| | |
|---|---|
| **Project ref no** | AAL-4-032 |
| **Project acronym** | T&Tnet |
| **Project full title** | T&Tnet: Travel & Transport solutions through emotional-social NETworking |
| **Dissemination level** | Restricted Document |
| **Date of delivery** | 12/3/2014 |
| **Deliverable number** | 1.4 |
| **Deliverable name** | Final functional requirements and API Specification for T&Tnet Services |
| **Type** | Report (R) |
| **Status** | Final |
| **WP contributing to the deliverable** | WP1 |
| **WP / Task responsible** | TELLU |
| **Other contributors** | ITAINNOVA, GEO, ISOIN |
| **Author(s)** | TELLU (Lars Thomas Boye, Knut Eilif Husa)<br>ITAINNOVA (David Escuin)<br>GEO (Elena Valari)<br>ISOIN |
| **Keywords** | Functional requirements, Architecture, API |
| **Abstract (for dissemination)** | The purpose of this deliverable is to specify the overall T&Tnet system architecture and functional requirements. The system is comprised of various sub-systems to be implemented by different partners, which must be interconnected and work together. Requirements are specified both with regards to what functionality the system will provide to end users, and with regards to what interfaces and functionality the various sub-systems will provide to users and to the other sub-systems. |

T&Tnet

Travel & Transport solutions through emotional-social NETworking

# T&Tnet: Travel & Transport solutions through emotional-social NETworking

AAL-4-032

| **Deliverable**<br>**D1.4** |
| --- |
| **Final functional requirements and API Specification for T&Tnet Services** |

Restricted Document

## VERSION HISTORY

| Version | Edited by | Date | Description |
|---|---|---|---|
| 0.1 | Lars Thomas Boye | 30.10.2014 | First version of D1.4, based on final version of R1.4. |
| 0.2 | GEO team | 3.11.2014 | Complete Tips and Social API chapters. |
| 0.3 | Lars Thomas Boye | 24.11.2014 | (Re)written chapters 2, 3, 4 and 6. |
| 0.35 | GEO team | 24.11.2014 | Review chapters 3 and 6. |
| 0.4 | David Escuin | 26.11.2014 | (Re)written chapter 9 |
| 0.9 | Lars Thomas Boye | 03.12.2014 | (Re)written chapter 7, intro and conclusion, reviewed 8-10. |
| 1.0 | Lars Thomas Boye | 10.12.2014 | Review by Victoria Cristancho-Lacroix and Angeliki Angeletou |

# Table of Contents

# 1 Context and background

## 1.1 Project overview

To sum up the T&Tnet project idea, it is journey planning and navigation for the elderly, with user interfaces, functionality, content and artificial intelligence specifically for this user group. It includes multi-modal route calculation in multiple countries, with public transport data. It includes crowd-sourcing of accessibility tips for locations and of emotional feedback for transit routes. It includes a web application to plan trips and access information. And it includes a mobile application giving tracking and navigation.

Compared to general purpose route planners such as what you get with Google Directions and Maps, our goals are to be more user-friendly, personalised and adaptive, and to provide accessibility-related content specifically to meet the needs of elderly and others with reduced mobility. Our system stores user preferences and feedback to be personalised and adaptive. It also tracks the user when travelling, and can notify a caregiver if a problem is detected or the user presses an SOS button in the app, if the user wants such features. By addressing accessibility and safety, we hope to make it possible for more people to travel freely and without anxiety.

## 1.2 Objectives of the document

This document is a software architecture description, specifying the sub-systems and components making up the T&Tnet system and how they are connected. The main objectives is to specify requirements, both with regards to what functionality the system will provide to end users, and with regards to what interfaces and functionality the various sub-systems will provide to users and to the other sub-systems. As the T&Tnet system is comprised of several sub-systems, developed by different partners in different corners of Europe, defining the roles of the sub-systems and the interaction between them is essential for the success of the project.

The initial version of this document, the internal report R1.4, was a result of the technical analysis, specification and early system design work in the T&Tnet project, leading to the specification of the first prototype system. The shared data and APIs of the

system has continued to be refined and extended throughout the project, and we now present the final version of the functional requirements and APIs, which have been implemented in the final prototype system.

## 1.3 Relation to other deliverables

The functional requirements and system design was informed by the User needs analysis (D1.1), Scenario story boards (D1.2) and User requirements (D1.3). A T&Tnet services mock up (D2.1) was also documented as part of the early system design work. This deliverable replaces the internal R1.4, which is the previous iteration of this document.

The T&Tnet prototype system has been implemented according to the requirements and specifications in this document. The prototype is documented in the deliverables of Work Package 2:

- D2.2 & D2.3: First & Final travel and transport infrastructure prototype. The two deliverables together document the route calculation part of the T&Tnet system. D2.2 describes the concepts of the multimodal transport infrastructure and its implementation in T&Tnet, and the data collected for each city involved in the project. D2.3 gives a detailed documentation of the final version of the Planner API.
- D2.4 & D2.5: First & Final system intelligence prototype. D2.4 documents the intelligence in the route calculation part of the system. The other content of D2.4 is replaced by the updated description in D2.5, describing the system intelligence server and the navigation.
- D2.6 & D2.7: First & Final journey planning and social collaboration prototype. The final version (D2.7) gives a complete documentation of the T&Tnet web application, describing the journey planning and social collaboration functionality. It also gives an overview of the Social API, which is described in more detail in chapter 8 of this document.
- D2.8 & D2.9: First & Final T&Tnet integrated prototype. The first version (D2.8) is now outdated, with updated versions of the content distributed

between this deliverable (API specifications) and D2.9 (Mobile application and system integration).

## 1.4 Structure of the document

After a short introduction to the user and system roles, chapter 3 describes the high-level functional requirements from the end user perspective. We then look at two key forms of shared data, user preferences and tips, which need clear definitions as they are accessed and updated by several sub-systems. Chapter 6 describes the flow and interaction involved in the main use cases. Chapter 7 describes the structure of the system. Chapters 6 and 7 together specify the overall architecture, giving the functional and API requirements for each sub-system. The rest of the document provides detailed API specifications.

# 2 System overview

Here we present the main actors of the T&Tnet system – the user roles and system parts. This gives a first overview of the architecture, elaborated on in the rest of the document.

## 2.1 User roles

There is one main type of end user for the T&Tnet system – the elderly person being assisted in travel. This will be referred to as the primary user. There are also various secondary and tertiary user roles, although these have not been the main focus in the project. One secondary user role is that of a caregiver – someone who provides support to the primary user. This can be a relative, friend or professional caregiver. The primary user may want to get in contact with a caregiver in case of a significant problem.

One tertiary user role is for entering content from "official" sources. While the primary idea for getting content such as tips into the system is crowd-sourcing (the primary user enters tips), other stakeholders such as travel agencies, tourist offices and hotels are potential sources of accessibility information about locations, and our system includes this possibility.

In addition, there will be various administrative roles, managing the sub-systems and their content, but these are not considered here.

## 2.2 Overall architecture

Figure 1 gives a simplified overview of the T&Tnet system, showing the main sub-systems. This sub-system division follows from the various types of functionality needed, how responsibility for implementing this functionality is divided amongst the partners in the project, and the technology platforms used for the implementations (a more detailed architecture description is given in chapter 7). A shortened name is given to each sub-system, to easily refer to them in this document.

A short description of each follows. The *main roles* listed are based on the functional parts of the system first outlined in the Description of Work, and which sub-systems implement these.

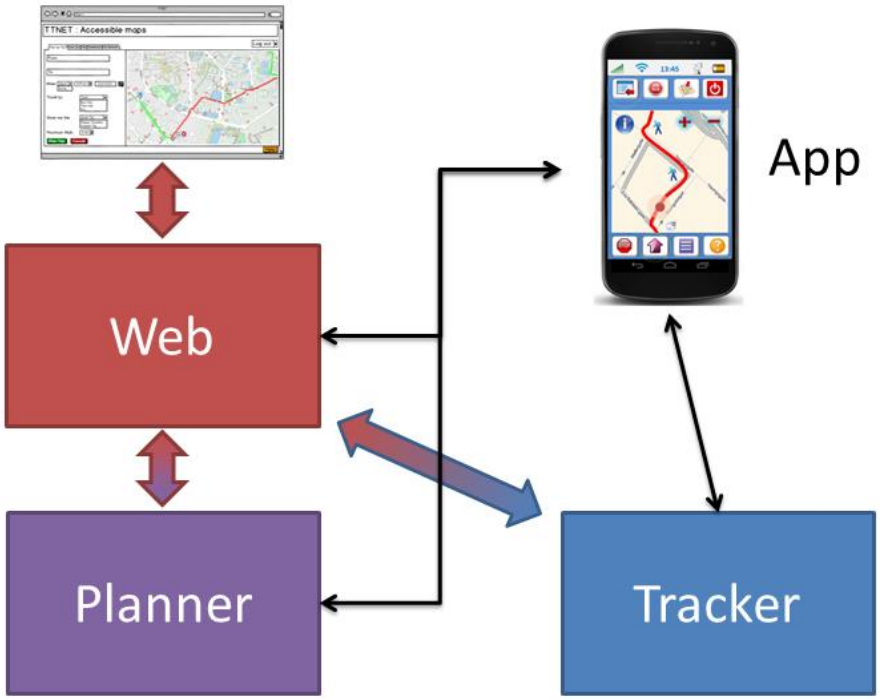


**Figure 1: System overview**

### 2.2.1 Web

Main roles: Social Collaboration Platform, Journey Planner, User interface

Responsible partner: GEO

Description: This sub-system includes a web application, database and API for access to the data. The web application is the T&Tnet front end for stationary use (as opposed to the mobile use of the app). The main functionalities available through the web interface include account management, journey planner with map, and entering of tips (accessibility-related location-bound information) and trip feedback. Both T&Tnet user accounts and tips are stored in this sub-system, and it has an API to retrieve and provide this data. The API is

referred to as the Social API, since it is part of the Social Collaboration Platform. Route calculation and storing of route feedback is handled by the Planner sub-system.

### 2.2.2 Planner

Main roles: Multimodal T&T infrastructure, Social Collaboration Platform

Responsible partner: ITAINNOVA

Description: A back-end sub-system providing the journey planner functionality accessible through the front-ends (Web and App). The front-ends allow the user to specify where to travel from and to, a time, and a number of travel preferences, and the Planner sub-system is responsible for calculating routes to suit the user input. It builds on the OpenTripPlanner platform, which provides the core planner algorithms for producing multi-modal routes, and adds a T&Tnet specific layer. The server instance has been set up with all the necessary map and transit data for the four pilot cities in the project. It is also responsible for storing routes planned by the user, and for storing feedback from trips. Feedback will be used to enrich route suggestions with information provided by those who have used those routes, to help the user select a route that suits their preferences. So it has an API for retrieval of planned routes and for giving feedback, in addition to requesting route planning.

### 2.2.3 App

Main roles: User interface, Navigation

Responsible partner: TELLU

Description: This is the T&Tnet smartphone application, running on the primary users' phones. It is the front end for mobile use - in connection with travelling and executing trips – and provides navigation based on planned routes. When a trip is active it uses sensors to track progress, both to provide navigation locally and to inform the server side. Much of the Journey Planner and Social Collaboration functionality is also available through this interface – getting a route for a specific destination, entering trip feedback and entering tips. It uses the APIs of all three other sub-systems.

### 2.2.4 Tracker

Main roles: System Intelligence

Responsible partner: TELLU

Description: This is a back-end, implemented by Tellu's SmartTracker sensor service platform. This sub-system provides tracking of users and System Intelligence. T&Tnet users are represented as tracked entities here, known as assets in SmartTracker. All user preferences are stored with the asset, allowing the reasoning engine of the platform to operate on them. It is the central store of sensor data such as positions, and makes it possible to share position data with a friend or caregiver. All sensor data are processed by a reasoning engine with configurable rule logic, which can result in actions such as preference improvements and raising of alarms.

# 3 Functional requirements

This chapter lists all the end user functionality implemented by the system, and specifies the sub-systems and prototype iterations for the implementation. This final list was established based on the results of the prototype iterations. The main difference from the initial list of functionalities was that friends-related function, aiming the coordinated use of the system for multiple primary users, such as travelling together and the support functions needed to connect friends, was removed.

## 3.1 Function specification

In this section we list the functionalities of the system, from the end user perspective, at a high level. The functionality is grouped in main categories, designated by letters, with a numbered list of functions in each category, so that we can refer to functions in this form: A.1, D.4, etc.

### *A. Manage user account*

The primary user needs to have an account in the system, with authorization data and preferences. So a new user must be able to create an account, and account details can be entered and later changed.

#### *A.1. Create user account*

An account in the system is created through the web, with an email address and password for authentication.

#### *A.2. Edit preferences*

The system may store a set of preferences for a user account. There are two categories of preferences, based on where they can be changed and how they are used.

