System Architecture and Overview of the Newfoundland Missing Persons App

Disant Upadhyay

September 16, 2023

Contents

1 Executive Summary

The Newfoundland Missing Persons App aims to offer a consolidated platform for the rapid dissemination of information regarding missing individuals in Newfoundland. This document serves as a comprehensive guide detailing the project's motivation, technical architecture, and deployment strategies, catering primarily to developers.

2 Project Overview and Motivation

A pressing societal issue that Newfoundland faces is the increasing number of individuals who go missing every year. The Newfoundland Missing Persons App stands as a beacon of hope, aiming to bridge this information gap. Its objectives include:

- Providing a centralized platform for storing detailed information about missing individuals.
- Speeding up the dissemination of crucial information.
- Raising community awareness and promoting collaborative search efforts.
- Offering advanced features to enhance user experience.

To achieve these goals, the application uses the Django framework, known for its scalability, robustness, and rich feature set.

3 Technical Overview with Django

3.1 Django Framework

Django, a high-level Python web framework, promotes rapid development and clean, pragmatic design. Its "batteries-included" philosophy provides built-in tools and functionalities, simplifying many common development tasks.

3.1.1 Directory Structure

Description:

- `/LostInNL/`: Main project directory.
- `/LostInNL/missingpersons/`: Django's primary application package.
- `models.py`: Defines database models.
- `views.py`: Houses route definitions, view functions, and core logic.
- `/static/`: Contains static files such as CSS, JS, and images.

- `/templates/`: Houses Django templates for dynamic HTML generation.
- `settings.py`: Contains configurations, including database connections.
- `urls.py`: Manages URL patterns and routes.
- `requirements.txt`: Enumerates essential Python packages.

```
/LostInNL/
|-- /missingpersons/
|   |-- models.py
|   |-- views.py
|   |-- /static/
|   |-- /templates/
|-- settings.py
|-- urls.py
|-- requirements.txt
```

3.1.2 Database Interaction

Django's ORM (Object-Relational Mapping) seamlessly integrates with databases, abstracting and simplifying many of the common database operations. For this project:

- A MySQL database on AWS RDS is utilized.
- AWS KMS encrypts database credentials.
- At runtime, Django decrypts the credentials.
- Direct RDS access is restricted, with only the EC2 instance permitted to connect.

3.2 Client-Side

The client side uses HTML for structural layout, CSS for styling, and JavaScript for dynamic interactions, liaising with the Django backend to retrieve and display data.

3.2.1 Directory Structure

Description:

- `base.html`: A base template extended by other templates.
- `index.html`: The application's primary access point.
- `listings.html, contact.html, resources.html, submit.html`: Serve different functionalities.
- `styles.css`: Determines the app's visual elements.

- `scripts.js`: Handles dynamic content and AJAX calls.

```
/client/
|-- base.html
|-- index.html
|-- listings.html
|-- contact.html
|-- resources.html
|-- submit.html
|-- styles.css
|-- scripts.js
|-- newfoundland_cover.jpg
```

4 Database Schema

The application relies on a relational database to store and manage data. The following subsections detail the structure of each table, outlining their columns and relationships.

4.1 Users Table

Column	**Type/Constraints**
id	INT, AUTO_INCREMENT, PRIMARY KEY
username	VARCHAR(255)
password	VARCHAR(255)
role	ENUM('user', 'admin')

4.2 Audit Trails Table

Column	**Type/Constraints**
id	INT, AUTO_INCREMENT, PRIMARY KEY
table_name	VARCHAR(255)
action	ENUM('insert', 'update', 'delete')
record_id	INT
modified_by	INT, FOREIGN KEY REFERENCES users(id)
modified_at	TIMESTAMP DEFAULT CURRENT_TIMESTAMP

4.3 Missing Persons Table

Column	Type/Constraints
id	INT, AUTO_INCREMENT, PRIMARY KEY
name	VARCHAR(255)
last_seen_date	DATE
last_seen_location	VARCHAR(255)
status	ENUM('missing', 'found')
picture_url	TEXT
contact_info	TEXT
height	FLOAT
weight	FLOAT
eye_color	VARCHAR(50)
case_number	VARCHAR(255)

4.4 Forums Table

Column	Type/Constraints
id	INT, AUTO_INCREMENT, PRIMARY KEY
missing_person_id	INT, FOREIGN KEY REFERENCES missing_persons(id)
title	TEXT

4.5 Comments Table

Column	Type/Constraints
id	INT, AUTO_INCREMENT, PRIMARY KEY
forum_id	INT, FOREIGN KEY REFERENCES forums(id)
content	TEXT

4.6 Media Table

Column	Type/Constraints
id	INT, AUTO_INCREMENT, PRIMARY KEY
missing_person_id	INT, FOREIGN KEY REFERENCES missing_persons(id)
media_url	TEXT

5 AWS Infrastructure

The app leverages AWS services for hosting, data storage, and security. The Django app resides on an EC2 instance, the database on RDS, and static/media files on S3. AWS IAM ensures secure access, and CloudWatch offers extensive logging and monitoring.

