Mavigo - Technical Documentation

Table of Contents

Documentation Version: 0.2.0
Last Updated: 2026-01-30
Git Branch: docs-for-v0.2

1. Team Presentation

Team Member
Marie BIBI
Raphael MEIMOUN
Seyed Amineddin MIRI
Malado SOW

2. Project Overview

2.1. Context

Mavigo is a personal public transportation assistant for the Paris metropolitan area (Île-de-France). The application helps users navigate the Parisian transit network with intelligent journey planning and quality-of-life features including accessibility support, Google Tasks integration, and disruption management.

2.2. Objectives

- **Simplify journey planning** across Île-de-France public transportation
- **Provide real-time information** on network disruptions and alternative routes
- **Integrate task management** with Google Tasks for location-aware productivity
- **Offer personalized suggestions** adapted to user accessibility and comfort preferences
- **Enable multi-stop optimization** to complete errands efficiently along transit routes

2.3. Main Features

2.3.1. Intelligent Journey Planning

- **Three-tier architecture**: Journey → JourneySegment → JourneyPoint for granular tracking
- **Comfort filtering**: Apply accessibility and comfort preferences (wheelchair, air conditioning, transfers, walking duration)
- **Multi-criteria optimization**: Balance speed, comfort, and accessibility needs
- **Real-time journey tracking**: Status lifecycle from PLANNED → IN_PROGRESS → COMPLETED/CANCELLED/REROUTED

2.3.2. Google Tasks Integration

- **OAuth2 authentication**: Secure Google account linking with token management
- **Location tagging**: Use `#mavigo: Location Name` syntax in task notes/titles

- **Automatic geocoding**: Convert location tags to coordinates via PRIM/BAN APIs
- **Task-optimized journeys**: Plan routes that pass by task locations (300m-900m radius detection)
- **Task-on-route detection**: Identify tasks along planned journey paths

2.3.3. Disruption Management

- **User-reported disruptions**: Report station or line disruptions during journeys
- **Point-level tracking**: Mark individual JourneyPoints as DISRUPTED
- **Automatic rerouting**: Generate alternative routes when disruptions are reported
- **Disruption types**: STATION (specific stop) or LINE (entire transit line)

2.3.4. Personalized Comfort Profiles

- **Default profile**: Per-user comfort settings (wheelchair, AC, max transfers, walking/waiting limits)
- **Named settings**: Save multiple comfort profiles with custom names
- **Direct path preference**: Option to prefer direct routes over transfers
- **Accessibility focus**: Wheelchair-accessible route filtering

2.3.5. Real-time Position Tracking

- **Journey lifecycle**: Track journey status from planning to completion
- **Segment-level details**: Monitor progress through individual journey segments
- **Disruption awareness**: Real-time integration of user-reported issues

2.4. Competitive Analysis

The Paris metropolitan transit app market is mature, with several established players offering journey planning and real-time information. Mavigo distinguishes itself through unique productivity features, advanced accessibility profiles, and community-driven intelligence that complement the core transit navigation experience.

2.4.1. Feature Comparison Matrix

The following comparison evaluates Mavigo against four major competitors across key functionality dimensions:

Feature Category	Citymapper	Google Maps	RATP	IdF Mobilités	Mavigo
Basic Route Planning	Yes	Yes	Yes	Yes	Yes
Real-time Transit Info	Yes	Yes	Yes	Yes	Yes
Wheelchair Routes	Yes	Yes	Yes	Yes	Yes

Feature Category	Citymapper	Google Maps	RATP	IdF Mobilités	Mavigo
Named Accessibility Profiles	No	No	No	No	**Yes**
Air Conditioning Filter	No	No	No	No	**Yes**
Comfort Settings	Limited	No	No	No	**Yes**
Google Tasks Integration	No	No	No	No	**Yes**
Location-Tagged Tasks	No	No	No	No	**Yes**
Multi-Waypoint w/ Tasks	No	Limited	No	No	**Yes**
User Disruption Reports	Limited	No	No	No	**Yes**
Point-Level Disruption	No	No	No	No	**Yes**
Auto Rerouting	Yes	Yes	No	No	**Yes**

Table Legend:

- **Yes** = Feature fully supported
- **Limited** = Partial support only
- **No** = Feature not available
- **Italic** = Mavigo's unique or superior implementation

2.4.2. Detailed Competitive Analysis

Accessibility and Personalization Leadership

While competitors like Citymapper and Google Maps offer basic wheelchair-accessible routing, these implementations typically function as a single toggle that filters results. Mavigo extends this concept significantly through its named comfort profile system, which allows users to create and save multiple accessibility configurations with distinct names. A user might maintain a "Full Accessibility" profile requiring wheelchair access and air conditioning on all segments, alongside a "Quick Commute" profile that relaxes some constraints for faster routes. This granularity extends to parameters such as maximum transfers, maximum walking duration, and maximum waiting time. The air conditioning requirement filter represents a comfort dimension absent from competitor offerings, directly addressing needs of users with temperature sensitivities or medical conditions. This multi-profile approach recognizes that accessibility needs are contextual and may vary based on time of day, weather conditions, or physical state.

Unique Productivity Integration

Mavigo is the only transit application offering native Google Tasks integration with location awareness. Users can tag tasks with locations using the `#mavigo: Location Name` syntax within Google Tasks notes or titles. The system automatically geocodes these locations and makes them available for journey optimization. When planning a route, users can include task locations as waypoints, and Mavigo generates optimized multi-stop itineraries that complete errands efficiently along transit routes. The task-on-route detection algorithm uses polyline densification and

proximity calculations to identify when existing tasks fall near a planned journey path, alerting users to opportunities to complete tasks without significant detours. While Google Maps offers multi-waypoint routing, it does not integrate with task management systems or provide task-aware optimization. This integration transforms transit planning from simple A-to-B routing into a productivity tool that helps users accomplish multiple objectives in a single trip.

Community-Driven Disruption Intelligence

The disruption reporting landscape in Paris transit apps is fragmented. Official apps from RATP and Île-de-France Mobilités rely exclusively on operator-provided disruption data, which can lag behind ground truth during developing incidents. Citymapper offers crowdsourced reporting in select markets, though its disruption model treats disruptions at the journey level rather than tracking specific affected infrastructure. Mavigo implements a two-tier disruption system supporting both station-specific and line-wide disruptions. When a user reports a station disruption, the system marks the specific JourneyPoint as DISRUPTED, creates a Disruption entity with timestamp and reporter metadata, and automatically requests a reroute from the next valid point to the destination. This point-level granularity provides precise context for rerouting algorithms and maintains a detailed disruption history. The automatic rerouting response ensures users receive alternative routes immediately upon reporting issues, minimizing journey delays.

Technical Architecture for Future Innovation

Mavigo's three-tier journey architecture differentiates it from competitors at a structural level. Where most transit apps model journeys as flat sequences of steps or legs, Mavigo implements a Journey entity containing JourneySegments, each containing multiple JourneyPoints. This hierarchy enables segment-level metadata such as air conditioning availability and line information, while point-level status tracking distinguishes between normal and disrupted points. The granularity provides richer data for machine learning applications, such as predicting disruption patterns or personalizing route recommendations based on historical preferences. The architecture also supports more sophisticated journey lifecycle management, with clear status transitions from PLANNED through IN_PROGRESS to COMPLETED, REROUTED, or CANCELLED states. This design represents an investment in long-term platform capabilities rather than minimum viable routing functionality.

2.4.3. Mavigo's Competitive Advantages

Mavigo offers distinct value propositions across multiple user segments:

- **Sole provider of Google Tasks integration**: Uniquely connects transit planning with task management through location-tagged tasks and optimized multi-waypoint routing
- **Most comprehensive accessibility features**: Multiple named comfort profiles, air conditioning filter, and granular timing controls exceed basic wheelchair routing
- **Community-powered disruption intelligence**: User-reported disruptions with point-level tracking and automatic rerouting provide real-time ground truth
- **Productivity-focused journey planning**: Task-on-route detection and optimization help users complete errands efficiently during transit trips
- **Advanced technical foundation**: Three-tier architecture enables future AI-driven features,

predictive routing, and personalized recommendations

- **Personalization-first design philosophy**: Named comfort settings and flexible preference system recognize that user needs vary by context

2.4.4. Market Positioning

Mavigo positions itself as the accessibility-first, productivity-integrated transit assistant for the Paris metropolitan area. While established competitors provide reliable basic routing and real-time transit information, Mavigo addresses underserved user segments with specific needs. For users with mobility challenges or accessibility requirements, the named profile system offers flexibility unavailable in single-toggle wheelchair filters. For productivity-conscious users who integrate tasks and errands into their daily commutes, the Google Tasks integration provides unique value. For community-minded users willing to contribute disruption intelligence in exchange for improved routing, the user reporting system creates a collaborative advantage. The technical architecture positions Mavigo for future differentiation through AI-powered features built on granular journey data.