- **Interface preferences**: These control the appearance and functionality of the mobile app. They are only edited in the mobile app, as the user should be able to see the effects of changing the preferences directly.
- **Travel preferences**: These are preferences related to the modes of travel, such as preferred transit modes, maximum transfers and walk distance. Some of these are available in the Journey Planner interface, but default values can be specified as part

of the user's preferences, and these default values will be used as pre-filled values in journey planning, so the user doesn't need to enter them each time a trip is planned unless there is a need to deviate from the personal default for a specific trip.

Note that this categorization is made here for technical reasons, but does not necessarily reflect how preferences is presented to the user (this is part of the user interaction design).

### *A.3. Reset password*

If a user forgets the password of the account, a new one can be requested. A generated password is sent to the user's email, along with a suggestion to change this to a new personal password.

## *B. Journey Planner*

These are the functionalities producing routes for navigation.

### *B.1. Browse map*

A map is available in both the web and mobile app. In addition to all the generic map data, T&Tnet-specific tips can be indicated on the map. The user can specify what types of tips to show (a checklist will be provided with limited categories). Selecting a tip on the map will show any additional information about that point, such as comments from users. The user can freely browse the map to learn about areas, as an aid while travelling or to support trip planning.

### *B.2. Plan trip*

This means to establish a trip plan for a future point in time. This is done through the web interface. The user specifies start and end points, either by clicking the map or by entering addresses. A departure or arrival time is specified. Travel preferences such as preferred transit modes and maximum walk distances can be specified. Travel preferences may have default values stored in the user preferences, which are used if left unchanged.

The system comes up with route suggestions based on all these parameters. The suggested routes are drawn on the map, with colours based on previous user feedback for the legs. The routes are listed, and selecting a route from the list brings up the full schedule

listing each leg of the route. The user can choose to accept one of the route suggestions, so that it is stored in the system and transferred to the mobile app.

### *B.3. Request navigation*

This is the mobile app equivalent of the trip planning described in the previous point. Rather than specifying a starting point and time, the current position and time is used as the starting point, and the user only specifies where they want to go (point on map). Travel preferences, with defaults from the user preferences, are available. Route suggestions are provided and selected between as described above, but accepting a route immediately starts navigation.

### *B.4. See planned trips*

Trips planned but not yet started can be listed. Selecting one, it can be displayed as when it was first produced, as lines on the map and listing the schedule of legs.

### *B.5. Cancel trip*

Related to the previous point, a planned trip not yet started can be cancelled, removing it from the system.

### *C. Navigation*

These are the functionalities guiding the user when travelling. All functions in the category are restricted to the mobile app.

### *C.1. Navigation guidance*

During a trip, the route and the user's positions within it can be displayed in several forms. The primary forms are map visualization and textual instructions. The route is shown on the map, indicating the separate legs (modes of transportation). The user's current position is indicated on the map, and the route indicates what part of it has been completed. Textual instructions are provided, and the instruction for the current step is always visible in the app. When walking (or bicycling), detailed step-by-step guidance is provided by the textual instructions. For transit, the guidance is which bus/tram/etc. to use, and to which stop. Additional navigation forms are designed and tested if needed and time permits. Navigation can be stopped at any time, cancelling the rest of the plan.

### *C.2. Deviation detection*

The position of the user is tracked whenever possible during a trip, and if the user is deviating from the route, either in space or time, this will be detected. There are two thresholds. Exceeding the first, the user is notified and guided to get back on track. If a further threshold is exceeded, where the current route is no longer feasible, the user is notified of this, and given a choice between a recalculated route to the current destination, or cancelling the navigation. Secondary users such as friends and relatives may be notified of continuing deviation.

### *C.3. Error notification*

The user is notified during a trip if the navigation functionality is compromised, such as from network or GPS unavailability. Any possible corrective activity from the user's part is described.

### *C.4. Route recalculation*

The route to the current destination is recalculated on a strong deviation, or if starting it from a different time or place than that planned. The user may also request a recalculation if they observe that conditions have changed.

### *C.5. Pause navigation*

The user may put the trip on hold, with no navigation or deviation detection until it is resumed. This can be useful when stationary/waiting, to preserve the battery of the phone or to extend the stay at a location. On resuming navigation, route recalculation may be necessary.

## *D. Social Collaboration*

These are the crowdsourcing functionalities, where users provide information about trip legs and locations.

### *D.1. Emotional feedback*

The user is asked for an emotional response to each leg of a route travelled by public transport. Seats availability and travel speed has been chosen as the attributes to gather feedback on. In the mobile app, the user is asked for this feedback after the trip is finished

or navigation is stopped. If the feedback is not given in the app, the user can give it on the web. Completed trips with incomplete feedback are listed here, and the user can indicate any problems or discomfort with the legs.

### *D.2. Enter tip*

A tip is an accessibility-related piece of information about a specific location. Tip types include the location of escalators, stairs, elevators and toilets. We also support temporal (time-limited) tips, for when a street or accessibility enabler is blocked, broken etc. Each tip is associated to an icon. To enter a tip, the icon is selected by pressing/clicking on it. In the web interface, the location is chosen by clicking the map or writing an address, while in the mobile app the current position is used. In addition, a comment may be provided, to describe the accessibility in more detail, or enter complementary information about the spot. Additional comments can also be provided for existing tips.

## *E. Communication*

Ways for the user to contact others, and for the system to contact users.

### *E.1. Send alarm*

The mobile app includes an alarm button, allowing the user to activate a signal when he is in trouble and/or needs help. The signal is sent to the Tracker server. Exactly how the alarm is handled is a system configuration tailored to each user, but it will typically include secondary user notification.

### *E.2. Initiate phonecall*

The alarm button in the mobile app may also initiate a direct call to one pre-configured contact, to quickly and easily call for help in an emergency.

### *E.3. Secondary user notification*

People who need not be users of the system in any other respect can be entered as contact persons (with phone number or email addresses) to a primary user, to receive a message (by SMS or email) when the primary user is lost or sends an alarm.

## 3.2 Function priority

The following table lists all the functions specified in the previous section. The columns for Web and App give the prototype iteration the function was specified for, for the web and mobile app respectively, showing when and where the functionality has been implemented.

| Code | Function | Web | App | Comments |
|---|---|---|---|---|
| A.1 | Create user account | 1 | - | |
| A.2 | Edit preferences | 2 | 1 | |
| A.3 | Reset password | 2 | - | |
| B.1 | Browse map | 1 | 1 | |
| B.2 | Plan trip | 1 | - | |
| B.3 | Request navigation | - | 2 | |
| B.4 | See planned trips | 2 | 1 | |
| B.5 | Cancel trip | 2 | 2 | |
| C.1 | Navigation guidance | - | 1 | Map mode initially, others may be added later |
| C.2 | Deviation detection | - | 1 | Improved for 2. iteration |
| C.3 | Error notification | - | 2 | |
| C.4 | Route recalculation | - | 2 | |
| C.5 | Pause navigation | - | 2 | If wanted as part of user interface |
| D.1 | Emotional feedback | 2 | 1 | |
| D.2 | Enter tip | 1 | 2 | |
| E.1 | Send alarm | - | 2 | |
| E.2 | Initiate phonecall | - | 2 | |
| E.3 | Secondary user notification | - | - | 2 (SMS/email sent from Tracker server) |

# 4 User preferences

In this chapter we discuss and document the user preferences in the T&Tnet system. This is one of the system aspects which spans all sub-systems and which therefore needs collaboration between all technical partners, as well as conceptual and user interface input from HCI partners. On the technical level, we need to coordinate what they are called internally and which values they are allowed to have between the systems which interact with them. On the HCI level, we need to specify which preferences will be available to the user in the various interfaces, and how they are presented there, making sure it is consistent between the web and mobile app interfaces.

## 4.1 About user preferences

We have defined two categories of user preferences. One is **interface preferences**, regarding how the user interacts with the system. In addition to GUI settings, we have included the specification of a contact person in this category. The other is **travel preferences**, which is directly related to trip parameters. It is important to understand this relationship. The **trip parameters** are parameters to guide the calculation of routes for a trip, such as preferred methods of transport. Some of these parameter values can be specified by the user while others cannot, because not all makes sense to a user. For instance, how fast the user walks is something that we want the system to try to learn, rather than ask the user.

Of those travel preferences the user can edit, some will also be available as trip parameters in the trip planning interface. The idea is that we store a user preference for each trip parameter, and if the trip parameter is also available in the planner user interface the preference value is used as a *pre-filled, default* value. So the trip parameters are already filled in based on preferences, but the user may change these parameter values if they wish, in case they want a different value for a particular trip. Labels of trip parameters should match labels of the corresponding preferences, as far as possible. So travel preferences and trip parameters need UI coordination, and both will be available in web and app, which also

needs coordination. This coordination is both what the preference/parameter is called in each language, as well as which values the user can choose between.

The travel preferences/parameters presented to the user are directly based on the parameters of the API call for route planning made to the Planner sub-system, which in turn is based on OpenTripPlanner. Since these are parameters for a software algorithm, and not all are understandable or relevant to a non-technical end user, we have limited the selection of which parameters to include as preferences.

The preferences are stored in SmartTracker (the tracker/intelligence sub-system), as part of the asset entry which represents the user in this sub-system. From here they can be accessed and updated by the web and mobile app sub-systems, through SmartTracker's REST API.

## 4.2 Preference listing

Table 1 gives an overview of the preferences. *Code* is the internal name – the name used in the SmartTracker asset entry. *UI name* is how the preference should be labelled for the user, or blank for preferences which should not be accessible to the user. This is an important point to agree on, and implement in the same way in all user interfaces. These are the English labels; translations in other languages must also be coordinated between web and mobile app. *Value* gives the data type used for the SmartTracker storage, as well as range of valid values (see detailed preference descriptions for more). *Default* value is what the preference should be initially, before the user or the system has edited it.

The preferences should also be presented in the same order in each user interface. Therefore, the order in this list is significant (those not intended for UI are put last and have no UI name). See the next sections for descriptions of the preferences.

**Table 1: User preferences**

| Code | UI name | Value | Default |
|---|---|---|---|
| Interface preferences | | | |
| highContrast | High contrast | Boolean | false |
| contactName | Alarm contact name | String | |
| contactPhone | Alarm contact phone number | String | |
| Travel preferences | | | |
| mode | Transport modes | Selected from list, see discussion | All except BICYCLE |
| maxWalkDistance | Maximum walk distance | Integer, 500-10000 meters | 1000 |
| stairs | Can use stairs | Boolean | true |
| elevators | Can use elevators | Boolean | true |
| walkSpeed | - | Floating-point (0.1 – 5) | 1.3 |

## 4.3 Interface preferences

Only one of these is really a user interface option, and so far only for the mobile app. Text size could have been another, except in Android this is a system preference and not something to control in an app. We include the contact person – who to contact with the "SOS" function of the app – in this category.

### 4.3.1 contactName

Description: The name of the person to contact with the "SOS" function of the app.

Value selection: Input box, for text

### 4.3.2 contactPhone

Description: The phone number to call with the "SOS" function of the app (should match the name).

Value selection: Input box, limit input to phone number characters if feasible

### 4.3.3 highContrast

Description: Turn on to use a high-contrast theme in the mobile app.

Value selection: Checkbox

## 4.4 Travel preferences

Travel preferences are directly based on the Planner API request. These are taken from OTPs API documentation[1], as well as trip parameters added by T&Tnet. We have selected the parameters which seem relevant to configure with preferences. Those that seem the most relevant for end user input should be available to the user when planning, as trip parameters, while the rest should be used directly from preferences. The internal names in SmartTracker match the OTP parameter names, making it easy to use them directly. Note that order in this section is alphabetical.

### 4.4.1 elevators

Description: Whether elevators can be included in calculated routes. This is one of the new parameters T&Tnet has added to the Planner. Note that it only concerns walk legs of routes, not general accessibility. Not many places have elevators to include in walk legs, and it is hard to imagine wanting to rule them out, but perhaps claustrophobia is one use case. We include it, with true as default value. That it is only for disabling if you really do not want elevators in walk legs is something which should be made clear in the preference interface.

Value selection: Checkbox

### 4.4.2 maxWalkDistance

Description: The maximum distance (in meters) the user is willing to walk. This parameter is a bit problematic. The planner will not produce routes with longer walk distance, so a low value may cause no routes due to the morphology of the city and the maps. Therefore, it needs to have a lower limit on what the user is allowed to specify (500 meters). The parameter value should be higher than the user's preferred walk distance, so that routes are produced even if they are suboptimal.

Value selection: Input box, for meters

[1] http://docs.opentripplanner.org/apidoc/0.11.0/resource_Planner.html

### 4.4.3 mode

Description: The set of modes that a user is willing to use. This is the most important travel preference, controlling what types of legs can be produced. The modes are as follows, with internal values and suggested labels:

**Table 2: Travel modes**

| Value | UI name |
|---|---|
| WALK | Walk |
| BICYCLE | Bicycle |
| TRAM | Tram |
| SUBWAY | Subway |
| RAIL | Rail |
| BUS | Bus |
| FERRY | Ferry |

Internally it is stored as a comma-separated list of values. OTP also has modes CAR, CABLE_CAR, GONDOLA and FUNICULAR, but they not are relevant for our trials. There are also the combo values TRANSIT, TRAINISH and BUSISH, but we will just include a “Select all” button in the user interface to tick all boxes (maybe except Bicycle). The default should be all.

