

# **Full-Project Report for**

## **IPC-NEXUS:**

### **How we decided this to be our problem Statement:**

Firstly we wanted to solve a real time problem, which solves a particular issue or helps for a certain cause within the society. Then we found a problem listed by Madhya Pradesh police.

### **PROBLEM-STATEMENT:**

Due to unavailability of legal experts in Police stations, whenever a complainant comes to police with his/her complaint to get a First Information Report(FIR) written sometimes police make mistakes in writing all the sections/acts due to which problems arise in the investigation later on.

**Description:** Whenever a complainant comes to the police station with his/her complaint to get the First Information Report written, the investigating officer sometimes due to ignorance or lack of experience is not able to follow the provisions and the First Information Report remains deficient. Due to which, later on, problems arise in the investigation and justice gets affected, as legal experts are not posted at police stations. If there will be any such software or app which can use Artificial Intelligence (AI) and immediately provide us the information regarding appropriate and correct sections and acts, as per the incident of the complainant. So there will be no errors regarding the sections and acts under the First Information Report and the investigation and justice will not be affected and will help in investigation.

## **HOW OUR PROJECT SOLVES THIS PROBLEM-STATEMENT:**

"Need to understand the legal implications of a crime? Our web application provides instant access to relevant Indian Penal Code (IPC) sections and punishments. Simply describe the incident or provide keywords, and our NLP-powered system will deliver precise legal information. Built with a modern React interface and a powerful Flask backend, the application also allows for secure complaint registration and retrieval using a SQL-ALCHEMY database and persistent volumes in kubernetes. Empowering users with legal clarity, this tool simplifies complex legal terminology and provides essential information at your fingertips."

## **LET'S DISCUSS THE FUNCTIONALITY IN DETAIL:**

### **Backend Technologies:**

1. **Flask (Python Web Framework):**
  - Core backend framework
  - Used for building RESTful APIs and handling HTTP requests
  - Version: 2.3.3
2. **Flask Extensions :**
  - **flask-cors**: For handling Cross-Origin Resource Sharing (CORS)
  - **flask-sqlalchemy**: For database ORM (Object-Relational Mapping)
  - **werkzeug**: WSGI web application library
3. **Machine Learning & Data Processing:**
  - **sentence-transformers**: For natural language processing and text embeddings
  - **torch**: PyTorch for deep learning operations
  - **pandas**: For data manipulation and analysis
  - **numpy**: For numerical computations
  - **huggingface hub**: For accessing pre-trained models
4. **Database:**
  - SQLAlchemy as the ORM
  - SQLite database (indicated by `law_enforcement.db`)
5. **Deployment & Server:**
  - **unicorn**: Production-grade WSGI server
  - Docker for containerization
  - Kubernetes (k8s) for orchestration

## Frontend Technologies:

1. **React Framework:**
  - Core frontend library
  - Version: 19.0.0
  - Using React Router for navigation
2. **UI Libraries & Components:**
  - react-bootstrap: For Bootstrap components in React
  - bootstrap: For styling and responsive design
  - @fortawesome/fontawesome-free: For icons
  - react-icons: Additional icon library
  - tailwindcss: Utility-first CSS framework
3. **Development Tools:**
  - vite: Modern build tool and development server
  - eslint: For code linting
  - postcss & autoprefixer: For CSS processing
4. **Data Handling:**
  - axios: For making HTTP requests
  - papaparse: For CSV file parsing

## DevOps & Infrastructure:

1. **Containerization:**
  - Docker for containerization
  - Docker Compose for multi-container applications
2. **Deployment:**
  - Kubernetes (k8s) for container orchestration
  - Nginx as a web server (indicated by nginx.conf)
3. **Environment Management:**
  - .env files for environment variables
  - Python-dotenv for environment variable management

## **Now lets firstly start with our DATABASE Approach:**

AS we were requested by our client for a minimum of 50 tables relationship. But we thought why not beyond:

So we came up with an idea of Enhanced Database Functionality for Dynamic Table Generation, Instead of manually creating a fixed number of tables, we have integrated a feature that allows authorized users to generate specific tables on demand. This functionality leverages provided keys and information to automatically create tables tailored to individual police stations and courts. Furthermore, it establishes seamless linkages with existing police and lawyer tables, ensuring data integrity and consistency across the database.

**Given the scale of India's legal and law enforcement infrastructure, with approximately 16,000 police stations and 2,000 courts (including Supreme Court, High Courts, District Courts, and local courts), this dynamic table generation functionality addresses a critical need for flexible data management.** This solution effectively provides the capability to create and manage data for a volume exceeding more than 300 times the originally requested number of tables, thereby demonstrating a substantial improvement in scalability and efficiency.

## **CODE:**

```
@app.route('/api/policeInfo', methods=['POST'])
def add_police_info():
    data = request.json
    state = data.get('state')
    pin_code = data.get('pinCode')
    station_number = data.get('stationNumber')
    station_location = data.get('stationLocation')
    police_id = data.get('policeId')

    if not all([state, pin_code, station_number, station_location,
police_id]):
        return jsonify({'message': 'All fields are required', 'success':
False}), 400
```

```

try:
    pin_code = int(pin_code)
    station_number = int(station_number)
except (ValueError, TypeError):
    return jsonify({'message': 'Pin Code and Station Number must be
numeric', 'success': False}), 400

    if not isinstance(state, str) or not isinstance(station_location, str) or
not isinstance(police_id, str):
        return jsonify({'message': 'State, Station Location, and Police ID
must be strings', 'success': False}), 400

    police = Police.query.filter_by(badge_id=police_id).first()
    if not police:
        return jsonify({'message': 'Police not found', 'success': False}), 404

    try:
        existing_station =
PoliceInfo.query.filter_by(station_number=station_number).first()
        if existing_station:
            return jsonify({'message': 'Station number already registered',
'success': False}), 400

        table_name = f"police_station_{station_number}"
        metadata = MetaData()

        inspector = inspect(db.engine)
        if not inspector.has_table(table_name):
            new_table = Table(
                table_name, metadata,
                Column('id', Integer, primary_key=True),
                Column('state', String(100), nullable=False),
                Column('pin_code', Integer, nullable=False),
                Column('station_number', Integer, nullable=False),
                Column('station_location', String(200), nullable=False),
                Column('police_id', String(20), nullable=False)
            )
            metadata.create_all(db.engine)
            print(f"Created new table: {table_name}")

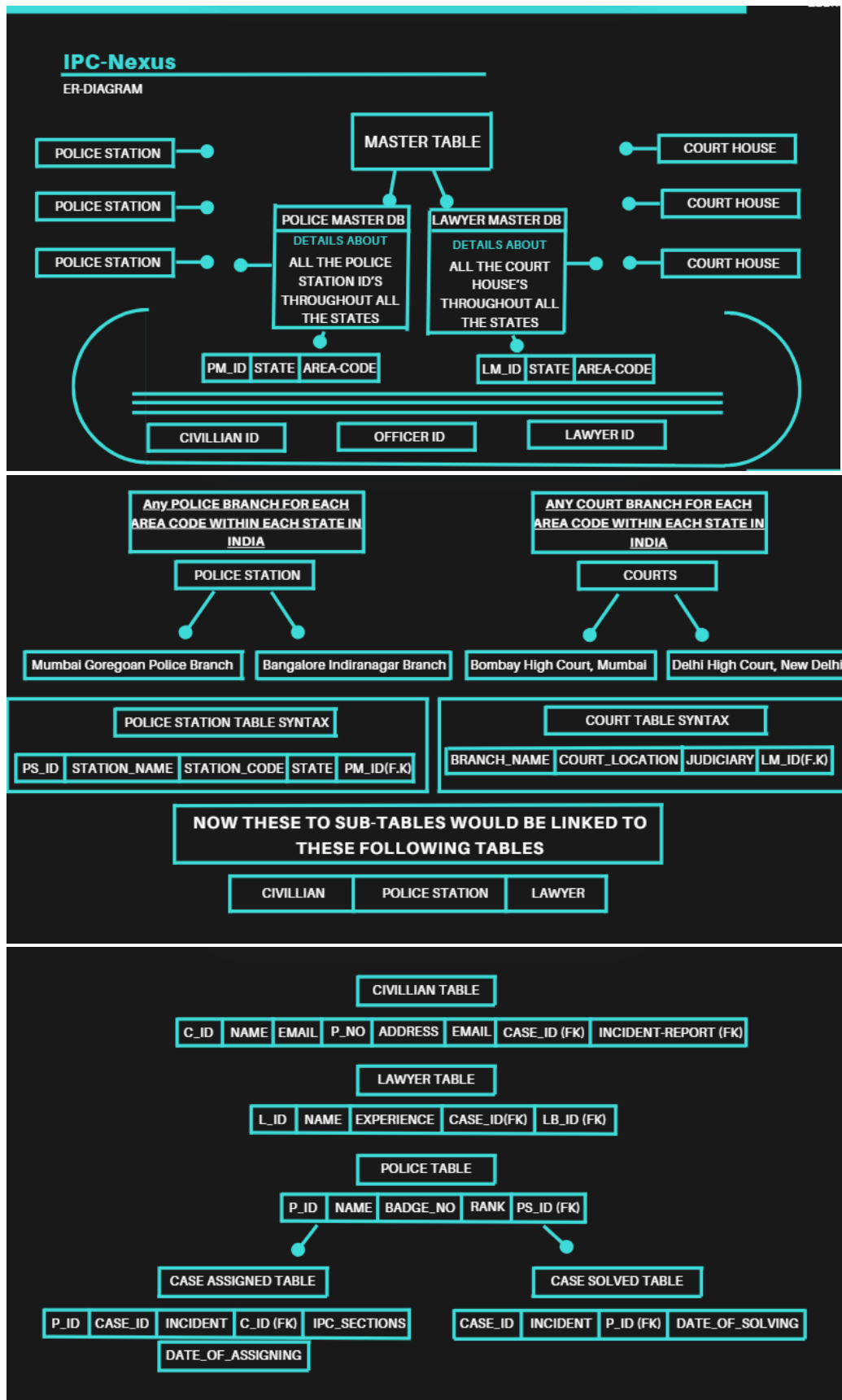
        with db.engine.connect() as connection:
            insert_stmt = text(
                f"INSERT INTO {table_name} (state, pin_code, station_number,
station_location, police_id) "
                f"VALUES (:state, :pin_code, :station_number,
:station_location, :police_id)"
            )

```

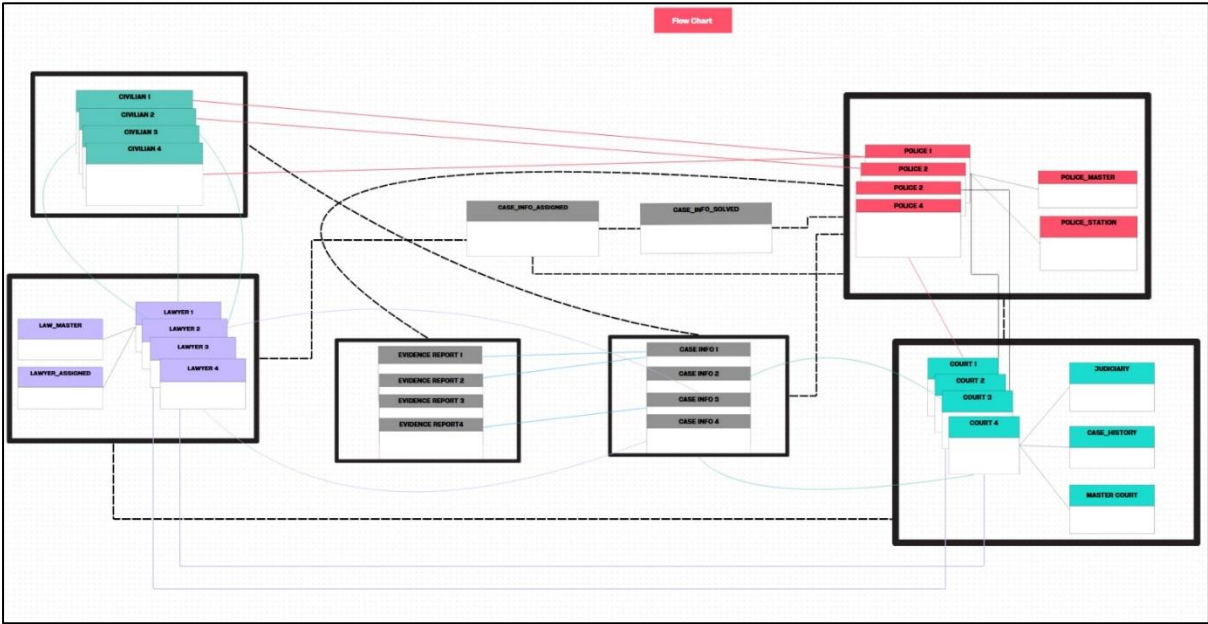
```
        connection.execute(
            insert_stmt,
            {"state": state, "pin_code": pin_code, "station_number":
station_number,
            "station_location": station_location, "police_id": police_id}
        )
        connection.commit()

    return jsonify({'message': f'Police-Station info registered
successfully in {table_name}', 'success': True}), 201
except Exception as e:
    db.session.rollback()
    return jsonify({'message': f'Error adding police-station info:
{str(e)}', 'success': False}), 500
```

## Schema For our Table:



ER-DIAGRAM:



DETAILED FLOW AND RELATIONSHIPS IN ER:

Summary of Key Relations:		
Table A	Table B	Relation Type
Civilian	Police	Many Civilians → 1 Police
Civilian	Lawyer_Assigned	1 Civilian → Many Assignments
Lawyer_Assigned	Lawyer	Many Assignments → 1 Lawyer
Lawyer_Assigned	Law_Master	Many Assignments → 1 Law Master
Case Info	Civilian, Police, Lawyer	Many Cases → 1 of each
Case Info	Evidence Report	1 Case → Many Reports
Case Info	Court	Many Cases → 1 Court
Court	Master Court	Many Courts → 1 Master Court
Court	Judiciary	Many Courts → 1 Judiciary
Police	Police_Master, Police_Station	Many → 1 each



## DATABASE STRUCTURE:

```
from flask import Flask, request, jsonify
from flask_sqlalchemy import SQLAlchemy
from flask_cors import CORS
from werkzeug.security import generate_password_hash, check_password_hash
import os

app = Flask(__name__)
CORS(app, resources={r"/api/*": {"origins": "*"}}, supports_credentials=True)

# Database config
DB_PATH = r"E:\Mthree_Project_final\Backend\Database\law_enforcement.db"
os.makedirs(os.path.dirname(DB_PATH), exist_ok=True)
app.config['SQLALCHEMY_DATABASE_URI'] = f'sqlite:/// {DB_PATH}'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db = SQLAlchemy(app)

# ----- MODELS -----

class Account(db.Model):
    __tablename__ = 'accounts'
    account_id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(50), unique=True, nullable=False)
    email = db.Column(db.String(100), unique=True, nullable=True)
    phoneno = db.Column(db.String(15), nullable=False)
    password_hash = db.Column(db.String(128), nullable=False)
    civilian = db.relationship('Civilian', backref='account', uselist=False)
    lawyer = db.relationship('Lawyer', backref='account', uselist=False)
    police = db.relationship('Police', backref='account', uselist=False)

class Civilian(db.Model):
    __tablename__ = 'civilians'
    civilian_id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(50), unique=True, nullable=False)
    account_id = db.Column(db.Integer, db.ForeignKey('accounts.account_id'),
    nullable=False)
    cases = db.relationship('Case', backref='civilian')
    # Removed: incidents = db.relationship('Incident', backref='civilian')

class Lawyer(db.Model):
    __tablename__ = 'lawyers'
    lawyer_id = db.Column(db.String(20), primary_key=True)
    account_id = db.Column(db.Integer, db.ForeignKey('accounts.account_id'),
    nullable=False)
    case = db.relationship('Case', backref='lawyer', uselist=False)
```

```

        court_location_id = db.Column(db.Integer,
db.ForeignKey('court_locations.court_id'))

class Police(db.Model):
    __tablename__ = 'police'
    badge_id = db.Column(db.Integer, primary_key=True)
    account_id = db.Column(db.Integer, db.ForeignKey('accounts.account_id'),
nullable=False)
    police_station_id = db.Column(db.Integer,
db.ForeignKey('police_stations.ps_id'))
    case_assignments = db.relationship('CaseAssign', backref='police')
    case_solved = db.relationship('CaseSolved', backref='police')

class LawyerInfo(db.Model):
    __tablename__ = 'lawyer_info'
    info_id = db.Column(db.Integer, primary_key=True)
    bar_id = db.Column(db.String(20), db.ForeignKey('lawyers.lawyer_id'),
nullable=False)
    branch_name = db.Column(db.String(100))
    state = db.Column(db.String(100))
    court_location = db.Column(db.String(100))
    judiciary = db.Column(db.String(100))
    judiciary_id = db.Column(db.String(50))
    lawyer = db.relationship('Lawyer', backref='info', uselist=False)

class PoliceInfo(db.Model):
    __tablename__ = 'police_station_info'
    id = db.Column(db.Integer, primary_key=True)
    state = db.Column(db.String(100), nullable=False)
    pin_code = db.Column(db.Integer, nullable=False)
    station_number = db.Column(db.Integer, nullable=False)
    station_location = db.Column(db.String(200), nullable=False)
    police_id = db.Column(db.String(20), nullable=False)

class Case(db.Model):
    __tablename__ = 'cases'
    case_id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(100), nullable=False)
    description = db.Column(db.Text, nullable=False)
    civilian_id = db.Column(db.Integer,
db.ForeignKey('civilians.civilian_id'))
    lawyer_id = db.Column(db.String(20), db.ForeignKey('lawyers.lawyer_id'))
    case_assign = db.relationship('CaseAssign', backref='case', uselist=False)
    case_solved = db.relationship('CaseSolved', backref='case', uselist=False)

class CaseAssign(db.Model):
    __tablename__ = 'case_assign'
    assign_id = db.Column(db.Integer, primary_key=True)

