { ...n, read: true } : n
          )
        );
      }

      if (notification.relatedPostId) {
        // First verify that the user is still authenticated
        try {
          await axios.get(`/api/user/by-identifier?value=${userId}`);
          // User is authenticated, proceed with navigation
          navigate('/home', {
            state: {
              openPost: notification.relatedPostId,
              fromNotification: true
            }
          });
        } catch (authErr) {
          // Authentication failed, redirect to login
          localStorage.removeItem('nirapod_identifier');
          navigate('/login');
        }
      }
    } catch (err) {
      console.error('Error handling notification:', err);
    }
  };

  useEffect(() => {
    fetchNotifications();
```

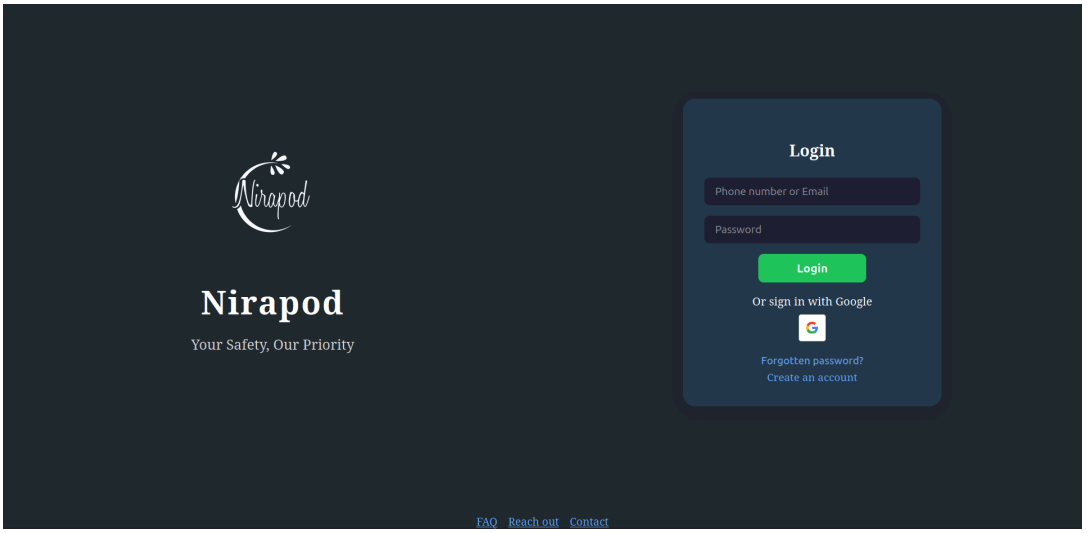
Notifications.js

frontend > src > pages > Notifications.js > Notifications > formatTime

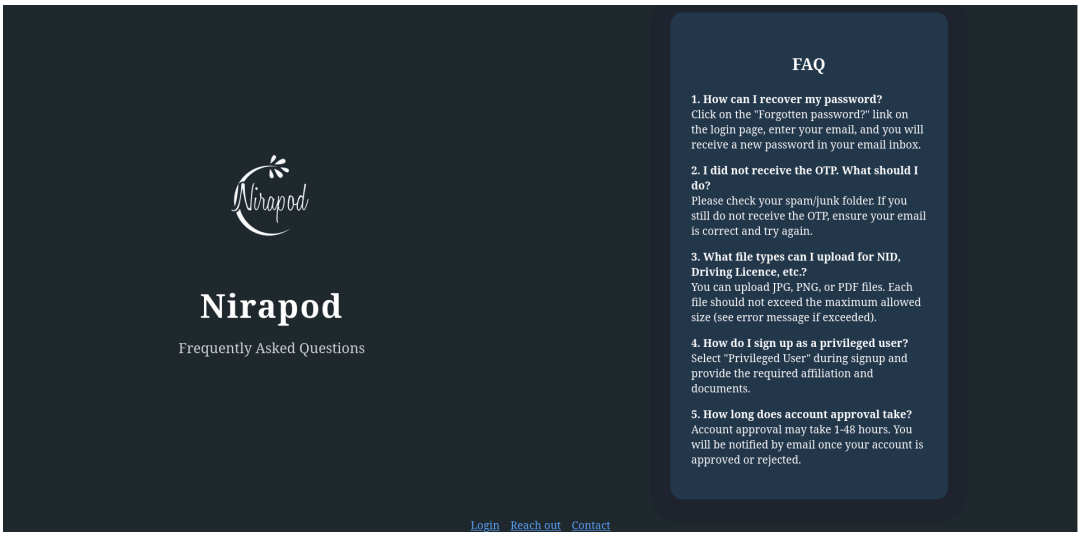
```
function Notifications() {

  const getNotificationStyle = (notification) => {
    let className = 'notification-item';
    if (!notification.read) className += ' notification-unread';
    return className;
  };

  const formatTime = (dateString) => {
    if (!dateString) return '';

    const date = new Date(dateString);
    if (isNaN(date.getTime())) return dateString; // Return original if invalid

    const now = new Date();
    const diff = now - date;

    if (diff < 60000) { // Less than 1 minute
      return 'Just now';
    }

    if (diff < 24 * 60 * 60 * 1000) {
      const hours = Math.floor(diff / (60 * 60 * 1000));
      if (hours < 1) {
        const minutes = Math.floor(diff / (60 * 1000));
        return `${minutes} minute${minutes !== 1 ? 's' : ''} ago`;
      }
      return `${hours} hour${hours !== 1 ? 's' : ''} ago`;
    }

    if (diff < 7 * 24 * 60 * 60 * 1000) {
      const days = Math.floor(diff / (24 * 60 * 60 * 1000));
      return `${days} day${days !== 1 ? 's' : ''} ago`;
    }

    return date.toLocaleDateString();
  };

  if (loading) return (
    <div className="notifications-wrapper">
      <h2 className="page-title">Notifications</h2>
      <div className="loading">Loading notifications...</div>
    </div>
  );

  if (error) return (
    <div className="notifications-wrapper">
      <h2 className="page-title">Notifications</h2>
      <div className="error">{error}</div>
    </div>
  );

  return (
    <div className="notifications-wrapper">
      <h2 className="page-title">Notifications</h2>
      {notifications.length === 0 ? (
        <div className="empty-state">
          No notifications yet
        </div>
      ) : (
        <div className="notifications-list">
          {notifications.map(notification => (
            <div
              key={notification.id}
              className={getNotificationStyle(notification)}
              onClick={() => handleClick(notification)}
            >
              <div className="notification-content">
                <div className="notification-message">{notification.message}</div>
                <div className="notification-time">{formatTime(notification.createdAt)}</div>
              </div>
            </div>
          ))}
        </div>
      )}
    </div>
  );
}

export default Notifications;
```

- # 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.

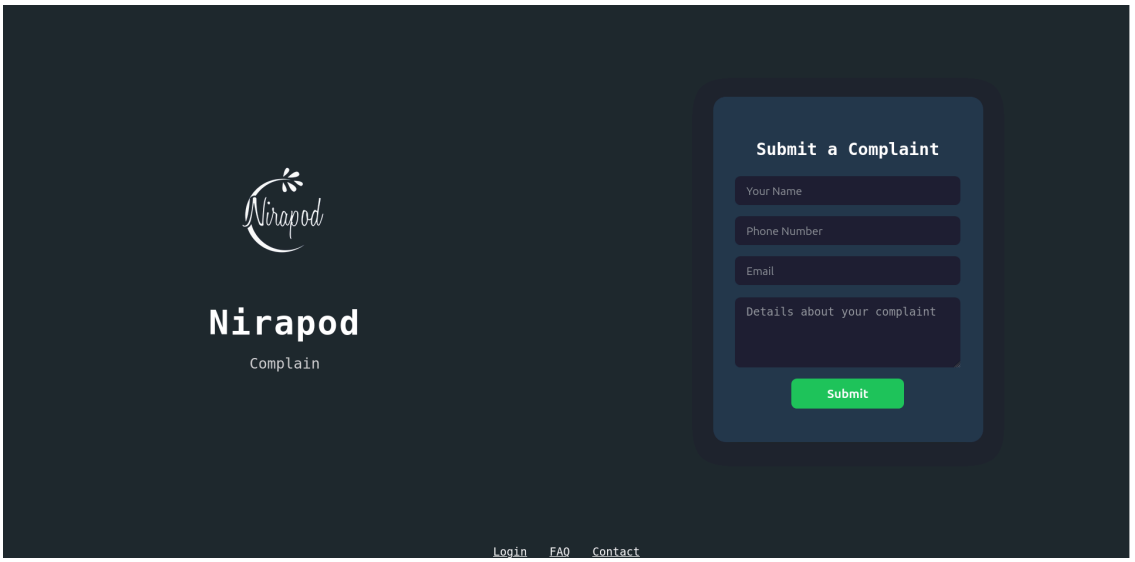


2. **FAQ page** - If a user encounters difficulties or feels lost while navigating the website, they can visit the FAQ (Frequently Asked Questions) page. The FAQ page provides answers to common questions and helps users resolve issues quickly without needing additional support.