Value selection: This should be multi-selection, so that each mode can be toggled individually.

### 4.4.4 stairs

Description: Whether stairs can be included in calculated routes. This is one of the new parameters T&Tnet added to the Planner. Note that it only concerns walk legs of routes, not general accessibility, something which should be made clear to the user.

Value selection: Checkbox

### 4.4.5 walkSpeed

Description: User's walking speed in meters/second. Not to be shown to the user. We have a rule in SmartTracker to modify it based on observed speed.

## 4.5 Dropped OTP parameters

The following are parameters of the OTP API which were originally included in our preferences list, but which have been removed because user configuration was unwanted or unneeded.

### 4.5.1 bikeSpeed

Description: The user's biking speed in meters/second. This is not a value which the user can relate to. If it were to be included it should be found by observation as with walkSpeed, but cycling is not a focus in T&Tnet.

### 4.5.2 maxTransfers

Description: The maximum number of transfers that a trip will be allowed. OTP documentation says "one plus the maximum number of boardings", with a numerical value, or "Any" to use the server's own limit. We don't see much need to enforce a specific limit – it is usually best to use the default "Any" and get the possible routes to select from.

### 4.5.3 minTransferTime

Description: The minimum time, in seconds, between successive trips on different vehicles. This parameter seeks the perfection of the OTP but it is unmanageable in practice. It is difficult for users to specify, as there are many variables involved, and also difficult to know without timing many transfers with a stopwatch. So the recommendation from those who work on the Planner part of the system is to use a fixed value between 60-90 seconds.

### 4.5.4 optimize

Description: The set of characteristics that the user wants to optimize for. The following table shows the options given by OTP.

**Table 3: OTP optimize values**

| Value | UI name |
|---|---|
| TRANSFERS | Fewest transfers |
| QUICK | Quick trip |
| SAFE | Bike friendly |

TRANSFERS is said to be obsolete, in favour of a transferPenalty parameter, and SAFE doesn't seem so relevant since it seems to be for bicycle. So we will always use QUICK, and not give a choice.

### 4.5.5 wheelchair

Description: Whether the trip must be wheelchair accessible. Although the exact meaning of this OTP parameter isn't clear, it should exclude stairs from walk legs, which is something we have the *stairs* parameter for.

# 5 Tips

One of the most important services of the T&Tnet platform is the "Tips". Tips are stored to the Geodatabase in the Web application sub-system. The management of the "Tips" are made in the Social API and are used from all the main sub-systems of the platform. In particular, the user can provide tips or get tips by using the mobile application or the web application. In addition, tips are used by the Planner sub-system in order to propose the routes to the users.

The tips categories which are supported by the first prototype are fully reconstructed based on the users' evaluation. The new tips categories are strictly related to the accessibility issues and made use of the existing tips on the Open Street Map. In addition, in the final version the tips duplication aspect was solved. In the next paragraphs, there is a more detailed description of the new tips categories and all the related issues.

## 5.1 Users profiles

The new tips categories are highly related to the different supported users' profiles. In particular, the T&Tnet system supports 3 types of user profiles which are described below:

- **Administrator user** (platform administrator): The user who manages and supports the T&Tnet platform. For example, approve users' requests etc.
- **Official users** (eg. councils, enterprises, etc. It will be validated by the administrator):
    - *Web app Interface*: In the registration page, there is a check box, so the user can ask to be an official user. At first, this user is not stored directly as official user, but the administrator of the platform will approve this request.
    - *Technical aspects:* Users table is modified and one more column is added for the type of user. Official user type value: 2.
- **VIALE end users**:
    - *Technical aspects*: Users table is modified and one more column is added for the type of user. VIALE user type value: 0.

## 5.2 Tips & Supported Services

There are two services in the T&Tnet platform which are related to the tips. The first service is used in order to provide / store new tips in the system and the second one to retrieve / get the stored tips. In the sequel, these services are described extensively.

### 5.2.1 Give Tip – Service

The users can provide geo-location information to the system. This will be done through the **T&Tnet** web application by using the "*Give Tip*" interface[2]. The main points of this service are given below:

- A simple list with choices is displayed to the user. There are no categories in the final version as in the first one. The categorization is internal and is made based on the choices of the list (especially icons). Figure 2 presents this simple list with the corresponding icons.
    - This list is classified into two parts:
        - Positive Tips
        - Negative Tips


**Figure 2: List of tips – icons**

[2] Deliverable D2.7 - Final Journey Planning and Social Collaboration Platform, section 4.3.3

- When the users selects the location for his tip (by clicking or giving address), a call to the nominatim API (https://nominatim.openstreetmap.org/) is made in order to get back info for this location based on the OSM. We do this call in order to use the existing info about the places which are provided by the OSM. As we mentioned before, in the T&Tnet platform will be stored only accessibility info which are related to places based on OSM.
    - **Technical Aspects**:
        - **Nominatim API call** (example): http://nominatim.openstreetmap.org/search?format=xml&q=41.648832227075005,-0.8849569378635777&addressdetails=1
        - **temporaltip** table is modified in order to store only the tips which can be used by the planner.
            - **Table schema:** (id, userID, icon_image, idTipOSM, typeTipOSM,) valueTypeTipOSM, datein, duration,comment, lon, lat, lonMP, latMP, road, city, county, number, geocoding)
        - **poitip** table is modified in order to store the rest of the tips, based on the different types. The distinction between the types of tips is made by the different icons.
            - **Table schema:** (id, userID, icon_image, idTipOSM, typeTipOSM,) valueTypeTipOSM, datein, comment, lon, lat, lonMP, latMP, road, city, county, number, geocoding)
        - **Internal tips categorization – icons**: Table 4 shows the internal categorization of the tips, in order to be used efficiently by the sub-systems of the platform. In addition, the green fields of the Table 4 indicate the type of tips which are used by the Planner API in order to produce the routes.

**Table 4: Internal tips categorization – icons numbering**

| No. | Tip Type-Name | Db Table | Icon value |
|---|---|---|---|
| **1.** | Elevator | poitip | 1 |
| **2.** | Escalator | poitip | 2 |
| **3.** | Stairs | poitip | 3 |
| **4.** | Toilets | poitip | 4 |
| **5.** | Bike | poitip | 5 |
| **6.** | Bench | poitip | 6 |
| **7.** | Broken Elevator | temporaltip & poitip | 7 |
| **8.** | Broken Escalator | poitip | 8 |
| **9.** | Slippery/Broken stairs | temporaltip & poitip | 9 |
| **10.** | Dysfunctional Toilets | poitip | 10 |
| **11.** | Dirty Benches | temporaltip & poitip | 11 |
| **12.** | Blocked road | temporaltip | 12 |
| **13.** | Cracked Tiles | poitip | 13 |
| **14.** | Traffic lights with insufficient time to cross | temporaltip & poitip | 15 |

### 5.2.2 Get Tips – Service

The users can look for tips in a specific area of interest through the T&Tnet platform by using the "Get Tips" service[3]. In this final version of the T&Tnet system, we reconstruct the previous version of this service by using 2 filtering levels. More specifically:

(i) the first filtering level is based on user profiles
   a. Official user
   b. VIALE user
   c. My tips

(ii) the second one based kind of the tips: The user can check / select which of the tips in the list (See Figure 2): wants to get back.

*Tips duplication*

[3] Deliverable D2.7 - Final Journey Planning and Social Collaboration Platform, section 4.3.2

Tips duplication is an important issue for the platform. We handle this issue in order to offer to users, the stored tips which are multiples for a specific geo-location in an easier way. Particularly, when for a geo-location there are more than one tip, the system will display only one time the icon-tip on the map and in the popup message for the specific tip will be shown all the stored comments. Figure 3 shows an example of how the system displays this kind of tips. The key point of this procedure is the call to the Nominatim API which is used in order to get the unique id of the tip-location based on the OSM.

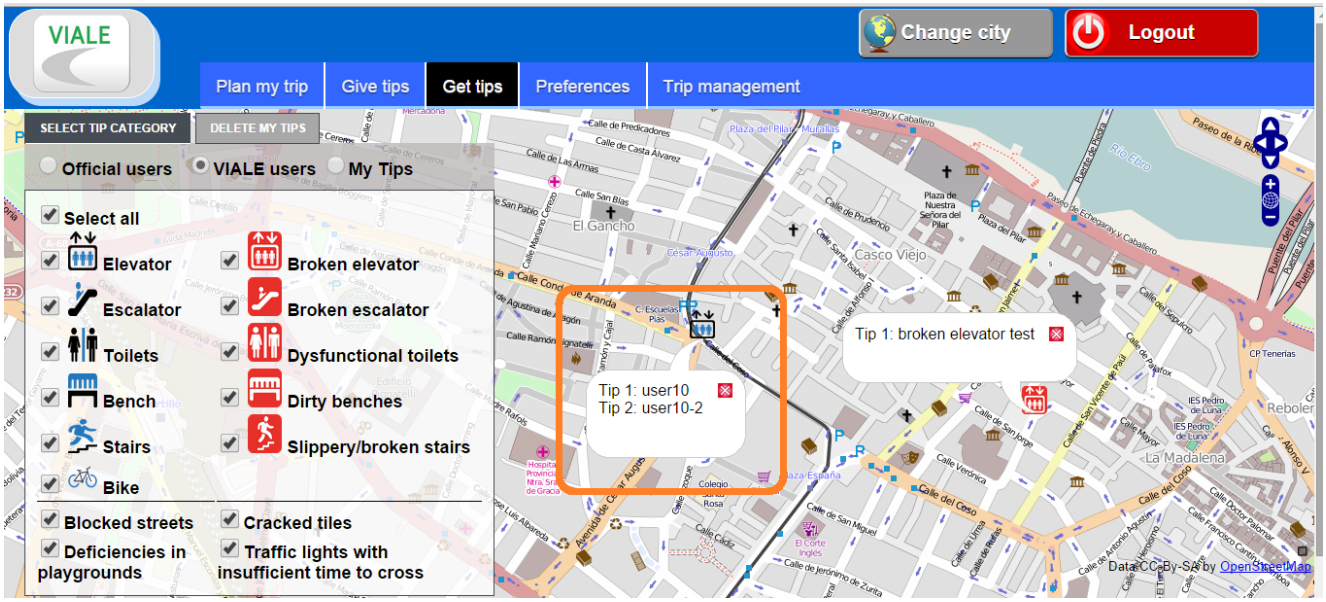


**Figure 3: Get tips View - Multiple comments in the same tip**

# 6 Use cases with sub-system interaction

Based on the functional requirements and the sub-systems which will implement them, the important use cases have been identified and a high-level system design has been done to work out all the interactions needed, both between the users and the system and between the sub-systems. The actors in these use cases are the user roles (Primary user and Caregiver) and the sub-systems (Web, Planner, Tracker, App) introduced in chapter 2. While this chapter focuses on the processes, it is complimented by chapter 7, focusing on the structure. The architecture decisions placing data and processing in specific sub-systems are explained in that chapter.

The five use cases are described using a common table schema. The main section is the Main Flow, where the use case is broken down into an ordered list of interactions. Each table is followed by a UML sequence diagram showing the interactions. The use cases are not fully complete with respect to the possible functionality and client type permutations, but near enough to cover all forms of sub-system interactions.

The sub-system interaction specification is a key result of the design process, and forms the bases for the interface/API-specification in the following chapters. The use cases and sequence diagrams are presented here in their final form, as implemented. The only significant change from the initial draft version of this chapter in R1.4 is the exclusion of the "Friend role" from the system, referring to the functionalities for travel with a friend. Other than that, minor refinements have been done in the implementation phases.

