Implementation and Test Deliverable (ITD)

Best Bike Paths (BBP)

Road Application

Project Name: Best Bike Paths (BBP)

Document Type: Implementation and Test Deliverable (ITD)

Version: 1.0

Date: January 31, 2026

Course: Software Engineering 2

Team Members

Name	Student ID	Email
Hu Xu Yan	Student ID	email@polimi.it
Team Member 2	Student ID	email@polimi.it
Team Member 3	Student ID	email@polimi.it

Politecnico di Milano
Academic Year 2025-2026

Table of Contents

1 Introduction

The Best Bike Paths (BBP) application is a comprehensive road condition monitoring and route planning system designed specifically for cyclists. The system enables users to:

- Report and track road conditions (potholes, surface quality, obstacles)
- Plan optimal cycling routes based on road quality data
- Contribute to a community-driven road quality database
- Receive intelligent route recommendations with safety scoring

This document describes the implementation details of the BBP prototype, including the architectural decisions, implemented features, testing procedures, and installation instructions.

1.1 Purpose

This ITD document serves as the technical reference for the BBP implementation, providing:

- Documentation of all implemented functionalities
- Justification for technology choices
- Comprehensive testing results
- Step-by-step installation guide for deployment

1.2 Definitions, Acronyms, and Abbreviations

Term	Definition
BBP	Best Bike Paths
OSRM	Open Source Routing Machine
API	Application Programming Interface
REST	Representational State Transfer
CRUD	Create, Read, Update, Delete
i18n	Internationalization
SPA	Single Page Application

2 Scope of the Document

This document covers the complete implementation of the BBP road application prototype, including:

1. **Backend Implementation**: FastAPI-based REST API server with in-memory data storage
2. **Frontend Implementation**: React-based Single Page Application with interactive maps
3. **Integration**: OSRM routing service integration for real road geometry
4. **Testing**: Unit tests and system-level test cases
5. **Deployment**: Installation and configuration instructions

2.1 Document Boundaries

This document focuses on:

- ✓ Implemented prototype features
- ✓ Technical architecture and code structure
- ✓ Testing methodology and results
- ✓ Installation procedures

This document does NOT cover:

- ✗ Production deployment configurations
- ✗ Performance optimization strategies
- ✗ Future enhancement roadmap

3 Implemented Requirements and Functions

3.1 Implemented Features Overview

Category	Feature	Status	Priority
User Management	User Registration	✓	High
	User Settings	✓	Medium
	Multi-language Support	✓	Medium
Segment Management	Create Road Segments	✓	High
	View Segments on Map	✓	High
	Segment Status Tracking	✓	High
Report System	Submit Condition Reports	✓	High
	Report Confirmation	✓	High
	Weighted Voting Aggregation	✓	High
	Batch Confirmation	✓	Medium
Route Planning	OSRM Route Integration	✓	High
	Generate & Evaluate Algorithm	✓	High
	Route Quality Scoring	✓	High
	Multiple Route Alternatives	✓	Medium
Trip Management	Create Trips	✓	High
	Trip History	✓	Medium
	Privacy Protection	✓	High
Auto Detection	Sensor-based Detection	✓	Medium
	Auto-confirm Reports	✓	Low
Weather Service	Weather Information	✓	Low
	Cycling Recommendations	✓	Low
Internationalization	English (en)	✓	Medium
	Chinese (zh)	✓	Medium
	Italian (it)	✓	Medium

3.2 Detailed Feature Descriptions

3.2.1 User Management

Implementation: Users can register with a username. The system stores user preferences including language, dark mode, notification settings, and default map configurations.

```
API Endpoints:
- POST /api/users - Create or get user
- GET /api/users - List all users
- GET /api/users/{user_id}/settings - Get user settings
```

```
- PUT /api/users/{user_id}/settings - Update user settings
- PATCH /api/users/{user_id}/settings - Partial update settings
```

Motivation: User management is essential for personalized experience and tracking individual contributions to the road condition database.

3.2.2 Road Segment Management

Implementation: Road segments are defined by start and end GPS coordinates with associated status (optimal, medium, suboptimal, maintenance) and optional obstacle information.