3. Technologies and Methodologies

3.1. Technical Stack

3.1.1. Backend

Technology	Version/Details
Java	21 (LTS)
Spring Boot	3.5.7
Spring Data JPA	Data persistence and ORM
Spring Security	OAuth2 with Google
H2 Database	Development database (in-memory)
Gradle	9.1 (build system)

3.1.2. Frontend

- HTML5
- CSS3
- JavaScript (Vanilla, no framework)

3.1.3. External APIs

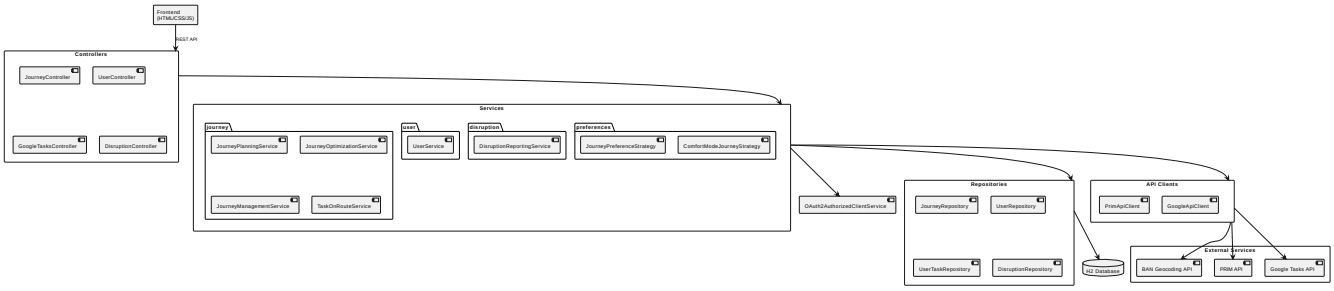
- **PRIM API**: Ile-de-France Mobilites for journey planning
- **Google Tasks API v1**: User task management with OAuth2
- **BAN Geocoding API**: Address and location geocoding
- **Spring Security OAuth2 Client**: OAuth2 authentication flow

3.2. Methodologies

- **Git Flow**: Feature branches, dev branch, main branch workflow
- **CI/CD**: GitHub Actions for continuous integration and testing
- **Code Review**: Mandatory pull requests with peer review
- **Testing**: Unit tests and integration tests with JUnit 5 and Spring Test
- **API Documentation**: OpenAPI/Swagger annotations (available at `/swagger-ui.html`)

