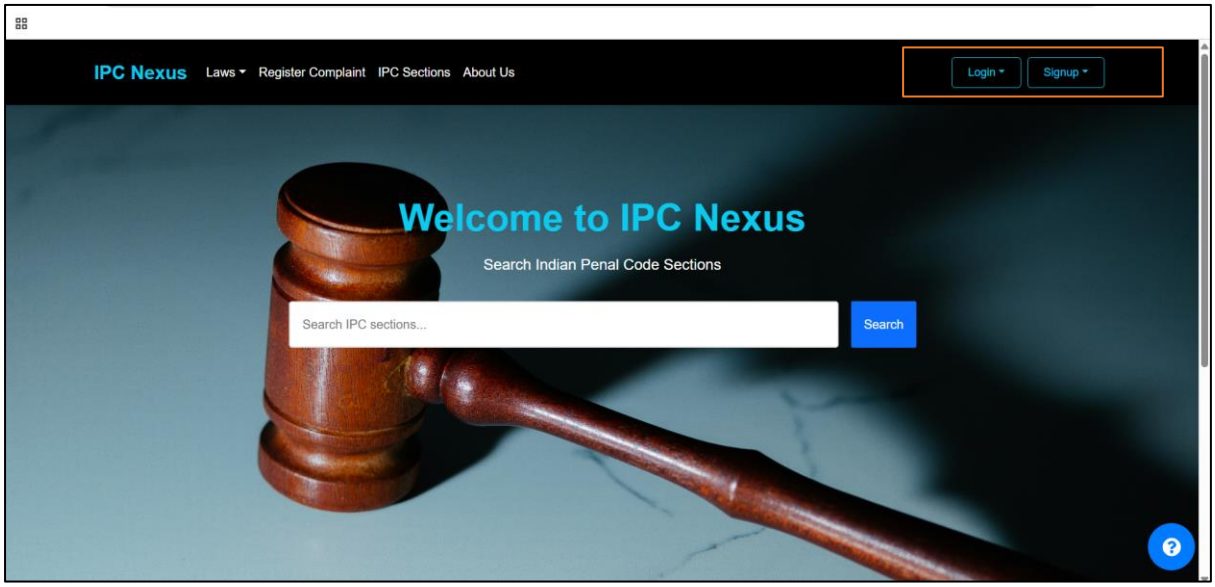
FRONT-END Login and Signup pages:

We added the functionality of three kinds of login and signup, depending on the type of user who wants to log in/sign up.

That is so that to generate token and permission for the user depending on their role, which is further discussed in the backend code.

So these are the interface looking on our website:

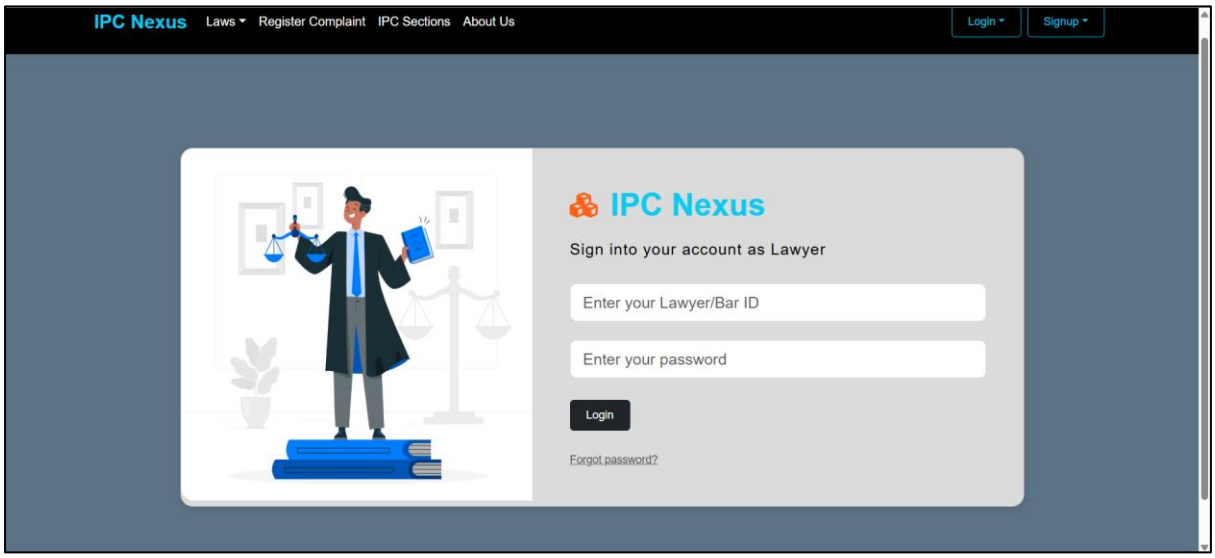


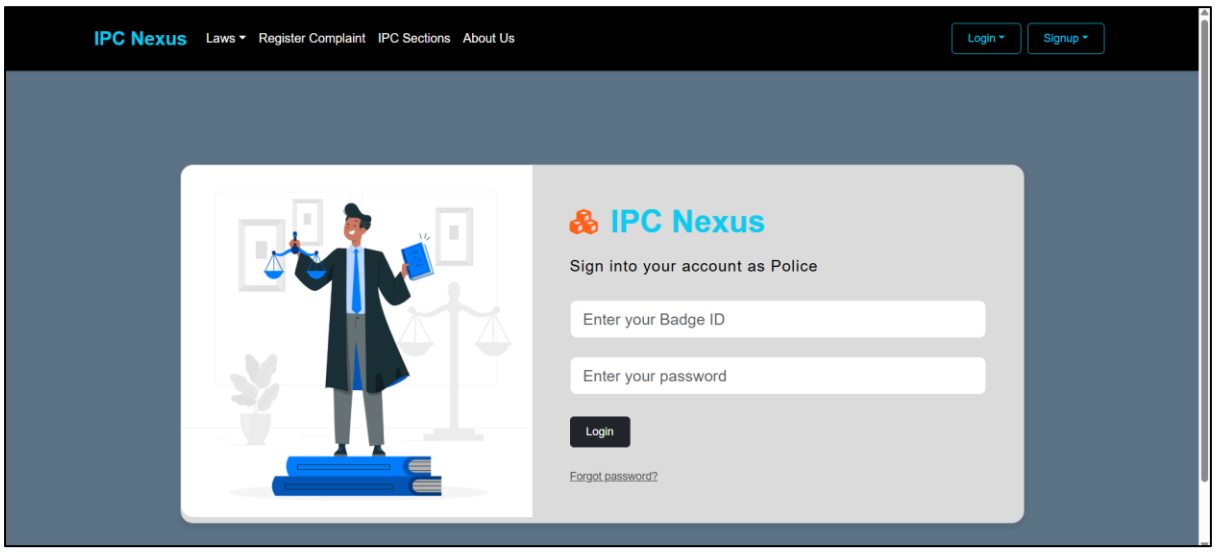
And when user clicks on either of those then it shows the following list of choices:

And clicking on each of these button would guide to different login /signup page according to user authority or role.
(so that only a designated person can make a change in the interface and only the designated person gets the authority to register complaints and assign sections).

And when we click on either of this we get the following setup pages:

And similarly when the user wants to login:





FRONT-END CODE:

LOGIN_form.jsx

```
import React, { useState } from 'react';
import axios from 'axios';
import auth_bg from "../../assets/auth_bg.jpg";

const LoginForm = ({ userType }) => {
  const [formData, setFormData] = useState({
    idOrUsername: '', // Username/Lawyer ID/Badge ID
    password: '',
  });

  const [message, setMessage] = useState('');

  const handleChange = (e) => {
    const { name, value } = e.target;
    setFormData({ ...formData, [name]: value });
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const payload = { ...formData, userType };
      const response = await axios.post('/api/login', payload); // Backend API
      setMessage(response.data.message || 'Login successful!');
    } catch (error) {
      setMessage(error.response?.data?.message || 'Login failed.');
    }
  };

  return (
    <section className="vh-100" style={{ backgroundColor: '#5C7285' }}>
      <div className="container py-5 h-100">
        <div className="row d-flex justify-content-center align-items-center
h-100">
          <div className="col col-xl-10">
          <div className="card" style={{ borderRadius: '1rem',
backgroundColor: '#DBDBDB' }}>

              <div className="row g-0">
                <div className="col-md-6 col-lg-5 d-none d-md-block">
                  <img
                    src={auth_bg}
                    alt="login form"
                    className="img-fluid"
                    style={{ borderRadius: '1rem 0 0 1rem' }}
                  />
```

```
                </div>
                <div className="col-md-6 col-lg-7 d-flex align-items-center">
                  <div className="card-body p-4 p-lg-5 text-black">
                    <form onSubmit={handleSubmit}>
                      <div className="d-flex align-items-center mb-3 pb-1">
                        <i className="fas fa-cubes fa-2x me-3" style={{ color:
'#ff6219' }}></i>
                        <span className="h1 fw-bold mb-0 text-info">IPC
Nexus</span>
                      </div>
                      <h5 className="fw-normal mb-3 pb-3" style={{
letterSpacing: '1px' }}>
                        Sign into your account as {userType}
                      </h5>
                      <div className="form-outline mb-4">
                        <input
                          type="text"
                          id="form2Example17"
                          className="form-control form-control-lg"
                          name="idOrUsername"
                          value={formData.idOrUsername}
                          onChange={handleChange}
                          placeholder={`Enter your ${
                            userType === 'Civilian'
                              ? 'Username'
                              : userType === 'Lawyer'
                              ? 'Lawyer/Bar ID'
                              : 'Badge ID'
                          }`}
                        />

                      </div>
                      <div className="form-outline mb-4">
                        <input
                          type="password"
                          id="form2Example27"
                          className="form-control form-control-lg"
                          name="password"
                          value={formData.password}
                          onChange={handleChange}
                          placeholder="Enter your password"
                        />

                      </div>
                      <div className="pt-1 mb-4">
                <button className="btn btn-dark btn-lg btn-block" type="submit">
                          Login
                        </button>
```

```
                    </div>
                    <a className="small text-muted" href="#!">
                      Forgot password?
                    </a>
      {message && <p className="mt-3 text-center text-danger">{message}</p>}
                  </form>
                </div>
              </div>
            </div>
          </div>
        </div>
      </section>
  );
};

export default LoginForm;
```