Status Classification:

Status	Description	Color
Optimal	Excellent road condition	Green
Medium	Fair condition, minor issues	Yellow
Suboptimal	Poor condition, caution advised	Orange
Maintenance	Under repair, avoid if possible	Red

3.2.3 Report System with Weighted Voting

Implementation: Users submit reports for road segments. The system uses a weighted voting algorithm to aggregate reports and automatically update segment status.

Weighting Algorithm:

Weight Calculation:
- Base weight: 1.0
- Fresh report (< 30 days): $\times 2.0$ multiplier
- Confirmed report: $\times 1.5$ multiplier

Status Determination:
- `negative_score >= 0.6` $\rightarrow$ "maintenance"
- `negative_score >= 0.3` $\rightarrow$ "medium"
- `positive_score > 0.7` $\rightarrow$ "optimal"
- Otherwise $\rightarrow$ "medium"

Motivation: Weighted voting ensures that recent and verified reports have greater influence on road status, improving data accuracy over time.

3.2.4 Route Planning with Quality Scoring

Implementation: The "Generate & Evaluate" algorithm provides intelligent route recommendations:

Phase 1: Candidate Generation
1. Direct route from OSRM
2. Routes via perpendicular waypoints (15% offset)

3. Deduplication of similar routes (>80% overlap)

Phase 2: Scoring

$$\text{Score} = \text{Distance} + \text{Penalty}$$

Factor	safety_first	balanced	shortest
Pothole	1200	500	100
Maintenance	10.0×length	4.0×length	0.8×length
Bad Road	5.0×length	2.0×length	0.3×length
Medium Road	1.5×length	0.5×length	0.1×length

Phase 3: Tagging

- "Recommended" - Top-ranked route
- "Alternative" - Other viable routes
- "Best Surface" - Quality score > 90
- "Fastest" - Shortest distance
- "Bumpy", "Road Work", "Poor Surface" - Warning tags

3.2.5 Privacy By Design

Implementation: Location privacy protection following GDPR principles:

1. **Location Obfuscation**: Start/end points fuzzed by ~150m
2. **Coordinate Truncation**: Public coordinates rounded to 3 decimal places (~100m precision)
3. **Trip Geometry Sanitization**: First and last 150m of trip routes obfuscated
4. **Private Data Separation**: Raw coordinates stored separately, only accessible by owner

3.3 Excluded Features and Justification

Feature	Reason for Exclusion
User Authentication (JWT)	Simplified for prototype; username-based identification sufficient for demo
Persistent Database	In-memory storage suitable for prototype; reduces deployment complexity
Real-time Notifications	Would require WebSocket infrastructure; deferred to future iteration
Mobile App	Web-based SPA provides cross-platform access; native app not required for prototype
Social Features	Core functionality prioritized; social features are enhancement, not essential
Payment Integration	Out of scope for academic prototype

4 Adopted Development Frameworks

4.1 Programming Languages

4.1.1 Backend: Python 3.10+

Version	Python 3.10 or higher
Paradigm	Multi-paradigm (OOP, Functional)

Advantages:
- ✓ Rapid development with clean, readable syntax
- ✓ Excellent library ecosystem (FastAPI, Pydantic, httpx)
- ✓ Strong type hints support for better code quality
- ✓ Easy integration with scientific computing libraries
- ✓ Large community and extensive documentation

Disadvantages:
- ✗ Slower execution compared to compiled languages
- ✗ Global Interpreter Lock (GIL) limits true parallelism
- ✗ Memory consumption higher than C/C++

Justification: Python's rapid development capabilities and the excellent FastAPI framework make it ideal for building REST APIs quickly while maintaining code quality through type hints.

4.1.2 Frontend: TypeScript 5.9

Version	TypeScript ~5.9.3
Paradigm	Object-oriented, Functional

Advantages:
- ✓ Static type checking catches errors at compile time
- ✓ Enhanced IDE support with autocompletion
- ✓ Better code maintainability and refactoring
- ✓ Seamless integration with React ecosystem
- ✓ Industry standard for modern web development

Disadvantages:
- ✗ Additional compilation step required
- ✗ Learning curve for developers new to static typing
- ✗ Type definitions may lag behind JavaScript libraries

Justification: TypeScript provides the type safety necessary for maintaining a complex React application while leveraging the vast JavaScript ecosystem.