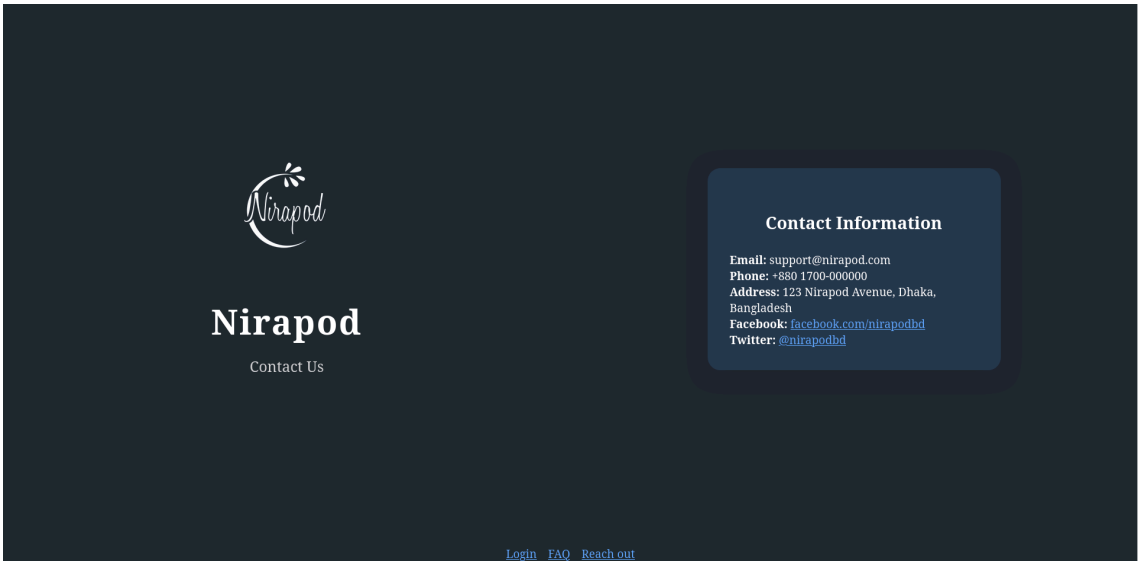


3. **ReachOut** -Users can submit complaints to the administrator regarding issues they encounter, such as login problems or other technical difficulties. When a complaint is submitted, the administrator receives an email notification containing the details of the issue. The administrator can then contact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

resolving the problem.

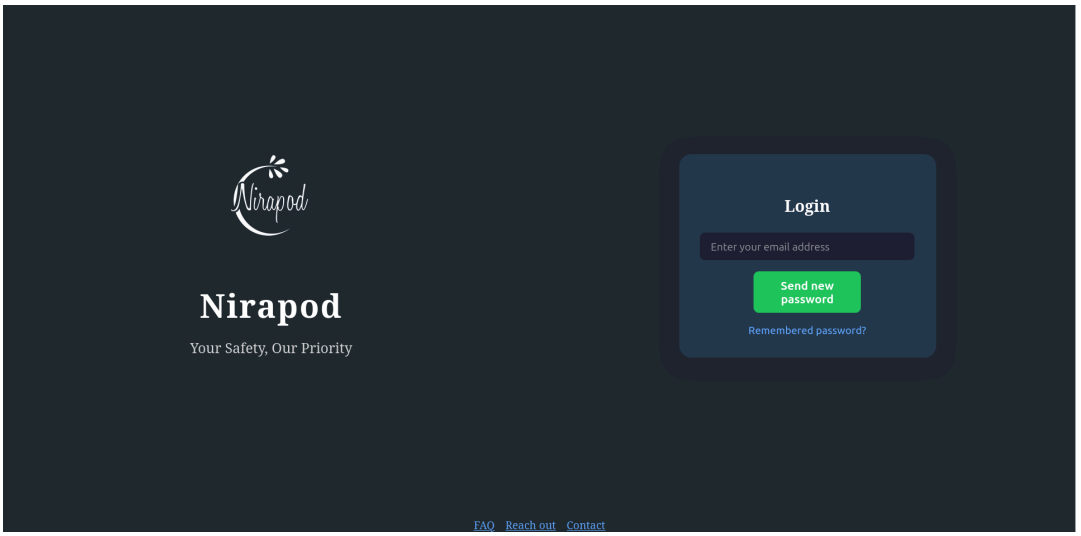


4. **Contract** - This page contains the official contact details of the organization, allowing users to reach out for support, inquiries, or further communication.

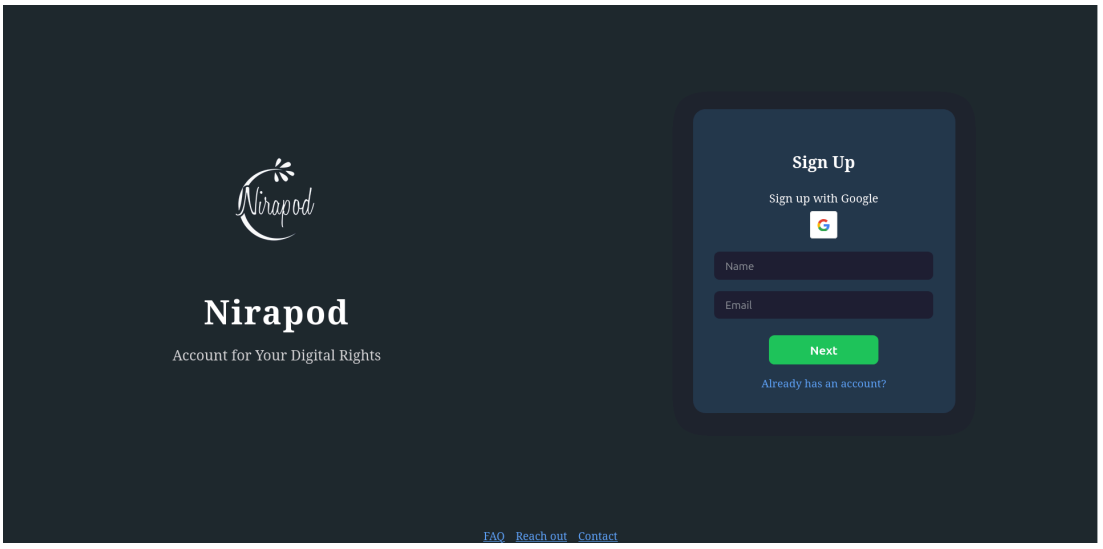


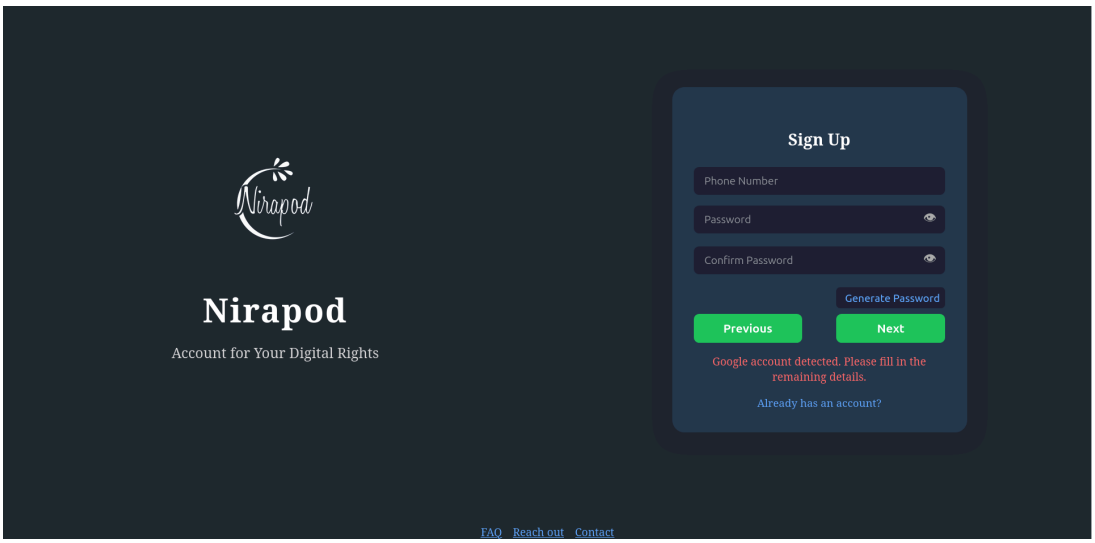
5. **Forget Password** - Users have the ability to reset their account password if needed. Upon requesting a password reset, the system will generate a new password and send it instantly to the user's registered email address. Users can

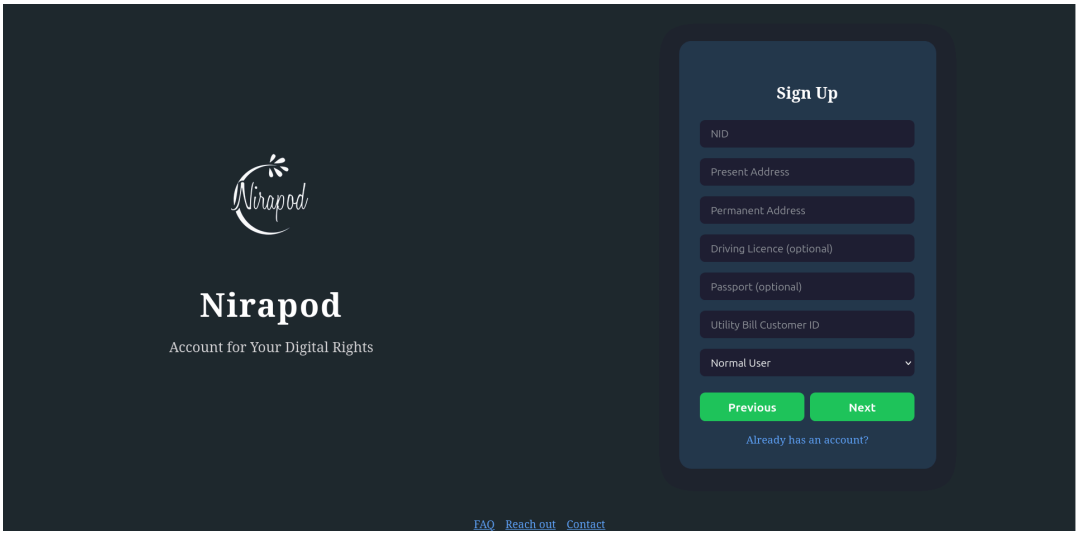
then use the new password to log in and update it after logging in.