SignupForm.jsx:

```
import React, { useState } from 'react';
import axios from 'axios';
import auth_bg from "../../assets/auth_bg.jpg";

const SignupForm = ({ userType }) => {
    const [formData, setFormData] = useState({
        username: '', // For Civilian
        email: '',
        phoneno: '',
        password: '',
        confirmPassword: '',
        id: '', // Lawyer ID / Badge ID for Lawyer and Police
    });

    const [message, setMessage] = useState('');

    const handleChange = (e) => {
        const { name, value } = e.target;
        setFormData({ ...formData, [name]: value });
    };

    const handleSubmit = async (e) => {
        e.preventDefault();
        if (formData.password !== formData.confirmPassword) {
            setMessage("Passwords don't match!");
            return;
```

```
        }

        try {
            const payload = { ...formData, userType };
            const response = await axios.post('/api/signup', payload); //
Backend API endpoint
            setMessage(response.data.message || 'Signup successful!');
        } catch (error) {
            setMessage(error.response?.data?.message || 'Signup failed.');
        }
    };

    return (
        <section className="vh-100" style={{ backgroundColor: '#5C7285' }}>
            <div className="container py-5 h-100 ">
   <div className="row d-flex justify-content-center align-items-center h-
100">
             <div className="col col-xl-10">
                  className="card" style={{ borderRadius: '1rem',
backgroundColor: '#DBDBDB' }}>

                    <div className="row g-0">
                       <div className="col-md-6 col-lg-5 d-none d-md-block">
                                  <img
                                      src={auth_bg}
                                      alt="signup form"
                                      className="img-fluid"
                                      style={{ borderRadius: '1rem 0 0
1rem', height: '100%', width: '100%', objectFit: 'cover' }}
                                  />

                               </div>
          <div className="col-md-6 col-lg-7 d-flex align-items-center">
                 <div className="card-body p-4 p-lg-5 text-black">
                    <form onSubmit={handleSubmit}>
                       <div className="d-flex align-items-center mb-3 pb-1">
     <i className="fas fa-cubes fa-2x me-3" style={{ color: '#ff6219' }}></i>
                <span className="h1 fw-bold mb-0 text-info">IPC Nexus</span>
                                          </div>
        <h5 className="fw-normal mb-3 pb-3" style={{ letterSpacing: '1px' }}>
                                              Signup as {userType}
                                          </h5>

                    {/* Conditionally Render Fields */}
                          {userType === 'Civilian' && (
                                <div className="form-outline mb-4">
                                    <input
                                       type="text"
```

```
                                    id="username"
                                    className="form-control form-control-lg"
                                    name="username"
                                    value={formData.username}
                                    onChange={handleChange}
                                    placeholder="Enter your username"
                                                />

                                            </div>
                                        )}
                    {(userType === 'Lawyer' || userType === 'Police') && (
                            <div className="form-outline mb-4">
                                <input
                                  type="text"
                                  id="id"
                                  className="form-control form-control-lg"
                                  name="id"
                                  value={formData.id}
                                  onChange={handleChange}
placeholder={`Enter your ${userType === 'Lawyer' ? 'Lawyer/Bar ID' : 'Badge
ID'}`}
                                                />

                                            </div>
                                        )}
                        <div className="form-outline mb-4">
                                <input
                                 type="email"
                                  id="email"
                                 className="form-control form-control-lg"
                                 name="email"
                                 value={formData.email}
                                 onChange={handleChange}
                                 placeholder="Enter your email"
                                                />

                          </div>
                         <div className="form-outline mb-4">
                          <input
                          type="text"
                          id="phoneno"
                          className="form-control form-control-lg"
                          name="phoneno"
                          value={formData.phoneno}
                          onChange={handleChange}
                          placeholder="Enter your phone number"
                                                />
                          </div>
```

```
                        <div className="form-outline mb-4">
                          <input
                            type="password"
                            id="password"
                            className="form-control form-control-lg"
                            name="password"
                            value={formData.password}
                            onChange={handleChange}
                            placeholder="Enter your password"
                            />
                          </div>
                        <div className="form-outline mb-4">
                            <input
                               type="password"
                               id="confirmPassword"
                               className="form-control form-control-lg"
                               name="confirmPassword"
                               value={formData.confirmPassword}
                               onChange={handleChange}
                               placeholder="Confirm your password"
                               />
                          </div>
                          <div className="pt-1 mb-4">
            <button className="btn btn-dark btn-lg btn-block" type="submit">
                                    Signup
                             </button>
                          </div>

   {message && <p className="mt-3 text-center text-danger">{message}</p>}
                                      </form>
                                   </div>
                                </div>
                             </div>
                          </div>
                       </div>
                    </div>
                 </section>
    );
};

export default SignupForm;
```