4.2 Frameworks

4.2.1 Backend Framework: FastAPI

Advantages:
- ✓ Automatic OpenAPI (Swagger) documentation generation
- ✓ Built-in request validation via Pydantic
- ✓ Native async/await support for high performance

- ✓ Dependency injection system
- ✓ Excellent developer experience

Disadvantages:
- ✗ Relatively new compared to Django/Flask
- ✗ Smaller plugin ecosystem
- ✗ Async programming complexity for beginners

4.2.2 Frontend Framework: React 19

Advantages:
- ✓ Component-based architecture promotes reusability
- ✓ Virtual DOM for efficient rendering
- ✓ Hooks API for state management
- ✓ Large ecosystem and community support
- ✓ Excellent developer tools

Disadvantages:
- ✗ Steep learning curve for beginners
- ✗ Frequent updates may require migration effort
- ✗ JSX syntax unconventional for some developers

4.2.3 Build Tool: Vite

Advantages:
- ✓ Extremely fast hot module replacement (HMR)
- ✓ Native ES modules support
- ✓ Optimized production builds
- ✓ Simple configuration

Disadvantages:
- ✗ Less mature than Webpack
- ✗ Some plugins may not be compatible

4.3 Libraries and Middleware

4.3.1 Backend Libraries

Library	Purpose	Version
uvicorn	ASGI server for FastAPI	Latest
pydantic	Data validation and serialization	Latest
httpx	HTTP client for OSRM API calls	Latest
python-multipart	Form data parsing	Latest

4.3.2 Frontend Libraries

Library	Purpose	Version
react-leaflet	React components for Leaflet maps	5.0.0
leaflet	Interactive map library	1.9.4

4.4 External APIs

4.4.1 OSRM (Open Source Routing Machine)

Base URL	`http://router.project-osrm.org`
Profile	Bicycle routing
Timeout	10 seconds

Endpoints Used:

```
GET /route/v1/bike/{coordinates}
Parameters:
- overview=full
- geometries=geojson
- alternatives=true/false
- steps=true
```

Advantages:
- ✓ Free public API for bicycle routing
- ✓ Real road geometry data
- ✓ Support for alternative routes
- ✓ Turn-by-turn navigation data

Disadvantages:
- ✗ Rate limits on public server
- ✗ No guaranteed uptime
- ✗ Limited to road network coverage

Fallback Strategy: When OSRM is unavailable, the system falls back to geometric interpolation using Haversine distance calculations.

5 Structure of the Source Code

5.1 Project Directory Structure

```
bbp-road-app/
├── README.md                    # Project documentation
├── backend/                     # Python FastAPI backend
│   ├── main.py                  # Main application (2144 lines)
│   ├── requirements.txt         # Python dependencies
│   └── test_routing.py          # Routing algorithm tests
├── frontend/                    # React TypeScript frontend
│   ├── index.html               # HTML entry point
│   ├── package.json             # Node.js dependencies
│   ├── tsconfig.json            # TypeScript configuration
│   ├── vite.config.ts           # Vite build configuration
│   └── src/                     # Source code
│       ├── main.tsx             # React entry point
│       ├── App.tsx              # Main application component
│       ├── AppContext.tsx       # Global state management
│       ├── api.ts               # API client functions
│       ├── Layout.tsx           # Page layout component
│       ├── DashboardPage.tsx    # Dashboard with statistics
│       ├── SegmentsPage.tsx     # Segment management
│       ├── ReportsPage.tsx      # Report submission
│       ├── RoutePlanningPage.tsx # Route planning interface
│       ├── TripsPage.tsx        # Trip creation
│       ├── TripHistoryPage.tsx  # Trip history view
│       ├── AutoDetectionPage.tsx # Auto-detection feature
│       ├── SettingsPage.tsx     # User settings
│       └── MapView.tsx          # Leaflet map component
└── node-v20.19.0-win-x64/       # Bundled Node.js runtime
```

5.2 Backend Architecture

The backend follows a **layered service architecture**:

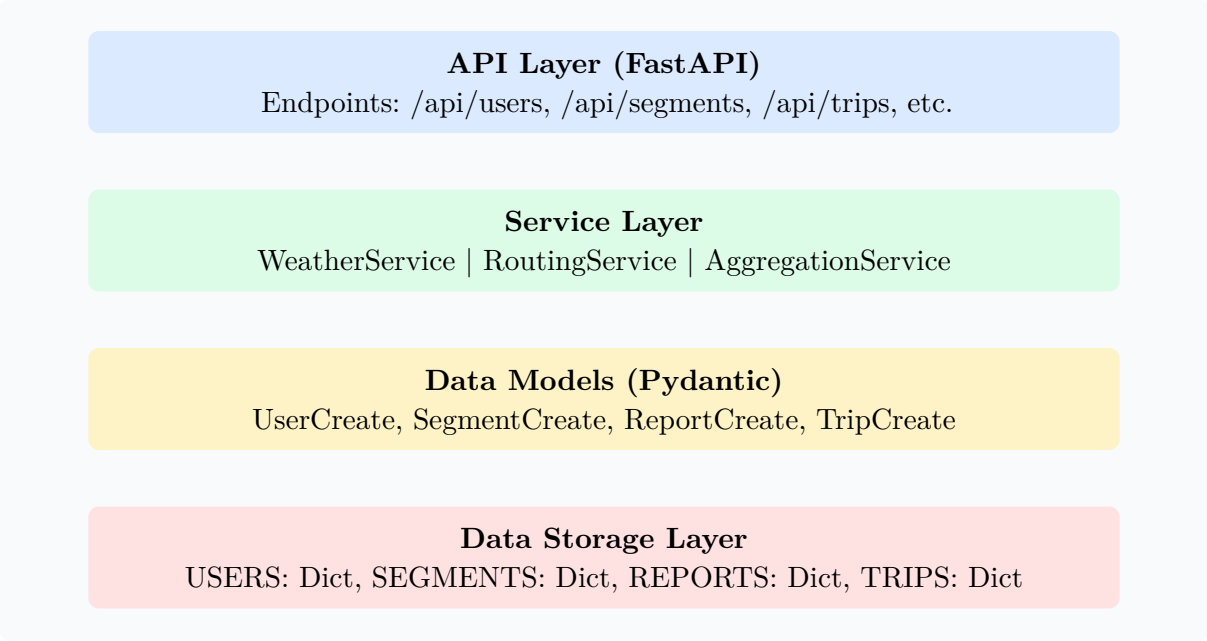

Figure 1: Backend Architecture Diagram

5.2.1 Module Organization (main.py sections)

Lines	Section	Description
1-130	i18n	Translation dictionaries and helper functions
131-230	Weather Service	Mock weather data generation
231-300	OSRM Integration	Route fetching from OSRM API
300-400	Route Utilities	Perpendicular waypoint calculation
400-500	Geo Utilities	Haversine distance, path generation
500-650	Privacy Helpers	Location obfuscation
650-720	Aggregation Service	Weighted voting algorithm
720-800	Data Storage	In-memory stores and Pydantic models
800-900	Demo Data	Sample data initialization
900-1200	CRUD Endpoints	Users, Segments, Reports, Trips
1200-1350	Auto-Detection	Sensor-based pothole detection
1350-1500	Settings API	User settings, weather endpoints
1500-1700	Route Preview	Multi-route preview, i18n API
1700-2144	Path Search	Generate & Evaluate algorithm

5.3 Frontend Architecture

The frontend follows a **component-based architecture** with centralized state management.

5.3.1 Component Descriptions

Component	Responsibility	Lines
App.tsx	Root component, routing logic, user state	~70

AppContext.tsx	Global state provider (dark mode, i18n)	~330
Layout.tsx	Navigation sidebar, header, page structure	~200
DashboardPage.tsx	Statistics cards, segment map	~314
SegmentsPage.tsx	Segment CRUD, map visualization	~241
ReportsPage.tsx	Report submission, confirmation	~281
RoutePlanningPage.tsx	Route search, multi-route display	~618
TripsPage.tsx	Trip creation with OSRM	~200
TripHistoryPage.tsx	Historical trip listing	~150
AutoDetectionPage.tsx	Sensor-based detection interface	~200
SettingsPage.tsx	User preferences configuration	~250
MapView.tsx	Reusable Leaflet map wrapper	~150
api.ts	HTTP client, type definitions	~337