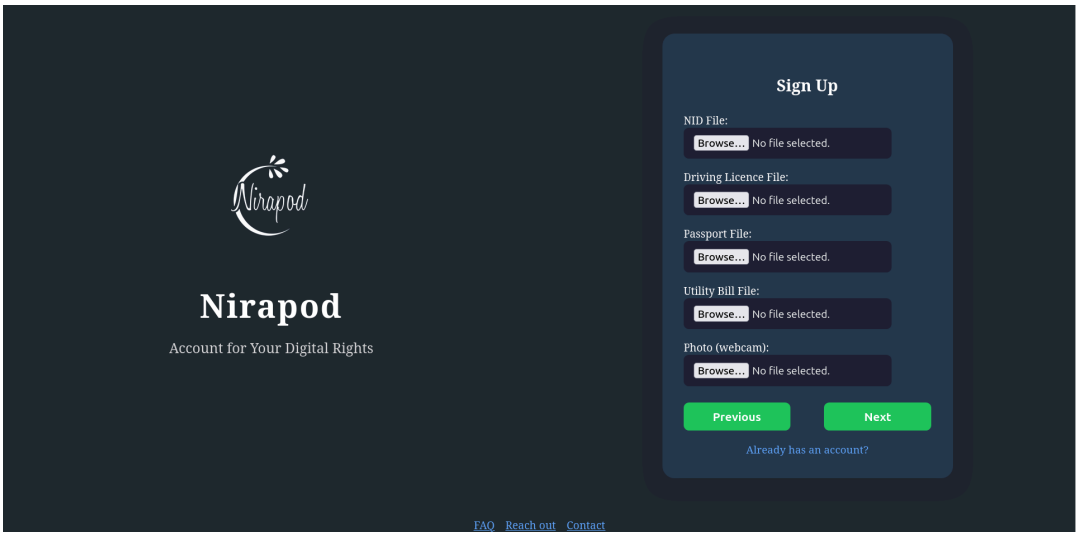


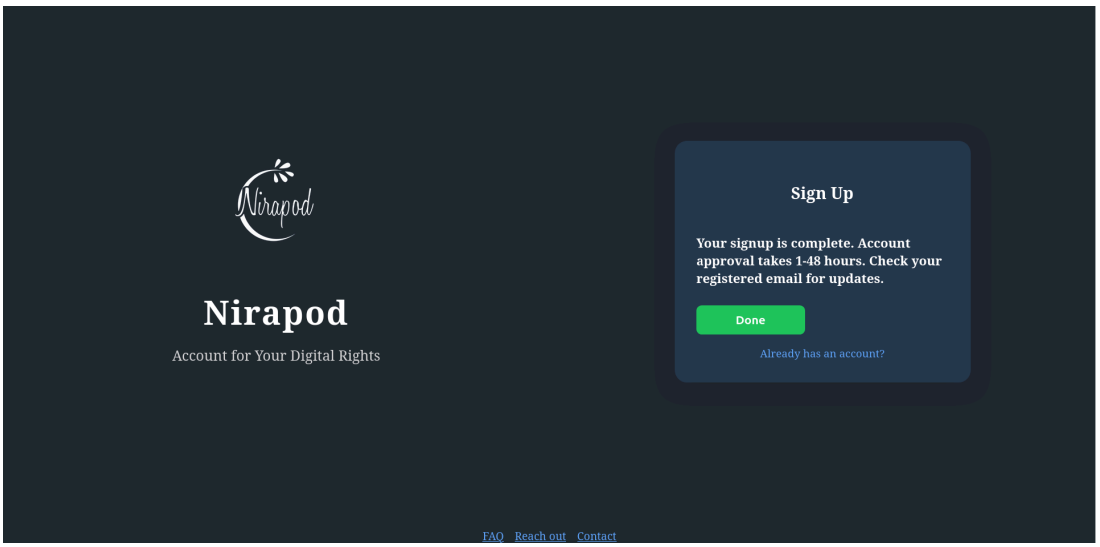
6. **Signup** - If a user does not have an account on the platform, they can easily create one through the signup process. During registration, an OTP (One-Time Password) will be sent to the user's provided email address for verification. Therefore, providing a valid and active email is mandatory. To 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").



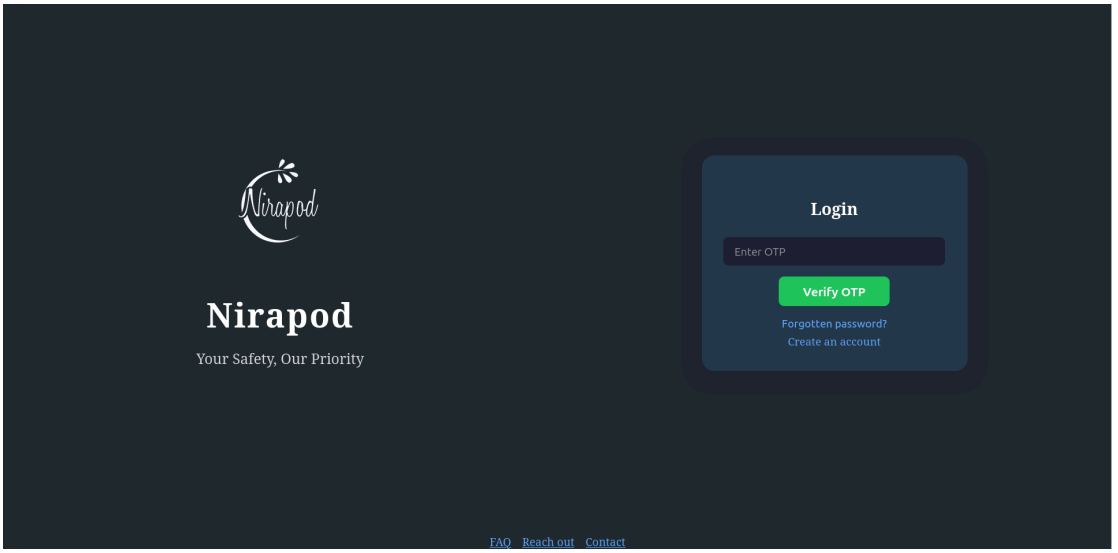

Nirapod
Account for Your Digital Rights
Sign Up
Phone Number
Password
Confirm Password
Generate Password
Previous
Next
Google account detected. Please fill in the remaining details.
Already has an account?
FAQ
Reach out
Contact


Nirapod
Account for Your Digital Rights
Sign Up
NID
Present Address
Permanent Address
Driving Licence (optional)
Passport (optional)
Utility Bill Customer ID
Normal User
Previous
Next
Already has an account?
FAQ
Reach out
Contact

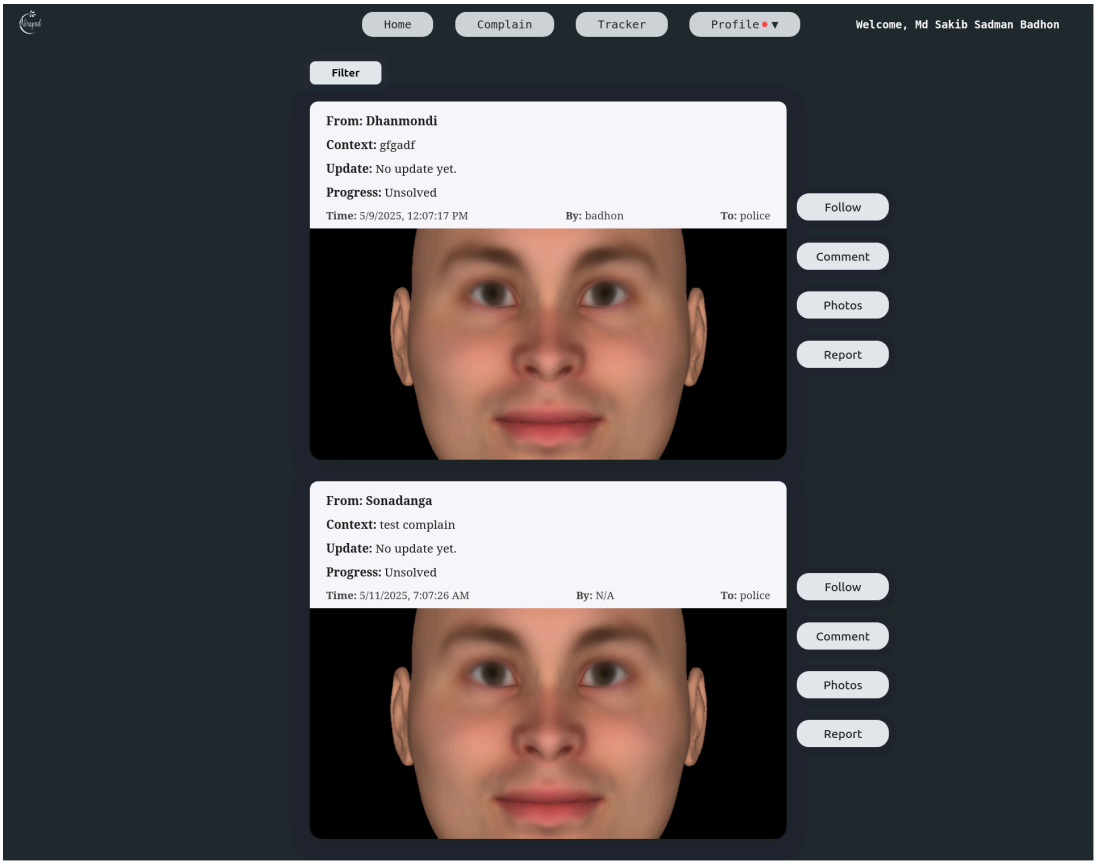

Nirapod
Account for Your Digital Rights
Sign Up
NID File:
Browse... No file selected.