```

```

    case_id = db.Column(db.Integer, db.ForeignKey('cases.case_id'))
    police_id = db.Column(db.String(20), db.ForeignKey('police.badge_id'))

class CaseSolved(db.Model):
    __tablename__ = 'case_solved'
    solved_id = db.Column(db.Integer, primary_key=True)
    case_id = db.Column(db.Integer, db.ForeignKey('cases.case_id'))
    police_id = db.Column(db.String(20), db.ForeignKey('police.badge_id'))

class PoliceStation(db.Model):
    __tablename__ = 'police_stations'
    ps_id = db.Column(db.Integer, primary_key=True)
    station_name = db.Column(db.String(100), nullable=False)
    police = db.relationship('Police', backref='station', uselist=False)
    master_id = db.Column(db.Integer, db.ForeignKey('police_master.pm_id'))

class PoliceMaster(db.Model):
    __tablename__ = 'police_master'
    pm_id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    stations = db.relationship('PoliceStation', backref='master')

class CourtLocation(db.Model):
    __tablename__ = 'court_locations'
    court_id = db.Column(db.Integer, primary_key=True)
    location = db.Column(db.String(100), nullable=False)
    master_id = db.Column(db.Integer, db.ForeignKey('lawyer_master.lm_id'))
    lawyers = db.relationship('Lawyer', backref='court_location')

class LawyerMaster(db.Model):
    __tablename__ = 'lawyer_master'
    lm_id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    courts = db.relationship('CourtLocation', backref='master')

class Incident(db.Model):
    __tablename__ = 'incidents'
    incident_id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100))
    email = db.Column(db.String(100))
    phone = db.Column(db.String(15))
    description = db.Column(db.Text, nullable=False)
    location = db.Column(db.String(100), nullable=False)
    address = db.Column(db.String(200))
    incident_date = db.Column(db.String(20), nullable=False)
    timestamp = db.Column(db.String(50))

class Support(db.Model):

```

```

__tablename__ = 'support'
support_id = db.Column(db.Integer, primary_key=True)
message = db.Column(db.Text, nullable=False)
account_id = db.Column(db.Integer, db.ForeignKey('accounts.account_id'))

# ----- ROUTES -----

@app.route('/api/lawyerInfo', methods=['POST'])
def add_lawyer_info():
    data = request.json
    print("Received lawyer info data:", data) # Debug log

    bar_id = data.get('barId')
    branch_name = data.get('branchName')
    state = data.get('state')
    court_location = data.get('courtLocation')
    judiciary = data.get('judiciary')
    judiciary_id = data.get('judiciaryId')

    # Validate required fields
    if not all([bar_id, state]): # Adjust required fields as needed
        print("Missing required fields")
        return jsonify({'message': 'Bar ID and State are required', 'success':
False}), 400

    # Check if lawyer exists
    lawyer = Lawyer.query.filter_by(lawyer_id=bar_id).first()
    if not lawyer:
        print(f"Lawyer not found for bar_id: {bar_id}")
        return jsonify({'message': 'Lawyer not found. Please sign up
first.', 'success': False}), 404

    try:
        # Check if branch_name already exists for this lawyer
        existing_branch = LawyerInfo.query.filter_by(bar_id=bar_id,
branch_name=branch_name).first()
        if existing_branch:
            return jsonify({'message': 'Court Branch already registered for
this lawyer', 'success': False}), 400
        lawyer_info = LawyerInfo(
            bar_id=bar_id,
            branch_name=branch_name,
            state=state,
            court_location=court_location,
            judiciary=judiciary,
            judiciary_id=judiciary_id

```

```

    )
    db.session.add(lawyer_info)
    db.session.commit()
    print(f"Lawyer info added for bar_id: {bar_id}")
    return jsonify({'message': 'Court Registered Successfully', 'success':
True}), 201
except Exception as e:
    db.session.rollback()
    print(f"Error adding lawyer info: {str(e)}")
    return jsonify({'message': f'Error adding lawyer info:
{str(e)}', 'success': False}), 500

@app.route("/api/civilian/complaint", methods=["POST"])
def register_complaint():
    data = request.json
    print("Received data:", data)

    name = data.get('name')
    email = data.get('email')
    phone = data.get('phone')
    description = data.get('description')
    location = data.get('location')
    address = data.get('address')
    incident_date = data.get('incidentDate')
    timestamp = data.get('timestamp')

    if not all([description, location, incident_date]):
        print("Missing required fields")
        return jsonify({'message': 'Description, location, and incident date
are required', 'success': False}), 400

    try:
        print("Creating incident object")
        incident = Incident(
            name=name,
            email=email,
            phone=phone,
            description=description,
            location=location,
            address=address,
            incident_date=incident_date,
            timestamp=timestamp
        )
        print("Adding to session")
        db.session.add(incident)
        print("Committing to database")
        db.session.commit()
        print(f"Incident created with ID: {incident.incident_id}")

```

```

        return jsonify({
            'message': 'Complaint registered successfully',
            'incident_id': incident.incident_id,
            'success': True
        }), 201
    except Exception as e:
        db.session.rollback()
        print(f"Error occurred: {str(e)}")
        return jsonify({'message': f'Error registering complaint: {str(e)}', 'success': False}), 500

@app.route("/api/civilian/login", methods=["POST"])
def login():
    data = request.json
    username = data.get('idOrUsername')
    password = data.get('password')
    if not username or not password:
        return jsonify({"message": "Username and Password required", "success": False}), 400
    account = Account.query.filter_by(username=username).first()
    if not account or not check_password_hash(account.password_hash, password):
        return jsonify({"message": "Invalid credentials", "success": False}), 401
    civilian = Civilian.query.filter_by(account_id=account.account_id).first()
    if not civilian:
        return jsonify({"message": "No civilian profile found", "success": False}), 404
    return jsonify({
        "message": "Login successful",
        "account_id": account.account_id,
        "civilian_id": civilian.civilian_id,
        "userType": "Civilian",
        "success": True
    }), 200

@app.route("/api/lawyer/login", methods=["POST"])
def lawyer_login():
    data = request.json
    lawyer_id = data.get('idOrUsername')
    password = data.get('password')
    if not lawyer_id or not password:
        return jsonify({"message": "Lawyer ID and Password required", "success": False}), 400
    lawyer = Lawyer.query.filter_by(lawyer_id=lawyer_id).first()
    if not lawyer:
        return jsonify({"message": "No lawyer profile found", "success": False}), 404

```

```

        account = Account.query.filter_by(account_id=lawyer.account_id).first()
        if not account or not check_password_hash(account.password_hash,
password):
            return jsonify({"message": "Invalid credentials", "success": False}),
401
        return jsonify({
            "message": "Lawyer login successful",
            "userType": "Lawyer",
            "success": True
        }), 200

@app.route("/api/police/login", methods=["POST"])
def police_login():
    data = request.json
    badge_id = data.get('idOrUsername')
    password = data.get('password')
    if not badge_id or not password:
        return jsonify({"message": "Badge ID and Password required", "success":
False}), 400
    police = Police.query.filter_by(badge_id=badge_id).first()
    if not police:
        return jsonify({"message": "No police profile found", "success":
False}), 404
    account = Account.query.filter_by(account_id=police.account_id).first()
    if not account or not check_password_hash(account.password_hash,
password):
        return jsonify({"message": "Invalid credentials", "success": False}),
401
    return jsonify({
        "message": "Police login successful",
        "userType": "Police",
        "success": True
    }), 200

@app.route("/api/civilian/signup", methods=["POST"])
def signup():
    data = request.json
    username = data.get('username')
    phoneno = data.get('phoneno')
    password = data.get('password')
    if not username or not password:
        return jsonify({'message': 'Username and Password required', 'success':
False}), 400
    existing_user = Account.query.filter_by(username=username).first()
    if existing_user:
        return jsonify({'message': 'User with same username
exists!', 'success': False}), 409
    hashed_password = generate_password_hash(password)

```

```

        account = Account(username=username, phoneno=phoneno,
password_hash=hashed_password)
        db.session.add(account)
        db.session.commit()
        civilian = Civilian(username=username, account_id=account.account_id)
        db.session.add(civilian)
        db.session.commit()
        return jsonify({"message": "Signup successful", "success": True}), 201

@app.route("/api/lawyer/signup", methods=["POST"])
def lawyer_signup():
    data = request.json
    lawyer_id = data.get('id')
    phoneno = data.get('phoneno')
    password = data.get('password')
    email = data.get('email')

    # Validate required fields
    if not all([lawyer_id, password, email, phoneno]):
        return jsonify({'message': 'All fields (Bar ID, Email, Phone Number,
and Password) are required', 'success': False}), 400

    # Validate phone number format
    if not phoneno.isdigit() or len(phoneno) != 10:
        return jsonify({'message': 'Phone number must be a 10-digit
number', 'success': False}), 400

    # Validate email format
    if '@' not in email or '.' not in email:
        return jsonify({'message': 'Please enter a valid email
address', 'success': False}), 400

    # Check if email already exists
    existing_email = Account.query.filter_by(email=email).first()
    if existing_email:
        return jsonify({'message': 'An account with this email already
exists', 'success': False}), 409

    # Check if lawyer ID already exists in Account table (username)
    existing_account = Account.query.filter_by(username=lawyer_id).first()
    if existing_account:
        return jsonify({'message': 'This Bar ID is already
registered', 'success': False}), 409

    # Check if lawyer ID already exists in Lawyer table
    existing_lawyer = Lawyer.query.filter_by(lawyer_id=lawyer_id).first()
    if existing_lawyer:

```



```

        return jsonify({'message': 'This Bar ID is already registered as a
lawyer', 'success': False}), 409

    try:
        hashed_password = generate_password_hash(password)
        account = Account(email=email, phoneno=phoneno,
password_hash=hashed_password, username=lawyer_id)
        db.session.add(account)
        db.session.commit()

        lawyer = Lawyer(lawyer_id=lawyer_id, account_id=account.account_id)
        db.session.add(lawyer)
        db.session.commit()

        return jsonify({"message": "Lawyer signup successful", "success":
True}), 201
    except Exception as e:
        db.session.rollback()
        return jsonify({'message': f'Error during signup: {str(e)}', 'success':
False}), 500

@app.route("/api/police/signup", methods=["POST"])
def police_signup():
    data = request.json
    badge_id = data.get('id')
    phoneno = data.get('phoneno')
    password = data.get('password')
    email = data.get('email')

    # Validate required fields
    if not all([badge_id, password, email, phoneno]):
        return jsonify({'message': 'All fields (Badge ID, Email, Phone Number,
and Password) are required', 'success': False}), 400

    # Validate badge ID format (assuming it should be numeric)
    try:
        badge_id = int(badge_id)
        if badge_id <= 0:
            return jsonify({'message': 'Badge ID must be a positive
number', 'success': False}), 400
        except ValueError:
            return jsonify({'message': 'Badge ID must be a valid
number', 'success': False}), 400

    # Check if email already exists
    existing_email = Account.query.filter_by(email=email).first()
    if existing_email:

```

```

        return jsonify({'message': 'An account with this email already
exists','success': False}), 409

    # Check if badge ID already exists in Account table (username)
    existing_account = Account.query.filter_by(username=str(badge_id)).first()
    if existing_account:
        return jsonify({'message': 'This Badge ID is already
registered','success': False}), 409

    # Check if badge ID already exists in Police table
    existing_police = Police.query.filter_by(badge_id=badge_id).first()
    if existing_police:
        return jsonify({'message': 'This Badge ID is already registered as a
police officer','success': False}), 409

    try:
        hashed_password = generate_password_hash(password)
        account = Account(email=email, phoneno=phoneno,
password_hash=hashed_password, username=str(badge_id))
        db.session.add(account)
        db.session.commit()

        police = Police(badge_id=badge_id, account_id=account.account_id)
        db.session.add(police)
        db.session.commit()

        return jsonify({"message": "Police signup successful","success":
True}), 201
    except Exception as e:
        db.session.rollback()
        return jsonify({'message': f'Error during signup: {str(e)}','success':
False}), 500

@app.route('/api/policeInfo', methods=['POST'])
def add_police_info():
    data = request.json
    state = data.get('state')
    pin_code = data.get('pinCode')
    station_number = data.get('stationNumber')
    station_location = data.get('stationLocation')
    police_id = data.get('policeId')
    if not all([state, pin_code, station_number, station_location,
police_id]):
        return jsonify({'message': 'All fields are required','success':
False}), 400

    try:
        pin_code = int(pin_code)

```

```

        station_number = int(station_number)
    except (ValueError, TypeError):
        return jsonify({'message': 'Pin Code and Station Number must be
numeric','success': False}), 400

    if not isinstance(state, str) or not isinstance(station_location, str) or
not isinstance(police_id, str):
        return jsonify({'message': 'State, Station Location, and Police ID
must be strings','success': False}), 400

    police = Police.query.filter_by(badge_id=police_id).first()
    if not police:
        return jsonify({'message': 'Police not found','success': False}), 404

    existing_station =
PoliceInfo.query.filter_by(station_number=station_number).first()
    if existing_station:
        return jsonify({'message': 'Station number already registered',
'success': False}), 200

    try:
        info = PoliceInfo(
            state=state,
            pin_code=pin_code,
            station_number=station_number,
            station_location=station_location,
            police_id=police_id
        )
        db.session.add(info)
        db.session.commit()
        return jsonify({'message': 'Police-Station info registered
successfully','success': True}), 201
    except Exception as e:
        db.session.rollback()
        return jsonify({'message': f'Error adding police-station info:
{str(e)}','success': False}), 500

if __name__ == "__main__":
    with app.app_context():
        # db.drop_all()
        db.create_all()
        app.run(debug=True)

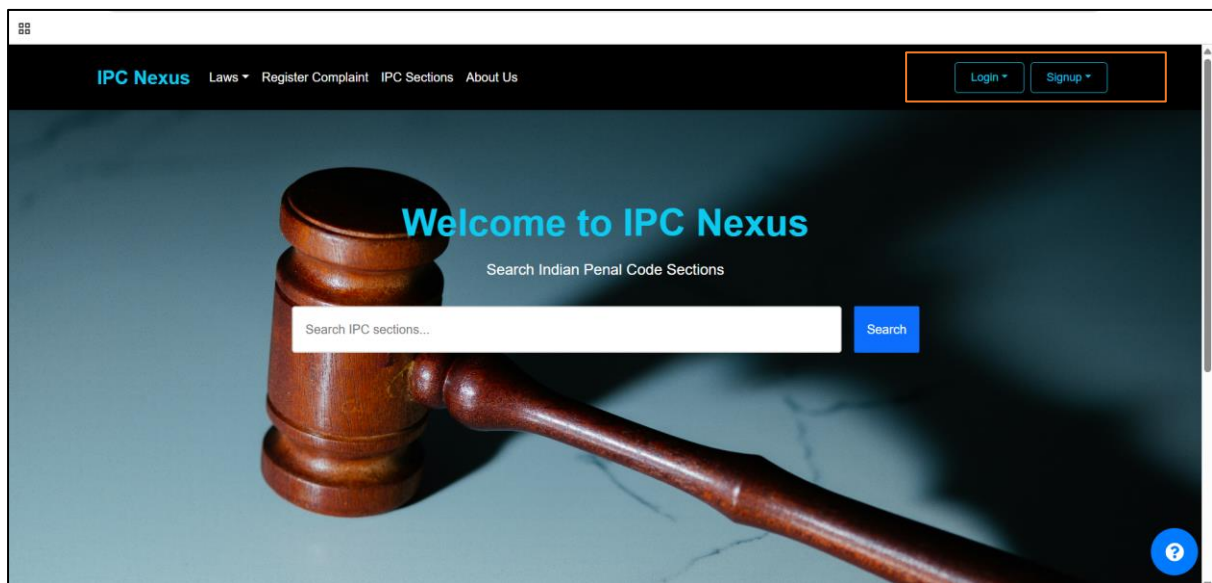
```

## FRONT-END Login and Signup pages:

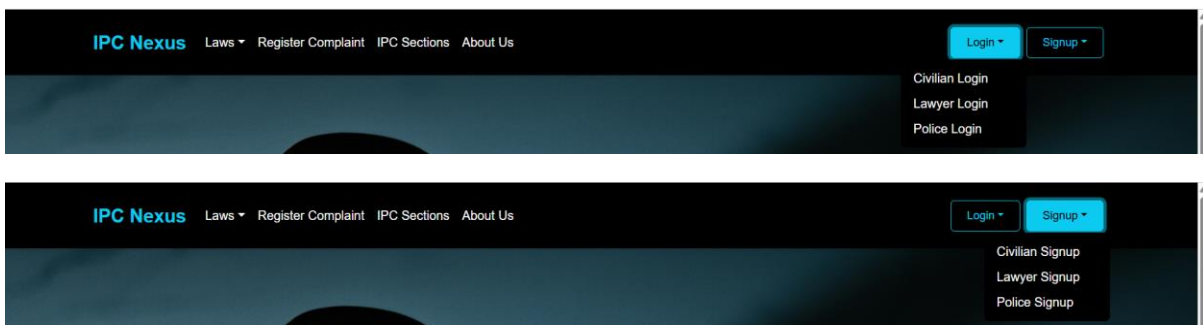
We added the functionality of three kinds of login and signup, depending on the type of user who wants to log in/signup.

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:

The screenshot shows the 'Signup as Civilian' page of the IPC Nexus application. The header includes the 'IPC Nexus' logo and navigation links for 'Laws', 'Register Complaint', 'IPC Sections', and 'About Us'. On the right side of the header are 'Login' and 'Signup' buttons. The main content area is split: the left side features an illustration of a judge in a black robe holding scales and a book, standing on a stack of books; the right side contains the signup form. The form is titled 'Signup as Civilian' and includes five input fields: 'Enter your username', 'Enter your email', 'Enter your phone number', 'Enter your password', and 'Confirm your password'. A 'Signup' button is located at the bottom of the form.

The screenshot shows the 'Signup as Lawyer' page of the IPC Nexus application. The header is identical to the previous page. The main content area is split: the left side features the same judge illustration; the right side contains the signup form. The form is titled 'Signup as Lawyer' and includes five input fields: 'Enter your Lawyer/Bar ID', 'Enter your email', 'Enter your phone number', 'Enter your password', and 'Confirm your password'. A 'Signup' button is located at the bottom of the form.

The screenshot shows the 'Signup as Police' page of the IPC Nexus application. The header is identical to the previous pages. The main content area is split: the left side features the same judge illustration; the right side contains the signup form. The form is titled 'Signup as Police' and includes five input fields: 'Enter your Badge ID', 'Enter your email', 'Enter your phone number', 'Enter your password', and 'Confirm your password'. A 'Signup' button is located at the bottom of the form.