## 6.1 Create and configure account

| | |
|---|---|
| Use Case Number | 1 |
| Use Case Name | Create and configure account |
| Actors | Primary user, Web, Tracker, App |
| Summary | Covers all the steps of creating and configuring a T&Tnet account through the web, as well as logging in with this account on the app for the first time. |
| Trigger/ intent | User enters the T&Tnet website front-end |
| Preconditions | The primary user is not yet registered in the system, but wishes to be a user. The primary user has an email address, and this is not yet registered as a user in the system. |
| **Flow of events:** | 1. Enters a valid user name (email address) and password. |

| | |
|---|---|
| (Main Flow) | 2. Web checks there is no user id that uses the same user name.<br>3. Web contacts Tracker to create account data for the user there – asset with preferences representing user and device representing the mobile app. The asset id is returned on success.<br>4. Web database adds this new user to the platform, along with Tracker asset id.<br>5. User is displayed the account was created successfully and he is granted access to the main page.<br>6. User opens the preferences page, to review and edit preferences. This is stored in the asset object in Tracker, which must be retrieved to populate the preferences form.<br>7. The asset comes with default values for preferences, which the user can modify.<br>8. When the user presses save on the web, the asset object must be updated from the preferences form and returned to the Tracker server for storage.<br>9. To start using the mobile app, the user logs in with the username and password used when creating the account.<br>10. The app contacts the Web server through its API to execute the login. If the username and password are correct, the Web returns the username, password and asset id identifying the user in Tracker.<br>11. The app then contacts the Tracker server and requests the asset object with preferences, based on the asset id. From the asset it also gets the associated device id used to identify sensor observations sent to Tracker. |
| Alternate flows | • User is requested to choose another password and name<br>• Password and name do not meet the correct format<br>• User abandons website |
| Exceptional flows | • Tracker operation fails: Account creation fails with error message. |
| Displayed information | Form to enter username and password. |
| Postconditions | The primary user has a configured account in the system, and may start using the mobile client and social collaboration platform. |
| Relation to other use cases | None of the rest of use cases can be performed unless this has been done successfully. |

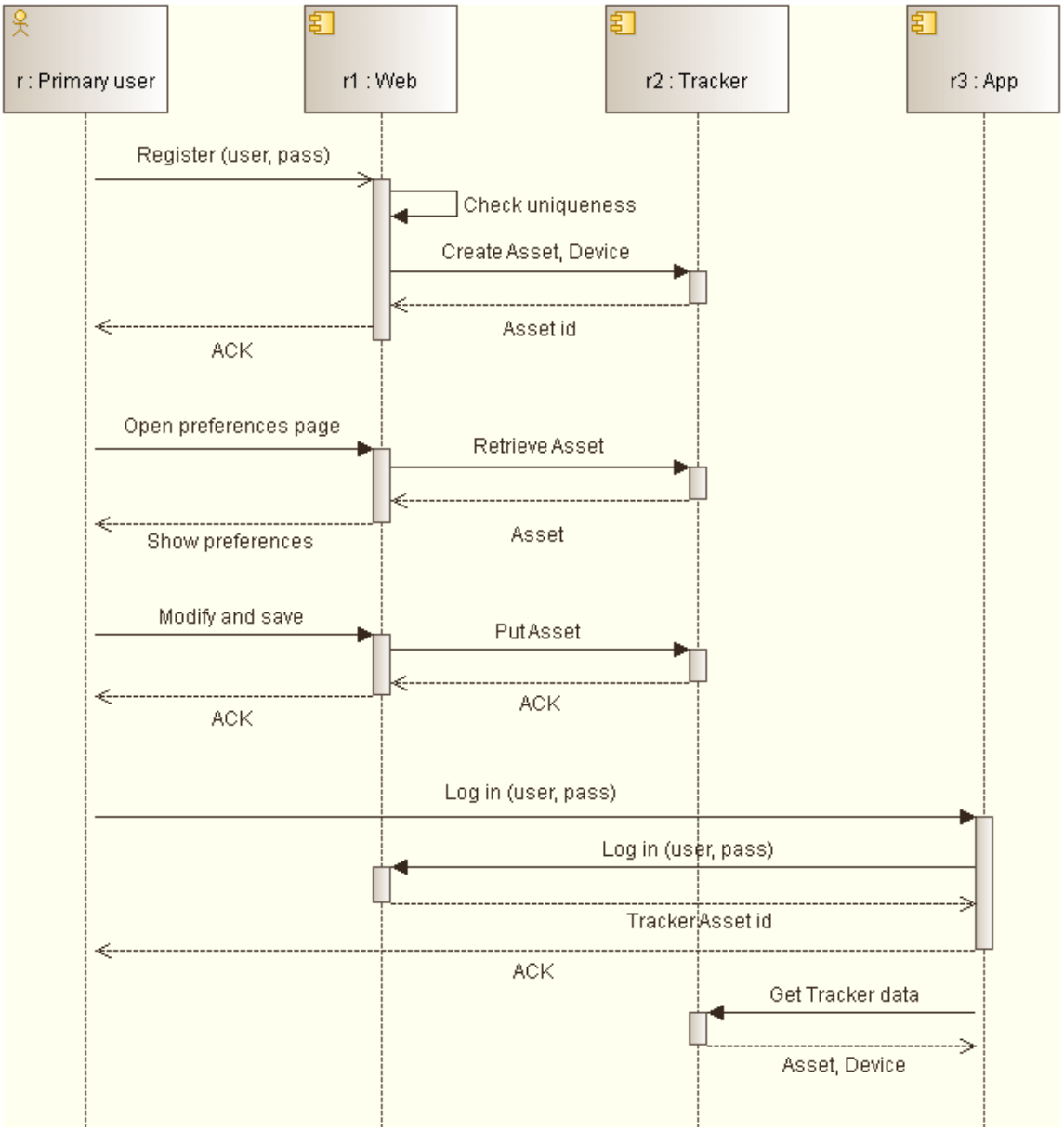


**Figure 4: Sequence diagram for use case 1 – Create and configure account**

## 6.2 Plan a trip

| | |
|---|---|
| Use Case Number | 2 |
| Use Case Name | Plan a trip |
| Actors | Primary user, Web, Planner, Tracker, App |
| Summary | User plans a trip through the web interface. Routes are calculated for the user to acknowledge. |
| Trigger/ intent | User pushes *new trip* |
| Preconditions | User wants to plan a trip, knowing where she wants to travel from and to (can write it down or pinpoint in a map). |

| | |
|---|---|
| **Flow of events:**<br>(Main Flow) | 1. User enters the Journey Planner (main page) of the web.<br>2. Default values for travel preferences are stored in the asset object. If not currently in memory, the web server must retrieve this from the Tracker server, to use as suggested values in the planner input fields.<br>3. The user provides departure and arrival points, and either a start or arrival time. The default travel preferences may also be modified for this trip.<br>4. The web server sends the parameters provided by the user along with user identification to the Planner sub-system.<br>5. The route calculation additionally needs tips for blocked locations within the area of the route (stored in the web server database).<br>6. The Planner calculates routes, and stores them in its database, marked as suggestions. The ACK to the Web contains the number of routes.<br>7. The Web retrieves the route suggestions from the Planner, and displays the routes.<br>8. User selects one of the routes proposed and acknowledges her request.<br>9. The Web informs the Planner of the route choice, by updating the status of the selected route to 'accepted' and deleting any others.<br>10. When the app is running (as a background process on the phone), it periodically polls the Planner for updated routes, to stay in sync. So the new route is automatically synced to the app, which can schedule notifications to the user based on the planned departure time. |
| Alternate flows | The route calculation may produce one, several or no potential routes. Possible reasons why a route can't be produced:<br>• Address cannot be found<br>• Impossible to route destination (lack of a transport mode selected, or other kind of reason) |
| Exceptional flows | The route planning will fail with an error message if one of the servers can't be reached. |
| Displayed information | User is shown a map with routes and possible alternatives. |
| Postconditions | Route is stored in the Planner with planned status, and synced to the phone if the app is running. |
| Relation to other use cases | *Request navigation* is the equivalent using the mobile app rather than the web. *Execute trip* at departure time. |

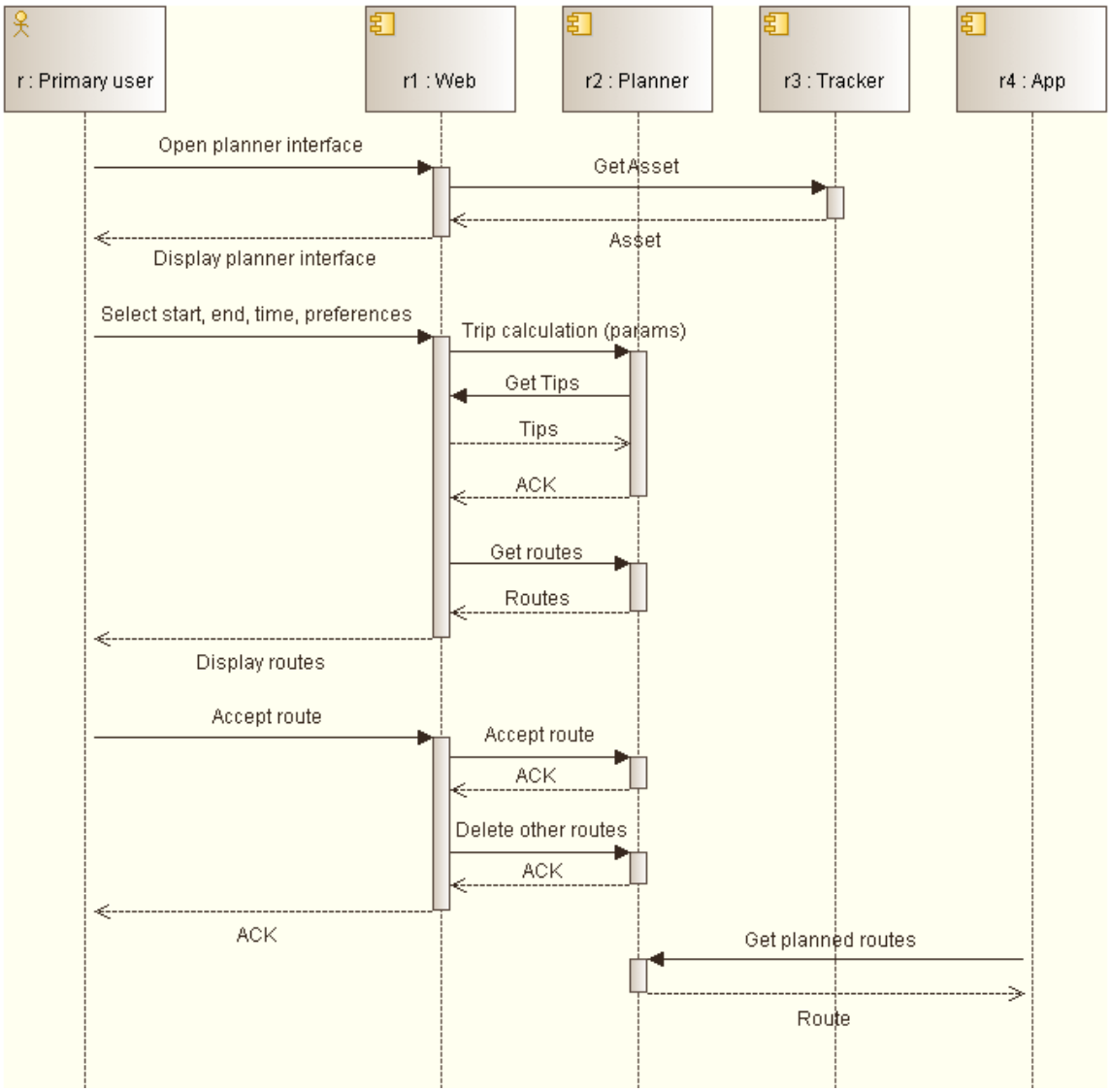


**Figure 5: Sequence diagram for use case 2 – Plan trip**

## 6.3 Request navigation

| | |
|---|---|
| Use Case Number | 3 |
| Use Case Name | Request navigation |
| Actors | Primary user, App, Web, Planner |
| Summary | User wants navigation guidance to get to a specific place as soon as possible, and requests navigation through the mobile app. Routes are calculated for the user to acknowledge. |
| Trigger/ intent | User wants navigation guidance, and pushes *new trip* in the app interface. |
| Preconditions | User has an account and is logged in on the app. User knows where she wants to go. |
| **Flow of events:** | 1. User enters the journey planner interface in the app. |

| | |
|---|---|
| (Main Flow) | 2. The journey planner includes a map. Tips are retrieved for the shown area to display the map. Tips are cached locally, but the Web server is also contacted to get updated tips.<br>3. The user selects a wanted destination (pointing on the map).<br>4. The App asks the Planner to do route calculation, with the current position and time as departure, user id, and the travel parameters from the user's preferences.<br>5. The route calculation additionally needs tips for blocked locations within the area of the route (stored in the web server database).<br>6. The Planner calculates routes, and stores them in its database, marked as suggestions. The ACK to the App contains the number of routes.<br>7. The App retrieves the routes and lists them for the user. The user can see route details and display the routes on the map.<br>8. User selects one of the routes proposed and acknowledges her request.<br>9. Any routes not chosen are deleted from the Planner with delete requests.<br>10. With a route successfully chosen, the app switches to navigation mode. The chosen route has its status changed to navigation, and this is communicated to the Planner. |
| Alternate flows | • The default travel preferences may also be modified for this trip.<br>• As with *Plan a trip*, a route may not be found (can't find address, no possible route to destination). |
| Exceptional flows | The route planning will fail with an error message if one of the servers can't be reached. |
| Displayed information | User is shown a map with routes and possible alternatives. |
| Postconditions | Route is stored in the Planner DB and the phone, and navigation is starting. |
| Relation to other use cases | *Plan a trip* is the equivalent using the web rather than the mobile app. Directly followed by *Execute trip* if a route is accepted. |