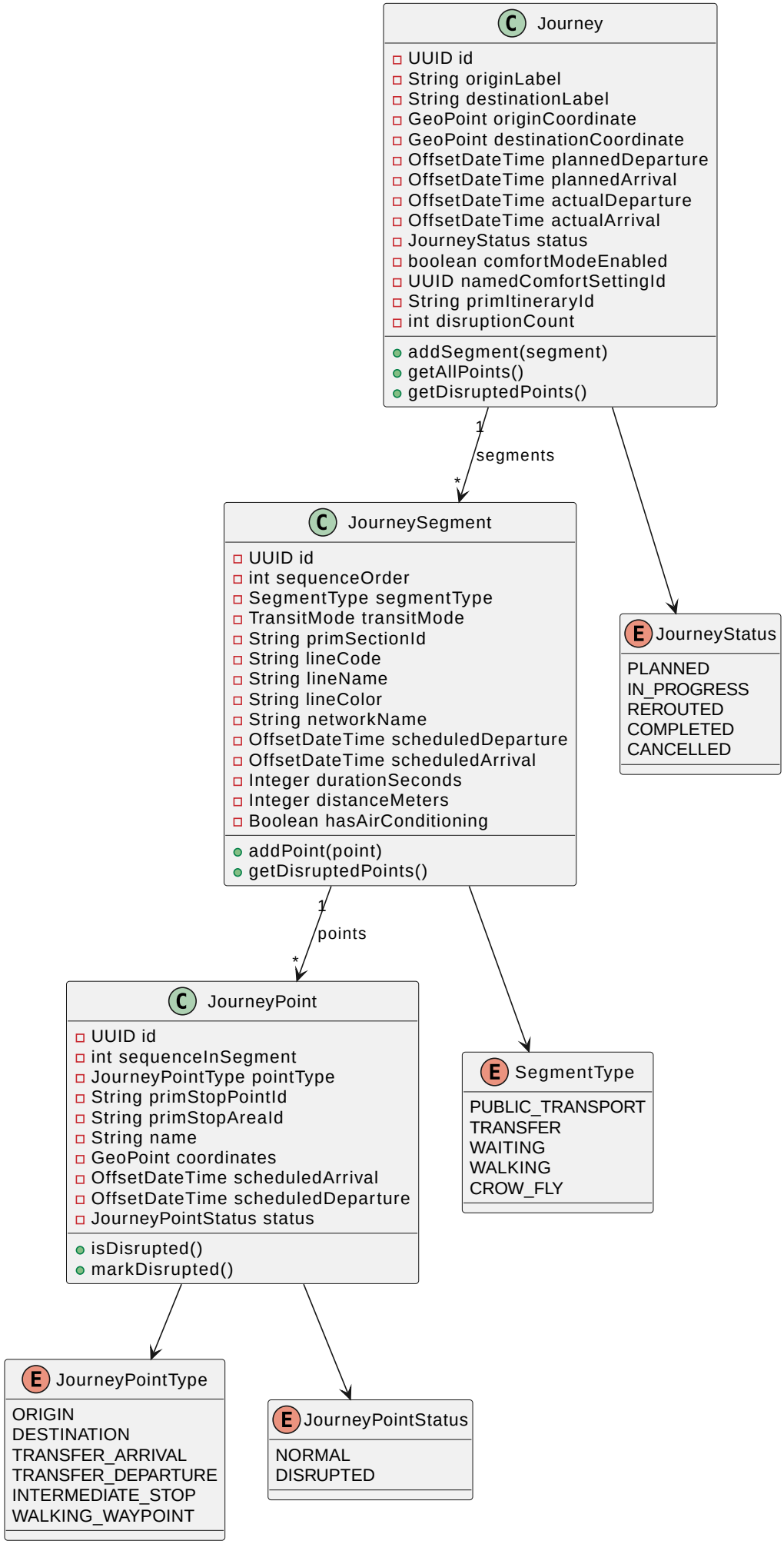
4. Architecture

4.1. Global Architecture

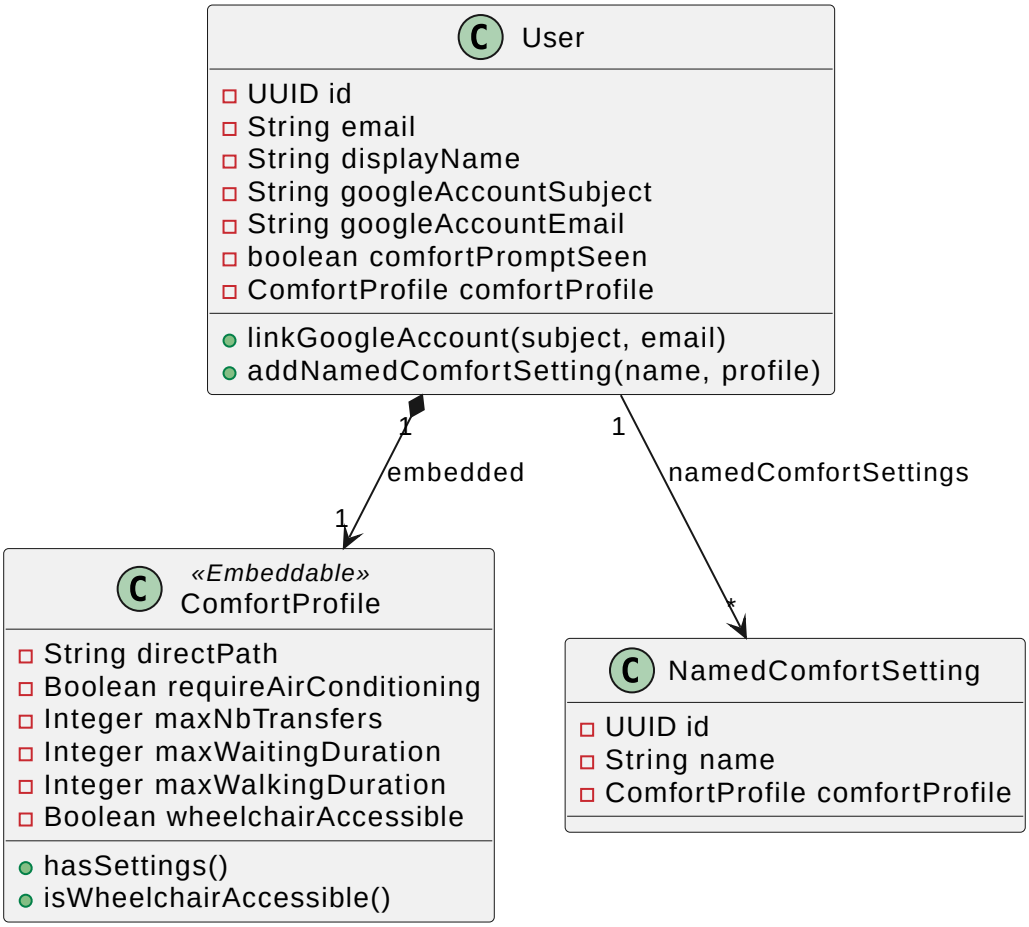


4.2. Class Diagrams - Core Domain Models

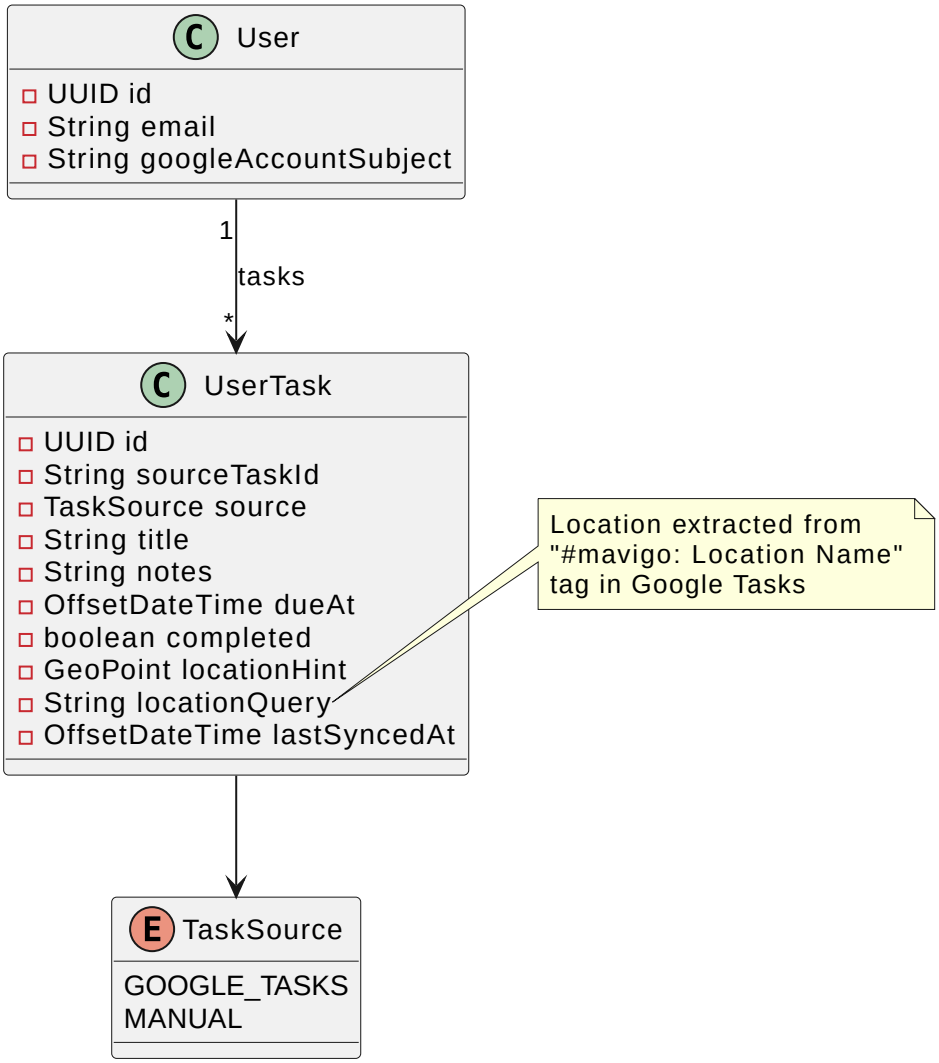
4.2.1. Journey Hierarchy


Journey
UUID id
String originLabel
String destinationLabel
GeoPoint originCoordinate
GeoPoint destinationCoordinate
OffsetDateTime plannedDeparture
OffsetDateTime plannedArrival
OffsetDateTime actualDeparture
OffsetDateTime actualArrival
JourneyStatus status
boolean comfortModeEnabled
UUID namedComfortSettingId
String primItineraryId
int disruptionCount
addSegment(segment)
getAllPoints()
getDisruptedPoints()
1
segments
*
JourneySegment
UUID id
int sequenceOrder
SegmentType segmentType
TransitMode transitMode
String primSectionId
String lineCode
String lineName
String lineColor
String networkName
OffsetDateTime scheduledDeparture
OffsetDateTime scheduledArrival
Integer durationSeconds
Integer distanceMeters
Boolean hasAirConditioning
addPoint(point)
getDisruptedPoints()
JourneyStatus
PLANNED
IN_PROGRESS
REROUTED
COMPLETED
CANCELLED
1
points
*
JourneyPoint
UUID id
int sequenceInSegment
JourneyPointType pointType
String primStopPointId
String primStopAreaId
String name
GeoPoint coordinates
OffsetDateTime scheduledArrival
OffsetDateTime scheduledDeparture
JourneyPointStatus status
isDisrupted()
markDisrupted()
SegmentType
PUBLIC_TRANSPORT
TRANSFER
WAITING
WALKING
CROW_FLY
JourneyPointType
ORIGIN
DESTINATION
TRANSFER_ARRIVAL
TRANSFER_DEPARTURE
INTERMEDIATE_STOP
WALKING_WAYPOINT
JourneyPointStatus
NORMAL
DISRUPTED