And similarly when the user wants to login:

IPC Nexus

Laws Register Complaint IPC Sections About Us

Login Signup



IPC Nexus

Sign into your account as Civilian

Enter your Username

Enter your password

Login

[Forgot password?](#)

IPC Nexus

Laws Register Complaint IPC Sections About Us

Login Signup



IPC Nexus

Sign into your account as Lawyer

Enter your Lawyer/Bar ID

Enter your password


Login

[Forgot password?](#)

IPC Nexus

Laws Register Complaint IPC Sections About Us

Login Signup



IPC Nexus

Sign into your account as Police

Enter your Badge ID

Enter your password

Login

[Forgot password?](#)

## 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.');
```

```

        />
      </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>
          </form>
        </div>
      </div>
    </div>
  </div>
</div>

```





```

        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.');
```

```

        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>

```

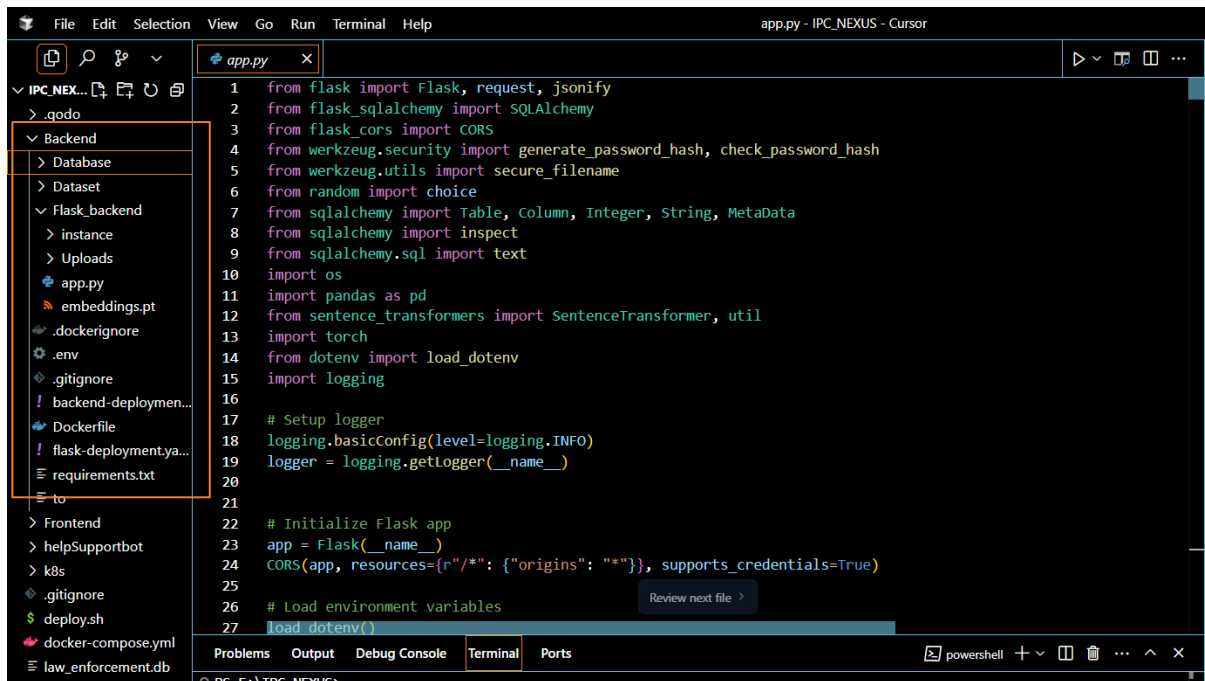
```

    </div>
    <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;
```

## BACKEND STRUCTURE:



The screenshot shows a code editor with the following structure:

- File Explorer (Left Sidebar):
  - IPC\_NEXUS
    - .qodo
    - Backend
      - Database
      - Dataset
      - Flask\_backend
        - instance
        - Uploads
        - app.py
        - embeddings.pt
    - .dockerignore
    - .env
    - .gitignore
    - backend-deploymen...
    - Dockerfile
    - flask-deployment.ya...
    - requirements.txt
    - to
    - Frontend
    - helpSupportbot
    - k8s
    - .gitignore
    - deploy.sh
    - docker-compose.yml
    - law\_enforcement.db
- Main Editor (app.py):

```
1 from flask import Flask, request, jsonify
2 from flask_sqlalchemy import SQLAlchemy
3 from flask_cors import CORS
4 from werkzeug.security import generate_password_hash, check_password_hash
5 from werkzeug.utils import secure_filename
6 from random import choice
7 from sqlalchemy import Table, Column, Integer, String, MetaData
8 from sqlalchemy import inspect
9 from sqlalchemy.sql import text
10 import os
11 import pandas as pd
12 from sentence_transformers import SentenceTransformer, util
13 import torch
14 from dotenv import load_dotenv
15 import logging
16
17 # Setup logger
18 logging.basicConfig(level=logging.INFO)
19 logger = logging.getLogger(__name__)
20
21
22 # Initialize Flask app
23 app = Flask(__name__)
24 CORS(app, resources={r"/*": {"origins": "*"}}, supports_credentials=True)
25
26 # Load environment variables
27 load_dotenv()
```
- Terminal (Bottom):
  - Problems
  - Output
  - Debug Console
  - Terminal (Active)
  - Ports

## Code:

```
from flask import Flask, request, jsonify
from flask_sqlalchemy import SQLAlchemy
from flask_cors import CORS
from werkzeug.security import generate_password_hash, check_password_hash
from werkzeug.utils import secure_filename
from random import choice
from sqlalchemy import Table, Column, Integer, String, MetaData
from sqlalchemy import inspect
from sqlalchemy.sql import text
import os
import pandas as pd
from sentence_transformers import SentenceTransformer, util
import torch
from dotenv import load_dotenv
import logging

# Setup logger
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

# Initialize Flask app
app = Flask(__name__)
CORS(app, resources={r"/*": {"origins": "*"}}, supports_credentials=True)

# Load environment variables
```

```

load_dotenv()

# Paths relative to Flask_backend/
DB_PATH = os.getenv("DB_PATH", os.path.join(os.path.dirname(__file__),
"../Database/law_enforcement.db"))
DATASET_PATH = os.getenv("DATASET_PATH",
os.path.join(os.path.dirname(__file__),
"../Dataset/ipc_sections_updated.csv"))
UPLOAD_FOLDER = os.getenv("UPLOAD_FOLDER",
os.path.join(os.path.dirname(__file__), "Uploads/Evidence"))
os.makedirs(UPLOAD_FOLDER, exist_ok=True)

app.config['SQLALCHEMY_DATABASE_URI'] = f'sqlite:/// {DB_PATH}'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
db = SQLAlchemy(app)

class ResourceLoader:
    _df = None
    _embeddings = None
    _model = None

    @staticmethod
    def get_df():
        if ResourceLoader._df is None:
            logger.info("Loading dataset...")
            if not os.path.exists(DATASET_PATH):
                raise FileNotFoundError(f"Dataset file not found at:
{DATASET_PATH}")
            ResourceLoader._df = pd.read_csv(DATASET_PATH)
            logger.info("Dataset loaded successfully!")
            return ResourceLoader._df

    @staticmethod
    def get_embeddings():
        if ResourceLoader._embeddings is None:
            embeddings_path = os.path.join(os.path.dirname(__file__),
"embeddings.pt")
            if not os.path.exists(embeddings_path):
                raise FileNotFoundError(f"Embeddings file not found at:
{embeddings_path}. Please generate it beforehand.")
            logger.info("Loading precomputed embeddings...")
            ResourceLoader._embeddings = torch.load(embeddings_path)
            logger.info("Precomputed embeddings loaded!")
            return ResourceLoader._embeddings

    @staticmethod
    def get_model():

```

```

        if ResourceLoader._model is None:
            logger.info("Loading NLP model from cache...")
            model_path = os.getenv("SENTENCE_TRANSFORMERS_HOME", "/app/model")

            if not os.path.isdir(model_path) or len(os.listdir(model_path)) ==
0:
                logger.error(f"Pre-cached model not found at: {model_path}.
Falling back to download.")
                ResourceLoader._model = SentenceTransformer("all-MiniLM-L6-
v2", device='cpu')
            else:
                ResourceLoader._model = SentenceTransformer(model_path,
device='cpu')
            logger.info("NLP model loaded from cache successfully!")

            logger.info("NLP model initialized!")
            return ResourceLoader._model

# Allowed file extensions for evidence
ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg', 'pdf', 'txt'}

def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in
ALLOWED_EXTENSIONS

# Models
class Evidence(db.Model):
    __tablename__ = 'evidence'
    evidence_id = db.Column(db.Integer, primary_key=True)
    complaint_id = db.Column(db.Integer, nullable=False)
    police_id = db.Column(db.Integer, db.ForeignKey('police.badge_id'),
nullable=True)
    lawyer_id = db.Column(db.String(20), db.ForeignKey('lawyers.lawyer_id'),
nullable=True)
    submitter_type = db.Column(db.String(10), nullable=False,
default='police')
    file_path = db.Column(db.String(200), nullable=False)
    upload_date = db.Column(db.DateTime, default=db.func.current_timestamp())

class Account(db.Model):
    __tablename__ = 'accounts'
    account_id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(50), unique=True, nullable=False)
    email = db.Column(db.String(100), unique=True, nullable=True)
    phoneno = db.Column(db.String(15), nullable=False)
    password_hash = db.Column(db.String(128), nullable=False)
    civilian = db.relationship('Civilian', backref='account', uselist=False)
    lawyer = db.relationship('Lawyer', backref='account', uselist=False)

```

```

    police = db.relationship('Police', backref='account', uselist=False)

class Civilian(db.Model):
    __tablename__ = 'civilians'
    civilian_id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(50), unique=True, nullable=False)
    account_id = db.Column(db.Integer, db.ForeignKey('accounts.account_id'),
nullable=False)
    cases = db.relationship('Case', backref='civilian')

class Lawyer(db.Model):
    __tablename__ = 'lawyers'
    lawyer_id = db.Column(db.String(20), primary_key=True)
    account_id = db.Column(db.Integer, db.ForeignKey('accounts.account_id'),
nullable=False)
    case = db.relationship('Case', backref='lawyer', uselist=False)
    court_location_id = db.Column(db.Integer,
db.ForeignKey('court_locations.court_id'))

class Police(db.Model):
    __tablename__ = 'police'
    badge_id = db.Column(db.Integer, primary_key=True)
    account_id = db.Column(db.Integer, db.ForeignKey('accounts.account_id'),
nullable=False)
    police_station_id = db.Column(db.Integer,
db.ForeignKey('police_stations.ps_id'))
    case_assignments = db.relationship('CaseAssign', backref='police')
    case_solved = db.relationship('CaseSolved', backref='police')

class LawyerInfo(db.Model):
    __tablename__ = 'lawyer_info'
    info_id = db.Column(db.Integer, primary_key=True)
    bar_id = db.Column(db.String(20), db.ForeignKey('lawyers.lawyer_id'),
nullable=False)
    branch_name = db.Column(db.String(100))
    state = db.Column(db.String(100))
    court_location = db.Column(db.String(100))
    judiciary = db.Column(db.String(100))
    judiciary_id = db.Column(db.String(50))
    lawyer = db.relationship('Lawyer', backref='info', uselist=False)

class PoliceInfo(db.Model):
    __tablename__ = 'police_station_info'
    id = db.Column(db.Integer, primary_key=True)
    state = db.Column(db.String(100), nullable=False)
    pin_code = db.Column(db.Integer, nullable=False)
    station_number = db.Column(db.Integer, nullable=False)
    station_location = db.Column(db.String(200), nullable=False)

```



```

    police_id = db.Column(db.String(20), nullable=False)

class Case(db.Model):
    __tablename__ = 'cases'
    case_id = db.Column(db.Integer, db.ForeignKey('incidents.incident_id'),
primary_key=True)
    title = db.Column(db.String(100), nullable=False)
    description = db.Column(db.Text, nullable=False)
    civilian_id = db.Column(db.Integer,
db.ForeignKey('civilians.civilian_id'))
    lawyer_id = db.Column(db.String(20), db.ForeignKey('lawyers.lawyer_id'))

class CaseAssign(db.Model):
    __tablename__ = 'case_assign'
    assign_id = db.Column(db.Integer, primary_key=True)
    case_id = db.Column(db.Integer, db.ForeignKey('cases.case_id'))
    police_id = db.Column(db.String(20), db.ForeignKey('police.badge_id'))

class CaseSolved(db.Model):
    __tablename__ = 'case_solved'
    solved_id = db.Column(db.Integer, primary_key=True)
    case_id = db.Column(db.Integer, db.ForeignKey('cases.case_id'))
    police_id = db.Column(db.String(20), db.ForeignKey('police.badge_id'))

class PoliceStation(db.Model):
    __tablename__ = 'police_stations'
    ps_id = db.Column(db.Integer, primary_key=True)
    station_name = db.Column(db.String(100), nullable=False)
    police = db.relationship('Police', backref='station', uselist=False)
    master_id = db.Column(db.Integer, db.ForeignKey('police_master.pm_id'))

class PoliceMaster(db.Model):
    __tablename__ = 'police_master'
    pm_id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    stations = db.relationship('PoliceStation', backref='master')

class CourtLocation(db.Model):
    __tablename__ = 'court_locations'
    court_id = db.Column(db.Integer, primary_key=True)
    location = db.Column(db.String(100), nullable=False)
    master_id = db.Column(db.Integer, db.ForeignKey('lawyer_master.lm_id'))
    lawyers = db.relationship('Lawyer', backref='court_location')

class LawyerMaster(db.Model):
    __tablename__ = 'lawyer_master'
    lm_id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)

```

```

    courts = db.relationship('CourtLocation', backref='master')

class Incident(db.Model):
    __tablename__ = 'incidents'
    incident_id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100))
    email = db.Column(db.String(100))
    phone = db.Column(db.String(15))
    description = db.Column(db.Text, nullable=False)
    location = db.Column(db.String(100), nullable=False)
    address = db.Column(db.String(200))
    incident_date = db.Column(db.String(20), nullable=False)
    timestamp = db.Column(db.String(50))

class Support(db.Model):
    __tablename__ = 'support'
    support_id = db.Column(db.Integer, primary_key=True)
    message = db.Column(db.Text, nullable=False)
    account_id = db.Column(db.Integer, db.ForeignKey('accounts.account_id'),
nullable=True)

# Routes
@app.route('/api/evidence', methods=['POST'])
def submit_evidence():
    if 'evidenceFile' not in request.files:
        return jsonify({'message': 'No file part in the request'}), 400

    file = request.files['evidenceFile']
    complaint_id = request.form.get('complaintId')
    submitter_id = request.form.get('submitterId')
    submitter_type = request.form.get('submitterType')

    if not all([complaint_id, submitter_id, submitter_type, file]):
        return jsonify({'message': 'Complaint ID, Submitter ID, Submitter
Type, and evidence file are required'}), 400

    if file.filename == '':
        return jsonify({'message': 'No selected file'}), 400

    if not allowed_file(file.filename):
        return jsonify({'message': 'File type not allowed. Allowed types: png,
jpg, jpeg, pdf, txt'}), 400

    try:
        complaint_id = int(complaint_id)

        if submitter_type == 'police':
            police_id = int(submitter_id)

```

```

        police = db.session.get(Police, police_id)
        if not police:
            return jsonify({'message': 'Invalid Police ID'}), 404
        case_assignment = CaseAssign.query.filter_by(case_id=complaint_id,
police_id=police_id).first()
        if not case_assignment:
            return jsonify({'message': 'This case is not assigned to
you'}), 403
        lawyer_id = None
        elif submitter_type == 'lawyer':
            lawyer_id = submitter_id
            lawyer = db.session.get(Lawyer, lawyer_id)
            if not lawyer:
                return jsonify({'message': 'Invalid Lawyer ID'}), 404
            case = Case.query.filter_by(case_id=complaint_id,
lawyer_id=lawyer_id).first()
            if not case:
                return jsonify({'message': 'This case is not assigned to
you'}), 403
            police_id = None
        else:
            return jsonify({'message': 'Invalid submitter type'}), 400

    filename = secure_filename(file.filename)
    file_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
    file.save(file_path)

    evidence = Evidence(
        complaint_id=complaint_id,
        police_id=police_id,
        lawyer_id=lawyer_id,
        submitter_type=submitter_type,
        file_path=file_path
    )
    db.session.add(evidence)
    db.session.commit()

    return jsonify({
        'message': 'Evidence submitted successfully',
        'evidence': {
            'evidence_id': evidence.evidence_id,
            'complaint_id': evidence.complaint_id,
            'police_id': evidence.police_id,
            'lawyer_id': evidence.lawyer_id,
            'submitter_type': evidence.submitter_type,
            'file_path': evidence.file_path,
            'upload_date': evidence.upload_date.isoformat(),

```

```

        'details': f"File: {filename}, Case ID:
{evidence.complaint_id}, Submitted by: {submitter_type.capitalize()}"
    }
    )), 201

except ValueError:
    return jsonify({'message': 'Complaint ID must be numeric'}), 400
except Exception as e:
    db.session.rollback()
    return jsonify({'message': f'Error submitting evidence: {str(e)}'}),
500

@app.route('/api/evidence/case/<int:case_id>', methods=['GET'])
def get_evidence_by_case(case_id):
    try:
        evidence_list = Evidence.query.filter_by(complaint_id=case_id).all()
        evidence_data = [{
            'evidence_id': evidence.evidence_id,
            'complaint_id': evidence.complaint_id,
            'police_id': evidence.police_id,
            'lawyer_id': evidence.lawyer_id,
            'submitter_type': evidence.submitter_type,
            'file_path': evidence.file_path,
            'upload_date': evidence.upload_date.isoformat(),
            'details': f"File: {os.path.basename(evidence.file_path)}, Case
ID: {evidence.complaint_id}, Submitted by:
{evidence.submitter_type.capitalize()}"
        } for evidence in evidence_list]
        return jsonify(evidence_data), 200
    except Exception as e:
        return jsonify({'message': f'Error fetching evidence: {str(e)}'}), 500

@app.route('/api/lawyer/cases/<string:lawyer_id>', methods=['GET'])
def get_lawyer_cases(lawyer_id):
    try:
        assigned_cases = Case.query.filter_by(lawyer_id=lawyer_id).all()
        assigned_cases_data = [{
            'case_id': case.case_id,
            'title': case.title,
            'description': case.description,
            'lawyer_id': case.lawyer_id
        } for case in assigned_cases]
        return jsonify({'assignedCases': assigned_cases_data}), 200
    except Exception as e:
        return jsonify({'message': f'Error fetching cases: {str(e)}'}), 500