5.0.1 System Architecture Diagram

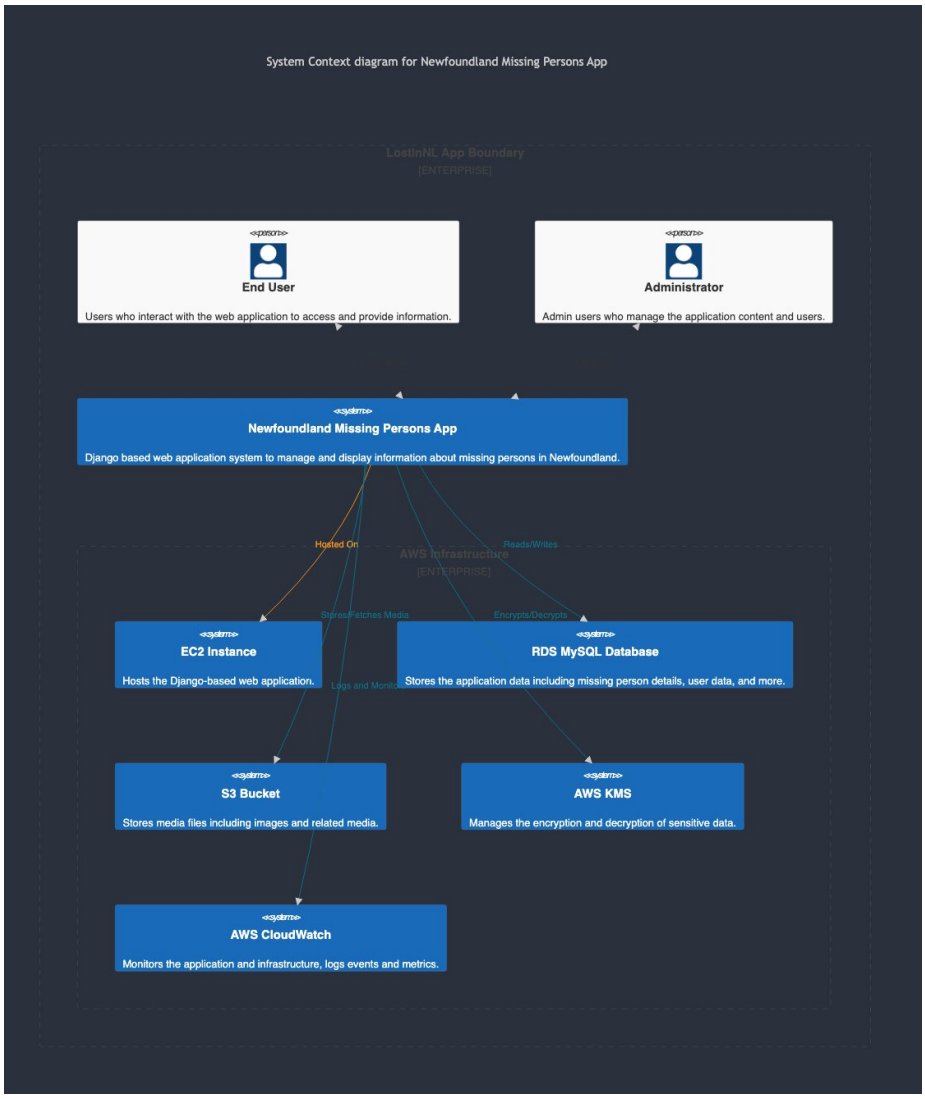

Figure 1: AWS System Architecture detailing interactions among EC2, RDS, S3, and other AWS services.

6 Setup, Deployment, and Maintenance

This section provides a detailed walkthrough, ensuring developers can efficiently set up, run, and maintain the application.

6.1 Local Deployment

1. Clone the repository.
2. Create a virtual environment: `python -m venv venv`
3. Activate the virtual environment and install dependencies: `pip install -r requirements.txt`
4. Run migrations: `python manage.py migrate`
5. Start the Django development server: `python manage.py runserver`

6.2 Server Deployment

1. Secure a domain and server.
2. Install and configure a web server, e.g., Nginx or Apache.
3. Set up the Django app with Gunicorn or a similar WSGI server.
4. Use Certbot for SSL to ensure data security.
5. Configure the server to manage static and media files, possibly using S3.
6. Access the domain to view the live application.

6.3 Routine Maintenance

1. Backup the RDS database regularly.
2. Update all software packages periodically.
3. Monitor server resources and upscale as necessary.

6.4 Monitoring with CloudWatch

1. Configure CloudWatch for detailed logging of the EC2 instance and RDS.
2. Track metrics like CPU and memory usage.
3. Set CloudWatch Alarms for resource constraints or anomalies.

7 Conclusion

The Newfoundland Missing Persons App stands as a testament to the power of technology in addressing societal challenges. This document offers a thorough technical guide to ensure that developers can effectively contribute to, deploy, and maintain this critical platform.