Software Test Plan (STP) for ORP – Outdoor Route Planner

Software Test Plan (STP) for ORP - Outdoor Route Planner

# 1. Introduction

This Software Test Plan outlines the testing strategy, test cases, tools, resources, and schedule required to verify and validate the Outdoor Route Planner (ORP) application. ORP is a full-stack application that integrates React for the frontend, Node.js and Express for the backend, and Python with Flask for AI-powered route optimization. This plan ensures all functional and non-functional requirements are met to provide a robust, secure, and user-friendly hiking route planner.

# 2. Test Items

- React frontend application for map-based route creation and user interactions.

- Node.js backend RESTful API for authentication and route management.

- Python Flask microservice for elevation-aware route optimization using A\* algorithm.

- Integration with external APIs including Google Maps, Elevation API.

- MongoDB Atlas database for persistent user and route data.

# 3. Features to be Tested

Authentication:

- Register and login functionality

- JWT token generation and storage

- Auto-logout on token expiration

- Error handling for invalid credentials

## Map Interface:

- Adding and removing up to 5 markers

- Draggable markers for real-time edits

- Terrain/satellite view toggle

- Live route drawing between points

## Route Optimization:

- Accuracy and performance of A\* pathfinding algorithm

- Correct cost calculation using elevation and distance

- Grid creation and bounding box validation

- Performance with maximum resolution and size

## Route Management:

- Save new routes to database with metadata

- Edit existing routes and store modified versions

- Delete confirmation and removal

- Mark/unmark as favorite

- Sorting, filtering, and searching user routes

## Statistics & Visualization:

- Total route length, elevation gain/loss

- Highest/lowest point identification

- Display in route detail cards and modals

## Download System:

- GPX export with valid XML schema

- JSON export with structured, readable format

- Download links and user feedback

## UI/UX:

- Responsive layout for desktop and mobile

- Intuitive navigation and visual cues

- Error states and progress indicators

# 4. Features Not to Be Tested

- In-depth testing of third-party APIs (e.g., Google Maps internals)

- Stress/load testing under thousands of concurrent users

- Penetration testing and vulnerability scanning (unless separately arranged)

# 5. Testing Strategy

## Unit Testing:

- Test each backend route and model

- Isolate and validate Python optimization logic

## Integration Testing:

- Test authentication flow end-to-end

- Ensure Python service integrates correctly with backend and frontend

## Manual UI Testing:

- Validate all user interactions

- Test layout responsiveness and error handling

Regression Testing:

- Re-run all tests after updates to ensure no new bugs were introduced

## Download Validation:

- Use online validators for GPX format

- Check completeness of JSON export

## Cross-browser Testing:

- Chrome, Firefox, Safari, and Edge supported

# Responsibilities

- Tomer: Frontend development and testing

- Richard: backend develepment and database testing

# 7. Schedule

|  |  |
| --- | --- |
| Phase | Dates |
| Unit Testing | May 11 – May 16 |
| Integration Testing | May 17 – May 24 |
| System Testing | May 25 – June 2 |
| Acceptance Testing | June 3 – June 13 |
| Final Review | June 14 |

# 8. Environment Requirements

- Node.js v16+, Python 3.12+

- MongoDB Atlas account and credentials

- Google Maps API Key with required services enabled

- Local testing on ports: 3000 (frontend), 3001 (backend), 5000 (Python)

# 9. Acceptance Criteria

- All critical and high-priority test cases must pass

- No open critical bugs at release

- All major workflows (login, create, edit, save, download routes) must work

- Real-time route optimization should return a path within 5 seconds

- Interface should function smoothly on all major browsers