Driving Licence File:
Browse... No file selected.
Passport File:
Browse... No file selected.
Utility Bill File:
Browse... No file selected.
Photo (webcam):
Browse... No file selected.
Previous
Next
Already has an account?
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.

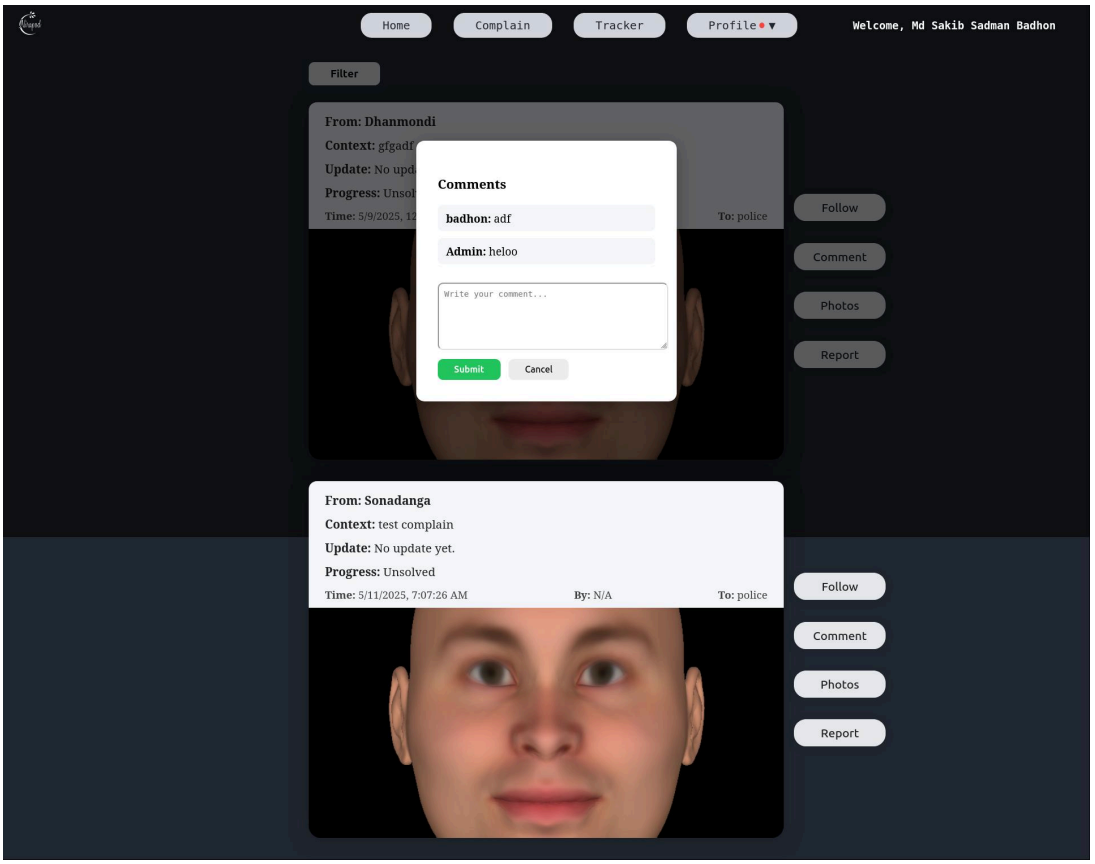


8. **Homepage** - After successfully logging in, users are redirected to the homepage. On the homepage, users can view all complaints — both those they have personally submitted and complaints made by other users, depending on their access permissions.

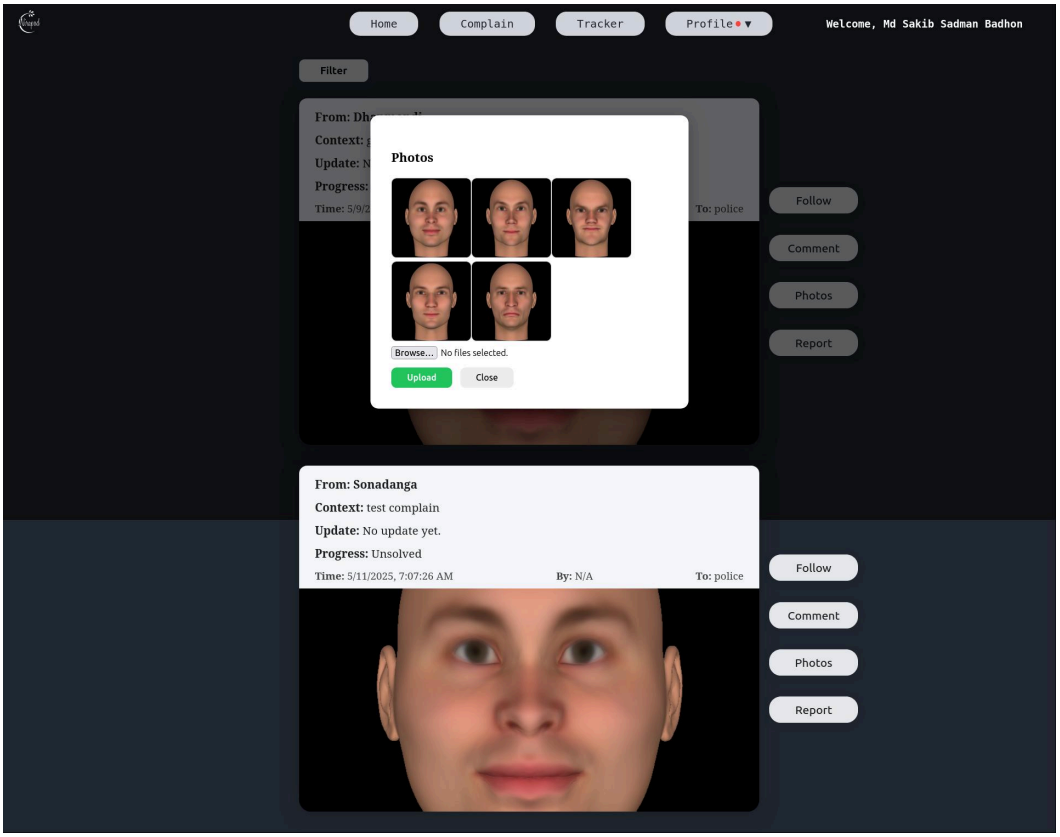


From the homepage, users also have the option to follow complaints. By following a post, users will receive notifications whenever a privileged user provides an update or when someone comments on that complaint. This

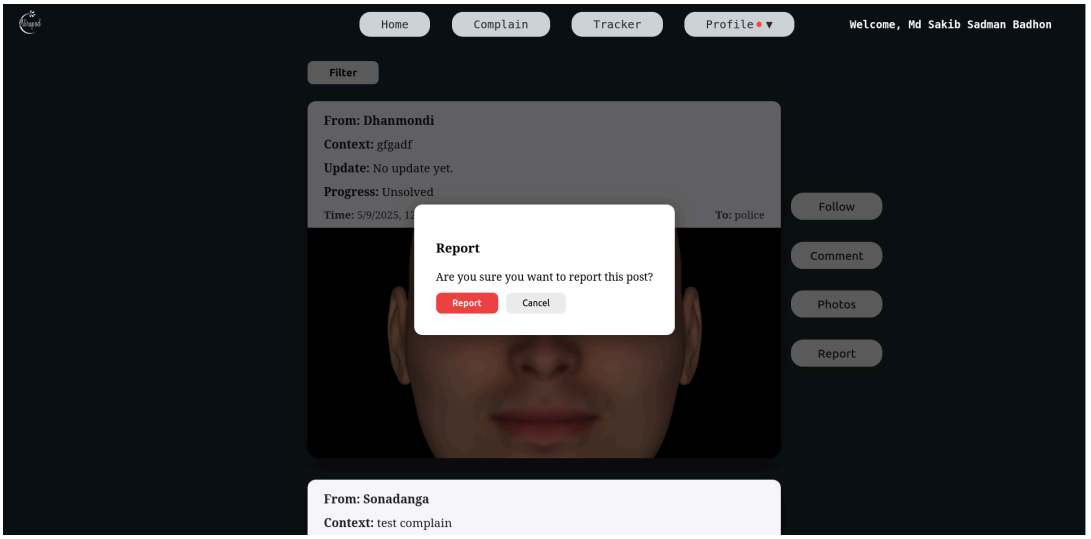
feature helps users stay informed about issues they are interested in.