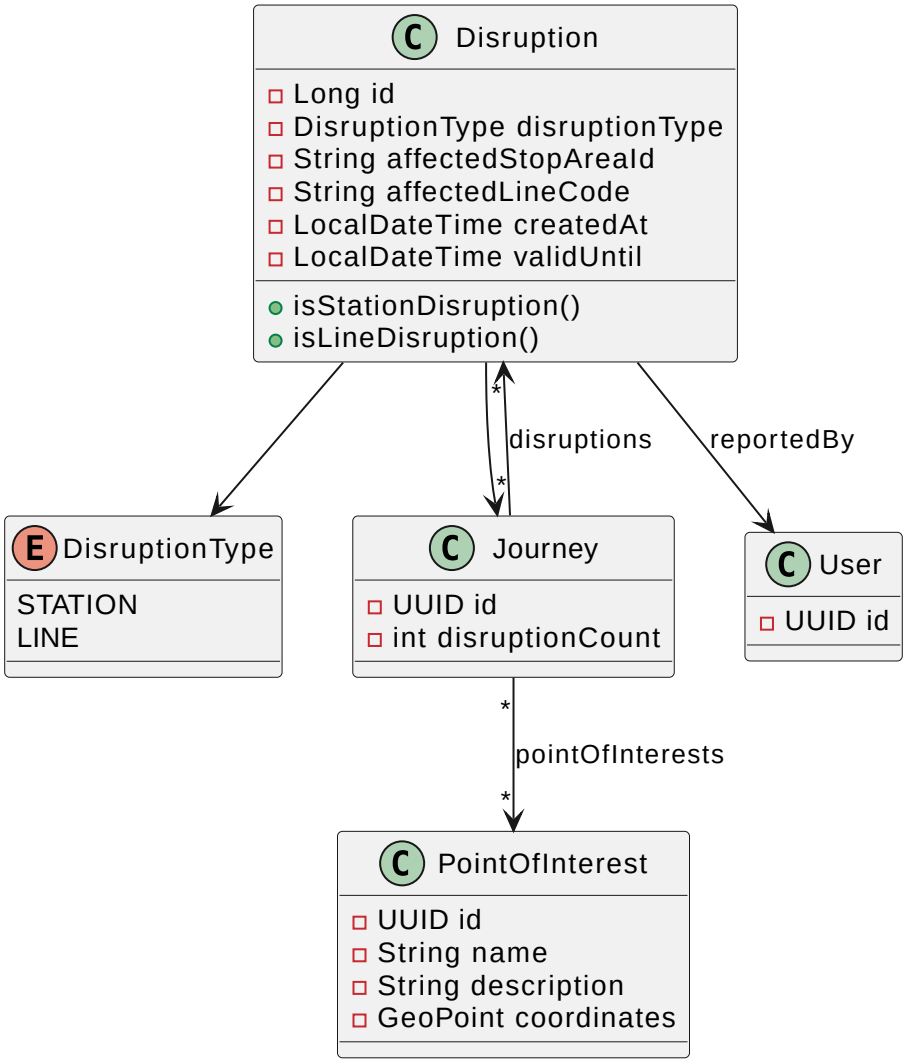
4.2.2. User and Comfort Profiles



4.2.3. Tasks and Google Integration

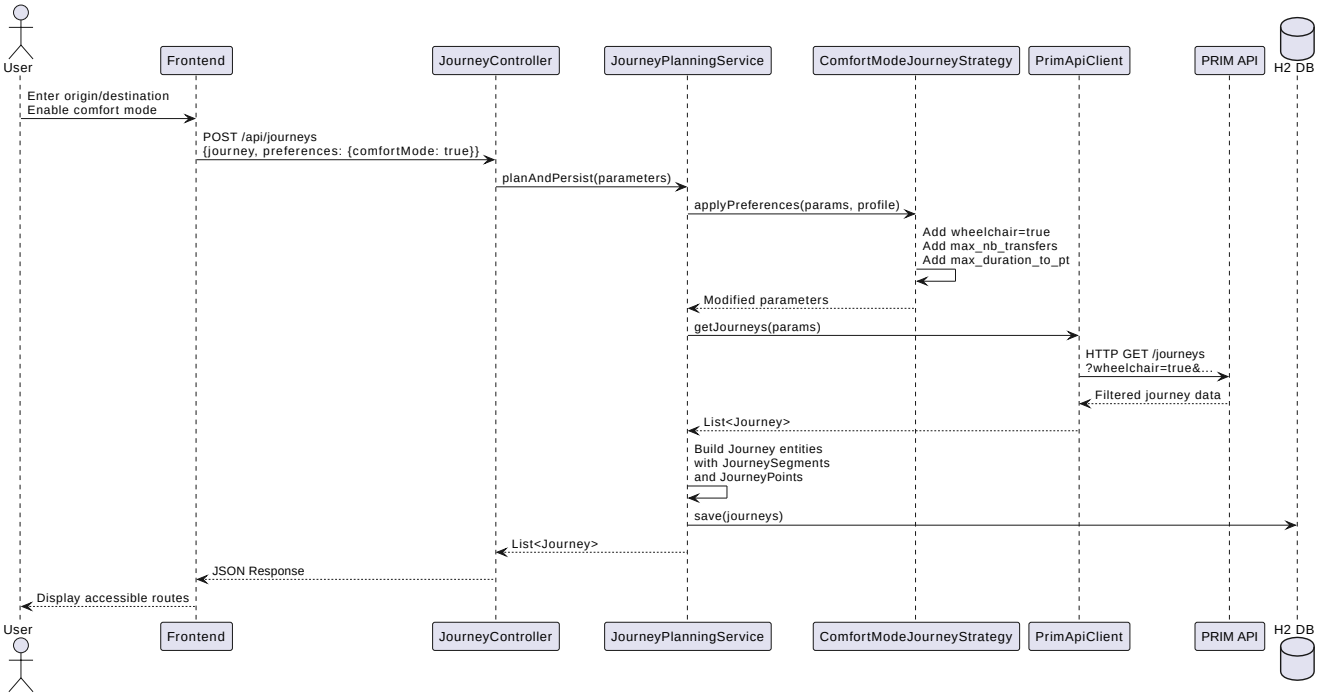


4.2.4. Disruptions and Points of Interest

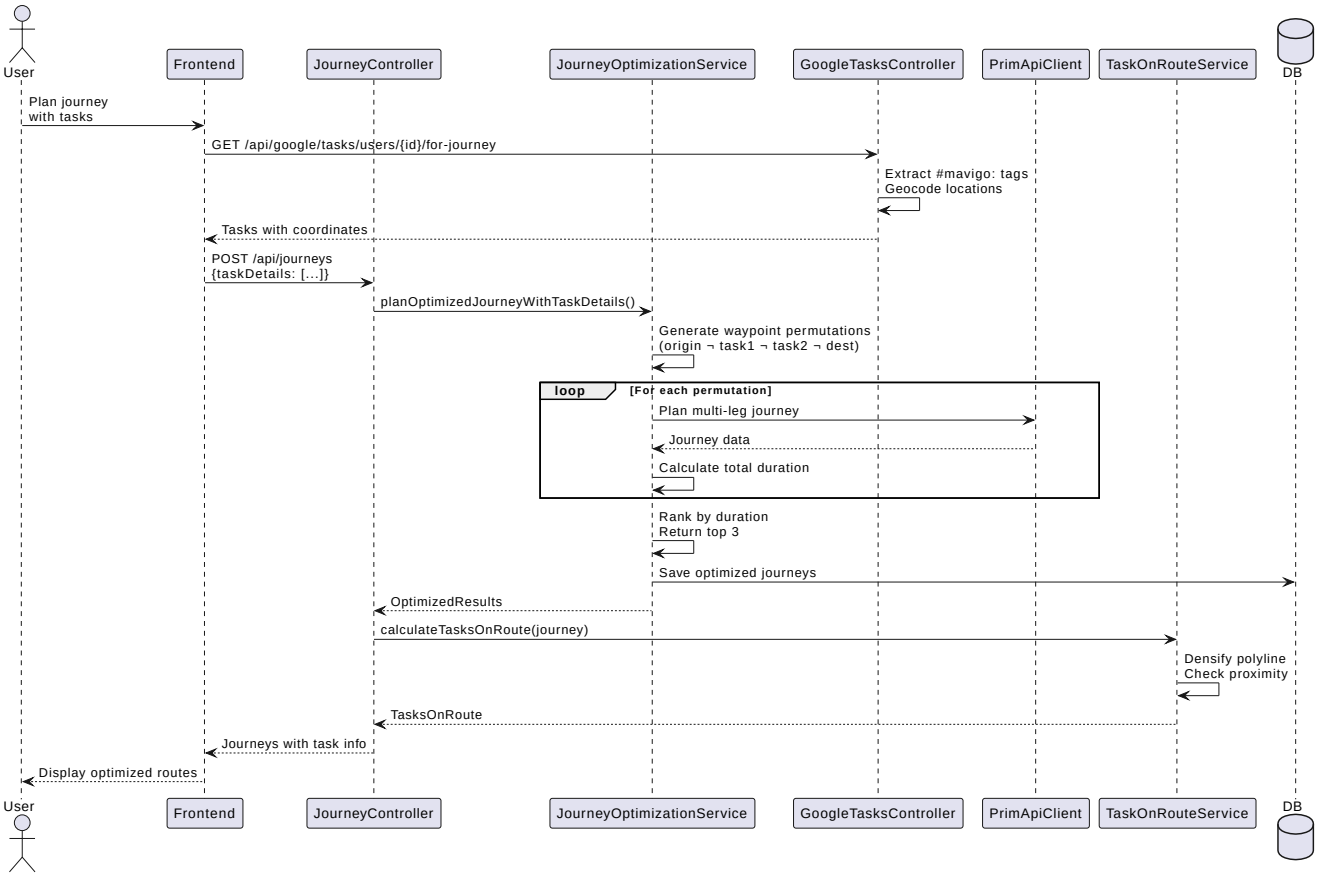

Disruption
Long id
DisruptionType disruptionType
String affectedStopAreaId
String affectedLineCode
LocalDateTime createdAt
LocalDateTime validUntil
isStationDisruption()
isLineDisruption()
*
*
disruptions
reportedBy
DisruptionType
STATION
LINE
Journey
UUID id
int disruptionCount
User
UUID id
*
*
pointOfInterests
PointOfInterest
UUID id
String name
String description
GeoPoint coordinates

4.3. Sequence Diagrams

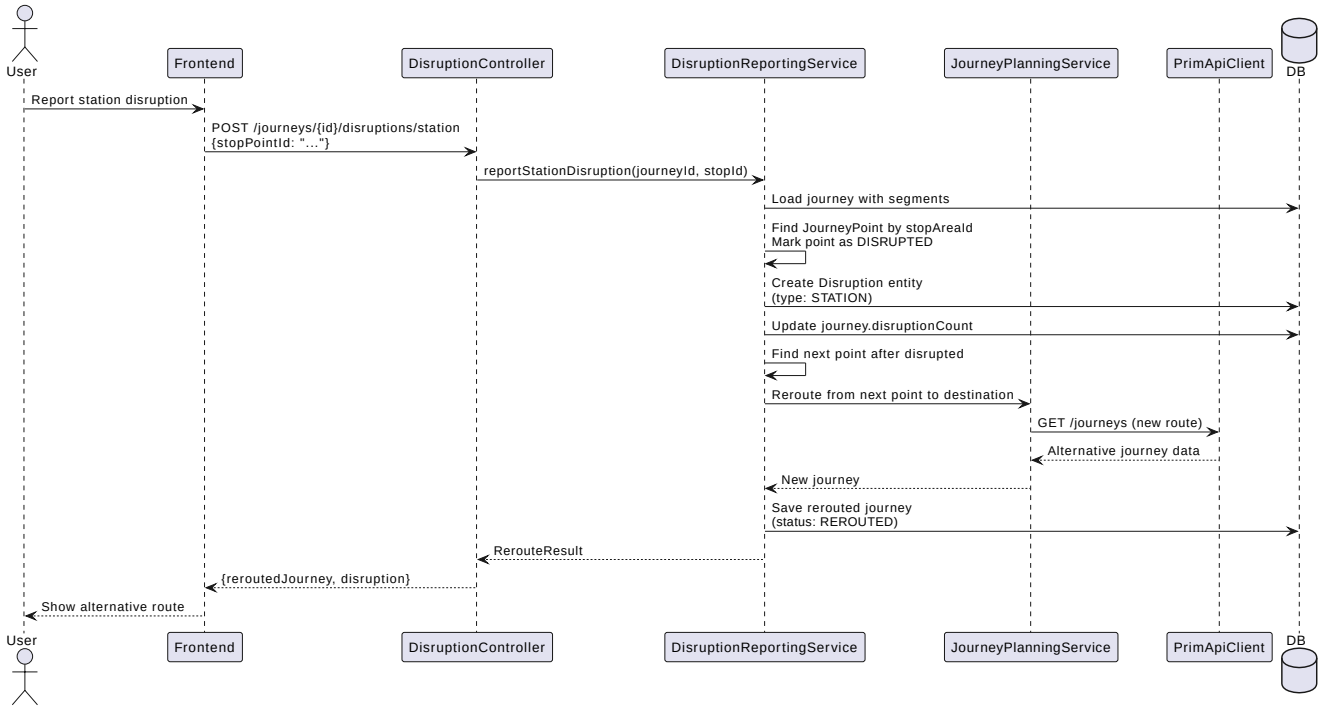
4.3.1. Journey Planning with Comfort Profile