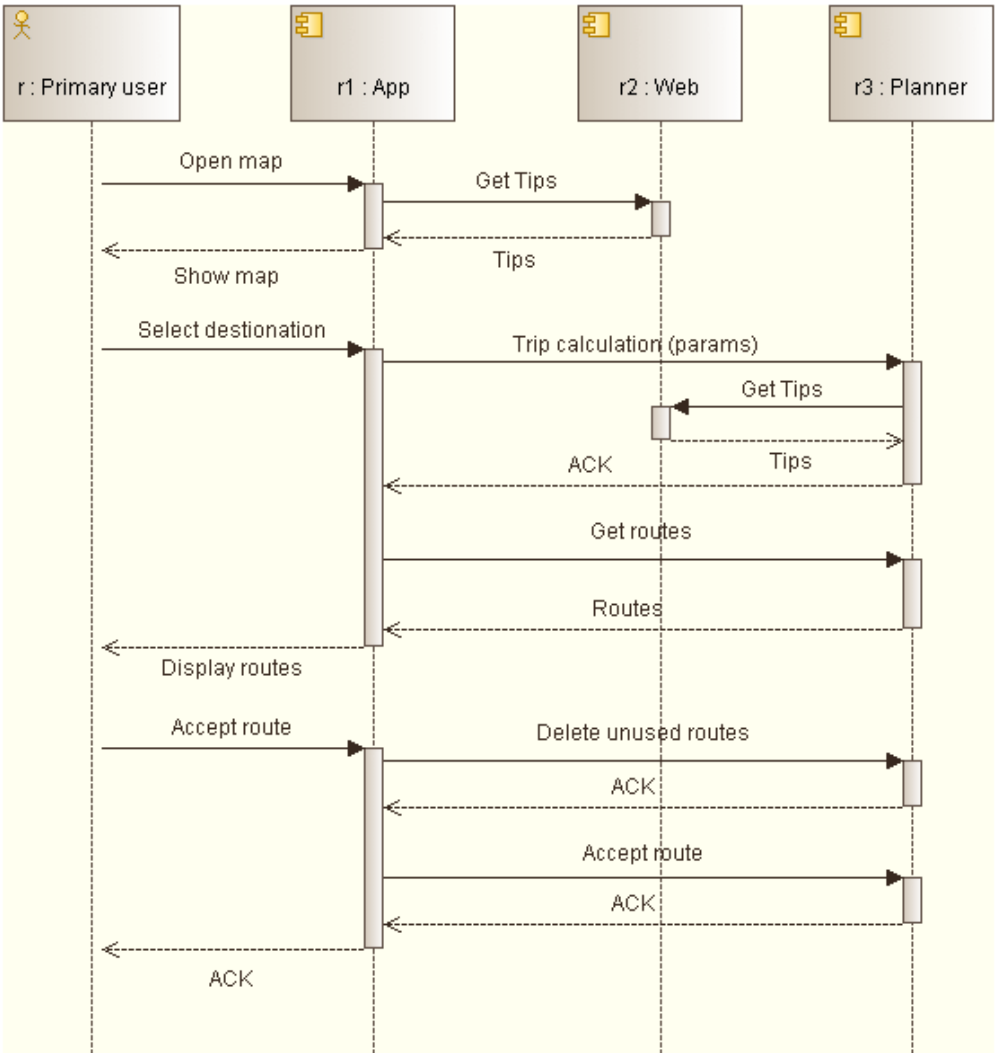


**Figure 6: Sequence diagram for use case 3 – Request navigation**

## 6.4 Trip execution and feedback

| | |
|---|---|
| Use Case Number | 4 |
| Use Case Name | Trip execution and feedback |
| Actors | Primary user, App, Web, Planner, Tracker, Caregiver |
| Summary | User is given navigation guidance based on a route. The system monitors progress and detects deviation, recalculating the route if navigation is to continue after significant deviation. The caregiver can be notified on repeated deviation. At completion, the user is asked for emotional feedback. |
| Trigger/ intent | Either selecting a pre-planned route to use, or the acceptance of a route suggested by the app based on navigation request. |
| Preconditions | User has an account and is logged in on the app. A route is present in the app, and the user has agreed to start the trip. |
| **Flow of events:** | The use case is made up of several sub-cases, each illustrated in |

| | |
|---|---|
| (Main Flow) | separate sections of the sequence diagram. First, initiation:<br>1. The user starts a trip, either a pre-planned trip or one produced by a *new trip* request (use case 3).<br>2. The App enters tracking mode. The Tracker server is informed that the user is now on a trip, and the route status is updated on the Planner server.<br>While on a trip, the App regularly reads the current position from the GPS sensor. Repeated at each position update – blue section in diagram:<br>3. The App displays the route (if the map is shown) and instructions for the current step.<br>4. The position is sent to the Tracker server, where it is persisted. It is also processed by the reasoning engine.<br>5. When a position update indicates a change in route leg, this is communicated to the Planner server, which keeps track of the route state.<br>User deviates from the route – red section of diagram:<br>6. When a position update shows the user is having trouble staying on the route or is falling behind the schedule, the user is notified and a deviation is reported to the Tracker server.<br>7. In this case the user doesn't get back on track, instead letting the deviation grow until the current schedule is no longer feasible. The app again sends a deviation event to the Tracker server.<br>8. The reasoning engine of the tracker server can be configured to count and take action on significant deviations. In this case it is reported to a caregiver via SMS.<br>9. The significant deviation means the old route can no longer be used for navigation. The app asks the user if she wants to continue the trip.<br>10. Choosing to continue, a recalculated route is needed. The App first sends a request to abort the trip to the Planner server, which updates the route state and returns the travel parameters used for the original route calculation.<br>11. The app is now ready to get a new route, from the current time and place and to the original destination. This follows the same procedure as in navigation request, starting with a trip calculation request to the Planner server, but there is no user interaction.<br>12. The Planner calculates routes, getting tips from the Web database, and returns the number of routes to the app. The app then retrieves the routes.<br>13. If the Planner produced multiple routes, the app selects one automatically. It compares each suggestion with the remainder of the old route, and selects the closest match (so that the user can continue as originally planned, if possible, without needing to choose again). |

| | |
|---|---|
| | 14. The selected route is acknowledged and the others deleted, in transactions with the Planner. Navigation resumes.<br>Trip completion and feedback:<br>15. The last position update shows that the user has reached the destination. She is informed of this, and navigation ends.<br>16. The Tracker server is notified that tracking has stopped, and a route status update (trip is finished) is sent to the Planner.<br>17. The user is asked to provide emotional feedback for each public transport leg of the route just completed.<br>18. The user provides his feeling on the trip, and this is sent to the Planner server, where it is stored. |
| Alternate flows | • If a pre-planned trip is used to start navigation, and the current time and place does not match that of the plan, a route recalculation must be done initially. This starts with the abort trip transaction, followed by the rest of the transactions in the red box.<br>• When the Tracker reasoning engine processes incoming observations, there may be other actions, such as modifications of user preferences.<br>• On the significant deviation, the user could choose to end the trip rather than get an updated route.<br>• The user can cancel the trip at any time.<br>• If there are no public transport legs, there is no request for emotional feedback.<br>• The user can reject providing feedback. In this case, feedback can be provided later using the web. |
| Exceptional flows | • Reading of position may fail for a period of time, in which case the App doesn't know exactly where the user is in relation to the route. In public transportation modes, the navigation algorithm assumes the trip goes as planned, until the GPS shows otherwise, but for walk mode a GPS update is needed to advance the route state.<br>• Higher priority events (calls) may interrupt the guidance. System restores it automatically when mobile app is resumed.<br>• If a new route could not be produced after the significant deviation, navigation ends with an error message. |
| Displayed information | User is shown instructions during the trip, the route on the map when the map is displayed, and a feedback interface with smileys afterwards. |
| Postconditions | The trip has been completed, and the tracked results are stored in the Tracker. The Planner has stored the user's feedback on the completed trip, to use this information to improve the service, and has updated the status of the route accordingly. |
| Relation to other use cases | Follows as a consequence of either *Plan a trip* or *Request navigation*. Feedback can also be given on the web, as in *Provide tips and feedback*. |

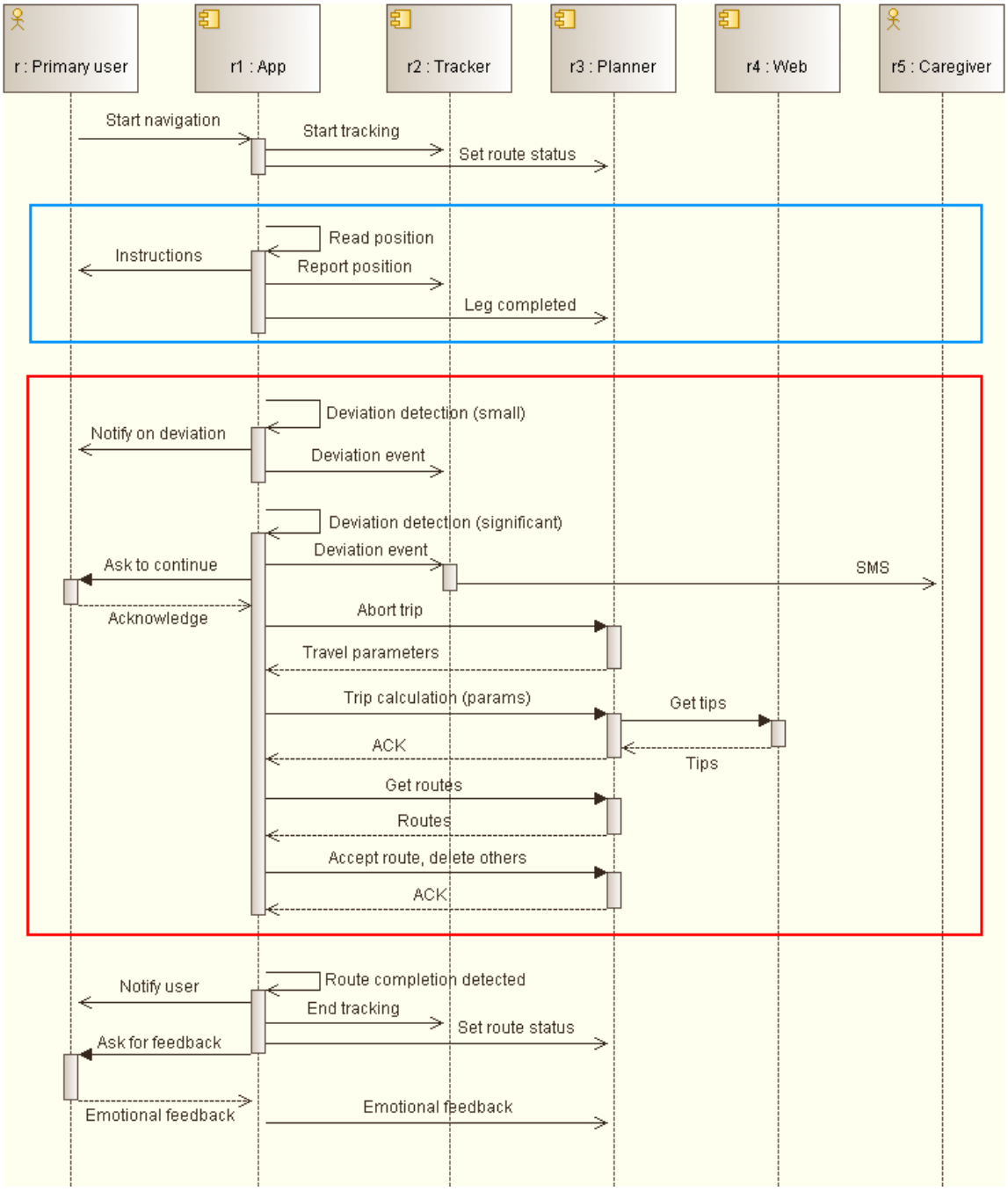


**Figure 7: Sequence diagram for use case 4 – Trip execution and feedback**

## 6.5 Provide tips and feedback

| | |
|---|---|
| Use Case Number | 5 |
| Use Case Name | Provide tips and feedback |
| Actors | Primary user, Web, Planner, Nominatim (third-party API) |
| Summary | The user uses the web to give feedback to a completed trip and |

| | |
|---|---|
| | enter accessibility information on a certain point on the map. |
| Trigger/ intent | User clicks on *Give Feedback* |
| Preconditions | User has completed a trip without giving emotional feedback, and has new accessibility information to add. |
| **Flow of events:**<br>(Main Flow) | 1. User enters the feedback web page.<br>2. Completed routes without feedback are fetched from the Planner server, and these are listed on the web.<br>3. The user selects a route and provides emotional feedback for each transit leg.<br>4. The feedback is sent to the Planner server, where it is stored, and the user is thanked for the input.<br>5. Next the user enters the *give tips* page, where a map is shown and the user can select a position by either clicking on the map or writing an address.<br>6. The user selects a point on the map. Tips are placed with reference to a place in OpenStreetMap data, and we use a service called Nominatim to get OpenStreetMap data based on map coordinates.<br>7. The user selects the type of tip by selecting an icon from a list of accessibility items, and may optionally write a comment.<br>8. The tip data is stored in the Web database. |
| Alternate flows | |
| Exceptional flows | |
| Displayed information | Feedback interface with smileys, map to enter tip. |
| Postconditions | The system has stored the feedback on the trip and the tip, and can use this information to improve the service. |
| Relation to other use cases | Providing feedback is done after Trip execution in use case 4, if feedback is not provided at the end of the trip. Note that providing tips can also be done in the app, using the same general procedure as outlined here. |