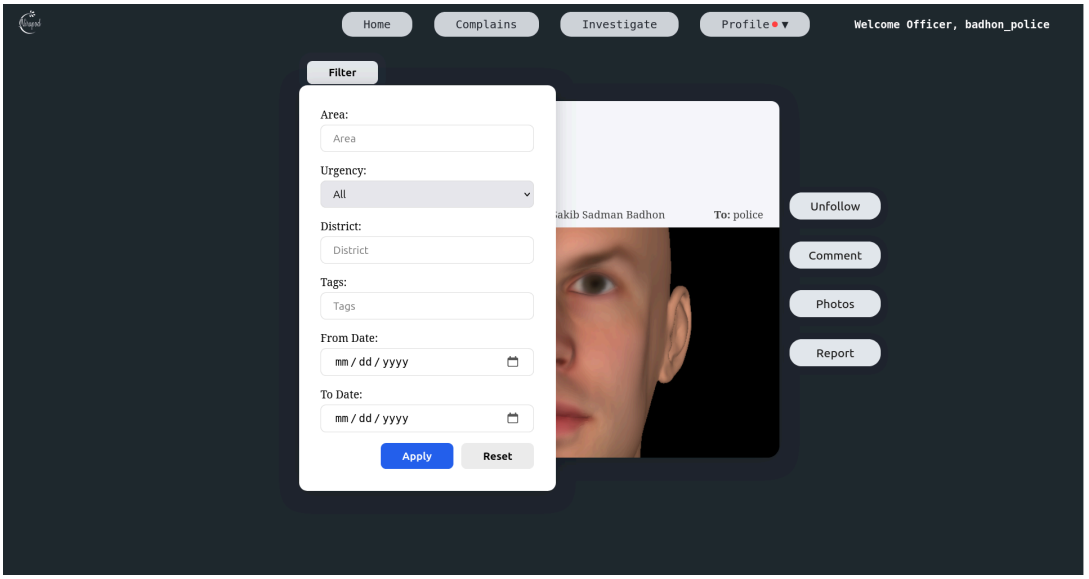
In the comment section of each complaint, users can view all comments made by other users. Users also have the ability to add their own comments to any post, allowing for discussion, feedback, or additional information sharing related to the complaint.



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.

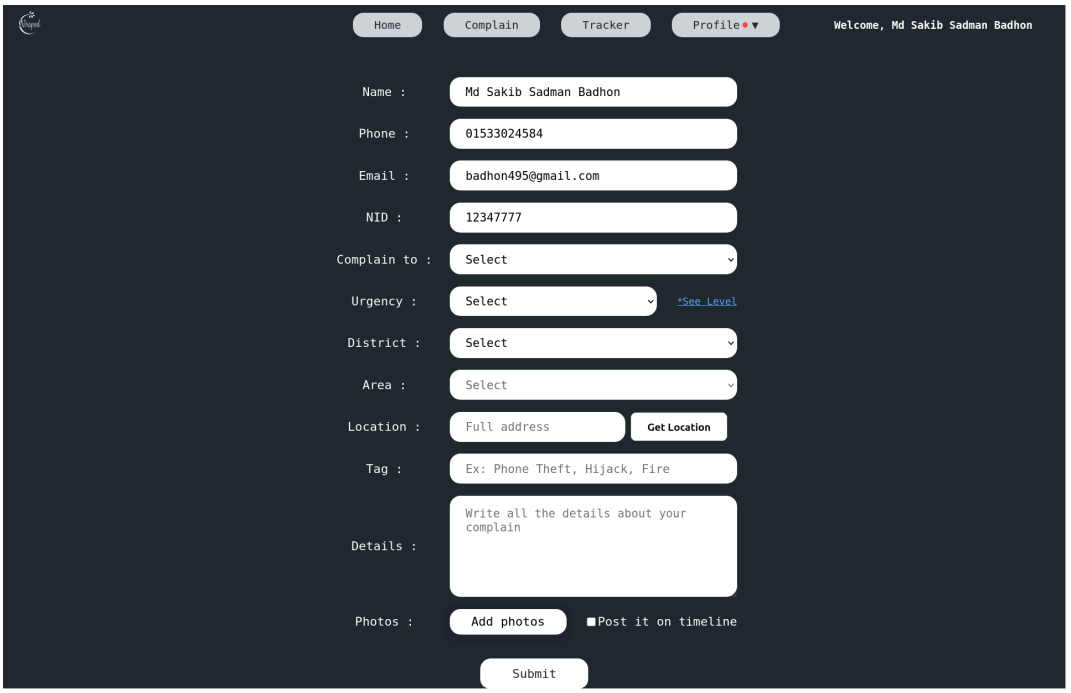


Users can report any post they find inappropriate or in violation of community guidelines by pressing the Report button. Once a report is submitted, the administrator gains access to all reports. The admin can then review the reports and take appropriate actions, such as deleting the report or removing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

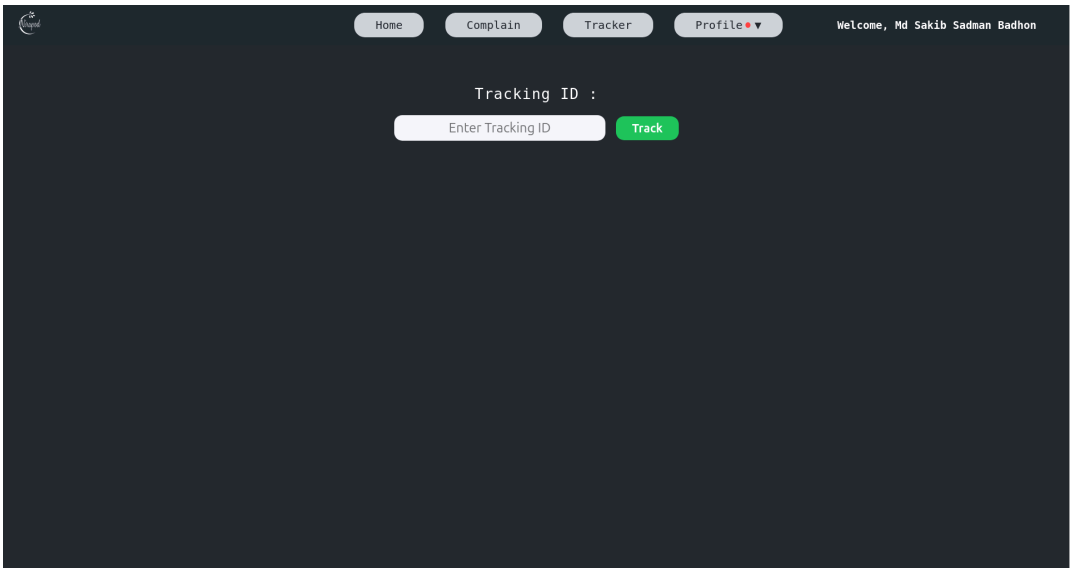


Users can use the Filter option to refine and sort posts based on various criteria. This allows users to view complaints that meet specific conditions, such as date, category, or status, making it easier to find relevant posts.

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.

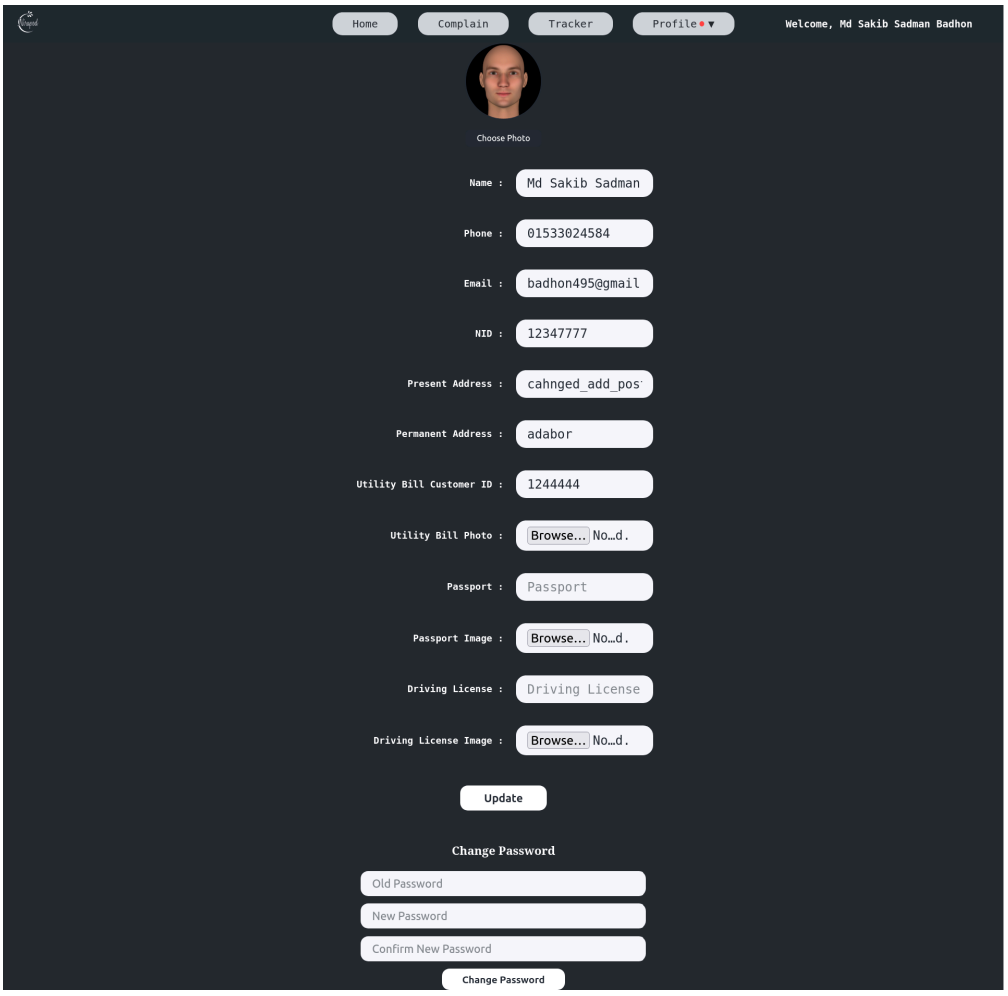


10. **Tracker -** After submitting a complaint, the user will receive a tracking number for their complaint. Using this tracking number, the user can check the status and receive updates regarding their complaint's progress at any time.