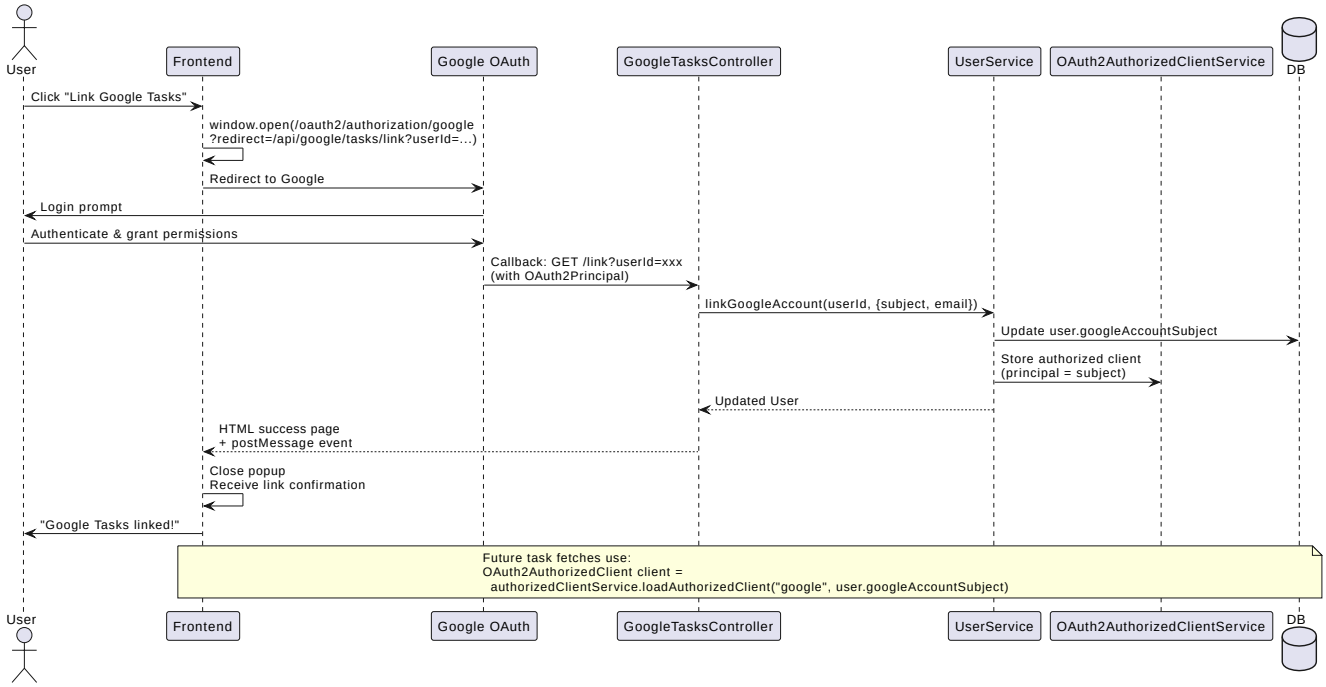
4.3.2. Task-Optimized Journey Planning



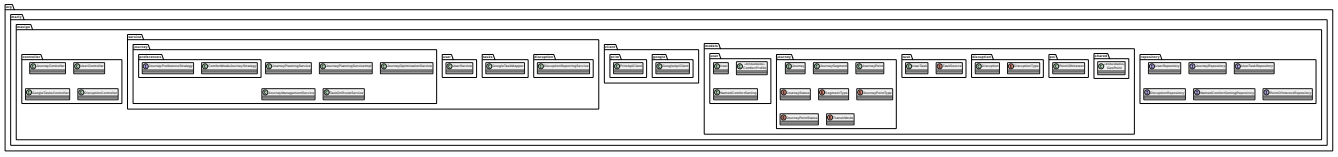
4.3.3. Disruption Reporting and Rerouting



4.3.4. Google Tasks OAuth2 and Task Linking



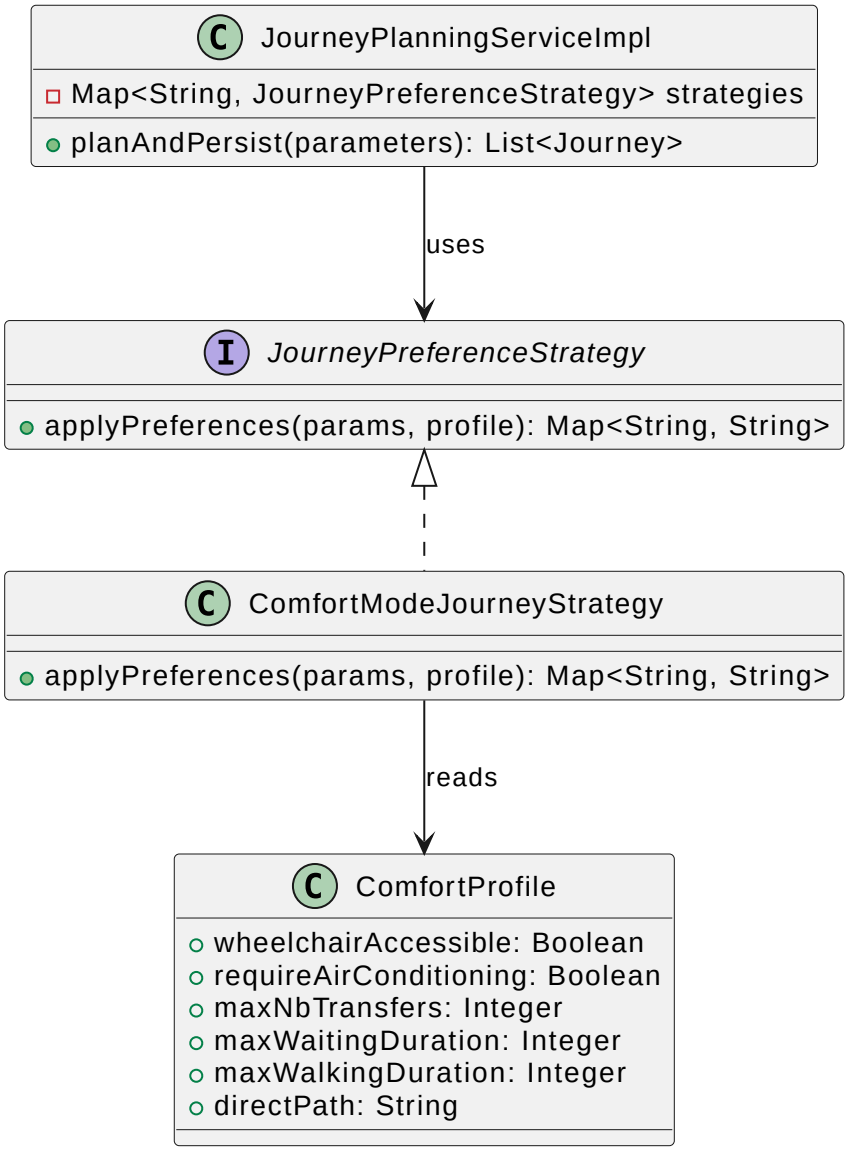
4.4. Package Structure



4.5. Journey Filtering Strategy Pattern

The application uses the **Strategy Pattern** to apply user preferences to journey planning. This allows flexible filtering of PRIM API results based on comfort profiles, accessibility needs, and other criteria.

4.5.1. Architecture



4.5.2. Comfort Parameters Applied to PRIM API

When comfort mode is enabled, the `ComfortModeJourneyStrategy` transforms user preferences into PRIM API query parameters:

Comfort Setting	PRIM Parameter	Description
`wheelchairAccessible = true`	`wheelchair=true`	Only return wheelchair-accessible routes
`maxNbTransfers`	`max_nb_transfers=N`	Limit the number of transfers in a journey

Comfort Setting	PRIM Parameter	Description
`maxWaitingDuration` (seconds)	`max_duration_to_pt=N`	Maximum walking time to first public transport stop
`maxWalkingDuration` (seconds)	`max_walking_duration_to_pt=N`	Maximum total walking duration in journey
`directPath = "only"`	`direct_path=only`	Prefer direct routes without transfers
`requireAirConditioning = true`	**(post-filter)**	Filter out segments without AC (applied after PRIM response)

4.5.3. Filtering Workflow

1. **User Request**: Frontend sends journey planning request with `comfortMode: true` or `namedComfortSettingId`
2. **Load Profile**: Service loads user's `ComfortProfile` or named setting
3. **Strategy Selection**: `JourneyPlanningServiceImpl` selects `ComfortModeJourneyStrategy`
4. **Parameter Transformation**: Strategy converts profile settings to PRIM API parameters
5. **API Call**: Enhanced parameters sent to PRIM API
6. **Post-filtering**: Air conditioning requirement applied to returned journeys
7. **Persistence**: Filtered journeys saved with `comfortModeEnabled = true`

4.6. Task Optimization Algorithm

The **Task Optimization** feature allows users to plan journeys that pass by locations where they have pending Google Tasks. This enables efficient multi-stop route planning.