DATA-BASE CODE STRUCTURE IN BACKEND:

```
from flask import Flask, request, jsonify
from flask_cors import CORS
from flask_sqlalchemy import SQLAlchemy
from flask_migrate import Migrate
from datetime import datetime
import bcrypt
import jwt
import os
from dotenv import load_dotenv
import traceback
from werkzeug.security import generate_password_hash, check_password_hash

# Load environment variables
load_dotenv()

app = Flask(__name__)
CORS(app)

# Database configuration
app.config['SQLALCHEMY_DATABASE_URI'] = os.getenv('DATABASE_URL',
'postgresql://localhost/ipc_nexus')
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
app.config['SECRET_KEY'] = os.getenv('JWT_SECRET_KEY', 'your-secret-key')

# Initialize SQLAlchemy and Migrate
db = SQLAlchemy(app)
migrate = Migrate(app, db)

# Models
class Account(db.Model):
    __tablename__ = 'accounts'
    account_id = db.Column(db.String(50), primary_key=True, unique=True,
nullable=False)
    password = db.Column(db.String(255), nullable=False)
    role = db.Column(db.String(20), nullable=False)  # 'police', 'lawyer',
'civilian'
    email = db.Column(db.String(120), unique=True, nullable=False)
    verified = db.Column(db.Boolean, default=False)
    created_at = db.Column(db.DateTime, default=datetime.utcnow)

class Civilian(db.Model):
    __tablename__ = 'civilians'
    account_id = db.Column(db.String(50),
db.ForeignKey('accounts.account_id'), primary_key=True)
    username = db.Column(db.String(50), unique=True, nullable=False)
    name = db.Column(db.String(100), nullable=False)
```

```
class Police(db.Model):
    __tablename__ = 'police'
    account_id = db.Column(db.String(50),
db.ForeignKey('accounts.account_id'), primary_key=True)
    police_name = db.Column(db.String(100), nullable=False)
    badge_number = db.Column(db.String(50), unique=True, nullable=False)
    department = db.Column(db.String(100))
    rank = db.Column(db.String(50))
    supervisor_id = db.Column(db.String(50),
db.ForeignKey('accounts.account_id'), nullable=True)

class Lawyer(db.Model):
    __tablename__ = 'lawyers'
    account_id = db.Column(db.String(50),
db.ForeignKey('accounts.account_id'), primary_key=True)
    lawyer_name = db.Column(db.String(100), nullable=False)
    bar_id = db.Column(db.String(50), unique=True, nullable=False)
    specialization = db.Column(db.String(100))
    experience_years = db.Column(db.Integer)

class Incident(db.Model):
    __tablename__ = 'incidents'
    incident_id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    description = db.Column(db.Text, nullable=False)
    incident_type = db.Column(db.String(50), nullable=False)
    title = db.Column(db.String(200), nullable=False)
    location = db.Column(db.String(200))
    incident_date = db.Column(db.DateTime, nullable=False)
    status = db.Column(db.String(50), default='Pending')
    created_at = db.Column(db.DateTime, default=datetime.utcnow)
    priority = db.Column(db.String(20), default='Medium')
    category = db.Column(db.String(100))
    resolution_notes = db.Column(db.Text)
    evidence_files = db.Column(db.JSON)

class PoliceIncidents(db.Model):
    __tablename__ = 'police_incidents'
    police_id = db.Column(db.String(50), db.ForeignKey('police.account_id'),
primary_key=True)
    incident_id = db.Column(db.Integer,
db.ForeignKey('incidents.incident_id'), primary_key=True)
    role_type = db.Column(db.String(20), nullable=False)  # 'assigned',
'registered'

class LawyerIncidents(db.Model):
    __tablename__ = 'lawyer_incidents'
```

```
    lawyer_id = db.Column(db.String(50), db.ForeignKey('lawyers.account_id'),
primary_key=True)
    incident_id = db.Column(db.Integer,
db.ForeignKey('incidents.incident_id'), primary_key=True)
    role_type = db.Column(db.String(20), nullable=False)  # 'assigned'

class CivilianIncidents(db.Model):
    __tablename__ = 'civilian_incidents'
    civilian_id = db.Column(db.String(50),
db.ForeignKey('civilians.account_id'), primary_key=True)
    incident_id = db.Column(db.Integer,
db.ForeignKey('incidents.incident_id'), primary_key=True)
    role_type = db.Column(db.String(20), nullable=False)  # 'filed', 'witness'

class Case(db.Model):
    __tablename__ = 'cases'
    case_id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    incident_id = db.Column(db.Integer,
db.ForeignKey('incidents.incident_id'), nullable=False)
    account_id = db.Column(db.String(50),
db.ForeignKey('accounts.account_id'), nullable=False)
    case_type = db.Column(db.String(20), nullable=False)  # 'filed',
'handled', 'solved'
    status = db.Column(db.String(50), nullable=False)  # e.g., 'pending',
'in_progress', 'closed'

class Support(db.Model):
    __tablename__ = 'support'
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    question = db.Column(db.Text, nullable=False)
    answer = db.Column(db.Text, nullable=False)

def generate_token(account_id, role):
    """Generate JWT token for authenticated users"""
    payload = {
        'account_id': account_id,
        'role': role,
        'exp': datetime.utcnow() + datetime.timedelta(days=1)
    }
    return jwt.encode(payload, app.config['SECRET_KEY'], algorithm='HS256')

def verify_token(token):
    """Verify JWT token"""
    try:
        payload = jwt.decode(token, app.config['SECRET_KEY'],
algorithms=['HS256'])
        return payload
    except jwt.ExpiredSignatureError:
```

```
        return None
    except jwt.InvalidTokenError:
        return None

@app.route('/api/auth/signup', methods=['POST'])
def signup():
    try:
        data = request.json
        required_fields = ['account_id', 'email', 'password',
'confirm_password', 'role', 'name']

        # Validate required fields
        for field in required_fields:
            if field not in data:
                return jsonify({'error': f'Missing required field: {field}'}),
400

        # Validate role
        if data['role'] not in ['civilian', 'police', 'lawyer']:
            return jsonify({'error': 'Invalid role'}), 400

        # Validate password
        if len(data['password']) < 8:
            return jsonify({'error': 'Password must be at least 8
characters'}), 400

        if data['password'] != data['confirm_password']:
            return jsonify({'error': 'Passwords do not match'}), 400