@app.route("/api/civilian/login", methods=["POST"])
def login():

```

```

data = request.json
username = data.get('idOrUsername')
password = data.get('password')
if not username or not password:
    return jsonify({"message": "Username and Password required"}), 400
account = Account.query.filter_by(username=username).first()
if not account or not check_password_hash(account.password_hash,
password):
    return jsonify({"message": "Invalid credentials"}), 401
civilian = Civilian.query.filter_by(account_id=account.account_id).first()
if not civilian:
    return jsonify({"message": "No civilian profile found"}), 404
return jsonify({
    "message": "Login successful",
    "account_id": account.account_id,
    "civilian_id": civilian.civilian_id,
    "userType": "Civilian"
}), 200

@app.route("/api/lawyer/login", methods=["POST"])
def lawyer_login():
    data = request.json
    lawyer_id = data.get('idOrUsername')
    password = data.get('password')
    if not lawyer_id or not password:
        return jsonify({"message": "Lawyer ID and Password required"}), 400
    lawyer = Lawyer.query.filter_by(lawyer_id=lawyer_id).first()
    if not lawyer:
        return jsonify({"message": "No lawyer profile found"}), 404
    account = Account.query.filter_by(account_id=lawyer.account_id).first()
    if not account or not check_password_hash(account.password_hash,
password):
        return jsonify({"message": "Invalid credentials"}), 401
    return jsonify({
        "message": "Lawyer login successful",
        "userType": "Lawyer"
    }), 200

@app.route('/api/police/cases/<int:badge_id>', methods=['GET'])
def get_police_cases(badge_id):
    try:
        assigned_cases = db.session.query(Case, CaseAssign).\
            join(CaseAssign, Case.case_id == CaseAssign.case_id).\
            filter(CaseAssign.police_id == badge_id).all()
        resolved_cases = db.session.query(Case, CaseSolved).\
            join(CaseSolved, Case.case_id == CaseSolved.case_id).\
            filter(CaseSolved.police_id == badge_id).all()

```

```

        assigned_cases_data = [{
            'case_id': case.Case.case_id,
            'title': case.Case.title,
            'description': case.Case.description,
            'lawyer_id': case.Case.lawyer_id
        } for case in assigned_cases]

        resolved_cases_data = [{
            'case_id': case.Case.case_id,
            'title': case.Case.title,
            'description': case.Case.description,
            'lawyer_id': case.Case.lawyer_id
        } for case in resolved_cases]

        return jsonify({
            'assignedCases': assigned_cases_data,
            'resolvedCases': resolved_cases_data
        }), 200
    except Exception as e:
        return jsonify({'message': f'Error fetching cases: {str(e)}'}), 500

@app.route("/api/police/complaint", methods=["POST"])
def register_police_complaint():
    data = request.json
    badge_id = data.get('badge_id')
    if not badge_id:
        return jsonify({'message': 'Police badge ID required'}), 400
    police = Police.query.filter_by(badge_id=badge_id).first()
    if not police:
        return jsonify({'message': 'Invalid police ID'}), 401

    name = data.get('name')
    email = data.get('email')
    phone = data.get('phone')
    description = data.get('description')
    location = data.get('location')
    address = data.get('address')
    incident_date = data.get('incidentDate')
    timestamp = data.get('timestamp')

    if not all([description, location, incident_date]):
        return jsonify({'message': 'Description, location, and incident date
are required'}), 400

    try:
        incident = Incident(
            name=name, email=email, phone=phone, description=description,
            location=location, address=address, incident_date=incident_date,

```

```

        timestamp=timestamp
    )
    db.session.add(incident)
    db.session.flush()

    lawyers = Lawyer.query.all()
    if not lawyers:
        return jsonify({'message': 'No lawyers available to assign'}), 500
    random_lawyer = choice(lawyers)

    case = Case(
        case_id=incident.incident_id,
        title=f"Case: {name or 'Unnamed'} - {incident_date}",
        description=description,
        lawyer_id=random_lawyer.lawyer_id
    )
    db.session.add(case)
    db.session.flush()

    case_assign = CaseAssign(case_id=case.case_id, police_id=badge_id)
    db.session.add(case_assign)
    db.session.commit()

    return jsonify({
        'message': 'Complaint registered and case assigned successfully',
        'incident_id': incident.incident_id,
        'case_id': case.case_id,
        'lawyer_id': random_lawyer.lawyer_id
    }), 201
except Exception as e:
    db.session.rollback()
    return jsonify({'message': f'Error registering complaint: {str(e)}'}),
500

@app.route("/api/police/login", methods=["POST"])
def police_login():
    data = request.json
    badge_id = data.get('idOrUsername')
    password = data.get('password')
    if not badge_id or not password:
        return jsonify({"message": "Badge ID and Password required"}), 400
    police = Police.query.filter_by(badge_id=badge_id).first()
    if not police:
        return jsonify({"message": "No police profile found"}), 404
    account = Account.query.filter_by(account_id=police.account_id).first()
    if not account or not check_password_hash(account.password_hash,
password):
        return jsonify({"message": "Invalid credentials"}), 401

```

```

        return jsonify({
            "message": "Police login successful",
            "userType": "Police",
            "badge_id": str(police.badge_id)
        }), 200

@app.route("/api/civilian/signup", methods=["POST"])
def signup():
    data = request.json
    username = data.get('username')
    phoneno = data.get('phoneno')
    password = data.get('password')
    if not username or not password:
        return jsonify({'message': 'Username and Password required'}), 400
    existing_user = Account.query.filter_by(username=username).first()
    if existing_user:
        return jsonify({'message': 'User with same username exists!'}), 409
    hashed_password = generate_password_hash(password)
    account = Account(username=username, phoneno=phoneno,
password_hash=hashed_password)
    db.session.add(account)
    db.session.commit()
    civilian = Civilian(username=username, account_id=account.account_id)
    db.session.add(civilian)
    db.session.commit()
    return jsonify({"message": "Signup successful"}), 201

@app.route("/api/lawyer/signup", methods=["POST"])
def lawyer_signup():
    data = request.json
    lawyer_id = data.get('id')
    phoneno = data.get('phoneno')
    password = data.get('password')
    email = data.get('email')

    if not all([lawyer_id, password, email, phoneno]):
        return jsonify({'message': 'All fields (Bar ID, Email, Phone Number,
and Password) are required'}), 400

    if not phoneno.isdigit() or len(phoneno) != 10:
        return jsonify({'message': 'Phone number must be a 10-digit number'}),
400

    if '@' not in email or '.' not in email:
        return jsonify({'message': 'Please enter a valid email address'}), 400

    existing_email = Account.query.filter_by(email=email).first()
    if existing_email:

```



```

        return jsonify({'message': 'An account with this email already
exists'}), 409

    existing_account = Account.query.filter_by(username=lawyer_id).first()
    if existing_account:
        return jsonify({'message': 'This Bar ID is already registered'}), 409

    existing_lawyer = Lawyer.query.filter_by(lawyer_id=lawyer_id).first()
    if existing_lawyer:
        return jsonify({'message': 'This Bar ID is already registered as a
lawyer'}), 409

    try:
        hashed_password = generate_password_hash(password)
        account = Account(email=email, phoneno=phoneno,
password_hash=hashed_password, username=lawyer_id)
        db.session.add(account)
        db.session.commit()

        lawyer = Lawyer(lawyer_id=lawyer_id, account_id=account.account_id)
        db.session.add(lawyer)
        db.session.commit()

        return jsonify({"message": "Lawyer signup successful"}), 201
    except Exception as e:
        db.session.rollback()
        return jsonify({'message': f'Error during signup: {str(e)}'}), 500

@app.route("/api/police/signup", methods=["POST"])
def police_signup():
    data = request.json
    badge_id = data.get('id')
    phoneno = data.get('phoneno')
    password = data.get('password')
    email = data.get('email')

    if not all([badge_id, password, email, phoneno]):
        return jsonify({'message': 'All fields (Badge ID, Email, Phone Number,
and Password) are required'}), 400

    try:
        badge_id = int(badge_id)
        if badge_id <= 0:
            return jsonify({'message': 'Badge ID must be a positive number'}),
400
    except ValueError:
        return jsonify({'message': 'Badge ID must be a valid number'}), 400

```

```

existing_email = Account.query.filter_by(email=email).first()
if existing_email:
    return jsonify({'message': 'An account with this email already
exists'}), 409

existing_account = Account.query.filter_by(username=str(badge_id)).first()
if existing_account:
    return jsonify({'message': 'This Badge ID is already registered'}),
409

existing_police = Police.query.filter_by(badge_id=badge_id).first()
if existing_police:
    return jsonify({'message': 'This Badge ID is already registered as a
police officer'}), 409

try:
    hashed_password = generate_password_hash(password)
    account = Account(email=email, phoneno=phoneno,
password_hash=hashed_password, username=str(badge_id))
    db.session.add(account)
    db.session.commit()

    police = Police(badge_id=badge_id, account_id=account.account_id)
    db.session.add(police)
    db.session.commit()

    return jsonify({"message": "Police signup successful"}), 201
except Exception as e:
    db.session.rollback()
    return jsonify({'message': f'Error during signup: {str(e)}'}), 500

@app.route('/api/lawyerInfo', methods=['POST'])
def add_lawyer_info():
    data = request.json
    bar_id = data.get('barId')
    branch_name = data.get('branchName')
    state = data.get('state')
    court_location = data.get('courtLocation')
    judiciary = data.get('judiciary')
    judiciary_id = data.get('judiciaryId')

    if not all([bar_id, state]):
        return jsonify({'message': 'Bar ID and State are required', 'success':
False}), 400

    lawyer = Lawyer.query.filter_by(lawyer_id=bar_id).first()
    if not lawyer:

```

```

        return jsonify({'message': 'Lawyer not found. Please sign up first.',
'success': False}), 404

    try:
        existing_branch = LawyerInfo.query.filter_by(bar_id=bar_id,
branch_name=branch_name).first()
        if existing_branch:
            return jsonify({'message': 'Court Branch already registered for
this lawyer', 'success': False}), 400

        table_name = f"lawyer_info_{branch_name.replace(' ', '_').lower()}"
        metadata = MetaData()

        inspector = inspect(db.engine)
        if not inspector.has_table(table_name):
            new_table = Table(
                table_name, metadata,
                Column('info_id', Integer, primary_key=True),
                Column('bar_id', String(20), nullable=False),
                Column('branch_name', String(100)),
                Column('state', String(100)),
                Column('court_location', String(100)),
                Column('judiciary', String(100)),
                Column('judiciary_id', String(50))
            )
            metadata.create_all(db.engine)
            print(f"Created new table: {table_name}")

            with db.engine.connect() as connection:
                insert_stmt = text(
                    f"INSERT INTO {table_name} (bar_id, branch_name, state,
court_location, judiciary, judiciary_id) "
                    f"VALUES (:bar_id, :branch_name, :state, :court_location,
:judiciary, :judiciary_id)"
                )
                connection.execute(
                    insert_stmt,
                    {"bar_id": bar_id, "branch_name": branch_name, "state": state,
                    "court_location": court_location, "judiciary": judiciary,
                    "judiciary_id": judiciary_id}
                )
                connection.commit()

            return jsonify({'message': f'Court Registered Successfully in
{table_name}', 'success': True}), 201
        except Exception as e:
            db.session.rollback()

```

```

        return jsonify({'message': f'Error adding lawyer info: {str(e)}',
'success': False}), 500

@app.route('/api/policeInfo', methods=['POST'])
def add_police_info():
    data = request.json
    state = data.get('state')
    pin_code = data.get('pinCode')
    station_number = data.get('stationNumber')
    station_location = data.get('stationLocation')
    police_id = data.get('policeId')

    if not all([state, pin_code, station_number, station_location,
police_id]):
        return jsonify({'message': 'All fields are required', 'success':
False}), 400

    try:
        pin_code = int(pin_code)
        station_number = int(station_number)
    except (ValueError, TypeError):
        return jsonify({'message': 'Pin Code and Station Number must be
numeric', 'success': False}), 400

    if not isinstance(state, str) or not isinstance(station_location, str) or
not isinstance(police_id, str):
        return jsonify({'message': 'State, Station Location, and Police ID
must be strings', 'success': False}), 400

    police = Police.query.filter_by(badge_id=police_id).first()
    if not police:
        return jsonify({'message': 'Police not found', 'success': False}), 404

    try:
        existing_station =
PoliceInfo.query.filter_by(station_number=station_number).first()
        if existing_station:
            return jsonify({'message': 'Station number already registered',
'success': False}), 400

        table_name = f"police_station_{station_number}"
        metadata = MetaData()

        inspector = inspect(db.engine)
        if not inspector.has_table(table_name):
            new_table = Table(
                table_name, metadata,
                Column('id', Integer, primary_key=True),

```

```

        Column('state', String(100), nullable=False),
        Column('pin_code', Integer, nullable=False),
        Column('station_number', Integer, nullable=False),
        Column('station_location', String(200), nullable=False),
        Column('police_id', String(20), nullable=False)
    )
    metadata.create_all(db.engine)
    print(f"Created new table: {table_name}")

    with db.engine.connect() as connection:
        insert_stmt = text(
            f"INSERT INTO {table_name} (state, pin_code, station_number,
station_location, police_id) "
            f"VALUES (:state, :pin_code, :station_number,
:station_location, :police_id)"
        )
        connection.execute(
            insert_stmt,
            {"state": state, "pin_code": pin_code, "station_number":
station_number,
            "station_location": station_location, "police_id": police_id}
        )
        connection.commit()

    return jsonify({'message': f'Police-Station info registered
successfully in {table_name}', 'success': True}), 201
    except Exception as e:
        db.session.rollback()
        return jsonify({'message': f'Error adding police-station info:
{str(e)}', 'success': False}), 500

@app.route("/api/support", methods=["POST"])
def help_support():
    try:
        data = request.json
        if not data:
            return jsonify({'message': 'No data provided', 'success': False}),
400

        message = data.get('question')
        if not message:
            return jsonify({'message': 'Question is required', 'success':
False}), 400
        account_id = data.get('account_id')
        info = Support(message=message, account_id=account_id)
        db.session.add(info)
        db.session.commit()
        return jsonify({
            'message': 'Question added successfully',

```

```

        'success': True,
        'support_id': info.support_id
    }), 201
except db.exc.IntegrityError as e:
    db.session.rollback()
    return jsonify({'message': 'Database integrity error', 'success':
False, 'error': str(e)}), 400
except Exception as e:
    db.session.rollback()
    return jsonify({'message': 'Error processing request', 'success':
False, 'error': str(e)}), 500

# NLP Routes
def get_relevant_sections(query, top_n=5):
    df = ResourceLoader.get_df()
    embeddings = ResourceLoader.get_embeddings()
    model = ResourceLoader.get_model()

    if not query.strip():
        return df[["Section", "Offense",
"Punishment"]].to_dict(orient="records")

    try:
        query_embedding = model.encode(query, convert_to_tensor=True)
        similarity_scores = util.pytorch_cos_sim(query_embedding,
embeddings).squeeze()
        top_indices = similarity_scores.argsort(descending=True)[:top_n]
        results = df.iloc[top_indices][["Section", "Offense",
"Punishment"]].copy()
        results = results[~results["Offense"].str.startswith("IPC Section")]
        return results.to_dict(orient="records")
    except Exception as e:
        print(f"Error in get_relevant_sections: {str(e)}")
        return []

@app.route("/nlp/search", methods=["POST"])
def search_sections():
    try:
        data = request.json
        query = data.get("query", "")
        print(f"Received search query: {query}")
        results = get_relevant_sections(query)
        print(f"Returning {len(results)} results")
        return jsonify(results), 200
    except Exception as e:
        print(f"Error in search_sections: {str(e)}")
        return jsonify({'error': str(e)}), 500

```

```

@app.route('/nlp/test', methods=['GET'])
def test():
    return jsonify({'message': 'Combined Backend with NLP is running!'}), 200

# Migration function
def migrate_evidence_table():
    with app.app_context():
        inspector = db.inspect(db.engine)
        if 'evidence' in inspector.get_table_names():
            columns = inspector.get_columns('evidence')
            police_id_nullable = any(col['name'] == 'police_id' and
col['nullable'] for col in columns)

            if not police_id_nullable:
                print("Migrating evidence table...")
                db.engine.execute(text("ALTER TABLE evidence RENAME TO
evidence_old"))
                db.create_all()
                migrate_sql = """
                INSERT INTO evidence (evidence_id, complaint_id, police_id,
lawyer_id, submitter_type, file_path, upload_date)
                SELECT evidence_id, complaint_id, police_id, lawyer_id,
                    COALESCE(submitter_type, 'police'), file_path,
upload_date
                FROM evidence_old
                """
                db.engine.execute(text(migrate_sql))
                db.engine.execute(text("DROP TABLE evidence_old"))
                print("Evidence table migrated successfully.")
            else:
                print("Evidence table schema is already up-to-date.")
        else:
            db.create_all()
            print("Created evidence table")

# Initialize database at startup
with app.app_context():
    db.create_all()
    migrate_evidence_table()

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000, debug=True)

```

## **SIMILARLY FOR FRONT-END MAIN COMPONENT: "HERO.JSX"**

```
import React, { useState, useEffect } from 'react';
// import SearchBar from './SearchBar';
import CrimeSearch from './CrimeSearch';
import '../styles/Components.css';

const Hero = () => {
  const [animatedText, setAnimatedText] = useState('');
  const message = 'Welcome to IPC Nexus';

  useEffect(() => {
    let i = 0;
    const interval = setInterval(() => {
      if (i <= message.length) {
        setAnimatedText(message.substring(0, i));
        i++;
      } else {
        clearInterval(interval);
      }
    }, 100);

    return () => clearInterval(interval);
  }, []);

  return (
    <div className="hero-container">
      <div className="hero-content">
        <div className="hero-text">
          <h1>{animatedText}</h1>
          <p className="hero-subtitle">Search Indian Penal Code Sections</p>
        </div>
        <div className="searchBar-container">
          <CrimeSearch />
        </div>
      </div>
    </div>
  );
};

export default Hero;
```



## NAVBAR.JSX:

```
import React, { useState, useEffect } from 'react';
import { Link } from 'react-router-dom';
import '../styles/Components.css';

const Navbar = ({ branches, userType }) => {
  const [isAuthenticated, setIsAuthenticated] =
    useState(localStorage.getItem('isAuthenticated') === 'true');
  const [username, setUsername] = useState(localStorage.getItem('username') ||
    'User');

  useEffect(() => {
    const handleStorageChange = () => {
      setIsAuthenticated(localStorage.getItem('isAuthenticated') === 'true');
      setUsername(localStorage.getItem('username') || 'User');
    };
    window.addEventListener('storage', handleStorageChange);
    return () => window.removeEventListener('storage', handleStorageChange);
  }, []);

  return (
    <nav className="navbar navbar-expand-lg navbar-dark bg-black shadow-sm">
      <div className="container">
        <Link className="navbar-brand fw-bold fs-4 text-info" to="/">
          IPC Nexus
        </Link>

        <button
          className="navbar-toggler"
          type="button"
          data-bs-toggle="collapse"
          data-bs-target="#navbarNav"
          aria-controls="navbarNav"
          aria-expanded="false"
          aria-label="Toggle navigation"
        >
          <span className="navbar-toggler-icon"></span>
        </button>

        <div className="collapse navbar-collapse" id="navbarNav">
          <ul className="navbar-nav me-auto">
            {/* Laws Dropdown - Visible for all */}
            <li className="nav-item dropdown">
              <Link
                className="nav-link dropdown-toggle text-white"
                to="#"
                id="branchesDropdown"
              >
```

```

        role="button"
        data-bs-toggle="dropdown"
        aria-expanded="false"
        style={{ transition: 'color 0.3s ease' }}
        onMouseEnter={(e) => (e.target.style.color = 'lightblue')}
        onMouseLeave={(e) => (e.target.style.color = 'white')}
      >
        Laws
      </Link>
      <ul className="dropdown-menu bg-black border-0" aria-
labelledby="branchesDropdown">
        {branches.map((branch, index) => (
          <li key={index}>
            <Link
              className="dropdown-item text-white"
              to={`/${branch.toLowerCase().replace(/\s/g, '-')}`}
              style={{ transition: 'color 0.3s ease' }}
              onMouseEnter={(e) => (e.target.style.color =
'lightblue')}}
              onMouseLeave={(e) => (e.target.style.color = 'white')}}
            >
              {branch}
            </Link>
          </li>
        ))}
      </ul>
    </li>

    { /* About Us - Visible for all */}
    <li className="nav-item">
      <Link className="nav-link text-white" to="/about">
        About Us
      </Link>
    </li>

    { /* Civilian Links */}
    {userType === 'Civilian' && isAuthenticated && (
      <>

        <li className="nav-item">
          <Link className="nav-link text-white" to="/IPCSections">
            IPC Sections
          </Link>
        </li>

      </>
    )}
  )}

```

```

    { /* Lawyer Links */
    {userType === 'Lawyer' && isAuthenticated && (
        <>

            <li className="nav-item">
                <Link className="nav-link text-white" to="/IPCSections">
                    IPC Sections
                </Link>
            </li>

            <li className="nav-item">
                <Link className="nav-link text-white" to="/caseInfo">
                    Case Info
                </Link>
            </li>
            <li className="nav-item">
                <Link className="nav-link text-white" to="/lawyerinfo">
                    Lawyer Info
                </Link>
            </li>
            <li className="nav-item">
                <Link className="nav-link text-white" to="/EvidenceReport">
                    Evidence Report
                </Link>
            </li>

        </>
    )}

    { /* Police Links */
    {userType === 'Police' && isAuthenticated && (
        <>

            <li className="nav-item">
                <Link className="nav-link text-white" to="/register-
complaint">
                    Register Complaint
                </Link>
            </li>
            <li className="nav-item">
                <Link className="nav-link text-white" to="/IPCSections">
                    IPC Sections
                </Link>
            </li>

            <li className="nav-item">
                <Link className="nav-link text-white" to="/EvidenceReport">

```

```

        Evidence Report
      </Link>
    </li>
    <li className="nav-item">
      <Link className="nav-link text-white" to="/caseInfo">
        Case Info
      </Link>
    </li>
    <li className="nav-item">
      <Link className="nav-link text-white" to="/policeinfo">
        Police Info
      </Link>
    </li>
  </ul>

  </>
)}
</ul>

{isAuthenticated ? (
  <div className="d-flex align-items-center">
    <span className="text-white me-3">Hi, {username}</span>
    <button
      className="btn btn-outline-danger"
      onClick={() => {
        localStorage.removeItem('isAuthenticated');
        localStorage.removeItem('userType');
        localStorage.removeItem('username');
        localStorage.removeItem('badge_id');
        window.location.href = '/login/civilian';
      }}
    >
      Logout
    </button>
  </div>
) : (
  <>
    <div className="dropdown">
      <button
        className="btn btn-outline-info dropdown-toggle me-2"
        type="button"
        id="loginDropdown"
        data-bs-toggle="dropdown"
        aria-expanded="false"
      >
        Login
      </button>
      <ul className="dropdown-menu dropdown-menu-end bg-black
border-0" aria-labelledby="loginDropdown">

```

```

        <li><Link className="dropdown-item text-white"
to="/login/civilian">Civilian Login</Link></li>
        <li><Link className="dropdown-item text-white"
to="/login/lawyer">Lawyer Login</Link></li>
        <li><Link className="dropdown-item text-white"
to="/login/police">Police Login</Link></li>
    </ul>
</div>
<div className="dropdown">
    <button
        className="btn btn-outline-info dropdown-toggle me-2"
        type="button"
        id="signupDropdown"
        data-bs-toggle="dropdown"
        aria-expanded="false"
    >
        Signup
    </button>
    <ul className="dropdown-menu dropdown-menu-end bg-black
border-0" aria-labelledby="signupDropdown">
        <li><Link className="dropdown-item text-white"
to="/signup/civilian">Civilian Signup</Link></li>
        <li><Link className="dropdown-item text-white"
to="/signup/lawyer">Lawyer Signup</Link></li>
        <li><Link className="dropdown-item text-white"
to="/signup/police">Police Signup</Link></li>
    </ul>
</div>
</>
    )}
</div>
</div>
</nav>
);
};

export default Navbar;

```

## POLICEINFO.JSX

```
import React, { useState } from 'react';
import axios from 'axios';
import '../styles/LoginInfo.css';
import states from '../states';

const PoliceInfo = () => {
  const [formData, setFormData] = useState({
    state: '',
    pinCode: '',
    stationNumber: '',
    stationLocation: '',
    policeId: ''
  });

  const [errors, setErrors] = useState({});
  const [message, setMessage] = useState('');

  const validate = () => {
    let tempErrors = {};

    if (!formData.state.trim()) tempErrors.state = 'State is required';
    if (!formData.stationLocation.trim()) tempErrors.stationLocation =
'Station Location is required';
    if (!formData.pinCode) tempErrors.pinCode = 'Pin Code is required';
    else if (!/^\\d+$/\\.test(formData.pinCode)) tempErrors.pinCode = 'Pin Code
must be a number';
    if (!formData.stationNumber) tempErrors.stationNumber = 'Station Number is
required';
    else if (!/^\\d+$/\\.test(formData.stationNumber)) tempErrors.stationNumber =
'Station Number must be a number';
    if (!formData.policeId) tempErrors.policeId = 'Police ID is required';
    else if (!/^\\d+$/\\.test(formData.policeId)) tempErrors.policeId = 'Police
ID must be a number';

    setErrors(tempErrors);
    return Object.keys(tempErrors).length === 0;
  };

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

  const handleSubmit = async (e) => {
    e.preventDefault();
    setMessage('');
```

```

    if (validate()) {
      try {
        const response = await
axios.post('http://localhost:5000/api/policeInfo', formData);
        setMessage(response.data.message || 'Data submitted successfully!');
        setFormData({
          state: '',
          pinCode: '',
          stationNumber: '',
          stationLocation: '',
          policeId: '',
        });
        setErrors({});
      } catch (error) {
        setMessage(error.response?.data?.message || 'Submission failed.');
      }
    }
  };

  return (
    <section className="vh-100 bg-custom">
      <div className="container py-5 h-100">
        <div className="row d-flex justify-content-center align-items-center
h-100">
          <div className="col-12 col-md-8 col-lg-6 col-xl-5">
            <div className="card bg-dark text-white" style={{ borderRadius:
'1rem' }}>
              <div className="card-body p-5 text-center">
                <h2 className="fw-bold mb-2 text-uppercase">Registered Police-
Station Information</h2>
                <p className="text-white-50 mb-5">Please enter your details
below</p>

                <form onSubmit={handleSubmit}>
                  <div className="form-outline form-white mb-3">
                    <select
                      id="state"
                      className="form-control form-control-lg"
                      name="state"
                      value={formData.state}
                      onChange={handleChange}
                    >
                      <option value="">Select State</option>
                      {states.map((state, index) => (
                        <option key={index} value={state}>
                          {state}
                        </option>
                      ))}
                    </select>
                  </div>
                </form>
              </div>
            </div>
          </div>
        </div>
      </div>
    </section>
  );

```

```

        </select>
        {errors.state && <small className="text-
danger">{errors.state}</small>}
      </div>

      <div className="form-outline form-white mb-3">
        <input
          type="text"
          id="pinCode"
          className="form-control form-control-lg"
          name="pinCode"
          value={formData.pinCode}
          onChange={handleChange}
          placeholder="Pin Code"
        />
        {errors.pinCode && <small className="text-
danger">{errors.pinCode}</small>}
      </div>

      <div className="form-outline form-white mb-3">
        <input
          type="text"
          id="stationNumber"
          className="form-control form-control-lg"
          name="stationNumber"
          value={formData.stationNumber}
          onChange={handleChange}
          placeholder="Station Number"
        />
        {errors.stationNumber && <small className="text-
danger">{errors.stationNumber}</small>}
      </div>

      <div className="form-outline form-white mb-3">
        <input
          type="text"
          id="stationLocation"
          className="form-control form-control-lg"
          name="stationLocation"
          value={formData.stationLocation}
          onChange={handleChange}
          placeholder="Station Location"
        />
        {errors.stationLocation && <small className="text-
danger">{errors.stationLocation}</small>}
      </div>

      <div className="form-outline form-white mb-3">

```



```

        <input
          type="text"
          id="policeId"
          className="form-control form-control-lg"
          name="policeId"
          value={formData.policeId}
          onChange={handleChange}
          placeholder="Police ID"
        />
        {errors.policeId && <small className="text-
danger">{errors.policeId}</small>}
      </div>

      <button className="btn btn-outline-light btn-lg px-5"
type="submit">
        Submit
      </button>
    </form>

    {message && (
      <p className="mt-3">
        <span className={message.includes('successfully') ? 'text-
success' : 'text-danger'}>
          {message}
        </span>
      </p>
    )}
  </div>
</div>
</div>
</div>
</div>
</section>
);
};

export default PoliceInfo;

```

## SCRIPT TO CREATE THIS WHOLE PROJECT FROM SCRATCH:

```
#!/bin/bash

# Colors for output
RED='\033[0;31m'
GREEN='\033[0;32m'
YELLOW='\033[1;33m'
NC='\033[0m' # No Color

# Function to print status messages
print_status() {
    echo -e "${GREEN}[*] $1${NC}"
}

print_error() {
    echo -e "${RED}[!] $1${NC}"
}

print_warning() {
    echo -e "${YELLOW}[!] $1${NC}"
}

# Check if Docker is installed
if ! command -v docker &> /dev/null; then
    print_error "Docker is not installed. Please install Docker first."
    exit 1
fi

# Check if Docker Compose is installed
if ! command -v docker-compose &> /dev/null; then
    print_error "Docker Compose is not installed. Please install Docker Compose first."
    exit 1
fi

# Check if kubectl is installed
if ! command -v kubectl &> /dev/null; then
    print_error "kubectl is not installed. Please install kubectl first."
    exit 1
fi

# Create project structure
print_status "Creating project structure..."
mkdir -p IPC_NEXUS/{Frontend,Backend,helpSupportbot,.qodo,k8s}
cd IPC_NEXUS

# Backend Setup
```

```
print_status "Setting up Backend..."
cd Backend

# Create requirements.txt
cat > requirements.txt << EOL
flask==2.3.3
flask-cors==4.0.0
flask-sqlalchemy==3.0.5
pandas==2.0.3
sentence-transformers==2.2.2
torch==2.0.1
werkzeug==2.3.7
python-dotenv==0.19.0
gunicorn==21.2.0
numpy==1.24.3
huggingface_hub==0.8.1
EOL

# Create .env file for backend
cat > .env << EOL
FLASK_APP=app.py
FLASK_ENV=development
DATABASE_URL=sqlite:///law_enforcement.db
SECRET_KEY=your-secret-key-here
EOL

# Create app.py
cat > app.py << EOL
from flask import Flask, request, jsonify
from flask_cors import CORS
from flask_sqlalchemy import SQLAlchemy
from sentence_transformers import SentenceTransformer
import pandas as pd
import numpy as np
import os
from dotenv import load_dotenv

# Load environment variables
load_dotenv()

app = Flask(__name__)
CORS(app)

# Database configuration
app.config['SQLALCHEMY_DATABASE_URI'] = os.getenv('DATABASE_URL')
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db = SQLAlchemy(app)
```

```

# Initialize the sentence transformer model
model = SentenceTransformer('all-MiniLM-L6-v2')

# Define database models
class Case(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(200), nullable=False)
    description = db.Column(db.Text, nullable=False)
    status = db.Column(db.String(50), default='open')
    created_at = db.Column(db.DateTime, server_default=db.func.now())

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

@app.route('/api/cases', methods=['GET'])
def get_cases():
    cases = Case.query.all()
    return jsonify([
        {
            'id': case.id,
            'title': case.title,
            'description': case.description,
            'status': case.status,
            'created_at': case.created_at.isoformat()
        } for case in cases])

@app.route('/api/cases', methods=['POST'])
def create_case():
    data = request.json
    new_case = Case(
        title=data['title'],
        description=data['description'],
        status=data.get('status', 'open')
    )
    db.session.add(new_case)
    db.session.commit()
    return jsonify({
        'id': new_case.id,
        'title': new_case.title,
        'description': new_case.description,
        'status': new_case.status,
        'created_at': new_case.created_at.isoformat()
    }), 201

@app.route('/api/analyze', methods=['POST'])
def analyze_text():
    data = request.json
    text = data.get('text', '')

```

```

# Generate embeddings
embedding = model.encode(text)

# Perform similarity analysis (example)
similarity_score = float(np.mean(embedding))

return jsonify({
    'embedding': embedding.tolist(),
    'similarity_score': similarity_score
})

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
EOL

# Create Dockerfile for backend
cat > Dockerfile << EOL
FROM python:3.9-slim

WORKDIR /app

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

EXPOSE 5000

CMD ["gunicorn", "--bind", "0.0.0.0:5000", "app:app"]
EOL

# Frontend Setup
print_status "Setting up Frontend..."
cd ../Frontend

# Create package.json
cat > package.json << EOL
{
  "name": "mthree_project_frontend",
  "private": true,
  "version": "0.0.0",
  "type": "module",
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "lint": "eslint .",
    "preview": "vite preview"
  }
}

```

```

    },
    "dependencies": {
      "@fortawesome/fontawesome-free": "^6.7.2",
      "@popperjs/core": "^2.11.8",
      "axios": "^1.8.3",
      "bootstrap": "^5.3.3",
      "papaparse": "^5.5.2",
      "react": "^19.0.0",
      "react-bootstrap": "^2.10.9",
      "react-dom": "^19.0.0",
      "react-icons": "^5.5.0",
      "react-router-dom": "^7.3.0"
    },
    "devDependencies": {
      "@eslint/js": "^9.21.0",
      "@types/react": "^19.0.10",
      "@types/react-dom": "^19.0.4",
      "@vitejs/plugin-react": "^4.3.4",
      "autoprefixer": "^10.4.21",
      "eslint": "^9.21.0",
      "eslint-plugin-react-hooks": "^5.1.0",
      "eslint-plugin-react-refresh": "^0.4.19",
      "globals": "^15.15.0",
      "postcss": "^8.5.3",
      "tailwindcss": "^4.0.12",
      "vite": "^6.2.2"
    }
  }
}
EOL

```

# Create .env file for frontend

```
cat > .env << EOL
```

```
VITE_API_URL=http://localhost:5000
```

```
EOL
```

# Create vite.config.js

```
cat > vite.config.js << EOL
```

```
import { defineConfig } from 'vite';
```

```
import react from '@vitejs/plugin-react';
```

```
export default defineConfig({
```

```
  plugins: [react()],
```

```
  server: {
```

```
    port: 3000,
```

```
    host: true
```

```
  }
```

```
});
```

```
EOL
```

```
# Create eslint.config.js
cat > eslint.config.js << EOL
import js from '@eslint/js';
import globals from 'globals';
import reactHooks from 'eslint-plugin-react-hooks';
import reactRefresh from 'eslint-plugin-react-refresh';
```

```
export default [
  js.configs.recommended,
  {
    files: ['**/*.js,jsx,mjs,cjs,ts,tsx'],
    plugins: {
      'react-hooks': reactHooks,
      'react-refresh': reactRefresh,
    },
    languageOptions: {
      globals: {
        ...globals.browser,
        ...globals.es2021,
      },
      parserOptions: {
        ecmaVersion: 'latest',
        sourceType: 'module',
        ecmaFeatures: {
          jsx: true,
        },
      },
    },
    rules: {
      'react-hooks/rules-of-hooks': 'error',
      'react-hooks/exhaustive-deps': 'warn',
      'react-refresh/only-export-components': [
        'warn',
        { allowConstantExport: true },
      ],
    },
  },
];
EOL
```

```
# Create postcss.config.js
cat > postcss.config.js << EOL
export default {
  plugins: {
    tailwindcss: {},
    autoprefixer: {},
  },
};
```

```
}  
EOL  
  
# Create tailwind.config.js  
cat > tailwind.config.js << EOL  
/** @type {import('tailwindcss').Config} */  
export default {  
  content: [  
    "./index.html",  
    "./src/**/*..{js,ts,jsx,tsx}",  
  ],  
  theme: {  
    extend: {},  
  },  
  plugins: [],  
}
```

EOL

