CS7CS3 Advanced Software Engineering Group Project

Functional and Technical Architecture

Project Name: Sustainable City Management

Group: 9
Aoife Walshe
Oisin Power
Parker Kavanagh
Darragh Hardiman Smyth
Charlie O'Malley
Diana Pylova

1. Development Approach: Extreme Programming (XP)

We will use Extreme Programming which focuses on delivering working software in short cycles with continuous feedback.

How We'll Work:

- Break work into 4 iterations (1 week iterations) with working features delivered every 3 weeks
- Each feature is described as a user story with clear acceptance criteria (what it needs to do)
- Regular stand-ups (15 min) to check progress and remove blockers
- All code is tested before it's merged to ensure quality
- Code reviews happen through pair programming, not separate reviews

Iteration Breakdown:

1. **Iteration 1**: Build foundation - data collection, external APIs, security, logging
2. **Iteration 2**: Build traffic and routing - traffic predictions, route optimization, environmental impact, alerts
3. **Iteration 3**: Build analytics and planning - data analysis, urban planning orchestration, bike-sharing, UI framework
4. **Iteration 4**: Integration and polish - end-to-end testing, performance optimization, security hardening, deployment

2. Pair Programming Strategy

All developers work in pairs for critical components. This ensures knowledge sharing and catches bugs early.

Pair Rotation:

- 3 pairs rotate every 2-3 stories to spread expertise across the team
- Pairs switch roles every 30 minutes to stay fresh

Team Pairing:

- Pair A: Aoife & Oisin (Infrastructure & Backend)
- Pair B: Parker & Darragh (Data & Algorithms)
- Pair C: Charlie & Diana (UI & Analytics)

Pairs rotate between iterations so everyone learns all parts of the system.

3. Coding Standards

Languages & Frameworks:

- Backend: Python and AWS
- Frontend: Python
- Database: MongoDB
- Real-time alerts: MQTT messaging

File Organization: Backend organized by component (traffic-management, route-planning, etc.) with each having controller, service, and database layers. Frontend organized by page and reusable components.

Naming Rules:

- Variables & functions: camelCase (e.g., getUserRoutes)
- Classes & components: PascalCase (e.g., TrafficManager)
- Database tables: snake_case plural (e.g., traffic_alerts)
- React components: Match filename to component name

Code Formatting:

- 2 space indentation
- Max 100 characters per line
- Use ESLint for automatic checking
- Use Prettier to auto-format code on save

Documentation:

- Add comments to all exported functions explaining what they do, inputs, outputs, and errors
- Each component folder has a README explaining its purpose and how to use it
- All API endpoints documented in Swagger/OpenAPI format

Testing:

- Minimum 80% test coverage (100% for security components)
- All public functions must have at least one passing test and one error test
- Tests organized alongside source code in __tests__ folders
- Use Pytest testing framework

Security:

- No hardcoded passwords or API keys - use environment variables
- All external input validated before use
- All passwords hashed with bcrypt
- HTTPS enforced on all APIs with TLS 1.3
- Data encrypted at rest (AES-256) and in transit
- User authentication via JWT tokens (1-hour expiry)

- Role-based access control (Admin, Planner, User)

Performance Targets:

- Page loads in <2 seconds
- API responses in <5 seconds for predictions, <1 second for data retrieval
- Real-time dashboards update every 30-60 seconds
- Database queries must complete in <200ms
- Cache traffic predictions for 24 hours, bike availability for 15 minutes

Git Workflow:

- All changes via pull requests with tests passing and code review approved
- Branch naming: feature/name, fix/issue, chore/task
- Commit messages: feat:, fix:, docs:, test:, refactor: prefixes
- Main branch protected - never commit directly

4. Development Environment

What You Need:

- Docker, MongoDB, Git, VS Code

Getting Started:

1. Clone repository
2. Copy .env.example to .env with your settings
3. Run npm install
4. Run docker-compose up to start database and services
5. Run npm test to verify everything works
6. Run npm run dev to start development server

Pre-commit hooks automatically check code formatting and run tests before allowing commits.