CHAPTER 4

System Requirement Specification

4.1 Software Requirement

4.1.1 Programming Languages

- **Python**

  The core programming language used for developing the backend with Django. It is required for creating models, views, and handling server-side logic.

4.1.2 Frameworks and Libraries

- **DjangoFramework**(Version 3.2 or higher):

  Used for building the server-side functionality of the web application. It provides built-in tools for handling databases, user authentication, and template rendering.
- **Bootstrap Framework**:

  Used for building responsive and user-friendly front-end interfaces.
- **Font Awesome**:

  Provides scalable vector icons used for adding aesthetic icons to buttons, links, and other UI components.
- **JavaScript Libraries**:
  - For handling client-side interactivity, such as capturing photos, speech-to-text, and location detection.

4.1.3 Database Management System

- **SQLite** (Default):

  A lightweight, built-in database for development and small-scale applications.

4.1.4 Development Tools

- **Integrated Development Environment (IDE):**
  - **VS Code**: Used for writing and managing Python and front-end code.
- **Browser Developer Tools:**
  - For debugging front-end issues, inspecting elements, and testing responsiveness.

4.1.5 Web Servers

- **Development Server:**
  - Django's built-in development server for local testing and debugging.
- **Production Server:**
  - **Gunicorn** or **uWSGI**: For serving the Django application in production.
  - **Nginx** or **Apache**: As a reverse proxy and for serving static and media files.

4.1.6 Operating System

- **Development Environment:**
  - Windows, macOS, or Linux.
- **Production Environment:**
  - Linux (e.g., Ubuntu, CentOS): Preferred for deploying web applications due to stability and performance.

4.1.7 Image Handling

- **Pillow Library:**

  Used for processing uploaded images, such as resizing or format conversions.

4.1.8 Media and Static File Management

- **AWS S3**, **Google Cloud Storage**, or **Azure Blob Storage** (Optional): For storing uploaded files and images in production.
- **Whitenoise Library**:

  For serving static files efficiently in production without additional configurations.

4.1.9 APIs and Integrations

- **Geolocation API:**
  For retrieving the user's location during complaint submission.
- **Speech-to-Text API:**
  - Web Speech API (built-in browser API) for converting audio input to text.

4.1.10 Deployment Tools

- **Docker:**
  For containerizing the application and ensuring consistent environments across development, testing, and production.
- **Version Control System:**
  - **Git**: For tracking code changes and collaborating with team members.
- **CI/CD Tools:**
  - **GitHub Actions** or **GitLab CI/CD**: For automated testing and deployment workflows.

4.1.11 Browsers

- **Supported Browsers:**
  - Google Chrome, Mozilla Firefox, Microsoft Edge, and Safari.

4.1.12 Package Management

- **pip(Python Package Index):**
  For installing required Python libraries and dependencies.

4.1.13 Hosting and Domain

- **Cloud Hosting:**
  - **AWS EC2**, **DigitalOcean**, **Heroku**, or **Google Cloud Platform** for deploying the web application.

4.1.14 Security and Authentication

- **SSL Certificates:**
  To secure data transmission over the internet.
- **Django Authentication Framework:**
  For user authentication, session management, and access control.

4.1.15 Backup and Monitoring Tools

- **Database Backup Tools:**
  - Tools like `pg_dump` (for PostgreSQL) or `mysqldump` (for MySQL) for regular backups.
- **Monitoring Tools:**
  - **New Relic** or **Sentry** for performance monitoring and error tracking.

4.1.16 Testing Tools

- **Django Test Framework:**
  For unit testing and integration testing of the application.
- **Selenium**:
  For end-to-end testing of user workflows.

4.2 Visual Studio Code

4.2.1 Code Editor

- **Writing Backend Code:**
  VS Code provides an efficient platform for writing Python code to develop models, views, and templates in Django.
- **Frontend Development:**
  It allows editing of HTML, CSS, and JavaScript files used for designing the user interface and interactivity.

4.2.2 Extensions for Enhanced Productivity

- **Python Extension:**
  - Adds support for Python syntax highlighting, debugging, and code linting.
  - Provides IntelliSense for auto-completion and suggestions, making Python development more efficient.
- **Django Extensions:**
  - Adds syntax highlighting for Django templates and supports Django-specific commands.
- **Live Server Extension:**
  - Automatically reloads the web page when changes are made, speeding up frontend development.
- **ESLint and Prettier Extensions:**
  - Help enforce coding standards and format frontend code (HTML, CSS, JavaScript).

4.2.3 Integrated Terminal

- **Running the Django Development Server:**
  - Developers can use the built-in terminal to start the Django server (`python manage.py runserver`) without switching to a separate terminal application.
- **Database Migrations:**
  - Execute database migration commands (`python manage.py makemigrations` and `python manage.py migrate`) directly within VS Code.
- **Package Installation:**
  - Install Python packages using `pip install` from the integrated terminal.

4.2.4 Version Control Integration

- **Git Integration:**
  - VS Code has built-in Git support for tracking changes, creating branches, and committing code.
- **Extensions for Git:**
  - Plugins like GitLens provide advanced Git functionalities, such as visualizing commit histories and resolving merge conflicts.