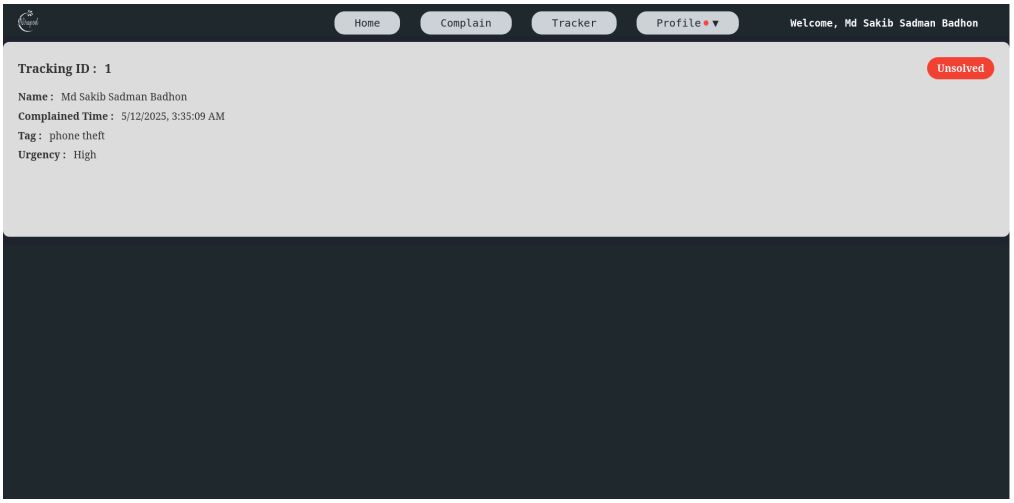


11. **Update profile** - The Update Profile option is available in the dropdown menu of the user's profile. Using this feature, users can update their profile information, such as their name, email, and address. However, certain details, such as the National ID (NID), cannot be updated by the user for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

verification purposes.



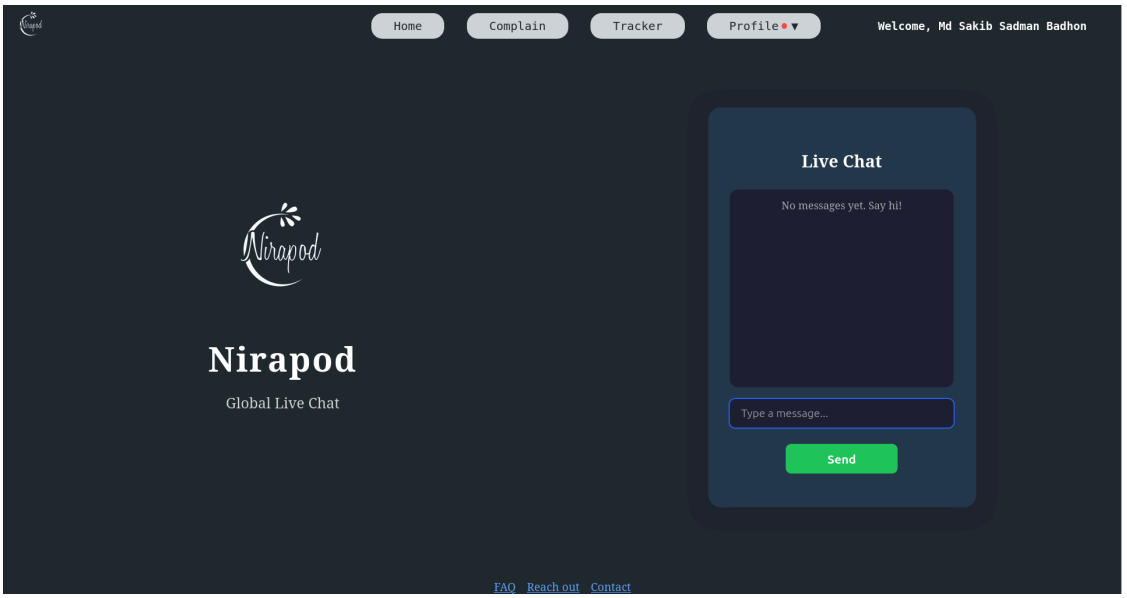
12. **Your Complaints** - In this section, users can view all the complaints they have made. This allows them to easily track and manage their submitted complaints.



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

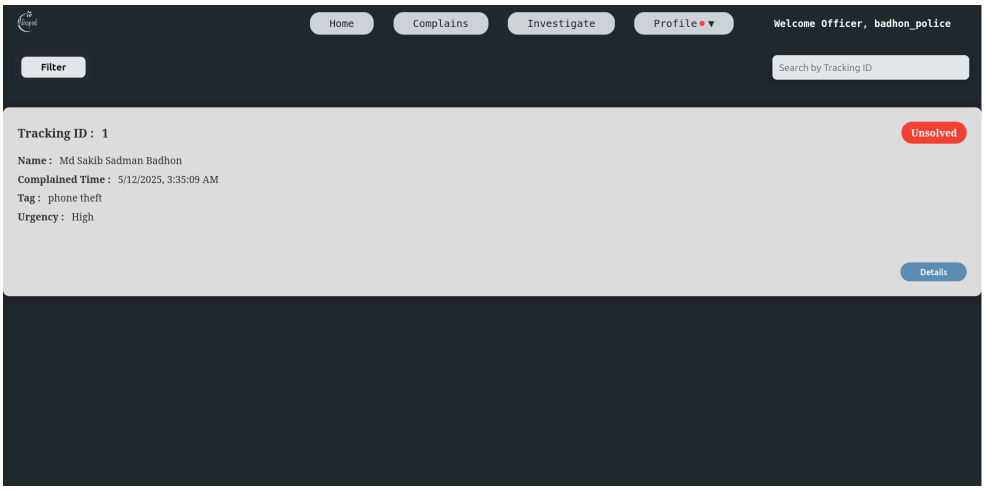
be notified of any new changes or activities related to the posts they are following.

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.

