Name	Student Number
Nicholas Dobson	u23671964
Shaylin Govender	u20498952
Lonwabo	u22516949
Mpho	u22668323
Aryan Mohanlall	u23565536

Traffic Guardian AWS Technical Installation Manual

Team: Quantum Quenchers

quantumquenchers@gmail.com

COS301

Capstone Project

University of Pretoria

1. Introduction

Traffic Guardian is a comprehensive AI-powered traffic incident detection and management platform consisting of three main components that work together to **provide real-time traffic monitoring** and i**ncident response capabilities**:

System Architecture Overview

Frontend Component (React.js)

- Modern web application built with React 19.1.0 and TypeScript 4.9.5
- Real-time dashboard with Material-UI components
- Interactive maps using Leaflet and React-Leaflet
- Live video streaming and incident visualisation
- Socket.io client for real-time updates

Backend API (Node.js)

- RESTful API server built with Express 4.18.2
- PostgreSQL database integration with SSL support
- Real-time communication using Socket.io 4.8.1
- Authentication and authorisation system
- Rate limiting and security middleware

AI Model Component (Python)

- Traffic incident detection using YOLO (You Only Look Once) models
- Computer vision processing with OpenCV 4.8.1.78
- Real-time video analysis and classification
- Machine learning pipeline with PyTorch

What Needs Installation

To run the **Traffic Guardian** system, you need to install and configure:

1. Development Environment: Node.js 18+, Python 3.10+, Git
2. Database: PostgreSQL 12+ with SSL support
3. Frontend Dependencies: React ecosystem, TypeScript, testing tools
4. Backend Dependencies: Express server, database drivers, security middleware
5. AI Model Dependencies: Python ML libraries, YOLO models, OpenCV
6. Development Tools: Code editors, linters, testing frameworks

Prerequisites

System Requirements

Minimum Hardware Requirements:

- RAM: 8GB (16GB recommended for AI model training)
- Storage: 10GB free disk space
- CPU: Multi-core processor (Intel i5/AMD Ryzen 5 or better)
- GPU: Optional but recommended for AI model inference (CUDA-compatible)

Supported Operating Systems:

- Windows 10/11
- macOS 10.15+ (Catalina or later)
- Ubuntu 20.04+ / Debian 10+
- CentOS 8+ / RHEL 8+

Required Software

1. Node.js (Version 18+)

Windows:

1. Download Node.js from https://nodejs.org/
2. Choose "LTS" version (18.x or later)
3. Run the installer with administrator privileges
4. Verify installation:

```
node --version npm --version
```

macOS:

```
# Using Homebrew (recommended)
brew install node@18

# Or download from nodejs.org
# Verify installation
node --version
npm --version
```

Linux (Ubuntu/Debian):

```
# Update package index
sudo apt update
# Install Node.js 18.x
curl -fsSL https://deb.nodesource.com/setup_18.x | sudo -E bash -
sudo apt-get install -y nodejs
# Verify installation
node --version
npm --version
```

Linux (CentOS/RHEL):

```
# Install Node.js 18.x
curl -fsSL https://rpm.nodesource.com/setup_18.x | sudo bash -
sudo yum install -y nodejs

# Verify installation
node --version
npm --version
```

2. Python (Version 3.10+)

Windows:

1. Download Python from https://python.org/downloads/
2. Choose Python 3.10 or later
3. Important: Check "Add Python to PATH" during installation
4. Verify installation:

```
python --version pip --version
```

macOS:

```
# Using Homebrew (recommended)
brew install python@3.10
# Or using pyenv
pyenv install 3.10.13
pyenv global 3.10.13
# Verify installation
python3 --version
pip3 --version
```

Linux (Ubuntu/Debian):

```
# Install Python 3.10
sudo apt update
sudo apt install python3.10 python3.10-venv python3.10-pip

# Verify installation
python3 --version
pip3 --version
```

Linux (CentOS/RHEL):

```
# Enable EPEL repository
sudo dnf install epel-release

# Install Python 3.10
sudo dnf install python310 python3-pip

# Verify installation
python3 --version
pip3 --version
```

3. Git (Latest Version)

Windows:

1. Download Git from https://git-scm.com/download/win
2. Run the installer with default settings
3. Verify: `git --version`

macOS:

```
# Using Homebrew
brew install git

# Or using Xcode Command Line Tools
xcode-select --install
```

Linux:

```
# Ubuntu/Debian
sudo apt install git

# CentOS/RHEL
sudo dnf install git
```

4. PostgreSQL (Version 12+)

Windows:

1. Download PostgreSQL from https://www.postgresql.org/download/windows/
2. Run the installer and remember the superuser password
3. Default port: 5432

macOS:

```
# Using Homebrew
brew install postgresql@14
brew services start postgresql@14

# Create database user
createuser -s postgres
```

Linux (Ubuntu/Debian):

```
# Install PostgreSQL
sudo apt update
sudo apt install postgresql postgresql-contrib

# Start PostgreSQL service
sudo systemctl start postgresql
sudo systemctl enable postgresql

# Create database user
sudo -u postgres createuser --superuser $USER
```

5. Development Tools (Optional but Recommended)

Visual Studio Code:

- Download from https://code.visualstudio.com/
- Recommended extensions:
  - TypeScript and JavaScript Language Features
  - Python
  - ESLint
  - Prettier
  - GitLens

Installation

Step 1: Clone the Repository

```
# Clone the main branch
git clone
https://github.com/COS301-SE-2025/traffic-guardian.git

# Navigate to project directory
cd traffic-guardian

# Verify repository structure
ls -la
```

Expected directory structure:

```
traffic-guardian/
├── API/                 # Backend API
├── frontend/           # React frontend
├── AI_Model_BB/        # AI model components
├── docs/               # Documentation
├── .github/            # GitHub Actions
└── README.markdown
```

Step 2: Database Setup

Option A: Local PostgreSQL Setup

1. Create Database:
    a. Connect to PostgreSQL as a superuser/owner

```
sudo -u postgres psql
```

.   b. Create database and user:

```
CREATE DATABASE traffic_guardian; CREATE USER traffic_user WITH
ENCRYPTED PASSWORD 'your_secure_password'; GRANT ALL PRIVILEGES ON
DATABASE traffic_guardian TO traffic_user; \q
```

2. Initialise Database Schema:

   Navigate to the API directory

```
cd API
```

Run database schema (if schema.sql exists)

```
psql -h localhost -d traffic_guardian -U traffic_user -f schema.sql
```

Option B: Remote PostgreSQL (AWS RDS/Other Cloud Provider)

If using a cloud database, ensure you have:

- Database host URL
- Database name
- Username and password
- SSL certificate (if required)

Step 3: Backend API Installation

```
# Navigate to API directory
cd API
# Install dependencies
npm install
# Create environment file
cp envExample.txt .env
```

Configure API Environment Variables (edit .env):

```
# Database Configuration
DATABASE_USERNAME=
DATABASE_HOST=
DATABASE_NAME=
DATABASE_PASSWORD=
DATABASE_PORT=5432
# API Keys (obtain from respective services)
# Server Configuration
PORT=
NODE_ENV=development
```

Get Required API Keys:

1. Weather API Key:
   - Visit WeatherAPI.com
   - Sign up for a free account
   - Get API key from the dashboard
2. TomTom API Key:
   - Visit TomTom Developer Portal
   - Create an account and get the API key

Verify Backend Installation:

```
# Test installation
npm test
# Start development server
npm run dev
```

Step 4: Frontend Installation

```
# Navigate to frontend directory
cd ../frontend
# Install dependencies
npm install
# Create environment file
cp envExample.txt .env
```

Configure Frontend Environment Variables (edit .env):

```
# API Configuration
REACT_APP_API_URL=http://localhost:5000
REACT_APP_SOCKET_URL=http://localhost:5000
# Map Configuration (if needed)
REACT_APP_MAP_API_KEY=your_map_api_key
```

Verify Frontend Installation:

```
# Run type checking
npm run type-check
# Run linting
npm run lint
# Run tests
npm test
# Start development server
npm start
```

Step 5: AI Model Setup

```
# Navigate to AI model directory
cd ../AI_Model_BB/Code

# Create Python virtual environment
python3 -m venv venv
# Activate virtual environment

# Windows:
venv\Scripts\activate
# macOS/Linux:
source venv/bin/activate

# Upgrade pip
pip install --upgrade pip

# Install dependencies
pip install -r requirements.txt
```

Download YOLO Models:

The system uses pre-trained YOLO models. Ensure these files exist:

- yolov5s.pt - YOLO v5 model
- yolov8s.pt - YOLO v8 model

If not present, they will be automatically downloaded on the first run.