4.6.1. Task-on-Route Detection

The `TaskOnRouteService` determines if a task location is "on the route" of a planned journey:

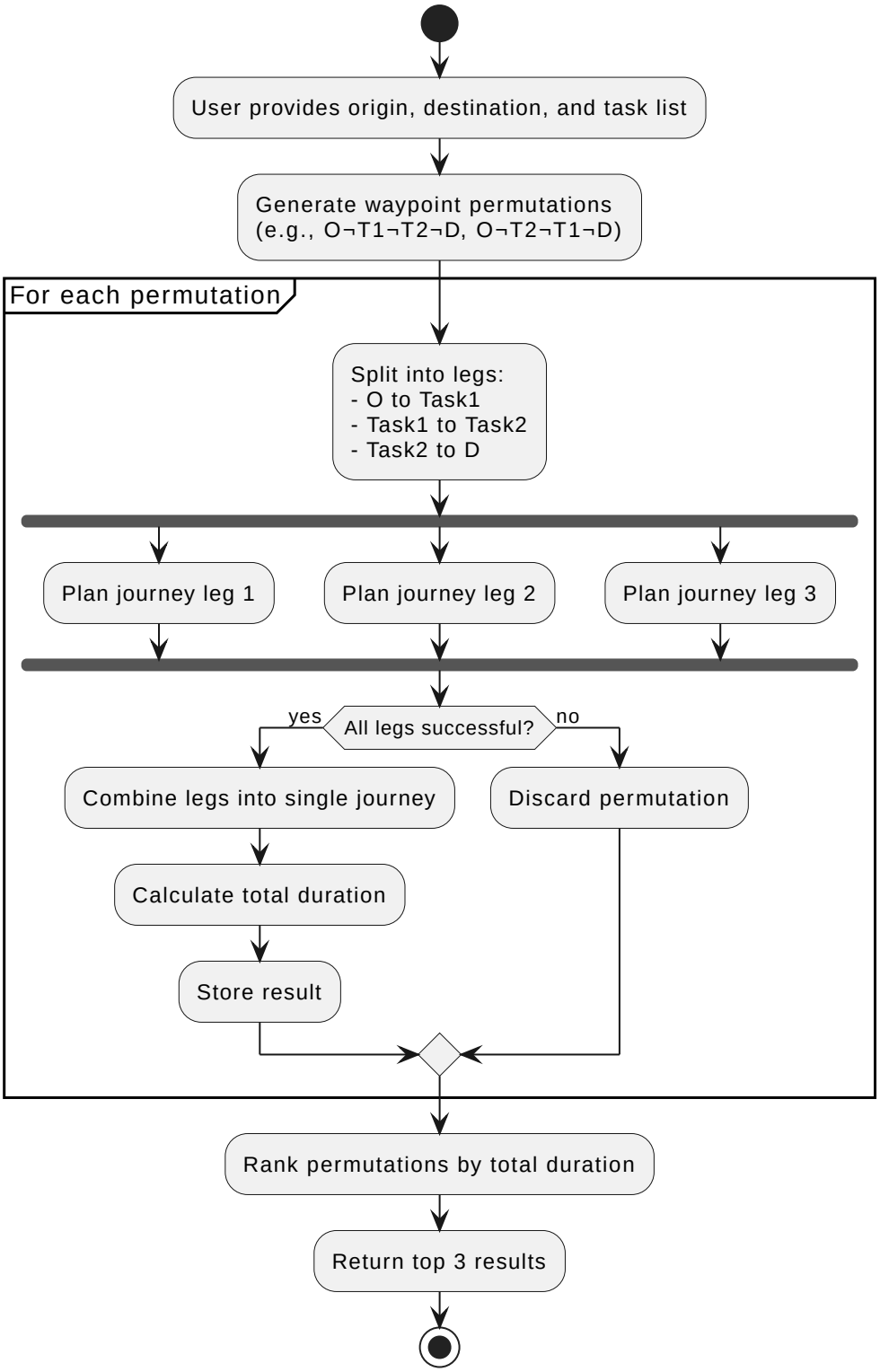
1. **Extract Route Points**: Get all `JourneyPoint` coordinates from journey segments
2. **Polyline Densification**: Insert intermediate points every ~200m to avoid gaps
3. **Proximity Calculation**: For each task location, find minimum distance to polyline
4. **Radius Threshold**:
    - **300m** for well-defined routes (many points)
    - **900m** for sparse routes (≤3 points) to avoid false negatives

```
// From JourneyController.java:134-162
double BASE_RADIUS_METERS = 300.0;
var baseRoutePoints = taskOnRouteService.extractRoutePoints(journey);
var polyline = taskOnRouteService.densify(baseRoutePoints, 200);
double radius = (polyline == null || polyline.size() <= 3) ? 900.0 :
BASE_RADIUS_METERS;

tasksOnRoute = tasks.stream()
    .filter(t -> t.getLocationHint() != null)
    .map(t -> {
        double d = taskOnRouteService.minDistanceMetersToPolyline(t.getLocationHint(),
polyline);
        return (d <= radius) ? JourneyResponse.fromTask(t, d) : null;
    })
    .filter(Objects::nonNull)
    .toList();
```

4.6.2. Multi-Waypoint Optimization

The `JourneyOptimizationService` plans optimized routes through multiple task locations:



4.6.3. Time Delta Calculation

Each optimized journey includes:

- **Base Duration**: Estimated time for direct route (origin → destination, no tasks)
- **Total Duration**: Actual time including all task waypoints
- **Added Time per Task**: `(totalDuration - baseDuration) / numberOfTasks`

This information helps users decide if the detour is worthwhile.

4.7. Disruption Management

The disruption management system allows users to report issues during their journeys and receive alternative routes.

4.7.1. Disruption Types

Type	Scope	Example
STATION	Specific stop/station	"Station Châtelet is closed due to incident"
LINE	Entire transit line	"Line 1 is experiencing delays"

4.7.2. Disruption Reporting Workflow

1. **User Reports**: During journey, user reports disruption (station or line)
2. **Point Marking**: System finds corresponding `JourneyPoint` and marks status as `DISRUPTED`
3. **Disruption Entity**: Creates `Disruption` record with:
    - Type (STATION/LINE)
    - Affected stop area ID or line code
    - Reporter (user)
    - Timestamp
4. **Update Count**: Increments `journey.disruptionCount`
5. **Find Continuation Point**: Identifies next valid point after disruption
6. **Reroute**: Requests new journey from continuation point to final destination
7. **Create Alternative**: Saves new journey with status `REROUTED`

4.7.3. Point-Level Disruption Tracking

Unlike traditional disruption systems that mark entire journeys as affected, Mavigo tracks disruptions at the individual point level:

- **Granularity**: Each `JourneyPoint` has a `status` field (NORMAL or DISRUPTED)
- **Precision**: Exactly which stop/station is affected is recorded
- **History**: Disrupted points remain in the journey history for debugging
- **Rerouting Context**: System knows exactly where to resume journey planning

4.7.4. Rerouting Process

```
// From DisruptionReportingService pattern:
1. Mark point as DISRUPTED
2. Create Disruption entity
3. Find next point: journey.getNextPointAfter(disruptedPoint)
```

```
4. Plan new route: nextPoint.coordinates → journey.destination
5. Save new journey with status = REROUTED
6. Return both disruption info and new journey to user
```

The user receives: * **Original Journey**: With disrupted point marked (for reference) * **Rerouted Journey**: New route avoiding the disruption * **Disruption Details**: What was reported, when, by whom

5. API Endpoints

5.1. Journey Planning API

Method	Endpoint	Description
POST	`/api/journeys`	Plan a new journey with optional comfort filtering and task optimization
GET	`/api/journeys/{id}`	Retrieve journey details by ID
POST	`/api/journeys/{id}/start`	Mark journey as IN_PROGRESS (sets actualDeparture timestamp)
POST	`/api/journeys/{id}/complete`	Mark journey as COMPLETED (sets actualArrival timestamp)
POST	`/api/journeys/{id}/cancel`	Mark journey as CANCELLED

5.1.1. Example: Plan Journey with Comfort Profile

Request:

```
POST /api/journeys
{
  "journey": {
    "userId": "550e8400-e29b-41d4-a716-446655440000",
    "originQuery": "Gare de Lyon",
    "destinationQuery": "Châtelet",
    "departureTime": "2026-01-30T14:30:00"
  },
  "preferences": {
    "comfortMode": true
  }
}
```

Response:

```
[
  {
    "id": "7c9e6679-7425-40de-944b-e07fc1f90ae7",
```

```
    "originLabel": "Gare de Lyon",
    "destinationLabel": "Châtelet",
    "plannedDeparture": "2026-01-30T14:30:00+01:00",
    "plannedArrival": "2026-01-30T14:47:00+01:00",
    "status": "PLANNED",
    "comfortModeEnabled": true,
    "segments": [
      {
        "sequenceOrder": 0,
        "segmentType": "PUBLIC_TRANSPORT",
        "lineCode": "14",
        "lineName": "Métro 14",
        "hasAirConditioning": true,
        "points": [...]
      }
    ],
    "tasksOnRoute": []
  }
]
```

5.1.2. Example: Plan Journey with Tasks

Request:

```
POST /api/journeys
{
  "journey": {
    "userId": "550e8400-e29b-41d4-a716-446655440000",
    "originQuery": "République",
    "destinationQuery": "Nation",
    "departureTime": "2026-01-30T15:00:00",
    "taskDetails": [
      {
        "id": "task-123",
        "title": "Buy groceries",
        "locationQuery": "Gare de Lyon",
        "locationHint": {"lat": 48.8443, "lng": 2.3730}
      }
    ]
  }
}
```

Response includes:

```
[
  {
    "id": "...",
    "includedTasks": [
```

```
      {
        "taskId": "task-123",
        "title": "Buy groceries",
        "locationQuery": "Gare de Lyon",
        "addedTimeSeconds": 420
      }
    ],
    "baseDurationSeconds": 1200,
    "totalDurationSeconds": 1620
  }
]
```

5.2. Comfort Profile API

Method	Endpoint	Description
GET	`/api/users/{userId}/comfort-profile`	Get user's default comfort profile
PUT	`/api/users/{userId}/comfort-profile`	Update default comfort profile
DELETE	`/api/users/{userId}/comfort-profile`	Reset comfort profile to defaults
GET	`/api/users/{userId}/comfort-settings`	List all named comfort settings
POST	`/api/users/{userId}/comfort-settings`	Create a new named comfort setting
PUT	`/api/users/{userId}/comfort-settings/{settingId}`	Update a named comfort setting
DELETE	`/api/users/{userId}/comfort-settings/{settingId}`	Delete a named comfort setting
POST	`/api/users/{userId}/comfort-prompt-seen`	Mark that user has seen comfort profile prompt