        # Check if account already exists
        if Account.query.filter_by(account_id=data['account_id']).first():
            return jsonify({'error': 'Account ID already exists'}), 400

        if Account.query.filter_by(email=data['email']).first():
            return jsonify({'error': 'Email already registered'}), 400

        # Hash password
        hashed_password = generate_password_hash(data['password'])

        # Create account
        account = Account(
            account_id=data['account_id'],
            email=data['email'],
            password=hashed_password,
            role=data['role']
        )
        db.session.add(account)
```

```
        # Handle role-specific data
        if data['role'] == 'civilian':
            if Civilian.query.filter_by(username=data['name']).first():
                return jsonify({'error': 'Username already taken'}), 400
            civilian = Civilian(
                account_id=data['account_id'],
                username=data['name'],
                name=data['name']
            )
            db.session.add(civilian)

        elif data['role'] == 'police':
            if 'badge_number' not in data:
                return jsonify({'error': 'Badge number is required for police
officers'}), 400
            if
Police.query.filter_by(badge_number=data['badge_number']).first():
                return jsonify({'error': 'Badge number already registered'}),
400

            police = Police(
                account_id=data['account_id'],
                police_name=data['name'],
                badge_number=data['badge_number'],
                department=data.get('department'),
                rank=data.get('rank'),
                supervisor_id=data.get('supervisor_id')
            )
            db.session.add(police)

        elif data['role'] == 'lawyer':
            if 'bar_id' not in data:
                return jsonify({'error': 'Bar ID is required for lawyers'}),
400
            if Lawyer.query.filter_by(bar_id=data['bar_id']).first():
                return jsonify({'error': 'Bar ID already registered'}), 400

            lawyer = Lawyer(
                account_id=data['account_id'],
                lawyer_name=data['name'],
                bar_id=data['bar_id'],
                specialization=data.get('specialization'),
                experience_years=data.get('experience_years')
            )
            db.session.add(lawyer)

        db.session.commit()
```

```
        return jsonify({
            'message': 'User registered successfully',
            'account_id': data['account_id']
        }), 201

    except Exception as e:
        db.session.rollback()
        print(f"Error in signup: {str(e)}")
        return jsonify({'error': str(e)}), 500

@app.route('/api/auth/login', methods=['POST'])
def login():
    try:
        data = request.json
        if not data or 'email' not in data or 'password' not in data:
            return jsonify({'error': 'Missing email or password'}), 400

        # Find account
        account = Account.query.filter_by(email=data['email']).first()
        if not account:
            return jsonify({'error': 'User not found'}), 404

        # Verify password
        if not check_password_hash(account.password, data['password']):
            return jsonify({'error': 'Invalid password'}), 401

        # Check verification status for lawyers and police
        if account.role in ['lawyer', 'police'] and not account.verified:
            return jsonify({'error': 'Account pending verification'}), 403

        # Generate token
        token = generate_token(account.account_id, account.role)

        # Prepare user data
        user_data = {
            'account_id': account.account_id,
            'email': account.email,
            'role': account.role
        }

        # Add role-specific data
        if account.role == 'civilian':
            civilian =
Civilian.query.filter_by(account_id=account.account_id).first()
            if civilian:
                user_data.update({
                    'username': civilian.username,
                    'name': civilian.name
```

```
                })
        elif account.role == 'lawyer':
            lawyer =
Lawyer.query.filter_by(account_id=account.account_id).first()
            if lawyer:
                user_data.update({
                    'name': lawyer.lawyer_name,
                    'bar_id': lawyer.bar_id,
                    'specialization': lawyer.specialization,
                    'experience_years': lawyer.experience_years
                })
        elif account.role == 'police':
            police =
Police.query.filter_by(account_id=account.account_id).first()
            if police:
                user_data.update({
                    'name': police.police_name,
                    'badge_number': police.badge_number,
                    'department': police.department,
                    'rank': police.rank
                })

        return jsonify({
            'token': token,
            'user': user_data
        }), 200

    except Exception as e:
        print(f"Error in login: {str(e)}")
        return jsonify({'error': str(e)}), 500

@app.route('/api/auth/verify', methods=['POST'])
def verify_user():
    try:
        data = request.json
        if not data or 'account_id' not in data or 'verified' not in data:
            return jsonify({'error': 'Missing required fields'}), 400

        account = Account.query.get(data['account_id'])
        if not account:
            return jsonify({'error': 'User not found'}), 404

        account.verified = data['verified']
        db.session.commit()

        return jsonify({'message': 'User verification status updated'}), 200

    except Exception as e:
```

```
        db.session.rollback()
        print(f"Error in verification: {str(e)}")
        return jsonify({'error': str(e)}), 500

@app.route('/api/auth/verify-token', methods=['POST'])
def verify_token_route():
    try:
        data = request.json
        if not data or 'token' not in data:
            return jsonify({'error': 'Missing token'}), 400

        payload = verify_token(data['token'])
        if payload:
            return jsonify({'valid': True, 'payload': payload}), 200