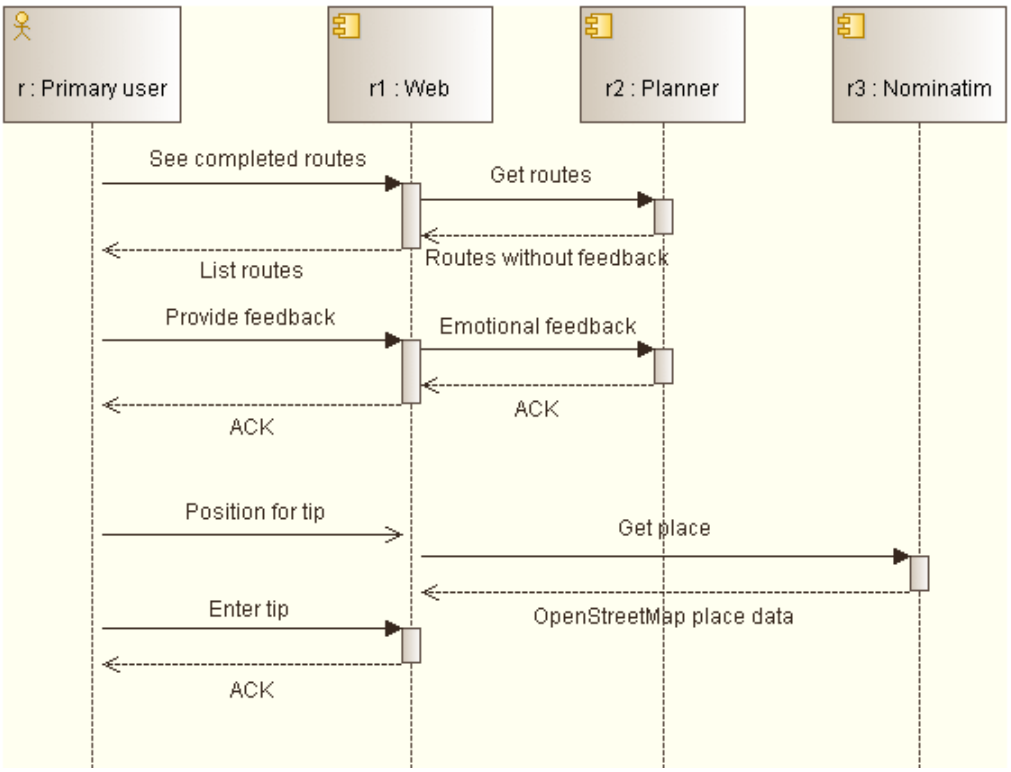


**Figure 8: Sequence diagram for use case 5 – Provide tips and feedback**

# 7 System architecture

This chapter describes the logical view of the system, giving an overview of the structure with components and interactions. We have already presented a division of the system into four sub-systems, here we give a more detailed architecture overview, with the server sub-systems divided into functional components and data stores which are described. The sub-system interactions, first specified in the use case chapter, are also shown in terms of these components, and the requirements for each sub-system API are listed at the end of the chapter. But first we describe the architecture decisions behind the system structure.

## 7.1 Architecture decisions

This main sub-system division followed from how responsibility for implementing the needed functionality was divided amongst the partners in the project. The key to a more detailed structure lies in the data – decisions regarding where to store and process the key types of data. There are three key forms of data, all of which are used in at least three of the four sub-systems: users (with preferences), routes and tips. These data and the resulting decisions are described here, as a key to understanding the architecture.

### 7.1.1 User accounts and profile data

The system needs to store various information about a user, and this information is used throughout the system. All of the server applications need a way to uniquely identify and authenticate a user, for secure access to the information. Both the Web/Social server and the Tracker server will hold parts of the user information, and make this available to the rest of the system through APIs. The server application of the T&Tnet web and social API holds the primary part of the user accounts, with login details. Routes and tips, discussed in the next sections, must also be tied to such a user account.

In SmartTracker, the user is a tracked entity, known as an *asset* in that system. An id uniquely identifying the asset must be stored with the user account in the Web sub-system, to link the user data in the two server applications. SmartTracker also stores the connection to a sensor device, positions and other sensor observations, and configurable logic for the reasoning engine, all tied to the asset. The other sub-systems will use SmartTracker's API

to access this data, and for this they need an authentication token issued by SmartTracker. The details are discussed in the Tracker API chapter. We want to avoid duplication of data, which creates synchronisation problems, so data will always be accessed from the sub-system responsible for it.

Each user has a set of preferences, such as for travel modes and accessibility. We have chosen to store them with the asset in SmartTracker, as the reasoning engine will read and potentially update the preferences continuously. This information will also be viewed and edited in the web interface, as well as in the mobile app, and it is input for route calculation, so preferences are related to all parts of the T&Tnet system. All sub-systems except the Planner must be able to access assets, using the asset id and authentication data stored with the primary account data in the Web sub-system.

### 7.1.2 Route calculation, representation and processing

Trip routes are an important core factor throughout most of the T&Tnet system, and have many aspects. A trip can be planned in advance, so the resulting route must be stored and transmitted to the mobile app where navigation takes place. During trip execution, the route data has two important functions. It is used to provide navigation for the user, such as drawing the route on a map and giving turn-by-turn instructions, and it is used to keep track of progress relative to the route, so that deviations can be detected and handled. The route might need updating during the trip, to account for user deviations or changing conditions. Afterwards, feedback is given for the route, and this must be processed and stored.

The Planner sub-system is responsible for producing the routes, with Open Trip Planner at its core. OTP produces routes in an *itinerary* data structure, where the itinerary representing the route as a whole is divided into legs of different transport modes which can be further divided into steps for detailed instructions. T&Tnet adds a layer on top of OTP, adding the concepts of users, accessibility preferences and emotional ratings from feedback, and also taking accessibility tips into account. T&Tnet also adds a layer to the data structure, with a route element which contains an augmented OTP itinerary and gives it a unique id.

For the navigation and tracking aspects, there is a question of client versus server for implementing the logic. The SmartTracker Reasoning Engine runs on the server side, and this will implement configurable logic and take actions on route deviation. However, tracking of route progress must be done in the mobile app to be able to provide navigation, and it should also be able to take action on route deviation as a connection to the server is not always available. For this reason, and to utilize limited resources where they are most needed, routes will not be represented or processed in the SmartTracker server. The route following logic is placed in the mobile app, and it is responsible for detecting deviations and sending such deviation events to SmartTracker to inform the reasoning engine.

A consequence of this decision is that server-side storage and processing of routes can be confined to the Planner sub-system, with route operations confined to its API. Planned and executed routes should be available both on the web and on the mobile app, at least as long as feedback has not been provided, so they need server-side storage. Whether requested from the web interface or from the mobile app, they need to be stored on the server. When providing a set of route suggestions, they are temporarily stored (until accepted or a new request is made), and if one route is accepted it is stored with a status of planned. The mobile app will poll for new planned routes regularly to be in sync with the server. On detection of route deviation, the mobile app must inform the server that the route should be discarded, and get back the parameters used to plan the original route, so that it can request a new route. This way the mobile app can request a new route using the same parameters used for the original route, even if that was planned from the web.

Route feedback can be given through both the app and web interfaces. It is the Planner sub-system which is responsible for processing and storing this feedback, so that it can use the result to enrich future route suggestions. When producing route suggestions, ratings resulting from feedback are included in the T&Tnet specific route data, to be visualised when routes are displayed to the user and help the user choose a route that matches his preferences.

### 7.1.3 Tips

Tips are accessibility-related location-specific information entered by users. It can be entered through the web, and the Web sub-system has been selected as responsible for storing the tips. Tips can also be added through the mobile app, and the app needs access to tips to display in the map. The Planner sub-system needs access to tips representing temporary blockage of routes. So all sub-systems need access to tips, and this must be handled by the Social API of the Web sub-system.

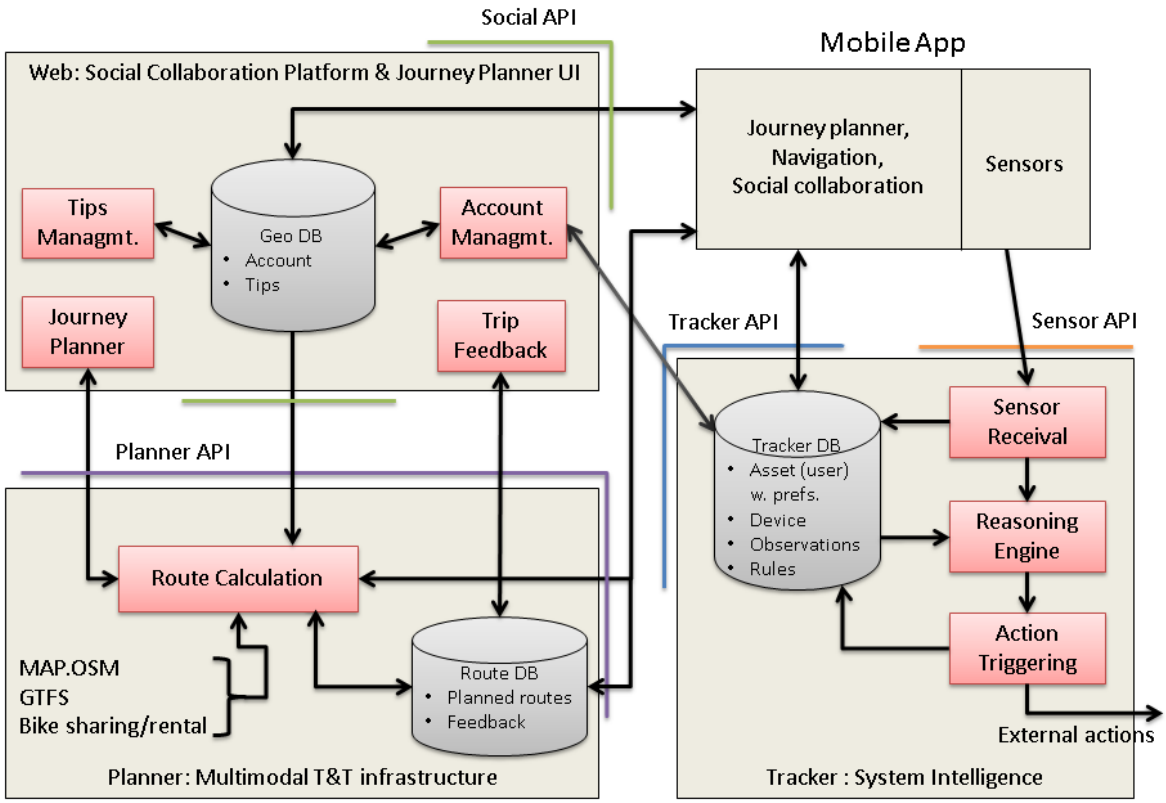


**Figure 9: Logical view of T&Tnet system**

## 7.2 System components

Figure 9 shows the system structure, resulting from the architecture decisions. The red boxes indicate the main functions of the server applications, and the grey cylinders indicate the data stores. The front ends are shown at the top of the diagram. The mobile app

is divided into two components for this view, as the sensor side is semi-independent of the rest. The web front end is an aspect of the server application (the function boxes are the functions available through the web user interface).

The figure also shows the main flows of data between the components (arrowheads indicate direction of data flows). APIs through which the sub-systems interact are indicated by the coloured lines - these are described at the end of this chapter. However, as the Web and Planner sub-systems are part of a common server application, data flow between them won't necessarily use HTTP-based APIs.

### 7.2.1 Geo DB

Two forms of data are housed in the Web sub-system: T&Tnet accounts, and tips. We refer to it as the Geo database because the tips are location bound (geo-information).

A T&Tnet user account has authentication information (username and password) and a user type in this database. It also has a reference to a SmartTracker asset, which represents the user in the SmartTracker service, with preferences and position.

The tips database has to be considered as a geo-database (information which is located by coordinates). It is for information on points of interest entered by users: the points themselves, and comments. This data is available and visible to all users of the system. See chapter 5 for the specification of tips, and section 8.2 for specification of the database tables of the Geo DB.