```
# Create public directory and add vite.svg  
mkdir -p public  
cat > public/vite.svg << EOL  
<svg xmlns="http://www.w3.org/2000/svg"  
xmlns:xlink="http://www.w3.org/1999/xlink" aria-hidden="true" role="img"  
class="iconify iconify:logos" width="31.88" height="32"  
preserveAspectRatio="xMidYMid meet" viewBox="0 0 256  
257"><defs><linearGradient id="IconifyId1813088fe1fbc01fb466" x1="-.828%"  
x2="57.636%" y1="7.652%" y2="78.411%"><stop offset="0%" stop-  
color="#41D1FF"></stop><stop offset="100%" stop-  
color="#BD34FE"></stop></linearGradient><linearGradient  
id="IconifyId1813088fe1fbc01fb467" x1="43.376%" x2="50.316%" y1="2.242%"  
y2="89.03%"><stop offset="0%" stop-color="#FFEA83"></stop><stop  
offset="8.333%" stop-color="#FFDD35"></stop><stop offset="100%" stop-  
color="#FFA800"></stop></linearGradient></defs><path  
fill="url(#IconifyId1813088fe1fbc01fb466)" d="M255.153 37.938L134.897  
252.976c-2.483 4.44-8.862 4.466-11.382.048L8.75 37.958c-2.746-4.814 1.371-  
10.646 6.827-9.671l120.385 21.517a6.537 6.537 0 0 0 2.322-.004l117.867-  
21.483c5.438-.991 9.574 4.796 6.877 9.62Z"></path><path  
fill="url(#IconifyId1813088fe1fbc01fb467)" d="M185.432.063L96.44 17.501a3.268  
3.268 0 0 0-2.634 3.014l-5.474 92.456a3.268 3.268 0 0 0 3.997 3.378l24.777-  
5.718c2.318-.535 4.413 1.507 3.936 3.838l-7.361 36.047c-.495 2.426 1.782 4.5  
4.151 3.781l15.304-4.649c2.372-.72 4.652 1.36 4.15 3.788l-11.698 56.621c-.732  
3.542 3.979 5.473 5.943 2.437l11.313-2.028l72.516-144.72c1.215-2.423-.88-5.186-  
3.54-4.672l-25.505 4.922c-2.396.462-4.435-1.77-3.759-4.114l16.646-57.705c.677-  
2.35-1.37-4.583-3.769-4.113Z"></path></svg>
```

EOL

```
# Create src directory and its contents  
mkdir -p src/components src/pages src/services src/utils
```



```

# Create main App component
cat > src/App.jsx << EOL
import React from 'react';
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import Navbar from './components/Navbar';
import Home from './pages/Home';
import CaseList from './pages/CaseList';
import CaseDetail from './pages/CaseDetail';
import Analysis from './pages/Analysis';
import 'bootstrap/dist/css/bootstrap.min.css';
import '@fortawesome/fontawesome-free/css/all.min.css';

function App() {
  return (
    <Router>
      <div className="App">
        <Navbar />
        <div className="container mt-4">
          <Routes>
            <Route path="/" element={<Home />} />
            <Route path="/cases" element={<CaseList />} />
            <Route path="/cases/:id" element={<CaseDetail />} />
            <Route path="/analysis" element={<Analysis />} />
          </Routes>
        </div>
      </div>
    </Router>
  );
}

export default App;
EOL

# Create Navbar component
cat > src/components/Navbar.jsx << EOL
import React from 'react';
import { Link } from 'react-router-dom';
import { Navbar as BootstrapNavbar, Nav, Container } from 'react-bootstrap';

function Navbar() {
  return (
    <BootstrapNavbar bg="dark" variant="dark" expand="lg">
      <Container>
        <BootstrapNavbar.Brand as={Link} to="/">IPC
Nexus</BootstrapNavbar.Brand>
        <BootstrapNavbar.Toggle aria-controls="basic-navbar-nav" />
        <BootstrapNavbar.Collapse id="basic-navbar-nav">

```

```

        <Nav className="me-auto">
          <Nav.Link as={Link} to="/">Home</Nav.Link>
          <Nav.Link as={Link} to="/cases">Cases</Nav.Link>
          <Nav.Link as={Link} to="/analysis">Analysis</Nav.Link>
        </Nav>
      </BootstrapNavbar.Collapse>
    </Container>
  </BootstrapNavbar>
);
}

export default Navbar;
EOL

# Create Home page
cat > src/pages/Home.jsx << EOL
import React from 'react';
import { Container, Row, Col, Card } from 'react-bootstrap';

function Home() {
  return (
    <Container>
      <Row className="mb-4">
        <Col>
          <h1>Welcome to IPC Nexus</h1>
          <p className="lead">Your comprehensive law enforcement analysis
platform</p>
        </Col>
      </Row>
      <Row>
        <Col md={4}>
          <Card className="mb-4">
            <Card.Body>
              <Card.Title>Case Management</Card.Title>
              <Card.Text>
                View and manage all your cases in one place.
              </Card.Text>
            </Card.Body>
          </Card>
        </Col>
        <Col md={4}>
          <Card className="mb-4">
            <Card.Body>
              <Card.Title>Analysis Tools</Card.Title>
              <Card.Text>
                Advanced analysis tools for case investigation.
              </Card.Text>
            </Card.Body>
          </Card>
        </Col>
      </Row>
    </Container>
  );
}

export default Home;
EOL

```

```

        </Card>
      </Col>
    <Col md={4}>
      <Card className="mb-4">
        <Card.Body>
          <Card.Title>Reports</Card.Title>
          <Card.Text>
            Generate detailed reports and insights.
          </Card.Text>
        </Card.Body>
      </Card>
    </Col>
  </Row>
</Container>
);
}

export default Home;
EOL

# Create CaseList page
cat > src/pages/CaseList.jsx << EOL
import React, { useState, useEffect } from 'react';
import { Table, Button, Container } from 'react-bootstrap';
import { Link } from 'react-router-dom';
import axios from 'axios';

function CaseList() {
  const [cases, setCases] = useState([]);

  useEffect(() => {
    const fetchCases = async () => {
      try {
        const response = await axios.get('http://localhost:5000/api/cases');
        setCases(response.data);
      } catch (error) {
        console.error('Error fetching cases:', error);
      }
    };
    fetchCases();
  }, []);

  return (
    <Container>
      <h2>Cases</h2>
      <Table striped bordered hover>
        <thead>
          <tr>

```

```

        <th>ID</th>
        <th>Title</th>
        <th>Status</th>
        <th>Actions</th>
      </tr>
    </thead>
    <tbody>
      {cases.map(case_ => (
        <tr key={case_.id}>
          <td>{case_.id}</td>
          <td>{case_.title}</td>
          <td>{case_.status}</td>
          <td>
            <Button
              as={Link}
              to={` /cases/${case_.id}`}
              variant="primary"
              size="sm"
            >
              View
            </Button>
          </td>
        </tr>
      ))}
    </tbody>
  </Table>
</Container>
);
}

export default Caselist;
EOL

# Create CaseDetail page
cat > src/pages/CaseDetail.jsx << EOL
import React, { useState, useEffect } from 'react';
import { useParams } from 'react-router-dom';
import { Card, Container, Row, Col } from 'react-bootstrap';
import axios from 'axios';

function CaseDetail() {
  const { id } = useParams();
  const [caseData, setCaseData] = useState(null);

  useEffect(() => {
    const fetchCase = async () => {
      try {

```

```

        const response = await
axios.get(`http://localhost:5000/api/cases/${id}`);
        setCaseData(response.data);
      } catch (error) {
        console.error('Error fetching case:', error);
      }
    };
    fetchCase();
  }, [id]);

  if (!caseData) {
    return <div>Loading...</div>;
  }

  return (
    <Container>
      <Card className="mb-4">
        <Card.Header>
          <h3>{caseData.title}</h3>
        </Card.Header>
        <Card.Body>
          <Row>
            <Col md={6}>
              <p><strong>Status:</strong> {caseData.status}</p>
              <p><strong>Created:</strong> {new
Date(caseData.created_at).toLocaleDateString()}</p>
            </Col>
          </Row>
          <Row>
            <Col>
              <h4>Description</h4>
              <p>{caseData.description}</p>
            </Col>
          </Row>
        </Card.Body>
      </Card>
    </Container>
  );
}

export default CaseDetail;
EOL

# Create Analysis page
cat > src/pages/Analysis.jsx << EOL
import React, { useState } from 'react';
import { Container, Form, Button, Card } from 'react-bootstrap';
import axios from 'axios';

```

```

function Analysis() {
  const [text, setText] = useState('');
  const [result, setResult] = useState(null);

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const response = await axios.post('http://localhost:5000/api/analyze', {
text });
      setResult(response.data);
    } catch (error) {
      console.error('Error analyzing text:', error);
    }
  };

  return (
    <Container>
      <h2>Text Analysis</h2>
      <Card className="mb-4">
        <Card.Body>
          <Form onSubmit={handleSubmit}>
            <Form.Group className="mb-3">
              <Form.Label>Enter text to analyze</Form.Label>
              <Form.Control
                as="textarea"
                rows={4}
                value={text}
                onChange={(e) => setText(e.target.value)}
              />
            </Form.Group>
            <Button variant="primary" type="submit">
              Analyze
            </Button>
          </Form>
        </Card.Body>
      </Card>

      {result && (
        <Card>
          <Card.Body>
            <h4>Analysis Results</h4>
            <p><strong>Similarity Score:</strong>
{result.similarity_score}</p>
          </Card.Body>
        </Card>
      )}
    </Container>
  );
}

```

```

    );
}

export default Analysis;
EOL

# Create main.jsx
cat > src/main.jsx << EOL
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';
import './index.css';

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
EOL

# Create index.css
cat > src/index.css << EOL
body {
  margin: 0;
  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto',
'Oxygen',
  'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',
  sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}

code {
  font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',
  monospace;
}
EOL

# Create index.html
cat > index.html << EOL
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>IPC Nexus</title>
  </head>

```

```
<body>
  <div id="root"></div>
  <script type="module" src="/src/main.jsx"></script>
</body>
</html>
EOL
```

```
# Create Dockerfile for frontend
```

```
cat > Dockerfile << EOL
```

```
FROM node:18-alpine
```

```
WORKDIR /app
```

```
COPY package*.json ./
```

```
RUN npm install
```

```
COPY . .
```

```
RUN npm run build
```

```
EXPOSE 3000
```

```
CMD ["npm", "run", "preview"]
```

```
EOL
```

```
# Create docker-compose.yml in root directory
```

```
cd ..
```

```
cat > docker-compose.yml << EOL
```

```
version: '3.8'
```

```
services:
```

```
  backend:
```

```
    build: ./Backend
```

```
    ports:
```

```
      - "5000:5000"
```

```
    environment:
```

```
      - FLASK_APP=app.py
```

```
      - FLASK_ENV=development
```

```
    volumes:
```

```
      - ./Backend:/app
```

```
    networks:
```

```
      - app-network
```

```
  frontend:
```

```
    build: ./Frontend
```

```
    ports:
```

```
      - "3000:3000"
```

```
    depends_on:
```



```
    - backend
  volumes:
    - ./Frontend:/app
  networks:
    - app-network

networks:
  app-network:
    driver: bridge
EOL

# Create .gitignore
cat > .gitignore << EOL
# Dependencies
node_modules/
__pycache__/
*.pyc

# Environment
.env
.venv
env/
venv/

# Build
dist/
build/

# IDE
.vscode/
.idea/

# Logs
*.log

# Database
*.db
EOL

# Create Kubernetes deployment files
print_status "Creating Kubernetes deployment files..."
cd k8s

# Create backend deployment
cat > backend-deployment.yaml << EOL
apiVersion: apps/v1
kind: Deployment
metadata:
```

```
name: backend
labels:
  app: ipc-nexus
  tier: backend
spec:
  replicas: 2
  selector:
    matchLabels:
      app: ipc-nexus
      tier: backend
  template:
    metadata:
      labels:
        app: ipc-nexus
        tier: backend
    spec:
      containers:
        - name: backend
          image: ipc-nexus-backend:latest
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 5000
          env:
            - name: FLASK_APP
              value: "app.py"
            - name: FLASK_ENV
              value: "production"
            - name: DATABASE_URL
              value: "sqlite:///law_enforcement.db"
            - name: SECRET_KEY
              valueFrom:
                secretKeyRef:
                  name: backend-secrets
                  key: secret-key
      resources:
        requests:
          memory: "256Mi"
          cpu: "200m"
        limits:
          memory: "512Mi"
          cpu: "500m"
---
apiVersion: v1
kind: Service
metadata:
  name: backend-service
spec:
  selector:
```

```

    app: ipc-nexus
    tier: backend
  ports:
  - port: 5000
    targetPort: 5000
  type: ClusterIP
EOL

# Create frontend deployment
cat > frontend-deployment.yaml << EOL
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
  labels:
    app: ipc-nexus
    tier: frontend
spec:
  replicas: 2
  selector:
    matchLabels:
      app: ipc-nexus
      tier: frontend
  template:
    metadata:
      labels:
        app: ipc-nexus
        tier: frontend
    spec:
      containers:
      - name: frontend
        image: ipc-nexus-frontend:latest
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 3000
        env:
        - name: VITE_API_URL
          value: "http://backend-service:5000"
        resources:
          requests:
            memory: "128Mi"
            cpu: "100m"
          limits:
            memory: "256Mi"
            cpu: "200m"
---
apiVersion: v1
kind: Service

```

```
metadata:
  name: frontend-service
```

```
spec:
```

```
  selector:
```

```
    app: ipc-nexus
```

```
    tier: frontend
```

```
  ports:
```

```
    - port: 80
```

```
      targetPort: 3000
```

```
  type: LoadBalancer
```

```
EOL
```

```
# Create ingress
```

```
cat > ingress.yaml << EOL
```

```
apiVersion: networking.k8s.io/v1
```

```
kind: Ingress
```

```
metadata:
```

```
  name: ipc-nexus-ingress
```

```
  annotations:
```

```
    nginx.ingress.kubernetes.io/rewrite-target: /
```

```
spec:
```

```
  rules:
```

```
    - host: ipc-nexus.local
```

```
      http:
```

```
        paths:
```

```
          - path: /api
```

```
            pathType: Prefix
```

```
            backend:
```

```
              service:
```

```
                name: backend-service
```

```
                port:
```

```
                  number: 5000
```

```
          - path: /
```

```
            pathType: Prefix
```

```
            backend:
```

```
              service:
```

```
                name: frontend-service
```

```
                port:
```

```
                  number: 80
```

```
EOL
```

```
# Create secrets
```

```
cat > secrets.yaml << EOL
```

```
apiVersion: v1
```

```
kind: Secret
```

```
metadata:
```

```
  name: backend-secrets
```

```
type: Opaque
```

```
data:
  secret-key: eW91ci1zZWNyZXQta2V5LWhlcmU= # Base64 encoded "your-secret-key-
here"
EOL

# Create configmap
cat > configmap.yaml << EOL
apiVersion: v1
kind: ConfigMap
metadata:
  name: ipc-nexus-config
data:
  FLASK_APP: "app.py"
  FLASK_ENV: "production"
  DATABASE_URL: "sqlite:///law_enforcement.db"
EOL

cd ..

# Make the script executable
chmod +x deploy.sh

print_status "Project structure created successfully!"

# Build Docker images
print_status "Building Docker images..."
docker build -t ipc-nexus-backend:latest ./Backend
docker build -t ipc-nexus-frontend:latest ./Frontend

# Deploy to Kubernetes
print_status "Deploying to Kubernetes..."

# Check if minikube is running, if not start it
if ! minikube status | grep -q "Running"; then
  print_status "Starting minikube..."
  minikube start
fi

# Load Docker images into minikube
print_status "Loading Docker images into minikube..."
minikube image load ipc-nexus-backend:latest
minikube image load ipc-nexus-frontend:latest

# Apply Kubernetes configurations
print_status "Applying Kubernetes configurations..."
kubectl apply -f k8s/configmap.yaml
kubectl apply -f k8s/secrets.yaml
kubectl apply -f k8s/backend-deployment.yaml
```

```
kubectl apply -f k8s/frontend-deployment.yaml
kubectl apply -f k8s/ingress.yaml

# Wait for deployments to be ready
print_status "Waiting for deployments to be ready..."
kubectl wait --for=condition=available --timeout=300s deployment/backend
kubectl wait --for=condition=available --timeout=300s deployment/frontend

# Get the service URL
print_status "Getting service URL..."
if command -v minikube &> /dev/null; then
    SERVICE_URL=$(minikube service frontend-service --url)
    print_status "Frontend is available at: ${SERVICE_URL}"
else
    print_status "Frontend service is running. Use kubectl to get the service URL."
    print_status "Run: kubectl get service frontend-service"
fi

print_status "Backend API is available at: http://backend-service:5000"

# Instructions for the user
echo -e "\n${GREEN}Setup completed successfully!${NC}"
echo -e "${YELLOW}To view logs, run:${NC} kubectl logs -l app=ipc-nexus"
echo -e "${YELLOW}To delete the deployment, run:${NC} kubectl delete -f k8s/"
echo -e "${YELLOW}To stop minikube, run:${NC} minikube stop"
echo -e "${YELLOW}To delete minikube, run:${NC} minikube delete"
```

## TO START THE APPLICATION:

```
hafsa_027@Dell:~$ minikube start
🔧 minikube v1.35.0 on Ubuntu 24.04 (amd64)
🔧 Using the docker driver based on existing profile

💡 The requested memory allocation of 3800MiB does not leave room for system overhead (total system memory: 3802MiB). You may face stability issues.
   Suggestion: Start minikube with less memory allocated: 'minikube start --memory=2200mb'

🔥 Starting "minikube" primary control-plane node in "minikube" cluster
📦 Pulling base image v0.0.46 ...
🔄 Restarting existing docker container for "minikube" ...
🔥 StartHost failed, but will try again: driver start: start: docker start minikube: exit status 1
stdout:

stderr:
Error response from daemon: failed to create task for container: failed to create shim task: OCI runtime create failed: runc create failed: unable to start
container process: error during container init: error setting cgroup config for procHooks process: failed to write "a": write /sys/fs/cgroup/devices
/docker/55b54e36a837faebb4c57c3c4128bda0a6579a5edf866ee661bc9ccdd934a54/devices.allow: invalid argument: unknown
Error: failed to start containers: minikube

🔄 Restarting existing docker container for "minikube" ...
📦 Preparing Kubernetes v1.32.0 on Docker 27.4.1 ...
🔍 Verifying Kubernetes components...
   • Using image registry.k8s.io/metrics-server/metrics-server:v0.7.2
   • Using image gcr.io/k8s-minikube/storage-provisioner:v5
```