5.2.1. Example: Update Comfort Profile

Request:

```
PUT /api/users/{userId}/comfort-profile
{
  "wheelchairAccessible": true,
  "requireAirConditioning": true,
  "maxNbTransfers": 2,
  "maxWalkingDuration": 600,
  "maxWaitingDuration": 300,
  "directPath": "indifferent"
}
```

Response:

```
{
  "wheelchairAccessible": true,
  "requireAirConditioning": true,
  "maxNbTransfers": 2,
  "maxWalkingDuration": 600,
  "maxWaitingDuration": 300,
  "directPath": "indifferent"
}
```

5.2.2. Example: Create Named Comfort Setting

Request:

```
POST /api/users/{userId}/comfort-settings
{
  "name": "Accessibility Priority",
  "comfortProfile": {
    "wheelchairAccessible": true,
    "maxNbTransfers": 1,
    "maxWalkingDuration": 300
  }
}
```

5.3. Google Tasks Integration API

Method	Endpoint	Description
GET	`/api/google/tasks/link?userId={id}`	Link Google account (OAuth2 callback endpoint)
GET	`/api/google/tasks/users/{userId}/lists`	List user's Google Task lists
GET	`/api/google/tasks/users/{userId}/default-list`	Get default task list (first list)
GET	`/api/google/tasks/users/{userId}/lists/{listId}/tasks`	Get tasks from a specific list with location enrichment
GET	`/api/google/tasks/users/{userId}/for-journey`	Get tasks suitable for journey optimization (with #mavigo: tags)
GET	`/api/google/tasks/users/{userId}/local`	Get locally stored tasks
POST	`/api/google/tasks/users/{userId}/lists/{listId}/tasks`	Create a new task with optional location
PATCH	`/api/google/tasks/users/{userId}/lists/{listId}/tasks/{taskId}/complete`	Mark task as completed
DELETE	`/api/google/tasks/users/{userId}/lists/{listId}/tasks/{taskId}`	Delete a task
GET	`/api/google/tasks/me`	Debug: Get current OAuth2 user info
GET	`/api/google/tasks/token`	Debug: Get OAuth2 token details

5.3.1. Location Tag Syntax

To associate a task with a location, add `#mavigo: Location Name` to the task notes or title:

Example:

```
Title: Buy milk
Notes: Get organic milk from the store near the station
#mavigo: Gare de Lyon
```

The system will: 1. Extract "Gare de Lyon" from the tag 2. Geocode it using PRIM/BAN APIs 3. Store coordinates in `UserTask.locationHint` 4. Use it for task-on-route detection and journey optimization

5.3.2. Example: Create Task with Location

Request:

```
POST /api/google/tasks/users/{userId}/lists/{listId}/tasks
{
```

```
  "title": "Pick up package",
  "notes": "At the post office",
  "locationQuery": "Gare du Nord",
  "due": "2026-02-01"
}
```

Response:

```
{
  "id": "google-task-xyz",
  "title": "Pick up package",
  "notes": "At the post office",
  "localTaskId": "550e8400-e29b-41d4-a716-446655440001",
  "locationResolved": true,
  "locationLat": 48.8809,
  "locationLng": 2.3553,
  "status": "needsAction"
}
```

5.4. Disruption Reporting API

Method	Endpoint	Description
GET	`/api/journeys/{journeyId}/lines`	Get all lines used in this journey
GET	`/api/journeys/{journeyId}/stops`	Get all stops in this journey
POST	`/api/journeys/{journeyId}/disruptions/station`	Report a station disruption and get reroute
POST	`/api/journeys/{journeyId}/disruptions/line`	Report a line disruption and get reroute

5.4.1. Example: Report Station Disruption

Request:

```
POST /api/journeys/{journeyId}/disruptions/station
{
  "stopPointId": "stop_point:IDFM:71234"
}
```

Response:

```
{
  "disruption": {
    "id": 42,
    "type": "STATION",
    "affectedStopAreaId": "stop_area:IDFM:71234",
    "createdAt": "2026-01-30T14:35:12"
  },
  "reroutedJourney": {
    "id": "new-journey-uuid",
    "status": "REROUTED",
    "originLabel": "Next valid stop",
    "destinationLabel": "Original destination",
    "segments": [...]
  },
  "originalJourney": {
    "id": "original-journey-uuid",
    "disruptedPoints": [
      {
        "id": "point-uuid",
        "name": "Châtelet",
        "status": "DISRUPTED"
      }
    ]
  }
```

```
}
```

5.5. User Management API

Method	Endpoint	Description
POST	`/api/users`	Create a new user
POST	`/api/users/login`	Login with email (simplified auth)
GET	`/api/users/{userId}`	Get user details
PUT	`/api/users/{userId}`	Update user information
DELETE	`/api/users/{userId}`	Delete user account

5.5.1. Example: Create User

Request:

```
POST /api/users
{
  "email": "user@example.com",
  "displayName": "John Doe"
}
```

Response:

```
{
  "id": "550e8400-e29b-41d4-a716-446655440000",
  "email": "user@example.com",
  "displayName": "John Doe",
  "googleAccountLinked": false,
  "comfortPromptSeen": false
}
```

6. Google Tasks Integration

6.1. Overview

Mavigo integrates with Google Tasks API v1 to allow users to:

- Link their Google account via OAuth2
- View and manage their Google Tasks within Mavigo
- Tag tasks with locations using `#mavigo: Location Name` syntax
- Plan journeys that include task locations as waypoints
- Get notified when tasks are near their planned routes

6.2. OAuth2 Configuration

6.2.1. Required Google Cloud Setup

1. **Create a Google Cloud Project**: https://console.cloud.google.com/
2. **Enable Google Tasks API**: In APIs & Services → Library → Search "Tasks API"
3. **Create OAuth2 Credentials**:
   - Navigate to APIs & Services → Credentials
   - Create OAuth 2.0 Client ID
   - Application type: Web application
   - Authorized redirect URIs:
     - `http://localhost:8080/login/oauth2/code/google` (development)
     - `https://your-domain.com/login/oauth2/code/google` (production)
4. **Note your credentials**:
   - Client ID: `your-client-id.apps.googleusercontent.com`
   - Client Secret: `your-client-secret`

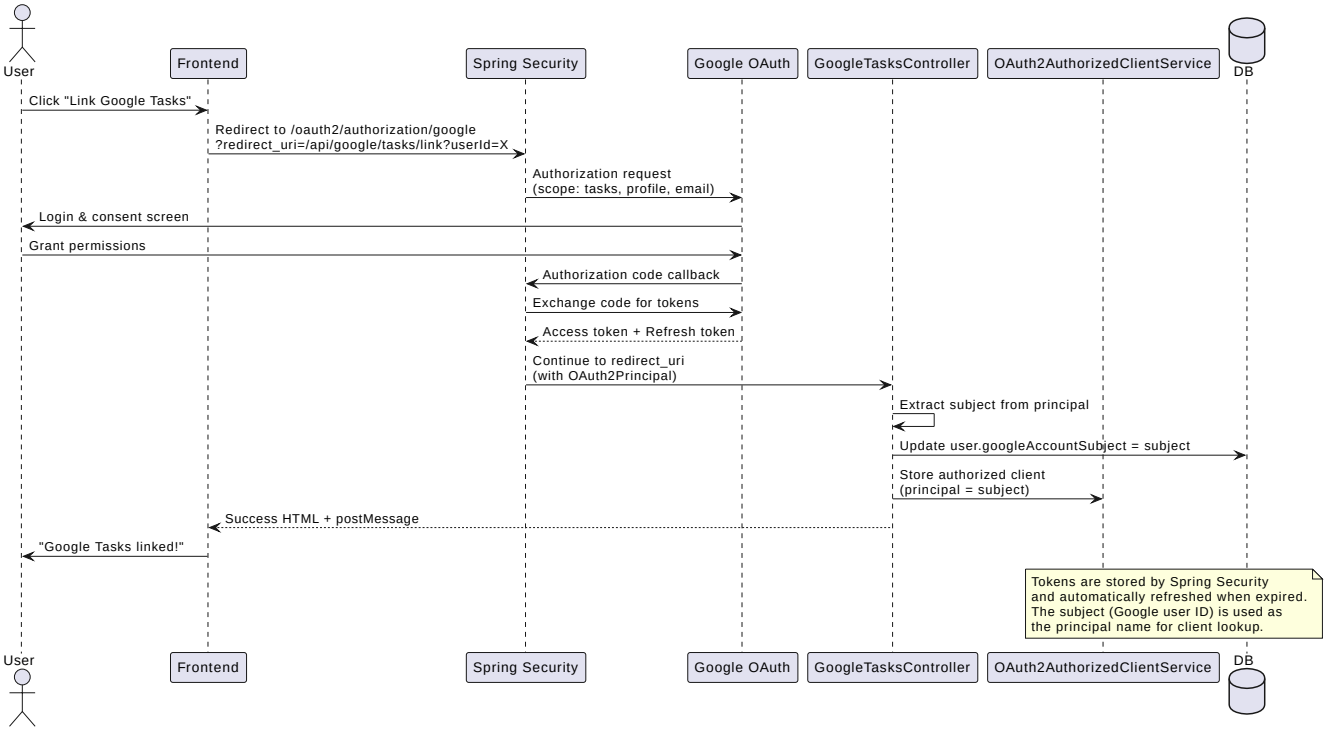
6.2.2. Application Configuration

Add to `application.properties` or environment variables:

```
# Google OAuth2
spring.security.oauth2.client.registration.google.client-id=${GOOGLE_CLIENT_ID}
spring.security.oauth2.client.registration.google.client-
secret=${GOOGLE_CLIENT_SECRET}
spring.security.oauth2.client.registration.google.scope=openid,profile,email,https://w
ww.googleapis.com/auth/tasks
spring.security.oauth2.client.registration.google.redirect-
uri={baseUrl}/login/oauth2/code/{registrationId}
```

```
# Google Tasks API Base URL
google.tasks.api.base-url=https://tasks.googleapis.com/tasks/v1
```