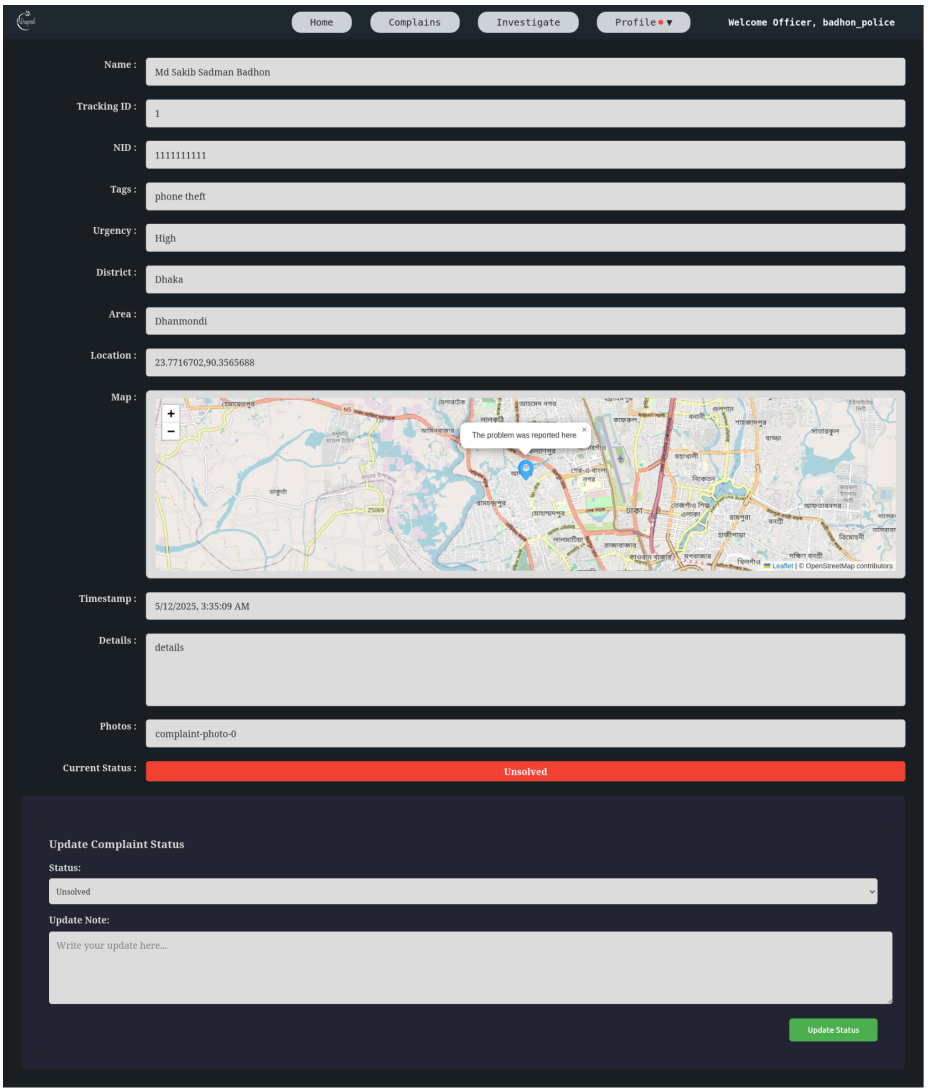


15. **Privileged user**
    a. **Login and signup** - Privileged users will follow the same login process as mentioned earlier. During the signup process, privileged users must select their user type as Privileged User and provide their specific verification ID (such as Police ID, etc.) to confirm their 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 compalit by clicking the Details button, not only that, by filter and search option, he can easily filter the complains by searching the tracking id or filtering based on the level and other

things.

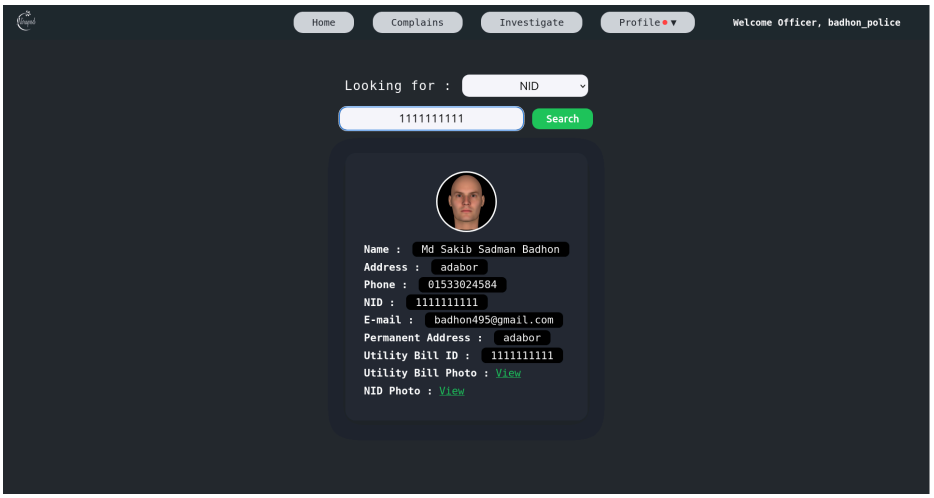


d. **Complain Details** - From here a privileged user can view the details of complaints directed to them. In addition to viewing, they have the ability to update the complaint status and provide information regarding the progress of resolving the issue. This feature enables privileged users to keep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

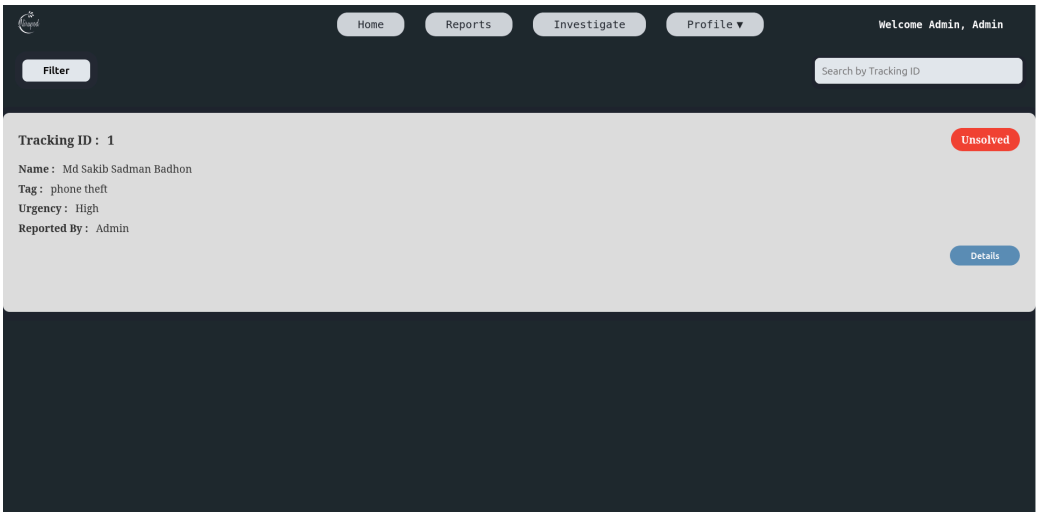


e. **Investigation** - The Investigation feature is exclusive to police officers, as they may require access to user details for various investigative purposes. Through this feature, police officers can search for a user and retrieve their details using identifiers such as National ID (NID), Driving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

and securely.

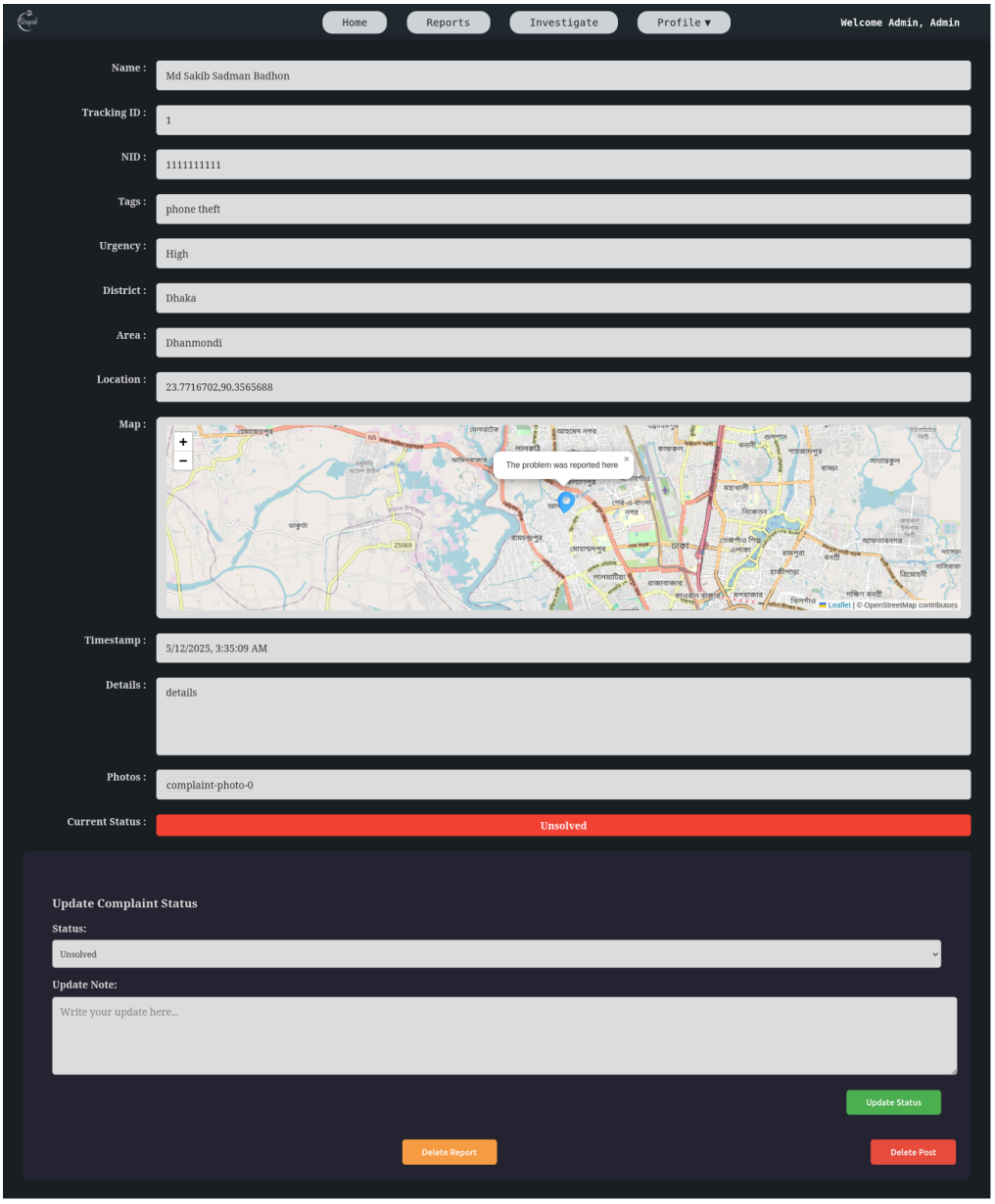


16. **Admin** - The Admin is the primary controller of the website. Admins have full access to all complaints, including the ability to delete any complaint submitted by users. Additionally, admins can view all reports made by users and take appropriate actions based on the nature of the report. This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



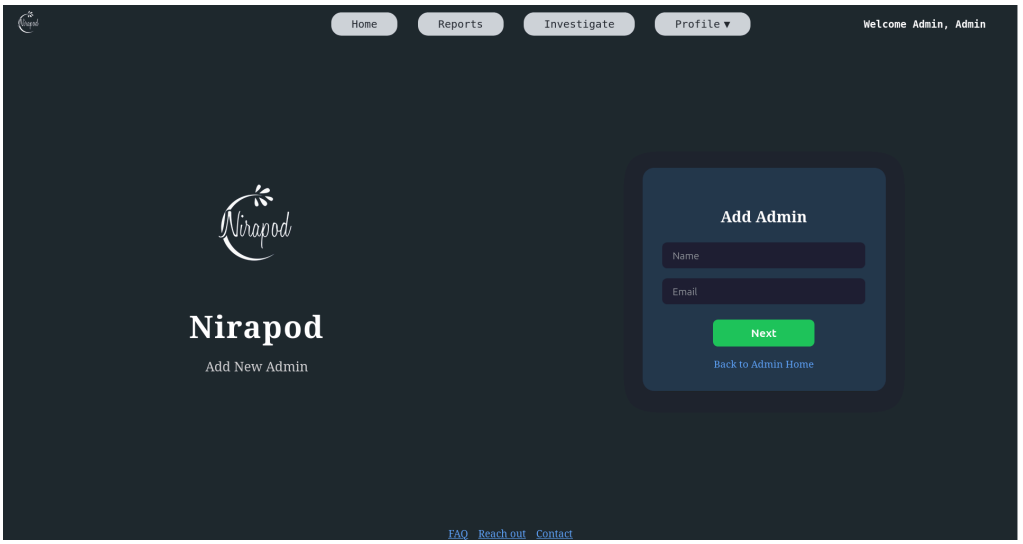
    b. **Report Details** -The Report Details page is similar to the complaint details page, but with two additional features specifically for admins.