```
hafsa_027@Dell:~$ minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured

hafsa_027@Dell:~$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
backend-6ff778f747-5cz5w            0/1     Running   2 (103s ago)  102m
frontend-8f48b7b9f-xqzzj           1/1     Running   4 (103s ago)  5h38m
prometheus-deployment-56d98cf486-bsxlc 1/1     Running   1 (103s ago)  90m

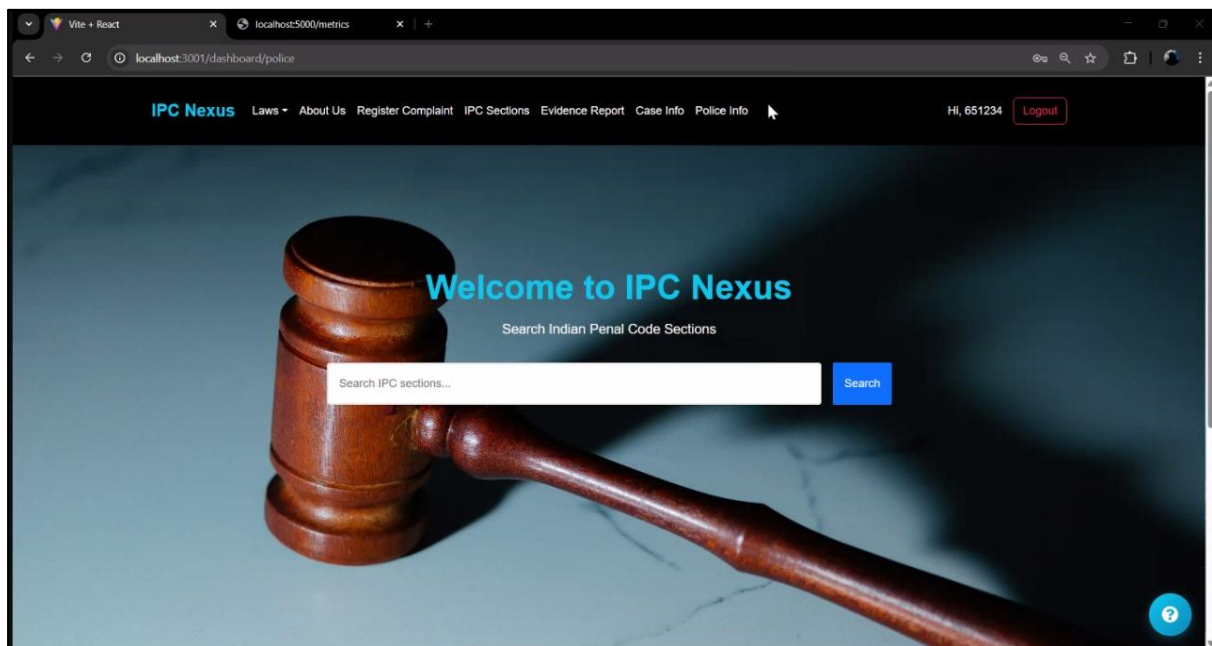
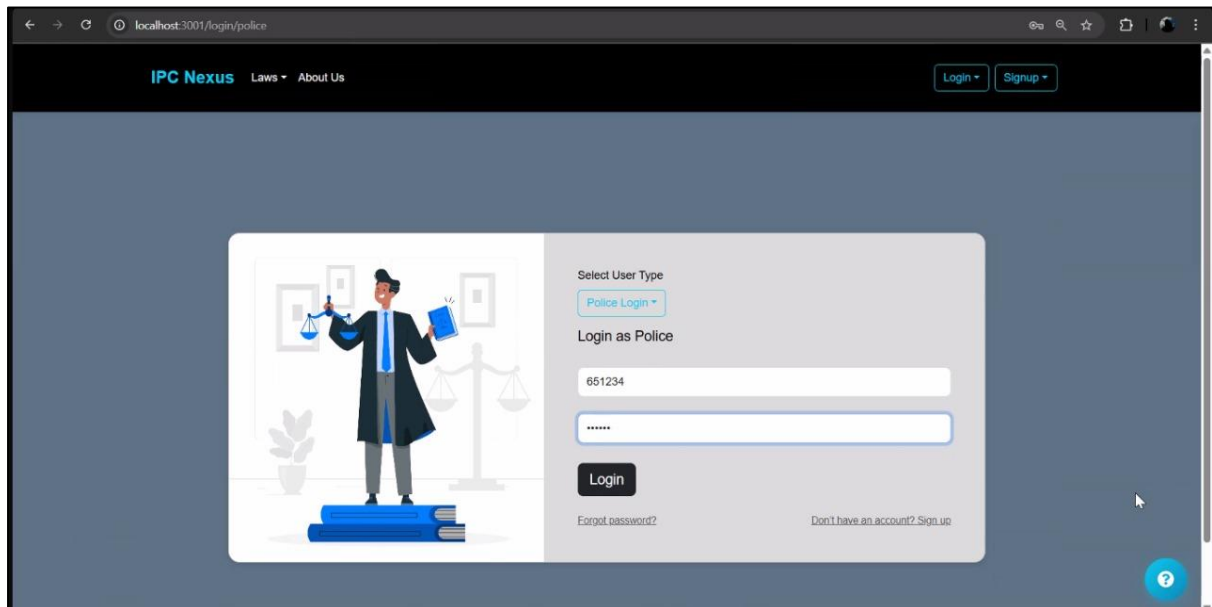
hafsa_027@Dell:~$ kubectl get services
NAME      TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
backend-service  ClusterIP  10.102.0.124 <none>        5000/TCP         23h
frontend-service NodePort    10.99.176.8   <none>        80:31935/TCP,9113:30674/TCP 22h
kubernetes      ClusterIP  10.96.0.1     <none>        443/TCP          23h
prometheus-service ClusterIP  10.99.69.230 <none>        9090/TCP         90m

hafsa_027@Dell:~$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
backend-6ff778f747-5cz5w            0/1     Running   2 (113s ago)  103m
frontend-8f48b7b9f-xqzzj           1/1     Running   4 (113s ago)  5h38m
prometheus-deployment-56d98cf486-bsxlc 1/1     Running   1 (113s ago)  90m

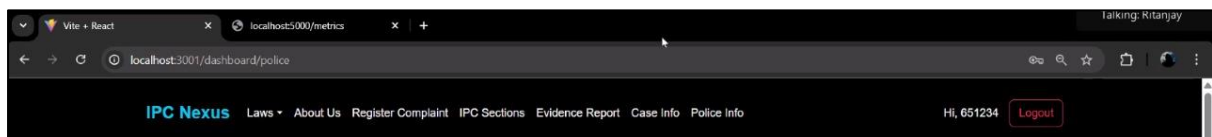
hafsa_027@Dell:~$ kubectl port-forward service/backend-service 5000:5000
Forwarding from 127.0.0.1:5000 -> 5000
Forwarding from [::1]:5000 -> 5000
```