### 7.2.2 Tips Management

Web interface functionality related to tips. Tips can be shown with icons in a map layer, and any comments associated with a tip can be displayed. Users can add tips. This functionality reads and writes tips from the Geo DB.

### 7.2.3 Account Management

This component represents both creation of accounts and editing of preferences through the web interface. User account data is stored both in the Geo DB of the Web and the Tracker DB of SmartTracker, so this functionality must interact with both data stores.

### 7.2.4 Journey Planner

This component represents the journey planner part of the web application, showing the map and letting the user plan trips. It invokes the Route Calculation module with the parameters provided by the user, such as start and end time and place. Route Calculation returns routes, which are displayed on the map. Once the user confirms a route, it must be stored in the Route DB.

### 7.2.5 Trip Feedback

The user can provide emotional feedback for previously completed trips through the web interface. The feedback is stored in the Route DB.

### 7.2.6 Route DB

This is the storage of route-related data. Two main forms of route-related data need to be stored. Firstly, a user is given a set of suggested routes that are temporarily stored. When a user accepts a route in the Journey Planner or Mobile App, it must be stored so that it will later be available for navigation and feedback. The other suggested routes are removed. Once feedback has been given or the plan has been cancelled, the route may be deleted. A route is owned by a user, and only available to that user.

The other form of data is the emotional feedback provided by users. The feedback aggregated from all the completed trips will be collectively available to T&Tnet users. Route calculation will then get input from this database to sort the different legs as green, yellow or red.

### 7.2.7 Route Calculation

The Route Calculation functionality builds on Open Trip Planner. OTP provides the basic route calculation algorithm, using map data, public transport data (GTFS) and real-time data if available. To this, a number of T&Tnet aspects of routes are added: the preferences of the user, shared and synchronised routes for two users, accessibility tips, and previous route feedback. The routes produced therefore contain much additional information compared to OTP planner output.

A route can be requested through the Journey Planner web interface, or through the mobile app interface, in both cases through the Planner API. The result may be several alternative routes for the user to choose between. An addition compared to OTP is that the server needs to associate an accepted route with a user and store it. The stored routes will then be available from the database through the Planner API.

### 7.2.8 Tracker DB

This is the database of the SmartTracker server, storing all the user-related information needed for tracking and reasoning. Firstly, there is the setup of the user, such as the profile with preferences and the sensor device (phone) used to track the user. The history of received sensor data is stored, at least for a time, so that it can be analysed. Data making up system intelligence is also stored here: the rules for the reasoning engine and rule state data. The relevant data from this database is available externally through an API, so that the Web server and mobile App can retrieve and insert data.

### 7.2.9 Mobile App Sensors

The sensor part of the mobile app has been separated from the other functionality in the architecture overview because this communicates with the server side through a separate API and as far as the SmartTracker server is concerned it could as well be realized by a stand-alone sensor device. As an app component, it manages the sensors of the phone and tries to keep track of the user's position during trips. It persists data locally, and transmits it to the server whenever possible, implementing the client side of the Sensor API. However, the sensor data is also vital for the Navigation functionality of the app.

### 7.2.10 Sensor Receival

Following the path of the sensor data, the SmartTracker server has components for receiving sensor data from devices. SmartTracker has support for a number of device-specific protocols, but in this case we are primarily using SmartTracker's native sensor API as we are using a custom mobile app as device. Sensor data are persisted in the database, and fed to the Reasoning Engine.

### 7.2.11 Reasoning Engine

The Reasoning Engine in SmartTracker is based on the Drools rule engine. Both simple service logic and more complex artificial intelligence can be created with rules. The rules process the account and state data associated with the user along with the sensor data.

### 7.2.12 Action Triggering

When a rule triggers, the resulting action can either be internal, updating the state associated with the user, or external, such as sending out an alert. Updating the persistent state can represent learning (the reasoning engine has inferred something new about the user).

### 7.2.13 Mobile App

Most functionality of the system will be available in the mobile app, which needs to use APIs to communicate with the various server sub-systems. The user data of the Web sub-system is polled and updated through the Social API. Likewise the user data stored in the SmartTracker server is polled and updated through its Tracker API. The app will also request routes through the Planner API.

## 7.3 Interfaces

In the Use Cases (chapter 6) and the architecture in the current chapter we have introduced the interactions between the different parts of the system. The interfaces between the sub-systems are very important, as these sub-systems are developed by different partners. Figure 10 identifies the interactions that cross sub-system boundaries. The requirements on the resulting sub-system interfaces are listed here. Numbers in brackets refer to the interaction numbers in the figure (showing which components interact). Based on these requirements, detailed API specifications are given in the following chapters.

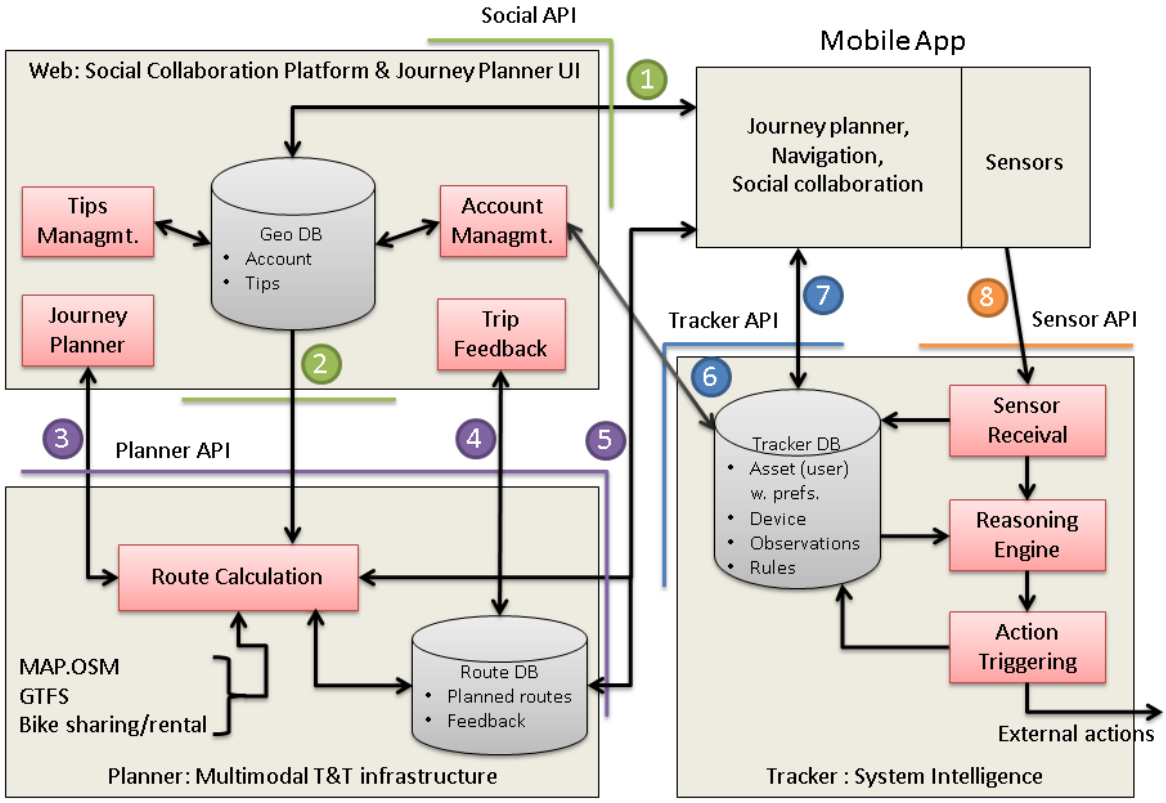


**Figure 10: Sub-system interactions**

### 7.3.1 Social API

Access to the Geo DB as needed by other sub-systems (this section is about machine-to-machine interfaces, so this is not a web user interface).

- LOGIN/GET account **[1]**: App provides authentication data, receives account data including SmartTracker user identifier.
- GET tips **[1, 2]**: Request tips for a map section (rectangular area).
- ADD tip **[1]**: App posts new tips/comments.

### 7.3.2 Planner API

For requesting new routes and interacting with stored routes, as well as posting route feedback.

- CREATE new route **[3, 5]**: Both web and app can make a request for route calculation, specifying user id, start and end points, start or end time and travel preferences.
- GET routes **[3, 5]**: Both web and app needs to retrieve route suggestions after route calculation, and to retrieve previously planned routes to display them and allow the user to cancel a planned trip. The mobile app also needs to retrieve plans to notify the user and start navigation. The web needs to retrieve finished routes without feedback, for the user to give feedback on the web.
- MODIFY route status **[3, 5]**: Both web and app needs to set a route as accepted after route calculation. The app also informs the server when a route is started and completed.
- DELETE planned route **[3, 5]**: To cancel a plan from web or app. This should also be used to delete unused suggestions after route planning.
- MODIFY leg status **[5]**: The mobile app informs the server when a leg is completed in the trip, so that the server knows which legs were completed and need feedback if the route is later aborted.
- ABORT route **[5]**: If a new route is needed during a trip, the mobile app calls the server to abort the route, and gets back the parameters used to plan it, to reuse them when creating a new route.
- ADD route feedback **[4, 5]**: Emotional feedback for route legs can be added from web and app.

### 7.3.3 Tracker API

Access to the SmartTracker domain objects, primarily the asset representing the user with position and preferences.

- GET asset **[6, 7]**: The user asset holding preferences is needed in the web to show/edit preferences, and by the app.
- MODIFY asset **[6, 7]**: Preferences may be edited on the web and app.
- CREATE account entities **[6]**: Asset and device entities are created from the web when creating a new account.

### 7.3.4 Sensor API

SmartTracker has an API for receiving observations from sensor devices. The Sensor API will not be detailed in this document, as Tellu is responsible for both sides of the interaction.

ADD observation **[8]**: The mobile app plays the role of sensor device, and will send such observations as positions, alarm button presses and route deviation.

# 8 Social API

## 8.1 Introduction

**Interface usage:** App, Planner

**Interface implementation responsibility:** GEO

The Social API is responsible for supporting interconnection and communication methods between the sub-systems (Planner and Mobile App) of the T&Tnet platform and the Web application. Furthermore, Social API is used in order to provide access to the GeoDB data which are required by the Planner and the Mobile App.

## 8.2 Geodatabase

The **Geodatabase** is highly related to the web application. In this DB, there are three tables which are connected to the Social API. These tables are presented in details below:

**Users'** table stores information about the user's account. In Table 5, there is a detailed description of User's table characteristics.

**Table 5: Users**

| Column | Description |
|---|---|
| **userID** | The user's id |
| **username** | The user's username (which is the first part of the registered email) for the registration/log in action |
| **password** | The user's secret code for the registration/log in action |
| **assetID** | This is the assetID which communicates with the SmartTracker |
| **email** | The user's email for the registration/log in action |
| **typeUser** | The type of user. Administrator (-1), VIALE user (typeUser =0), Official User (typeUser=2) |

**TemporalTip** table stores information about the tips which can be used by the **planner API**. In Table 6, there is a detailed description of the characteristics of the temporaltip table.

**Table 6: temporaltip table structure**

| Column | Description |
|---|---|
| **id** | The unique id for each record |
| **userID** | The user's id who add the tip |
| **idTipOSM** | The id of the tip location based on the OSM. This value is retrieved from the OSM after the call to the **Nominatim API.** |
| **typeTipOSM** | The type of the tip location based on the OSM. This value is retrieved from the OSM after the call to the **Nominatim API**. |
| **valueTypeTipOSM** | The type's value of the tip location based on the OSM. This value is retrieved from the OSM after the call to the **Nominatim API**. |
| **icon_image** | The icon value of the tip. |
| **datein** | The date of the tip insertion |
| **duration** | The expected duration time of the tip |
| **comment** | The user's comment for a specific point (location) (default value null) |
| **lon, lat** | The coordinates of the tip's location |
| **road** | The road name of the tip's location (default value null) |
| **number** | The road number of the tip's location (default value null) |
| **city** | The city of the tip's location (default value null) |
| **country** | The country of the tip's location (default value null) |
| **geocoding** | This field stores information about the type of the tip's insertion in the platform |

**PoITip** table stores the rest of the tips (accessibility info about the selected locations), based on the different types. The distinction between the types of tips is made by the different icons. In Table 7, there is a detailed description of **PoITip**'s table characteristics.