4.2.5 Debugging

- **Built-in Debugger:**
  - The VS Code debugger allows developers to set breakpoints, inspect variables, and step through code, which is particularly useful for diagnosing issues in Django views and scripts.
- **Django-Specific Debugging:**
  - Debug configurations can be set up to work seamlessly with Django's development server.

4.2.6 Code Formatting and Linting

- **Static Code Analysis:**
  - Extensions like Pylint and Flake8 ensure adherence to Python coding standards (PEP 8).
- **Auto-Formatting Tools:**
  - Prettier or Black can be integrated into VS Code to format Python, HTML, and JavaScript code automatically.

4.2.7 Collaboration

- **Live Share Extension:**

- Allows multiple developers to collaborate on the same codebase in real-time, making pair programming or team debugging sessions easier.

4.2.8 Project Management

- **Workspace Management:**
  - Organize the project files, such as the Django application directory structure (e.g., `models.py`, `views.py`, `templates`, and `static` files).
- **Task Automation:**
  - Automate repetitive tasks like running tests or starting the server using the task runner.

4.2.9 Testing Support

- **Test Runner Integration:**
  - Run unit tests and integration tests directly from VS Code.

4.2.10 Customization

- **Themes and Keybindings:**
  - Developers can customize the look and feel of the editor and adjust keybindings to suit their workflow.
- **Snippets:**
  - Use custom code snippets for repetitive tasks, such as creating a Django model, view, or template structure.

Why Use VS Code for This Application?

- Lightweight and fast.
- Extensive plugin ecosystem tailored for Python, Django, and web development.
- User-friendly interface and excellent community support.
- Cross-platform compatibility (Windows, macOS, Linux).

4.3 Location API's

4.3.1. Location API

The **Location API** is a web browser feature (part of the **Geolocation API**) that allows developers to retrieve the geographical location (latitude and longitude) of a user's device. It is commonly used in web and mobile applications to provide location-based services.

Features of Location API

- **Access Device Location:**
  It fetches the geographical location of a device using GPS, Wi-Fi, or IP address data.
- **Real-Time Updates:**
  Supports tracking location changes in real-time.
- **Cross-Browser Support:**
  Most modern browsers, such as Chrome, Firefox, Edge, and Safari, support the Location API.

Key Methods

1. **navigator.geolocation.getCurrentPosition()**
   - Retrieves the device's current position (latitude and longitude).
2. **navigator.geolocation.watchPosition()**
   - Tracks location changes and continuously updates the position.
3. **navigator.geolocation.clearWatch()**
   - Stops tracking location updates

Use Case in the Web Application

- Fetches the user's current latitude and longitude when they raise a complaint.
- Displays the location to confirm its accuracy.
- Enables storing geolocation data in the backend database for future reference.

Advantages

- Easy to implement using JavaScript.
- Directly integrates into browsers without external dependencies.

Limitations

- Requires user consent for access.
- Accuracy depends on the device's GPS capabilities or internet connection.
- Cannot convert coordinates into human-readable addresses (requires a service like OpenCage API).

4.3.2 OpenCage API

The **OpenCage API** is a **Geocoding API** that converts geographical coordinates (latitude and longitude) into human-readable addresses and vice versa. It is widely used for mapping, navigation, and location-based services.

Features of OpenCage API

- **Forward Geocoding:**
  Converts addresses or place names into geographic coordinates.
- **Reverse Geocoding:**
  Converts coordinates (latitude and longitude) into human-readable addresses.
- **Multi-Language Support:**
  Returns location data in multiple languages.
- **Time Zone Information:**
  Provides timezone details for the queried location.
- **Country Restriction:**
  Limits results to specific countries, if desired.

Integration with the Application

- Converts the latitude and longitude retrieved from the Location API into a meaningful address (e.g., "123 Main St, City, Country").
- Stores the address in the database alongside the complaint details.
- Displays the resolved address to the user for clarity.

How It Works

1. **Make an API Request:**
    - Send a request to OpenCage API with coordinates or an address as input.
2. **Receive a Response:**
    - The API returns detailed location information, including:
        - Address components (city, state, country, postal code).
        - Nearby landmarks or features.
        - Timezone details.

Advantages

- Provides highly accurate and detailed geocoding data.
- Supports both free and paid plans for scalable usage.
- Can be easily integrated with APIs or frameworks like Django.

Limitations

- Requires an API key for access.
- Free tier has request limits (e.g., 2,500 requests/day).
- Dependency on third-party service availability.

Comparison: Location API vs. OpenCage API

Feature	Location API	OpenCage API
Purpose	Fetch latitude and longitude of the device.	Convert coordinates to addresses or vice versa.
Source	Browser or device (GPS, Wi-Fi, IP).	External API with geocoding capabilities.
Integration	JavaScript in the frontend.	Requires HTTP API calls (backend/frontend).
Dependency	No external API needed.	Requires an API key and external service.
Output	Coordinates (latitude, longitude).	Address details, timezone, and other metadata.