        else:
            return jsonify({'valid': False, 'error': 'Invalid or expired
token'}), 401

    except Exception as e:
        print(f"Error in token verification: {str(e)}")
        return jsonify({'error': str(e)}), 500

@app.route('/api/incidents', methods=['POST'])
def create_incident():
    try:
        # Verify token and get user role
        token = request.headers.get('Authorization')
        if not token:
            return jsonify({'error': 'No token provided'}), 401

        token_data = verify_token(token)
        if not token_data or not token_data.get('valid'):
            return jsonify({'error': 'Invalid token'}), 401

        user_role = token_data['payload'].get('role')
        if user_role != 'police':
            return jsonify({'error': 'Only police officers can create
incidents'}), 403

        data = request.json
        required_fields = ['title', 'description', 'incident_type',
'location', 'incident_date']

        # Validate required fields
        for field in required_fields:
            if field not in data:
                return jsonify({'error': f'Missing required field: {field}'}),
400
```

```
        # Create incident
        incident = Incident(
            title=data['title'],
            description=data['description'],
            incident_type=data['incident_type'],
            location=data['location'],
            incident_date=datetime.strptime(data['incident_date'], '%Y-%m-
%d'),
            priority=data.get('priority', 'Medium'),
            category=data.get('category')
        )
        db.session.add(incident)

        # Create police-incident relationship
        police_incident = PoliceIncidents(
            police_id=token_data['payload'].get('account_id'),
            incident_id=incident.incident_id,
            role_type='registered'
        )
        db.session.add(police_incident)

        db.session.commit()

        return jsonify({
            'message': 'Incident created successfully',
            'incident_id': incident.incident_id
        }), 201

    except Exception as e:
        db.session.rollback()
        print(f"Error creating incident: {str(e)}")
        return jsonify({'error': str(e)}), 500

@app.route('/api/incidents', methods=['GET'])
def get_incidents():
    try:
        # Verify token and get user role
        token = request.headers.get('Authorization')
        if not token:
            return jsonify({'error': 'No token provided'}), 401

        token_data = verify_token(token)
        if not token_data or not token_data.get('valid'):
            return jsonify({'error': 'Invalid token'}), 401

        user_role = token_data['payload'].get('role')
        account_id = token_data['payload'].get('account_id')
```

```
        # Get incidents based on user role
        if user_role == 'police':
            incidents = db.session.query(Incident, PoliceIncidents).join(
                PoliceIncidents, Incident.incident_id ==
PoliceIncidents.incident_id
            ).filter(PoliceIncidents.police_id == account_id).all()
        elif user_role == 'lawyer':
            incidents = db.session.query(Incident, LawyerIncidents).join(
                LawyerIncidents, Incident.incident_id ==
LawyerIncidents.incident_id
            ).filter(LawyerIncidents.lawyer_id == account_id).all()
        else:  # civilian
            incidents = db.session.query(Incident, CivilianIncidents).join(
                CivilianIncidents, Incident.incident_id ==
CivilianIncidents.incident_id
            ).filter(CivilianIncidents.civilian_id == account_id).all()

        # Format incidents
        incidents_list = []
        for incident, relation in incidents:
            incident_data = {
                'id': incident.incident_id,
                'title': incident.title,
                'description': incident.description,
                'incident_type': incident.incident_type,
                'location': incident.location,
                'incident_date': incident.incident_date.isoformat(),
                'status': incident.status,
                'created_at': incident.created_at.isoformat(),
                'priority': incident.priority,
                'category': incident.category
            }
            incidents_list.append(incident_data)

        return jsonify(incidents_list), 200

    except Exception as e:
        print(f"Error fetching incidents: {str(e)}")
        return jsonify({'error': str(e)}), 500

@app.route('/api/incidents/<incident_id>', methods=['PUT'])
def update_incident(incident_id):
    try:
        # Verify token and get user role
        token = request.headers.get('Authorization')
        if not token:
            return jsonify({'error': 'No token provided'}), 401
```

```
        token_data = verify_token(token)
        if not token_data or not token_data.get('valid'):
            return jsonify({'error': 'Invalid token'}), 401

        user_role = token_data['payload'].get('role')
        if user_role not in ['police', 'lawyer']:
            return jsonify({'error': 'Only police officers and lawyers can
update incidents'}), 403

        data = request.json
        incident = Incident.query.get(incident_id)

        if not incident:
            return jsonify({'error': 'Incident not found'}), 404

        # Update fields if provided
        if 'status' in data:
            incident.status = data['status']
        if 'resolution_notes' in data:
            incident.resolution_notes = data['resolution_notes']

        # Handle lawyer assignment
        if user_role == 'police' and 'assigned_lawyer_id' in data:
            lawyer_incident = LawyerIncidents(
                lawyer_id=data['assigned_lawyer_id'],
                incident_id=incident_id,
                role_type='assigned'
            )
            db.session.add(lawyer_incident)

        db.session.commit()

        return jsonify({'message': 'Incident updated successfully'}), 200

    except Exception as e:
        db.session.rollback()
        print(f"Error updating incident: {str(e)}")
        return jsonify({'error': str(e)}), 500

# Create database tables
with app.app_context():
    db.create_all()

if __name__ == '__main__':
    app.run(debug=True, port=5001)
```