6.3. OAuth2 Flow



6.4. Location Tag Processing

6.4.1. Syntax

Add `#mavigo: Location Name` anywhere in task notes or title:

Examples:

```
Title: Buy milk #mavigo: Gare de Lyon
Notes: Pick up dry cleaning at the shop near station #mavigo: Châtelet
```

6.4.2. Processing Workflow

1. **Tag Extraction**: Regex pattern `(?i)mavigo:\s*([^\n]+)` extracts location
2. **Geocoding**: Location query sent to PRIM API's `/places` endpoint
3. **Coordinate Storage**: First valid result stored in `UserTask.locationHint`
4. **Error Handling**: If geocoding fails, task is saved without coordinates (non-blocking)

```
// From GoogleTasksController.java:258-277
private String extractLocationTag(TaskDto dto) {
```

```
    Pattern LOCATION_TAG = Pattern.compile("(?i)#mavigo:\\s*([^\\n#]+)");
    String notes = dto.notes() == null ? "" : dto.notes();
    String title = dto.title() == null ? "" : dto.title();

    // Check notes first, then title
    Matcher m = LOCATION_TAG.matcher(notes);
    if (m.find()) return m.group(1).trim();

    m = LOCATION_TAG.matcher(title);
    if (m.find()) return m.group(1).trim();

    return null;
}
```

6.5. Synchronization Strategy

- **Pull-based**: Frontend requests tasks from Google via Mavigo backend
- **On-demand geocoding**: Location tags processed during fetch
- **Local cache**: `UserTask` entities store geocoded coordinates
- **Lazy refresh**: No automatic polling; user triggers sync by viewing tasks
- **Bi-directional**: Changes in Google Tasks (completion, deletion) sync to local DB

6.6. Security Considerations

- **Token Storage**: OAuth2 tokens stored in `OAuth2AuthorizedClientService` (in-memory or JDBC-backed)
- **Token Refresh**: Spring Security automatically refreshes expired access tokens using refresh token
- **Principal Isolation**: Each user's Google account identified by unique `subject` claim
- **Scope Limitation**: Only requests necessary scopes (`tasks`, `profile`, `email`)

7. Installation Guide

7.1. Prerequisites

- **Java 21** or higher (LTS version recommended)
- **Gradle 9.1+** (or use included Gradle wrapper)
- **PRIM API access**: Contact Ile-de-France Mobilités for API credentials
- **Google Cloud account**: For OAuth2 and Google Tasks integration (optional but recommended)
- **Git**: For cloning the repository

7.2. Quick Start

```
# Clone the repository
git clone https://github.com/Aminmiri82/devops-project-MARLY.git
cd devops-project-MARLY/Mavigo

# Copy environment template
cp local.env.example local.env

# Edit local.env with your API keys (see Configuration section below)
nano local.env

# Run the application
./gradlew bootRun
```

The application will be available at **http://localhost:8080**

7.3. Configuration

7.3.1. Required Environment Variables

Create a `local.env` file in the `Mavigo/` directory with the following variables:

```
# PRIM API (Required)
PRIM_API_KEY=your-prim-api-key-here
PRIM_API_BASE_URL=https://prim.iledefrance-mobilites.fr/marketplace/v2

# Google OAuth2 (Required for Google Tasks integration)
GOOGLE_CLIENT_ID=your-client-id.apps.googleusercontent.com
GOOGLE_CLIENT_SECRET=your-client-secret

# Database (H2 in-memory - default)
SPRING_DATASOURCE_URL=jdbc:h2:mem:mavigodb
SPRING_DATASOURCE_USERNAME=sa
SPRING_DATASOURCE_PASSWORD=

# Application Server
SERVER_PORT=8080
SPRING_PROFILES_ACTIVE=dev
```

7.3.2. API Keys Setup

PRIM API

1. **Request access**: Contact https://prim.iledefrance-mobilites.fr/
2. **Create an application**: Register your app in their marketplace

3. **Get API key**: Copy the generated API key
4. **Set environment variable**: Add to `local.env` as `PRIM_API_KEY`

Google OAuth2

Follow the steps in **Section 6 - Google Tasks Integration** to:

1. Create Google Cloud project
2. Enable Tasks API
3. Create OAuth2 credentials
4. Configure redirect URIs:
   - Development: `http://localhost:8080/login/oauth2/code/google`
   - Production: `https://your-domain.com/login/oauth2/code/google`
5. Add Client ID and Secret to `local.env`

7.3.3. Optional Configuration

```
# Logging
LOGGING_LEVEL_ORG_MARLY_MAVIGO=DEBUG
LOGGING_LEVEL_ORG_SPRINGFRAMEWORK_SECURITY=DEBUG

# OAuth2 Client Service (for production)
# Use JDBC instead of in-memory storage
SPRING_SECURITY_OAUTH2_CLIENT_REGISTRATION_GOOGLE_SCOPE=openid,profile,email,https://w
ww.googleapis.com/auth/tasks

# Google Tasks API
GOOGLE_TASKS_API_BASE_URL=https://tasks.googleapis.com/tasks/v1
```

7.4. Building the Application

```
# Build JAR file
./gradlew build

# Skip tests during build
./gradlew build -x test

# The JAR will be in build/libs/mavigo-0.2.0.jar
```

7.5. Running Tests

```
# Run all tests
./gradlew test
```

```
# Run specific test class
./gradlew test --tests "JourneyPlanningServiceTest"

# Run with coverage report
./gradlew test jacocoTestReport
# Coverage report: build/reports/jacoco/test/html/index.html
```

7.6. Deployment

7.6.1. Local Development

```
# Run with live reload (using Spring DevTools)
./gradlew bootRun

# Run with specific profile
./gradlew bootRun --args='--spring.profiles.active=dev'
```

7.6.2. Production

```
# Build production JAR
./gradlew clean build -Pproduction

# Run the JAR
java -jar build/libs/mavigo-0.2.0.jar

# With environment variables
PRIM_API_KEY=xxx GOOGLE_CLIENT_ID=yyy java -jar build/libs/mavigo-0.2.0.jar
```

7.6.3. Docker (Optional)

```
FROM eclipse-temurin:21-jre-alpine
WORKDIR /app
COPY build/libs/mavigo-0.2.0.jar app.jar
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "app.jar"]
```

```
# Build Docker image
docker build -t mavigo:0.2.0 .

# Run container
docker run -p 8080:8080 \
  -e PRIM_API_KEY=xxx \
  -e GOOGLE_CLIENT_ID=yyy \
```

```
  -e GOOGLE_CLIENT_SECRET=zzz \
  mavigo:0.2.0
```

7.7. Troubleshooting

7.7.1. Common Issues

1. PRIM API 401 Unauthorized

```
Solution: Verify PRIM_API_KEY is correct and has not expired
Check: logs for "PRIM unauthorized" messages
```

2. Google OAuth2 redirect mismatch

```
Error: redirect_uri_mismatch
Solution: Ensure redirect URI in Google Cloud Console exactly matches:
  http://localhost:8080/login/oauth2/code/google
```

3. H2 Database console not accessible

```
Solution: Add to application.properties:
  spring.h2.console.enabled=true
Access at: http://localhost:8080/h2-console
JDBC URL: jdbc:h2:mem:mavigodb
```

4. Port 8080 already in use

```
Solution: Change port in local.env:
  SERVER_PORT=8081
Or kill existing process:
  lsof -ti:8080 | xargs kill -9
```

8. License

This project is licensed under the Apache License 2.0. See the LICENSE file for more details.

9. Appendix

9.1. Architecture Evolution

This documentation reflects **Mavigo v0.2**, which represents a major architectural evolution from v0.1:

Key Changes:

- **Journey Model**: Migrated from simple `Journey → List<Leg>` to three-tier `Journey → JourneySegment → JourneyPoint` hierarchy
- **Comfort Profiles**: Added comprehensive accessibility and comfort filtering system
- **Google Integration**: Full OAuth2 integration with Tasks API and location tagging
- **Disruption System**: Point-level disruption tracking with automatic rerouting
- **Task Optimization**: Multi-waypoint journey planning for task locations

9.2. Entity Counts

- **Core Entities**: 10 (User, Journey, JourneySegment, JourneyPoint, UserTask, Disruption, PointOfInterest, NamedComfortSetting, ComfortProfile, GeoPoint)
- **Enums**: 6 (JourneyStatus, SegmentType, JourneyPointType, JourneyPointStatus, TransitMode, DisruptionType, TaskSource)
- **Controllers**: 4 (JourneyController, UserController, GoogleTasksController, DisruptionController)
- **Services**: 10+ major service classes
- **API Endpoints**: ~50 endpoints across 5 categories

9.3. Contact and Support

- **Repository**: https://github.com/Aminmiri82/devops-project-MARLY
- **Issues**: https://github.com/Aminmiri82/devops-project-MARLY/issues
- **PRIM API Documentation**: https://prim.iledefrance-mobilites.fr/
- **Google Tasks API**: https://developers.google.com/tasks

Generated on 2026-01-30 for Mavigo v0.2.0