Verify AI Model Installation:

```
# Test Python dependencies
python -c "import cv2, torch, ultralytics; print('All dependencies
installed successfully')"

# Run basic tests
cd ../Testing
python -m pytest run_tests.py -v
```

Configuration

Environment Configuration Files

Backend API Configuration (API/.env)

```
# Database
DATABASE_USERNAME=traffic_user
DATABASE_HOST=localhost
DATABASE_NAME=traffic_guardian
DATABASE_PASSWORD=your_password
DATABASE_PORT=5432
DATABASE_SSL=false

# External APIs
WEATHERAPI=your_weather_api_key
TOMTOMAPI=your_tomtom_api_key

# Server
PORT=5000
NODE_ENV=development
CORS_ORIGIN=http://localhost:3000

# Security
JWT_SECRET=your_jwt_secret_key
SESSION_SECRET=your_session_secret
```

Frontend Configuration (frontend/.env)

```
# API Endpoints
REACT_APP_API_URL=http://localhost:5000
REACT_APP_SOCKET_URL=http://localhost:5000

# Feature Flags
REACT_APP_ENABLE_DEV_TOOLS=true
REACT_APP_LOG_LEVEL=debug
```

SSL Configuration (Production)

For production deployment with SSL:

Backend SSL Configuration:

```
# In API/.env
DATABASE_SSL=true
HTTPS_ENABLED=true
SSL_CERT_PATH=/path/to/certificate.crt
SSL_KEY_PATH=/path/to/private.key
```

Database Migration (If Applicable)

```
# Navigate to API directory
cd API

# Run migrations (if migration scripts exist)
npm run migrate

# Or manually run schema
psql -h localhost -d traffic_guardian -U traffic_user -f schema.sql
```

Deployment & Running

Development Mode

Method 1: Run All Components Separately

Terminal 1 - Backend API:

```
cd API
npm run dev
# Server will start on http://localhost:5000
```

Terminal 2 - Frontend:

```
cd frontend
npm start
# Application will open at http://localhost:3000
```

Terminal 3 - AI Model (if needed):

```
cd AI_Model_BB/Code
source venv/bin/activate  # or venv\Scripts\activate on Windows
python incident_detection_system.py
```

Method 2: Using Process Manager (Recommended)

Create a start-dev.sh script in the root directory:

```
#!/bin/bash

# Start backend API
cd API && npm run dev &
BACKEND_PID=$!

# Start frontend
cd ../frontend && npm start &
FRONTEND_PID=$!

# Wait for processes
wait $BACKEND_PID $FRONTEND_PID
```

Make executable and run:

```
chmod +x start-dev.sh
./start-dev.sh
```

Production Mode

Backend Production Build:

```
cd API
NODE_ENV=production npm start
```

Frontend Production Build:

```
cd frontend
npm run build
# Serve build files with a web server like nginx or serve
npx serve -s build -l 3000
```

Verification

System Health Checks

1. Backend API Verification

```
# Check API health
curl http://localhost:5000/health

# Expected response:
# {"status": "healthy", "timestamp": "2025-01-15T10:30:00Z"}

# Test database connection
curl http://localhost:5000/api/incidents

# Expected: JSON response with incidents data or empty array
```

2. Frontend Verification

Open a browser and navigate to: http://localhost:3000

Verify:

- Page loads without errors
- Console shows no critical errors
- Socket.io connection established
- Navigation works correctly

3. AI Model Verification

```
cd AI_Model_BB/Code
source venv/bin/activate

# Test model loading
python -c "
from incident_detection_system import load_model
model = load_model()
print('Model loaded successfully')
"
```