        i. Delete Post: This feature allows the admin to delete a user's post if it violates community guidelines or is deemed inappropriate.
        ii.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



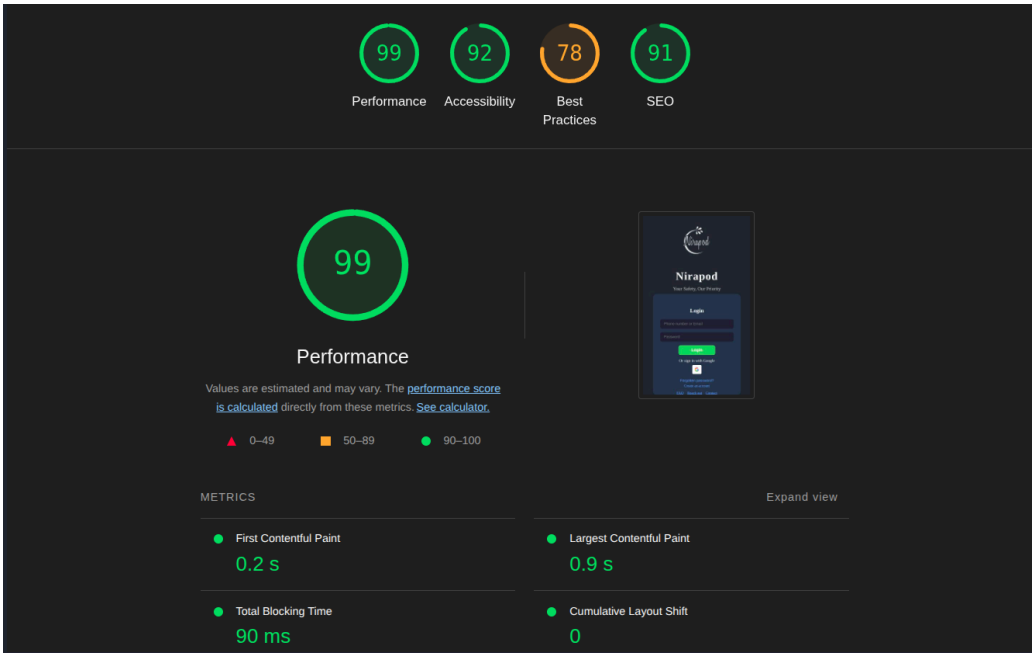
c. **Add Admin** - The System Admin has the ability to appoint a new admin. To add a new admin, the system admin must provide the Name and Email of the individual being appointed. This process is similar to the signup process for normal or privileged users, with the key difference being the role assigned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

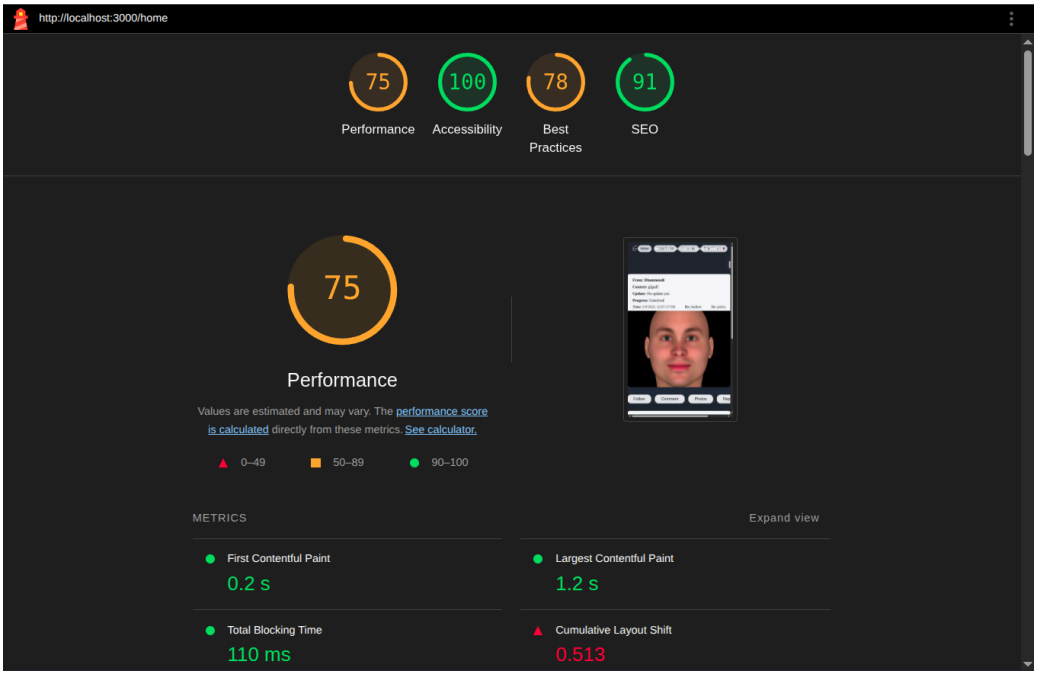
based on their role.



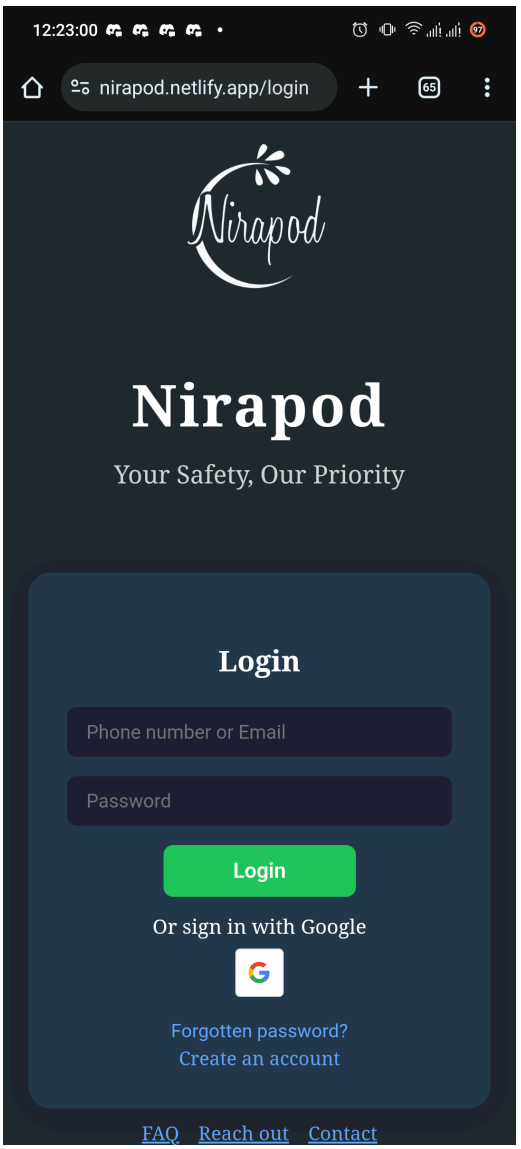
- **Performance and Network Analysis**

1. Lightouse 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. Spring Boot Tutorial for Beginners [2025]
6. PostgreSQL Tutorial for Beginners
7. React Tutorial for Beginners