5.4 Data Flow Diagram

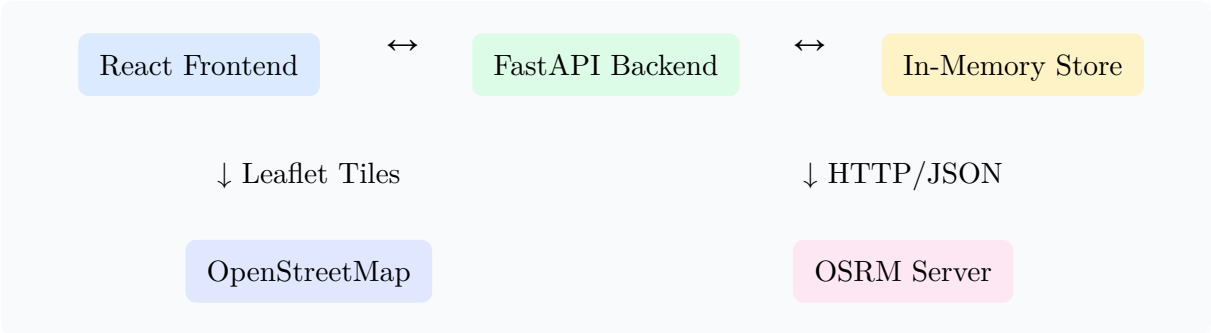

Figure 2: System Data Flow

6 Testing

6.1 Testing Strategy

The testing strategy follows the Test Plan outlined in the Design Document (DD), with focus on:

1. **Unit Testing**: Individual function testing
2. **Integration Testing**: API endpoint testing
3. **System Testing**: End-to-end user scenario testing

6.2 Test Environment

Component	Configuration
Backend	Python 3.10+, FastAPI TestClient
Frontend	Manual testing in Chrome/Firefox
OSRM	Public server (router.project-osrm.org)
Map Tiles	OpenStreetMap

6.3 System Test Cases

6.3.1 Test Case 1: User Registration and Login

Test ID	STC-001
Objective	Verify user can register and login successfully
Preconditions	Backend server running, Frontend accessible
Input	Username: “testuser”
Steps	1. Navigate to login page 2. Enter username “testuser” 3. Click “Login” button
Expected Output	User redirected to dashboard, username displayed in header
Actual Result	✓ PASSED - User successfully logged in

6.3.2 Test Case 2: Create Road Segment

Test ID	STC-002
Objective	Verify user can create a new road segment
Preconditions	User logged in
Input	Start: (1.3521, 103.8198), End: (1.3621, 103.8298), Status: “optimal”
Steps	1. Navigate to Segments page 2. Enter coordinates

	3. Select status 4. Click “Create”
Expected Output	New segment appears in list and on map
Actual Result	✓ PASSED - Segment created with correct coordinates

6.3.3 Test Case 3: Submit Condition Report

Test ID	STC-003
Objective	Verify user can submit a condition report
Preconditions	User logged in, at least one segment exists
Input	Segment ID: 1, Note: “Pothole near intersection”
Steps	1. Select segment from list 2. Navigate to Reports 3. Enter note 4. Click “Submit”
Expected Output	Report appears in list, aggregation updated
Actual Result	✓ PASSED - Report submitted successfully

6.3.4 Test Case 4: Report Confirmation

Test ID	STC-004
Objective	Verify report confirmation updates aggregation
Preconditions	At least one unconfirmed report exists
Input	Report ID to confirm
Steps	1. Find unconfirmed report 2. Click “Confirm” button
Expected Output	Report marked as confirmed, weighted score updated
Actual Result	✓ PASSED - Confirmation status updated

6.3.5 Test Case 5: Route Planning with OSRM

Test ID	STC-005
Objective	Verify route planning returns multiple alternatives
Preconditions	OSRM server accessible
Input	Origin: (1.3521, 103.8198), Destination: (1.332, 103.903), Preference: “balanced”