```
hafsa_027@Dell:~$ kubectl port-forward service/frontend-service 3001:80
Forwarding from 127.0.0.1:3001 -> 80
Forwarding from [::1]:3001 -> 80
```

## HOW OUR WEBSITE ACTUALLY WORKS AND LOOKS:

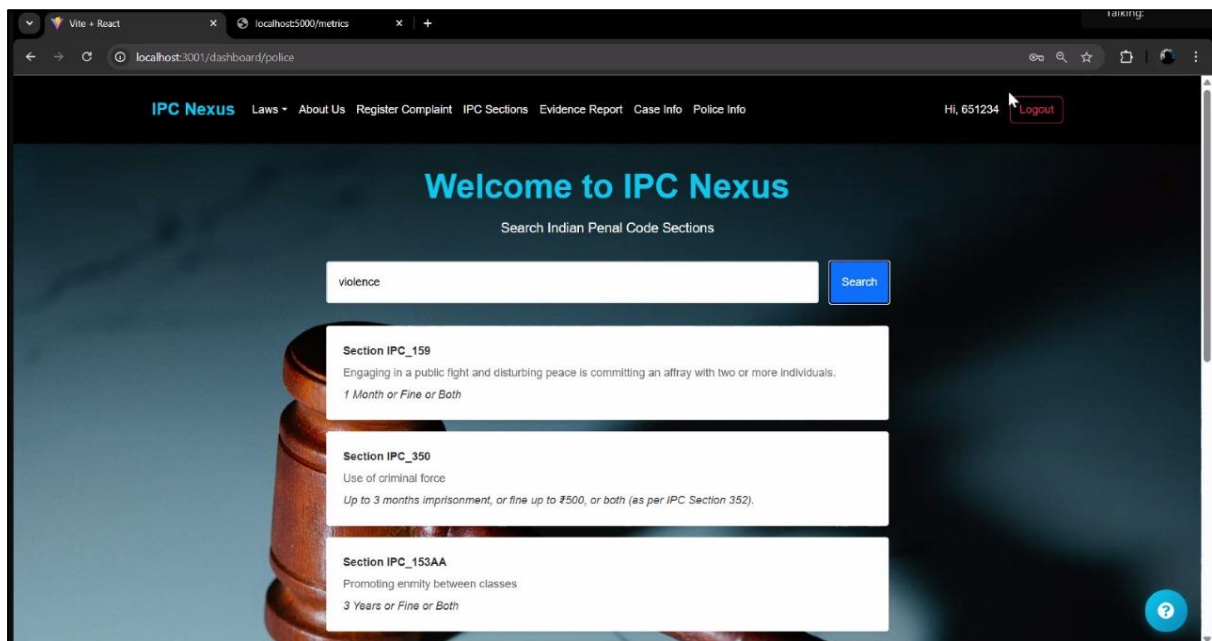


## And if we notice all the tabs here:





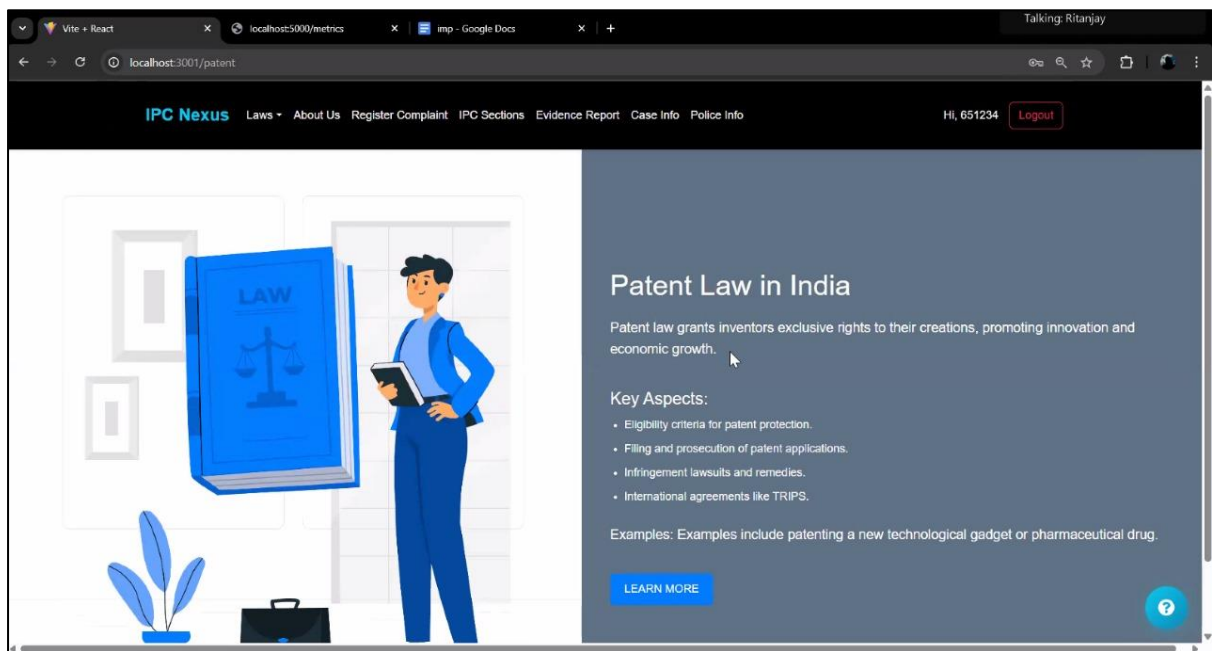
The user can directly search in the search bar, for the specific crime or incident And it would return all possible Indian Penal code Sections with punishment and fine for that crime:  
(So when user clicks enter or click on search icon all the related results are returned.)



After this we can go to second tab to study about various laws in detail:



And user clicks on any of the field then:



**The next tab that we have is the register complaint tab that returns the FRI form in digital format for entering the incident:**

The screenshot shows the same web browser window with the URL `localhost:3001/register-complaint`. The header and navigation links are identical. The main content area features an illustration of three people (a man, a woman, and a police officer) standing next to a large red clipboard with a checklist. To the right, the heading "Register Complaint" is displayed. Below it, there is a form with the following fields: "Name", "Email", "Phone", "dd-mm-yyyy" (with a calendar icon), "Location", "Address", and "Description of the Incident" (with a text area icon). A blue "Submit" button is at the bottom of the form. A small blue circle with a question mark is in the bottom right corner.

And when user fills these DIGITAL FIR's:

Complaint registered successfully! Case ID: 3

### Register Complaint

Ritanjay

Ritanjay04@gmail.com

9708397166

01-04-2025

Sundar Nagar

Himachal Pradesh

Violence

Submit

**They are submitted to the assigned cases for the specific police officer who is logged in, as they registered the complaint:**

IPC Nexus Laws About Us Register Complaint IPC Sections Evidence Report Case Info Police Info Hi, 651234 Logout

Assigned Cases

Resolved Cases

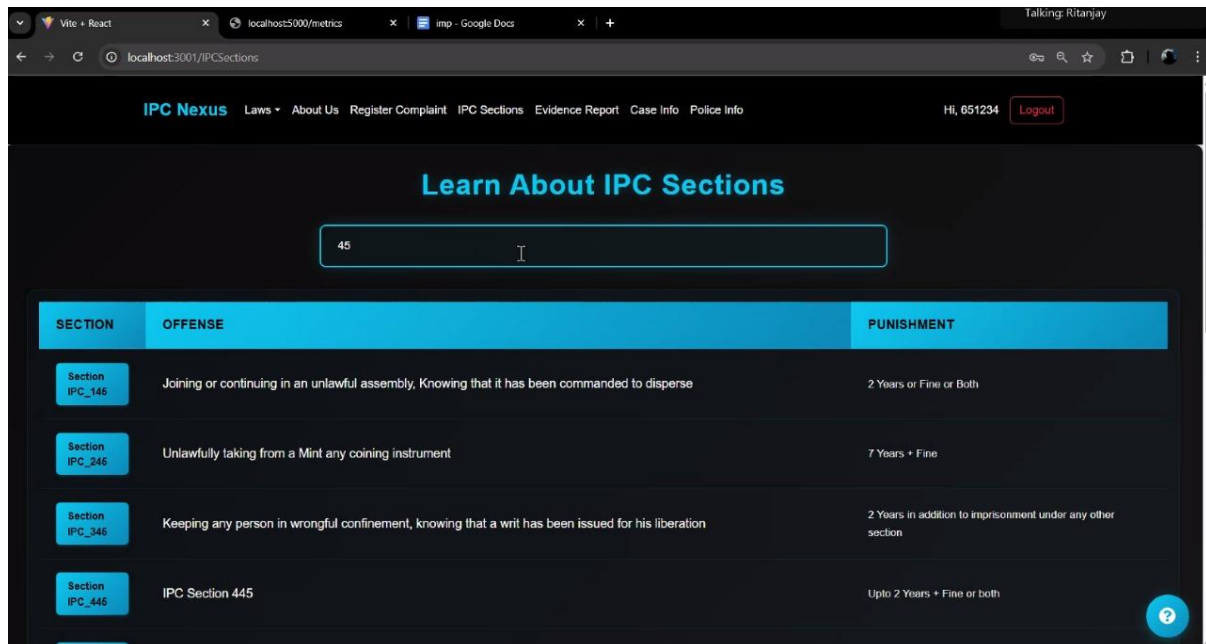
#### Assigned Cases

Case ID: 3  
Title: Case: Ritanjay - 2025-04-01  
Description: Violence  
Lawyer ID: 789456

Case ID: 4  
Title: Case: Ritanjay - 2025-04-01  
Description: Violence  
Lawyer ID: 789456

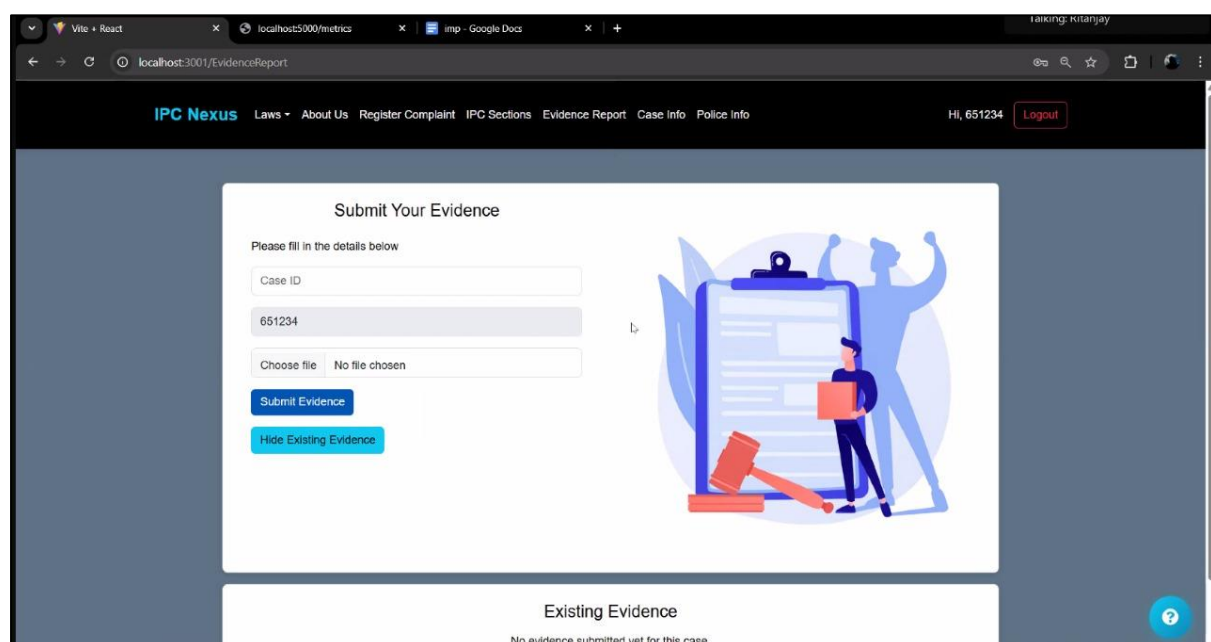
And each police officer would have their own assigned cases.

Then we have an ipc section tab which returns data related to ipc section offense and lastly punishment...  
here a person can specifically search for specific “sections”.



Here also we are using sentence transformer , so that when we are searching for specific ipc section then all related or starting from it are returned. As in above example we just typed 45, so it returning all ipc sections starting with 45 in it.

If a police officer wants submit some evidence file related to a case:



**Here after mentioning the case id and file:**

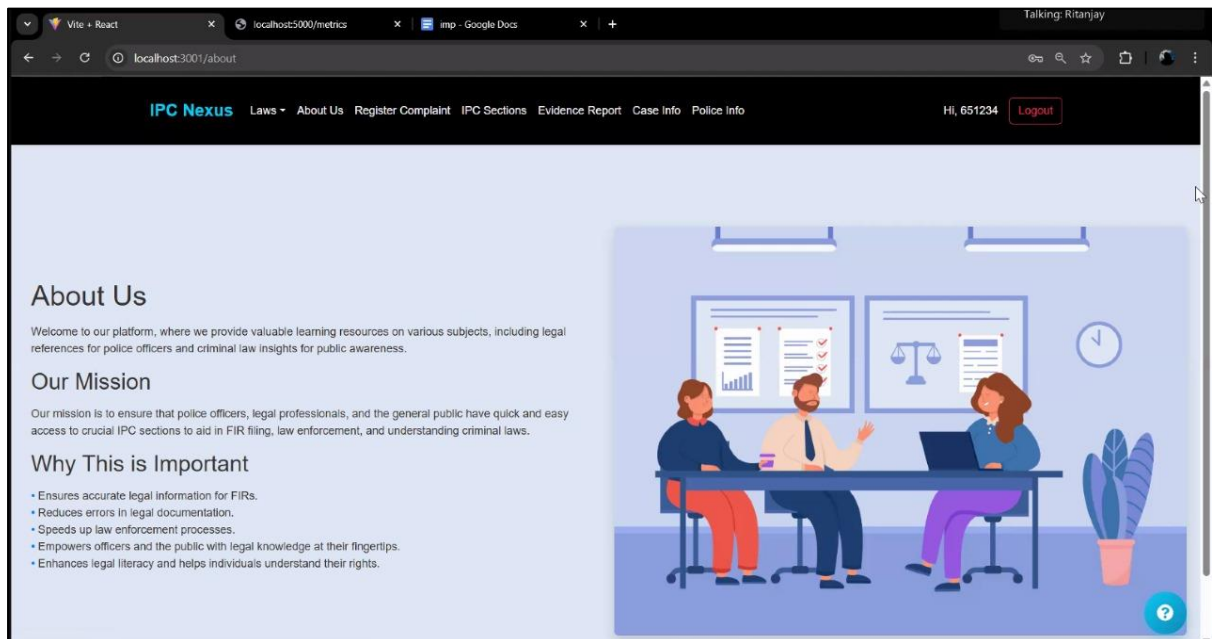
The screenshot shows a web browser window with the URL `localhost:3001/EvidenceReport`. The page has a dark header with the logo "IPC Nexus" and navigation links: "Laws", "About Us", "Register Complaint", "IPC Sections", "Evidence Report", "Case Info", and "Police Info". The user is logged in as "Hi, 651234" with a "Logout" button. The main content area is titled "Submit Your Evidence" and contains a form with the instruction "Please fill in the details below". The form has three input fields: a text field with "3", a text field with "651234", and a file selection field showing "Choose file" and "ipc.txt". Below the fields are two buttons: "Submit Evidence" (blue) and "Show Existing Evidence" (light blue). To the right of the form is an illustration of a person holding a folder next to a large clipboard with a gavel. A help icon (?) is in the bottom right corner.

**And then officer can see that below as well:**

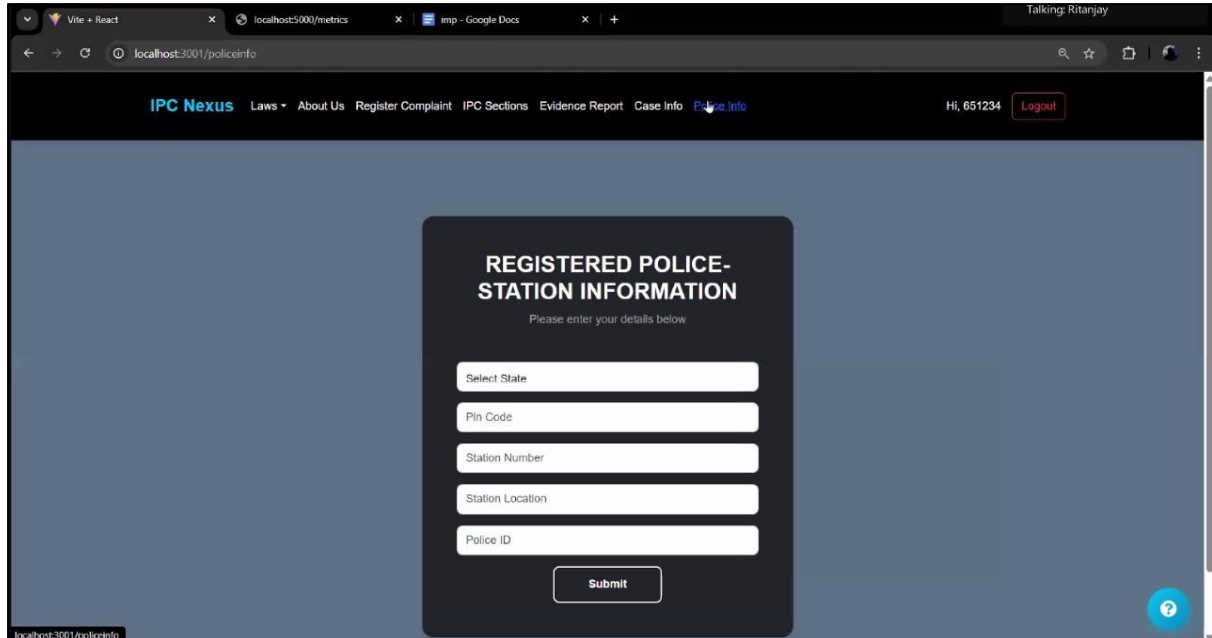
This screenshot shows the same "Submit Your Evidence" form as the previous image, but with an additional section below it titled "Existing Evidence". This section contains a table with one row of evidence. The table has two columns: "Evidence 1:" and "File: ipc.txt, Case ID: 3, Submitted by: Police". The "Submit Evidence" button is now disabled, and a new "Hide Existing Evidence" button (light blue) has appeared. The rest of the page, including the header and navigation, remains the same.

Existing Evidence	
Evidence 1:	File: ipc.txt, Case ID: 3, Submitted by: Police

Then in last tab we have about us page, describing our mission:

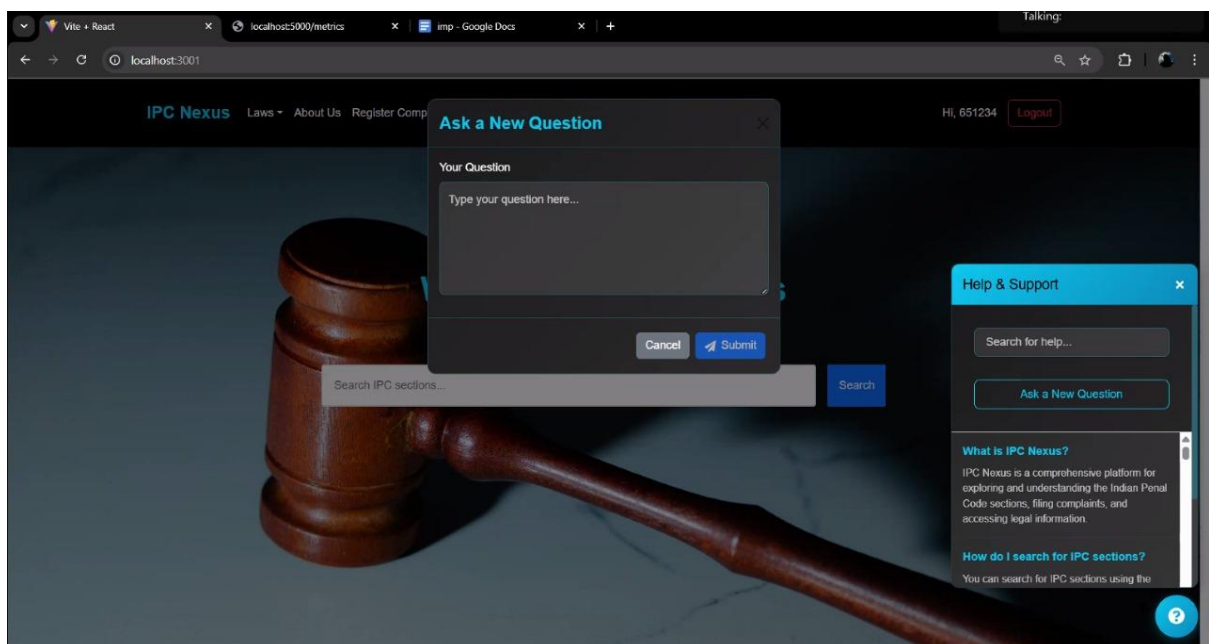
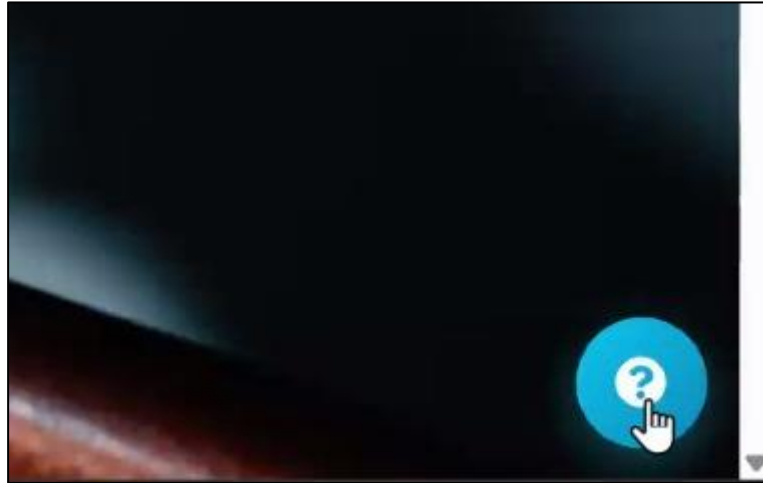


and then we have an police info tab , from where if police officer is new to website and want to register their police station then they can register their police station:



Then we also have created bot to help bot through our website:

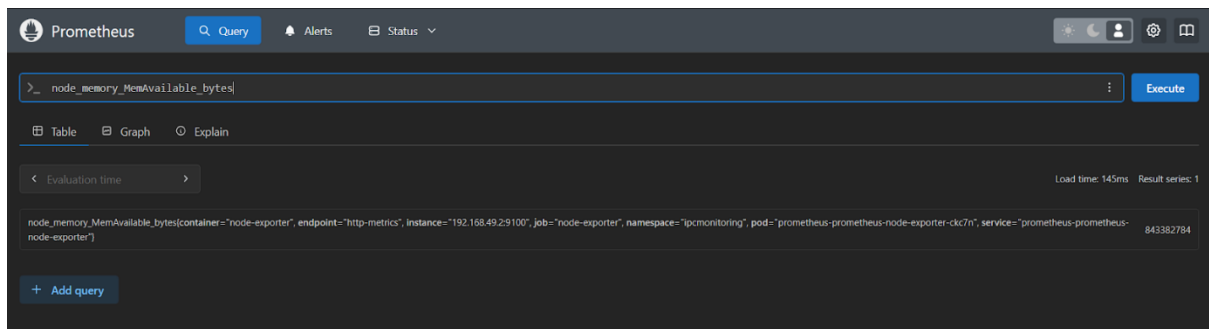
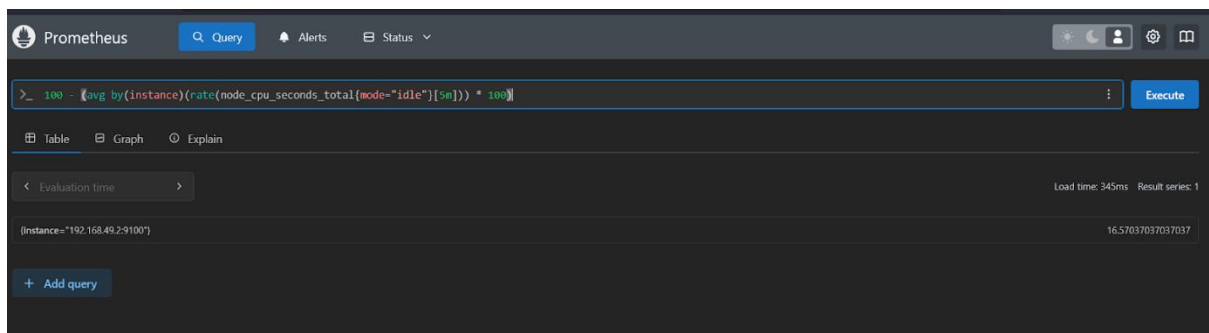
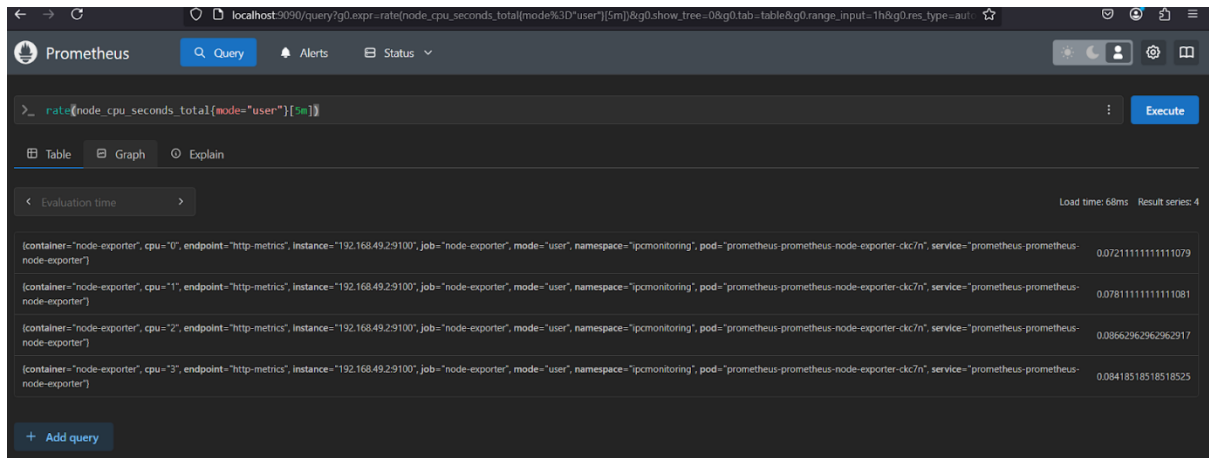
you can click on the blue icon on the bottom right to access it and if u have some extra question then u can submit it from there as well:





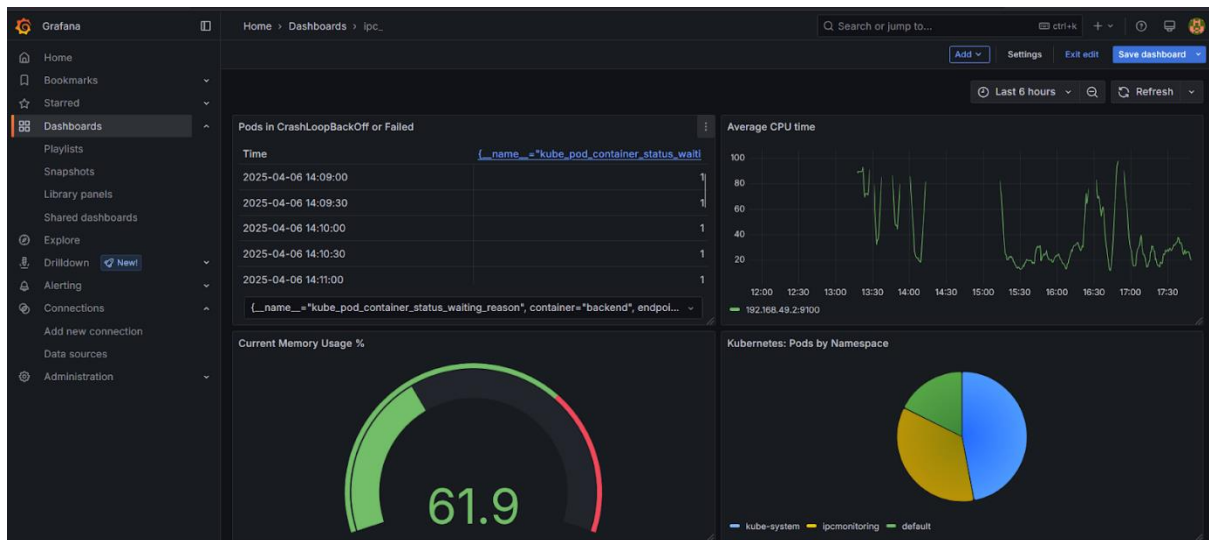
# GRAFANA AND PROMETHEUS DASHBORDS AND PANEL:

## Prometheus interface images:





## Grafana dashboard and panels:



This is the "Data sources" configuration page for "prometheus-1" in Grafana. It shows settings for the data source, including "Cache level", "Incremental querying (beta)", "Disable recording rules (beta)", "Other" settings, and "Exemplars". A success message at the bottom indicates that the Prometheus API was successfully queried.

**Cache level** ☐ Low

**Incremental querying (beta)** ☐ ☒

**Disable recording rules (beta)** ☐ ☒

**Other**

**Custom query parameters** ☐ Example: max\_source\_resolution=5m&timeout=...

**HTTP method** ☐ POST

**Use series endpoint** ☐ ☒

**Exemplars**

☒ Successfully queried the Prometheus API.

Next, you can start to visualize data by [building a dashboard](#), or by querying data in the [Explore view](#).