Running Tests

Backend Tests:

```
cd API
npm test
```

Frontend Tests:

```
cd frontend
npm test
npm run test:coverage
```

AI Model Tests:

```
cd AI_Model_BB/Testing
source ../Code/venv/bin/activate
python -m pytest run_tests.py -v
```

End-to-End Tests:

```
cd frontend
npx cypress run
```

Troubleshooting

Common Issues

1. Node.js Version Issues

Problem: npm install fails with version conflicts

Solution:

```
# Check Node.js version
node --version

# If version < 18, update Node.js
# Clear npm cache
npm cache clean --force

# Delete node_modules and reinstall
rm -rf node_modules package-lock.json
npm install
```

2. Python Dependencies Issues

Problem: pip install -r requirements.txt fails

Solutions:

```
# Update pip
pip install --upgrade pip

# Install system dependencies (Ubuntu/Debian)
sudo apt install python3-dev build-essential

# Install system dependencies (macOS)
xcode-select --install

# Install system dependencies (Windows)
# Install Visual Studio Build Tools
```

3. Database Connection Issues

Problem: Cannot connect to PostgreSQL

Solutions:

```
# Check PostgreSQL is running
sudo systemctl status postgresql  # Linux
brew services list | grep postgresql  # macOS

# Check connection with psql
psql -h localhost -U traffic_user -d traffic_guardian

# Verify .env file configuration
cat API/.env
```

4. Port Already in Use

Problem: Error: listen EADDRINUSE: address already in use :::3000

Solutions:

```
# Find process using port
lsof -i :3000  # macOS/Linux
netstat -ano | findstr :3000  # Windows

# Kill process
kill -9 <PID>  # macOS/Linux
taskkill /F /PID <PID>  # Windows

# Or use different port
PORT=3001 npm start
```

5. CORS Issues

Problem: Frontend cannot connect to the backend API

Solution:

```
# Check CORS configuration in API/.env
CORS_ORIGIN=http://localhost:3000

# Or allow all origins for development (not recommended for production)
CORS_ORIGIN=*
```

6. SSL/TLS Issues (Production)

Problem: Database SSL connection fails

Solutions:

```
# Check SSL configuration
DATABASE_SSL=true

# For development, disable SSL
DATABASE_SSL=false

# For production, ensure SSL certificates are valid
```

Getting Help

1. **Check System Logs**:
   - **Backend**: Check console output where npm run dev is running
   - **Frontend**: Check browser console (F12)
   - **AI Model**: Check Python script output
2. **Verify Environment Variables**:
   Check if .env files exist and have correct values

```
cat API/.env cat frontend/.env
```

3. **Test Individual Components**:
   Test backend API only

```
cd API && npm run dev
```

4. **Test frontend only:**

```
cd frontend && npm start
```

5. **Test database connection:**

```
psql -h localhost -U traffic_user -d traffic_guardian -c "SELECT
version();"
```

6. **Check Dependencies**:
   Verify all required software is installed:

```
node --version python3 --version psql --version git
--version
```

User Manual

Once the system is successfully installed and running, please refer to the comprehensive user manual for detailed instructions on how to use the Traffic Guardian system here

The user manual covers:

- System navigation and dashboard overview
- Incident management workflows
- Real-time monitoring features
- Alert configuration and management
- Analytics and reporting tools
- Administrative functions

Quick Start Guide

1. **Access the System**: Navigate to http://localhost:3000 in your web browser
2. **Login/Register**: Create an account or use provided credentials
3. **Dashboard**: View the main dashboard with real-time traffic data
4. **Monitor Incidents**: Use the incident management interface to view and manage traffic incidents
5. **Configure Alerts**: Set up automated alerts for specific incident types
6. **Analyse Data**: Use the analytics section to review historical data and trends

Support and Maintenance

Regular Maintenance Tasks

Update Dependencies:

Update Node.js dependencies:

```
npm update
```

Update Python dependencies:

```
pip list --outdated pip install --upgrade package_name
```

Database Maintenance:

Regular database backup:

```
pg_dump traffic_guardian > backup_$(date +%Y%m%d).sql
```

Getting Support

For technical support and questions:

- GitHub Issues
- Team Contact: Quantum Quenchers Development Team
- Documentation: Check the ReadMe for additional documentation