Traffic Accidents Management API

Comprehensive Technical Report & Implementation Documentation

Author: Jaouher
Role: IT/BA Junior Student

January 17, 2026

Abstract

This report details the development lifecycle of the Traffic Accidents Management API, a secure, scalable, and AI-assisted system for reporting, validating, and analyzing traffic incidents. It highlights the API design, database modeling, administrative dashboards, role-based access control, and deployment strategies. The system leverages AI tools for rapid frontend prototyping (A0) and backend optimization (Google Gemini) to accelerate development while maintaining high code quality.

Contents

Chapter 1

Problem Understanding

1.1 Overall Purpose

The Traffic Accidents Management API provides a centralized backend for reporting and managing traffic incidents. It replaces slow, error-prone manual reporting with a secure, real-time digital system, improving response times and data integrity.

1.2 Target Audience

- **Citizens (Public):** Report accidents with location, description, and optional evidence.
- **Traffic Officers (Partners):** Validate reports and update statuses after field verification.
- **Administrators (Internal):** Manage users, approve Officers, and monitor system health.

1.3 Inputs and Outputs

- **Inputs:** Registration details, accident geolocation, descriptions, images, comments, and status updates.
- **Outputs:** JSON responses including authentication tokens, accident report lists, filtered statistics, and system alerts.

Chapter 2

Requirements & Planning

2.1 Functional Features

- Citizen Accident Reporting with geolocation and optional photo evidence.
- State-Machine Status Tracking: Reported → Under Review → Verified/Invalid.
- Real-time comment and discussion threads on accident reports.
- Input validation with Marshmallow to ensure consistent data quality.

2.2 Administrative Features

- Tiered User Registration: Citizens vs Pending Officers.
- Role-Based Access Control (RBAC) using JWT tokens.
- Officer Validation Queue and Admin Audit Dashboard.
- Database CLI access via Docker for direct data management.

2.3 Technical/Infrastructure Features

- Docker Containerization for portable, reproducible deployments.
- Persistent SQLite volumes for data safety.
- Application Factory Pattern for modular Flask architecture.
- AI-Assisted Development: A0 for frontend prototyping, Google Gemini for backend logic and optimization.

2.4 Constraints

- JWT-secured endpoints, password hashing with PBKDF2.
- Enforced enums and validation for roles and statuses.

- Dockerized deployment for consistent dev/prod environments.

Chapter 3

API Design

3.1 Resources & Relationships

- **User:** One Role (Admin, Officer, User), one Status (Pending, Approved, Rejected).
- **Accident:** Belongs to a reporter, modifiable by Officer/Admin, includes location, description, status, and evidence.

3.2 Endpoint Specification

Table 3.1: Key API Endpoints

Endpoint	**Method**	**Access Level**	**Description**
/auth/register	POST	Public	Register a new account.
/auth/login	POST	Public	Login and receive JWT.
/accidents	POST	User	Submit a new accident report.
/accidents	GET	Public/User	List visible accidents.
/accidents/{id}	PATCH	Officer	Verify or Reject a report.
/accidents/{id}/comments	POST	User	Add a comment to an accident.
/admin/users	GET	Admin	List all system users.
/admin/approve	POST	Admin	Approve a pending Officer.

Chapter 4

Data Modeling

4.1 Database Schema

4.1.1 User Table

```
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True, nullable=False)
    password = db.Column(db.String(200), nullable=False)
    role = db.Column(db.Enum(UserRole), default=UserRole.USER)
    status = db.Column(db.Enum(UserStatus), default=UserStatus.PENDING)
```

4.1.2 Accident Table

```
class Accident(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    description = db.Column(db.String(500))
    location = db.Column(db.String(100))
    status = db.Column(db.String(20), default='Reported')
    reporter_id = db.Column(db.Integer, db.ForeignKey('user.id'))
```

4.2 Enums and Validation

- **UserRole:** `ADMIN`, `OFFICER`, `USER`
- **UserStatus:** `PENDING`, `APPROVED`, `REJECTED`

Chapter 5

Implementation

5.1 Technology Stack

- **Language:** Python 3.10
- **Framework:** Flask (Microframework)
- **ORM:** SQLAlchemy
- **Serialization:** Marshmallow
- **Containerization:** Docker & Docker Compose
- **Frontend AI:** A0.dev
- **Backend AI:** Google Gemini

5.2 Code Structure

```
/traffic_accidents
|-- /instance (Database)
|-- /models (Database classes)
|-- /routes (API endpoints)
|-- app.py (Entry point)
|-- config.py (Env settings)
|-- Dockerfile
|-- docker-compose.yml
```

Chapter 6

Deployment & Infrastructure

6.1 Docker Configuration

```
FROM python:3.10-slim
WORKDIR /app
RUN apt-get update && apt-get install -y --no-install-recommends \
    build-essential sqlite3 \
    && rm -rf /var/lib/apt/lists/*
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
EXPOSE 5000
CMD ["python", "app.py"]
```

Listing 6.1: Production Dockerfile

6.2 Database Management via CLI

```
docker exec -it traffic_api sqlite3 instance/data.db
UPDATE user SET role = 'ADMIN', status = 'APPROVED' WHERE username = '
   jaouher';
```

Chapter 7

Testing & Security

7.1 Security Implementation

- JWT authentication and Role-Based Access Control.
- PBKDF2 password hashing using passlib.
- Endpoint guards for Admin, Officer, and User roles.

7.2 Testing Strategy

1. Unit Testing: Model logic and hashing functions.
2. Integration Testing: Full API flows using Insomnia/Postman.
3. Manual Verification: SQLite CLI checks for enums and relationships.

Chapter 8

Conclusion

The Traffic Accidents Management API provides a secure, scalable, and AI-assisted backend system for civic traffic reporting. By integrating role-based access control, Dockerized deployments, and AI-assisted development tools (A0 and Gemini), the system ensures rapid development, high security, and maintainable architecture. The platform is ready for real-world usage and future expansion, including analytics, notifications, and monetization features.