Steps	1. Navigate to Route Planning 2. Enter coordinates 3. Select preference 4. Click "Search Routes"
Expected Output	1-3 routes displayed with quality scores and tags
Actual Result	✓ PASSED - Multiple routes returned with scoring

6.3.6 Test Case 6-10: Additional Tests

Test ID	Objective	Result
STC-006	Route Quality Scoring (Safety First)	✓ PASSED
STC-007	Weather Information Display	✓ PASSED
STC-008	Language Switching	✓ PASSED
STC-009	Data Aggregation Trigger	✓ PASSED
STC-010	Auto-Detection Simulation	✓ PASSED

6.4 API Testing Results

Endpoint	Method	Result
`/api/users`	POST	✓ PASSED
`/api/users`	GET	✓ PASSED
`/api/segments`	GET	✓ PASSED
`/api/segments`	POST	✓ PASSED
`/api/segments/{id}/reports`	POST/GET	✓ PASSED
`/api/reports/{id}/confirm`	POST	✓ PASSED
`/api/segments/{id}/aggregate`	GET	✓ PASSED
`/api/aggregation/trigger`	POST	✓ PASSED
`/api/trips`	POST/GET	✓ PASSED
`/api/path/search`	POST	✓ PASSED
`/api/weather`	GET	✓ PASSED
`/api/users/{id}/settings`	GET/PUT/PATCH	✓ PASSED
`/api/i18n/translations`	GET	✓ PASSED

7 Installation Instructions

7.1 Prerequisites

Requirement	Version	Notes
Python	3.10+	Required for backend
Node.js	20.x	Bundled in package
npm	9.x+	Comes with Node.js
Web Browser	Chrome/Firefox/Edge	For frontend access
Internet	Required	For OSRM API and map tiles

7.2 Quick Start (Windows)

7.2.1 Step 1: Extract the Package

```
# Navigate to the delivery folder
cd C:\path\to\DeliveryFolder\bbp-road-app
```

7.2.2 Step 2: Start Backend Server

```
# Open PowerShell in backend directory
cd backend

# Create virtual environment (first time only)
python -m venv .venv

# Activate virtual environment
.venv\Scripts\activate

# Install dependencies (first time only)
pip install -r requirements.txt

# Start the server
uvicorn main:app --host 127.0.0.1 --port 8000
```

The backend will be running at: `http://127.0.0.1:8000`

API documentation available at: `http://127.0.0.1:8000/docs`

7.2.3 Step 3: Start Frontend Development Server

```
# Open a NEW PowerShell window
cd C:\path\to\DeliveryFolder\bbp-road-app\frontend

# Use bundled Node.js
$env:PATH = "..\node-v20.19.0-win-x64\node-v20.19.0-win-x64;" + $env:PATH

# Install dependencies (first time only)
npm install
```

```
# Start development server
npm run dev
```

The frontend will be running at: `http://localhost:5173`

7.2.4 Step 4: Access the Application

1. Open a web browser
2. Navigate to `http://localhost:5173`
3. Enter any username to login (e.g., "alice" for demo data)

7.3 Detailed Installation (Linux/macOS)

7.3.1 Backend Setup

```
# Navigate to backend
cd bbp-road-app/backend

# Create virtual environment
python3 -m venv .venv

# Activate (Linux/macOS)
source .venv/bin/activate

# Install dependencies
pip install -r requirements.txt

# Run server
uvicorn main:app --host 127.0.0.1 --port 8000
```

7.3.2 Frontend Setup

```
# Navigate to frontend
cd bbp-road-app/frontend

# Install Node.js dependencies
npm install

# Run development server
npm run dev
```

7.4 Troubleshooting

Issue	Solution
Port 8000 in use	Change port: `uvicorn main:app --port 8001`
CORS errors	Ensure backend is running on 127.0.0.1:8000
npm install fails	Delete `node_modules` and `package-lock.json`, retry
Map not loading	Check internet connection (requires OpenStreetMap tiles)

OSRM timeout	Increase timeout in code or use fallback routes
Python not found	Install Python 3.10+ and add to PATH