IPC Nexus Backend Documentation:

Overview

IPC Nexus is an advanced backend architecture developed using Flask, facilitating user authentication, role-based access control, and structured case management for civilians, lawyers, and law enforcement personnel. The system integrates robust authentication mechanisms, incident creation and tracking functionalities, and secure data management to uphold legal and procedural integrity. ⚖️🔒📜

Technologies Utilized

- **Flask** – A micro web framework for Python.
- **Flask-CORS** – Enables secure Cross-Origin Resource Sharing for frontend interaction.
- **Flask-SQLAlchemy** – ORM layer for seamless database management with PostgreSQL.
- **Flask-Migrate** – Facilitates database schema migrations via Alembic.
- **PostgreSQL** – A scalable relational database system used for transactional integrity.
- **JWT Authentication** – Implements token-based authentication for secure user session management.
- **Bcrypt** – Ensures cryptographic security through password hashing.
- **dotenv** – Manages environment configurations to maintain security best practices. 🛠️🔑📡

Database Schema

1. Account Model

The Account model encapsulates user authentication credentials, authorization roles, and verification status. 🏛️🔐📜

- **Schema Fields:**
    - account_id (Primary Key) – Unique identifier for user records.
    - email (Unique) – Acts as the primary credential for authentication.
    - password – Stores the cryptographically hashed user password.
    - role (Enum) – Defines user type as Civilian, Lawyer, or Police Officer.
    - verified (Boolean) – Signifies verification status for lawyers and law enforcement personnel.

- created_at (Timestamp) – Tracks account registration timestamps.