8 Effort Spent

8.1 Individual Effort

Team Member	Task	Hours
Hu Xu Yan	Backend API Development	XX
Hu Xu Yan	Frontend Development	XX
Hu Xu Yan	OSRM Integration	XX
Hu Xu Yan	Testing	XX
Hu Xu Yan	Documentation	XX
Hu Xu Yan Total		**XX**
Team Member 2	Tasks	XX
Team Member 2 Total		**XX**
Team Member 3	Tasks	XX
Team Member 3 Total		**XX**

8.2 Effort Summary

Phase	Hours	Percentage
Requirements Analysis	X	X%
Design	X	X%
Implementation	X	X%
Testing	X	X%
Documentation	X	X%
Total	**XX**	**100%**

8.3 Generative AI Usage Declaration

In accordance with course requirements, we declare the use of Generative AI (GitHub Copilot / Claude) in the following capacities:

Task	AI Tool	Input	Output	Verification
Code Docs	Copilot	Add docstrings	Function docs	Manual review
ITD Draft	Claude	Project code + requirements	Document content	Fact verification

Test Cases	Claude	Generate test cases	Test templates	Execution
Code Review	Copilot	Code snippets	Improvements	Assessment

Verification Process:

1. All AI-generated code was reviewed for correctness
2. AI-generated documentation was verified against actual implementation
3. Test cases were executed to confirm validity
4. No AI output was used without human verification

9 References

9.1 Technical Documentation

1. **FastAPI Documentation** - https://fastapi.tiangolo.com/
2. **React Documentation** - https://react.dev/
3. **TypeScript Documentation** - https://www.typescriptlang.org/docs/
4. **Vite Documentation** - https://vitejs.dev/
5. **Leaflet Documentation** - https://leafletjs.com/reference.html
6. **React-Leaflet Documentation** - https://react-leaflet.js.org/
7. **OSRM API Documentation** - http://project-osrm.org/docs/v5.24.0/api/
8. **Pydantic Documentation** - https://docs.pydantic.dev/

9.2 Course Materials

1. **RASD Document** - Requirements Analysis and Specification Document (BBP Project)
2. **DD Document** - Design Document (BBP Project)
3. **Course Slides** - Software Engineering 2, Politecnico di Milano

9.3 Standards

1. **REST API Design Guidelines** - https://restfulapi.net/
2. **GeoJSON Specification** - RFC 7946
3. **Polyline Encoding** - Google Polyline Algorithm

9.4 Tools

1. **Visual Studio Code** - IDE
2. **Git** - Version Control
3. **Postman** - API Testing
4. **Chrome DevTools** - Frontend Debugging

10 Appendix A: API Endpoint Reference

10.1 User Endpoints

Method	Endpoint	Description
POST	`/api/users`	Create or get user
GET	`/api/users`	List all users
GET	`/api/users/{id}/settings`	Get user settings
PUT	`/api/users/{id}/settings`	Update all settings
PATCH	`/api/users/{id}/settings`	Partial update settings

10.2 Segment Endpoints

Method	Endpoint	Description
GET	`/api/segments`	List all segments
POST	`/api/segments`	Create segment
GET	`/api/segments/{id}/aggregate`	Get aggregation result
POST	`/api/segments/{id}/auto-detect`	Run auto-detection
POST	`/api/segments/{id}/apply-detection`	Apply detection result

10.3 Report Endpoints

Method	Endpoint	Description
GET	`/api/segments/{id}/reports`	List reports for segment
POST	`/api/segments/{id}/reports`	Create report
POST	`/api/reports/{id}/confirm`	Confirm report
POST	`/api/reports/batch-confirm`	Batch confirm

10.4 Trip Endpoints

Method	Endpoint	Description
GET	`/api/trips`	List trips
POST	`/api/trips`	Create trip
GET	`/api/trips/{id}`	Get trip details
DELETE	`/api/trips/{id}`	Delete trip

10.5 Route Planning Endpoints

Method	Endpoint	Description
POST	`/api/routes`	Preview routes
POST	`/api/path/search`	Search routes with scoring

10.6 Utility Endpoints

Method	Endpoint	Description
GET	`/api/stats`	Get statistics
GET	`/api/weather`	Get weather data
GET	`/api/i18n/translations`	Get translations
GET	`/api/i18n/languages`	Get available languages
POST	`/api/aggregation/trigger`	Trigger bulk aggregation

Document generated: January 31, 2026
End of Document