Model Implementation:

```python
from flask_sqlalchemy import SQLAlchemy
from datetime import datetime
from flask_bcrypt import Bcrypt

bcrypt = Bcrypt()
db = SQLAlchemy()

class Account(db.Model):
    __tablename__ = 'accounts'
    account_id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(120), unique=True, nullable=False)
    password = db.Column(db.String(256), nullable=False)
    role = db.Column(db.Enum('Civilian', 'Lawyer', 'Police', name='role_types'), nullable=False)
    verified = db.Column(db.Boolean, default=False)
    created_at = db.Column(db.DateTime, default=datetime.utcnow)
```

2. Incident Model

The Incident model facilitates structured storage and retrieval of case-related data.
🚓📄📍

- **Schema Fields:**
    - incident_id (Primary Key) – Unique identifier for each reported incident.
    - title – Concise summary of the case.
    - description – In-depth elaboration of the reported event.
    - incident_type – Categorization of the incident.
    - location – Geographical context of the occurrence.
    - incident_date (Timestamp) – Date and time when the event transpired.
    - status (Enum) – Tracks case progress (Open, Under Investigation, Resolved).
    - priority (Enum) – Assigns urgency level (Low, Medium, High).
    - category – Defines the nature of the case (e.g., Theft, Assault, Fraud).
    - resolution_notes – Documents case resolution details.
    - evidence_files – Stores references to multimedia evidence.

Model Implementation:

```
class Incident(db.Model):
    __tablename__ = 'incidents'
    incident_id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(255), nullable=False)
    description = db.Column(db.Text, nullable=False)
    incident_type = db.Column(db.String(50), nullable=False)
    location = db.Column(db.String(255), nullable=False)
    incident_date = db.Column(db.DateTime, nullable=False)
    status = db.Column(db.Enum('Open', 'Under Investigation', 'Resolved', name='incident_status'), default='Open')
    priority = db.Column(db.Enum('Low', 'Medium', 'High', name='incident_priority'), default='Medium')
    category = db.Column(db.String(100), nullable=False)
    resolution_notes = db.Column(db.Text)
    evidence_files = db.Column(db.ARRAY(db.String))
```

Authentication & Role-Based Access Control

1. User Registration (POST /api/auth/signup)

Facilitates account creation by securely hashing user passwords before persisting them in the database. 🔑📧🔒

Endpoint Implementation:

```
from flask import Flask, request, jsonify
import jwt
import os
from flask_jwt_extended import create_access_token

app = Flask(__name__)
app.config['SECRET_KEY'] = 'your_secret_key'

@app.route('/api/auth/signup', methods=['POST'])
def signup():
    data = request.get_json()
    hashed_password = bcrypt.generate_password_hash(data['password']).decode('utf-8')
    new_user = Account(email=data['email'], password=hashed_password, role=data['role'])
    db.session.add(new_user)
    db.session.commit()
    return jsonify({'message': 'User registered successfully'}), 201
```

2. User Authentication (POST /api/auth/login)

Validates user credentials and generates a JWT access token. 🔐🔄📜

Endpoint Implementation:

```python
@app.route('/api/auth/login', methods=['POST'])
def login():
    data = request.get_json()
    user = Account.query.filter_by(email=data['email']).first()
    if user and bcrypt.check_password_hash(user.password, data['password']):
        token = create_access_token(identity={'user_id': user.account_id, 'role': user.role})
        return jsonify({'token': token})
    return jsonify({'message': 'Invalid credentials'}), 401
```

Incident Management

1. Incident Creation (POST /api/incidents)

Restricted to law enforcement personnel to ensure procedural legitimacy in case registration. 🚔 🔍 📄

Endpoint Implementation:

```python
from flask_jwt_extended import jwt_required, get_jwt_identity

@app.route('/api/incidents', methods=['POST'])
@jwt_required()
def create_incident():
    data = request.get_json()
    current_user = get_jwt_identity()
    user = Account.query.get(current_user['user_id'])
    if user.role != 'Police':
        return jsonify({'message': 'Unauthorized'}), 403
    new_incident = Incident(
        title=data['title'],
        description=data['description'],
        incident_type=data['incident_type'],
        location=data['location'],
        incident_date=datetime.strptime(data['incident_date'], '%Y-%m-%d'),
        priority=data['priority'],
        category=data['category']
    )
    db.session.add(new_incident)
    db.session.commit()
    return jsonify({'message': 'Incident created successfully'}), 201
```

Security Considerations

- **Cryptographic Password Storage:** Bcrypt ensures irreversible password hashing.
- **JWT-Based Authentication:** Secure session handling through tokenization.

- **Granular Access Control:** Endpoint permissions are defined based on user roles.
- **Administrative Verification:** Lawyers and police officers must undergo validation.
- **Data Encryption:** Sensitive data is encrypted to prevent unauthorized access.
- **Audit Logging:** Tracks critical operations for accountability. 🔒 🛡️ 📜

Conclusion

IPC Nexus delivers a secure and scalable backend solution, integrating structured authentication and incident management tailored for law enforcement, legal professionals, and civilians. With role-based access control, JWT authentication, and encrypted data storage, the system upholds both security and procedural rigor, ensuring reliability and compliance with regulatory standards. 🚀 📜 🛠️