**Table 7: PoITip Table**

| Column | Description |
|---|---|
| **id** | The unique id for each record |
| **userID** | The user's id who add the tip |
| **icon_image** | The icon value of the tip. |
| **idTipOSM** | The id of the tip location based on the OSM. This value is retrieved from the OSM after the call to the **Nominatim API**. |
| **typeTipOSM** | The type of the tip location based on the OSM. This value is retrieved from the OSM after the call to the **Nominatim API**. |
| **valueTypeTipOSM** | The type's value of the tip location based on the OSM. This value is retrieved from the OSM after the call to the **Nominatim API**. |
| **datein** | The date of the tip insertion |
| **comment** | The user's comment for a specific point (location) (default value null) |
| **lon, lat** | The coordinates of the tip's location |
| **road** | The road name of the tip's location (default value null) |
| **number** | The road number of the tip's location (default value null) |
| **city** | The city of the tip's location (default value null) |
| **country** | The country of the tip's location (default value null) |
| **geocoding** | This field stores information about the type of the tip's insertion in the platform |

**Example: Nominatim API call:**

http://nominatim.openstreetmap.org/search?format=xml&q=41.648832227075005,-0.8849569378635777&addressdetails=1

**Response:**

```
<searchresults timestamp="Thu, 25 Sep 14 11:51:43
+0000" attribution="Data © OpenStreetMap contributors, ODbL 1.0.
http://www.openstreetmap.org/copyright"querystring="41.648832227075005,-
0.8849569378635777" polygon="false" exclude_place_ids="19547116" more_url
="http://nominatim.openstreetmap.org/search?format=xml&exclude_place_ids=
19547116&accept-language=el-
GR,el;q=0.8,en;q=0.6&addressdetails=1&q=41.648832227075005%2C-
0.8849569378635777">
<place place_id="19547116" osm_type="node" osm_id="1792256947" place_ran
k="30" boundingbox="41.6488761901855,41.6488800048828,-
0.884980022907257,-0.884979963302612" lat="41.6488776"lon="-
0.88498" display_name="Novodabo, 12, Plaza Aragón, Zaragoza, El Gancho,
Σαραγόσα, Aragon, 50004,
Ισπανία" class="amenity" type="restaurant" importance="0.001"icon="http:/
/nominatim.openstreetmap.org/images/mapicons/food_restaurant.p.20.png">
<restaurant>Novodabo</restaurant>
<house_number>12</house_number>
<pedestrian>Plaza Aragón</pedestrian>
<residential>Zaragoza</residential>
<suburb>El Gancho</suburb>
<city>Σαραγόσα</city>
<county>Σαραγόσα</county>
<state>Aragon</state>
<postcode>50004</postcode>
<country>Ισπανία</country>
<country_code>es</country_code>
</place>
</searchresults>
```

**idTipOSM → place_id (19547116)**

**typeTipOSM → restaurant**

**valueTypeTipOSM → Novodabo**

**TokenAPI** table stores tokens which are generated by the platform when the mobile app makes a request to the Social API. Every token corresponds to a particular user and has a specific life time to the platform.

**Table 8: TokenAPI Table**

| Column | Description |
|---|---|
| **idUser** | The user's id who adds the tip |
| **token** | The unique token |
| **timeGen** | The date and time of the token's generation |
| **used** | This field stores information of the usage of the generated token |

## 8.3 Supported Methods

There are three main methods which are supported: (i) user's authentication method (ii) retrieve tips service (iii) give tip service. More specifically, the supported services are given below:

### 8.3.1 registerUser

This method is used in order to send account data to the mobile application. Particularly, mobile app provides as input: (i) username (ii) password. The method is responsible to check if this authentication data are correct and the user has a valid account to the T&Tnet platform. If the authentication data are correct, this method generates a token which will be used by the mobile application for every other request to the Social API. As a result, the output of this method is the account data with a unique token. In the **Error! Reference source not found.**, there is a detailed description of the method's parameters.

**PathParams**: {username}, {password}

**Path**: rest/socialApi/registerGet/{username}/{password}

**Example**:

http://ttnetgeo.isoin.es:8080/TTNetSocialApi/rest/socialAPI/registerGet/elenaTest/3333

**Table 9: registerGet Method**

| Property | Description |
|---|---|
| username | The user's username. |
| password | The user's password. |

### 8.3.2 retrieveTips

This method is responsible for retrieving the tips from the **Geodatabase** in order to send them back to the mobile app. The retrieved tips correspond to a specific rectangular area. Thus, the input parameters of this method are two points (lon, lat) in order to specify the map area. The one point refers to the north-west corner of the rectangular area and the second one to the south-east corner. Additional inputs will be provided to the method regarding the tip's icon. In the table below, there is a detailed description of the method's parameters.

**PathParams**: {tokenUser},{cornerNW},{CornerSE},{ typeUserSelected },
{ tipsCategSelected }

- {typeUserSelected }: type of the user who provided the tips
- {tipsCategSelected }: String which stores the selected tip categories. The tip categories are represented by the icon numbers, which are separated by ','. ex. 1,2,10

**Path**:
rest/socialAPI/retrieveTips/{tokenUser}/{cornerNW}/{CornerSE}/{typeUserSelected}/{tipsCategSelected}

**Example**:
http://localhost:8080/RestApiTTNet/rest/socialAPI/retrieveTips/RTSIOHJ3Z7Z9ZR7/-96471.591400292,-5426.3563219/5107253.5053769,6709689.40331/0/1,2,3,4,5,6,7,8,9,10

**Table 10: retrieveTips Method**

| Property | Description |
|---|---|
| tokenUser | The authentication unique token |
| cornerNW | The (lat,lon) of the north-west corner of the rectangular area |
| cornerSE | The (lat,lon) of the south-west corner of the rectangular area |
| typeUserSelected | This property corresponds to the "type of user" who provides the tips and takes value "0" for VIALE users, "2" for Official users and "-1" for "my tips" option (tips which are provided by the user who requests the tips). |
| tipsCategSelected | This property corresponds to the "selected tip categories" and takes value from [1-15]. In particular, this property is string which stores the selected tip categories, separated by ','. Example: 2,10,15 |

### 8.3.3 giveTip

This method is responsible for storing the tip in the **Geodatabase,** giving through the mobile app The input parameters of this method are (i) user's authentication token (ii) location - point (longitude, latitude) (iii) tip category (iv) the icon of the tip (v) comment (optional) (vi) id Tip OSM, which is the id of the tip which is retrieved after the call to the nominatim (https://nominatim.openstreetmap.org/) API for the specific location (vii) type Tip OSM, which is the type of the tip which is retrieved after the call to the nominatim API for the specific location (viii) value Type Tip OSM, which is type value of the tip which is retrieved after the call to the nominatim API for the specific location.

**PathParams**:{tokenUser},{lon},{lat},{comment},{iconImageP},{idTipOSM},{typeTipOSM},{valueTypeTipOSM}

- { idTipOSM}: the id of the tip location based on the OSM
- { typeTipOSM}: the type of the tip location based on the OSM.

- {valueTypeTipOSM}: the type's value of the tip location based on the OSM

**Path**:
rest/socialAPI/giveTip/{tokenUser},{lon},{lat},{comment},{iconImageP},{idTipOSM},{typeTipOSM},{valueTypeTipOSM}

**Example**:
http://localhost:8080/RestApiTTNet/rest/socialAPI/giveTip/FK61DO5NLIGBREA/-0.88391820000004/41.654028/slipperstairs/9/90669869/tram/Calle del Coso

**Table 11: giveTip method**

| Property | Description |
|---|---|
| tokenUser | The authentication unique token |
| lon, lat | The location of the tip. |
| comment | The comment (string) provided by the user for the specific location – tip. |
| iconImageP | This property specifies the category of the tip and takes value from [1-15]. |
| idTipOSM | This property corresponds to the "id" of the tip based on the OSM. This id is retrieved from the corresponding call the **Nominatim API**. |
| typeTipOSM | This property corresponds to the "type" of the tip based on the OSM. This type is retrieved from the corresponding call the **Nominatim API**. |
| valueTypeTipOSM | This property corresponds to the "type's value" of the tip based on the OSM. This type's value is retrieved from the corresponding call the **Nominatim API**. |

# 9 Planner API

## 9.1 Introduction

**Interface usage:** App, Web

**Interface implementation responsibility:** ITA

Responsible for calculating routes based on input from available public transport agencies, mobility data sources, route tables, emotional feedback, user profile and accessibility-related data. Next, an update of the previously defined methods is provided. For further information over the latest methods, refers to the D2.2 Final Travel and transport infrastructure prototype.

## 9.2 Route database

A route is defined as the itinerary given by OTP (JSON or XML representation) with coloured-legs according to historical emotional feedback. A route can be labelled as:

1. to-be-decided: the route has just been calculated and user must decide what to do
2. accepted: the route is accepted by the user.
3. navigation: the routes is being used by the user.
4. finished: the user has finished the complete itinerary of route.
5. feedbacked: the user has provided feedback to the route. The route is considered as feedbacked if at least one route leg is given feedback.

*Routes table*

| Column | Description |
|---|---|
| idroute | A unique id associated to the route |
| iduser | The user that made the request for a trip plan. |
| datetime | The date and time when user made the request. |
| label | The label associated to the trip (1-5) |

| | |
|---|---|
| startinglocation | The starting location (latitude, longitude) |
| endinglocation | The end location (latitude, longitude) |
| startinglocationname | The starting location description |
| endinglocationname | The end location description |
| departure time | The time when route starts |
| itinerary | The response returned by OTP with coloured-legs |

*Legs table*

| Column | Description |
|---|---|
| idleg | A unique id that identifies a sequence of transport stops for each public transport line |
| transport mode | The transport mode of the leg |
| line | The associated public transport line |
| agency | The transport agency providing the service |
| startlocation | starting stop |
| endlocation | ending stop |

*Feedback table*

| Column | Description |
|---|---|
| idleg | A unique id that identifies a sequence of transport stops for each public transport line. |
| idroute | The id route which contains the leg in which feedback is provided |
| datetime | The date and time when feedback is provided |
| seats availability | Green, yellow or red. |

| | |
|---|---|
| speed | Green, yellow or red. |
| hour | The hour when the trip started |

## 9.3 Supported methods

### 9.3.1 getroutes

The method returns a set of routes of the user of a specific label. In case the departure time of the route is greater than current time, the planned route is not returned.

Parameters

| Property | Description |
|---|---|
| iduser | The id of user that requests the planned routes. |
| type | The label of the routes to be returned. Example: “to-be-decided”, “accepted”, ”completed” .. |

Results: The method returns the planned routes labeled as *type*

### 9.3.2 changeroutestatus

The method receives the idroute of the route and a new status for the route.

Parameters

| Property | Description |
|---|---|
| iditinerary | The id of route |
| iduser | The id of user that requests the planned routes. |
| type | The new status for the route. Keep in mind that the status changes by increasing the value in one unit (from 1 to 5). |

Results: The method returns ACK.

### 9.3.3 deleteroute

The method deletes the route from the system.

Parameters

| Property | Description |
|---|---|
| iduser | The id of user that requests the planned routes. |
| iditinerary | The id of route |

Results: The method returns ACK.

### 9.3.4 tripcalculation

This method will make a request for a trip plan. The call includes the two points (latitude/longitude or the address), the user id, the user preferences (max walking distance, max number of transfers, accessibility issues) and updated accessibility-related information from Geo-DB. The method will also evaluate trips based on historical feedback provided by users. The user must indicate in his/her preferences the form they want the routes to be displayed.

The emotional feedback can be classified at 4 time periods:

- From 07:00 to 10:00
- From 10:00 to 14:00
- From 14:00 to 19:00
- From 19:00 to 0:00

This classification is a good approximation to the natural way that transport public agencies schedule their routes.

Example:

http://193.144.226.70:8080/ita-ttnet-planner/planner-api/tripcalculation?&iduser=itaplanner&token=0DGDZOP63YHHPGB&routerId=zaragoza&bannedPlaces=&mode=,TRANSIT,WALK&maxWalkDistance=5000&bannedStreets=,&toPlace=41.67241471709407,-0.8589274404785212&fromPlace=41.63470585142353,-

0.9481913564941579,&toPlace2=&fromPlace2=&stairs=true&elevators=true&waitAtBeginningFactor=0.2&date=2014-10-16

Parameters

| Property | Description |
|---|---|
| Token | The token |
| routerId | The name of the city where the route will be calculated (zaragoza, oslo, paris, vienna) |
| fromPlace | The start location (latitude/longitude or address) |
| fromPlace2 | The start location used for synchrotrips. |
| toPlace | The end location (latitude/longitude or address) |
| toPlace2 | The end location used for shared trips |
| iduser | An object containing the id of the user making the request. |
| date | The date that the trip should depart or arrive. When empty, the current date will be considered. |
| time | The time that the trip should depart or arrive. When empty, the current mete will be considered. |
| arriveBy | Whether the trip should depart or arrive at the specified date and time |
| walkSpeed | The user's walking speed in meters/second. |
| maxWalkDistance | The maximum distance (in meters) the user is willing to walk |
| stairs | true or false . If the value is false the streets with stairs will be avoided in the calculation process |
| Elevator | True or false |
| bannedFrom | The start location (latitude/longitude) of a street that will be avoided in the calculation process |
| bannedTo | The end location (latitude/longitude) of a street that will be avoided in the calculation process |
| bannedStreets | A list of streets' names that will be avoided in the calculation process |

| | |
|---|---|
| bannedPlaces | A list of locations (latitude/longitude) that will be avoided in the calculation process |
| mode | The set of modes that a user is willing to use.<br><br>Possible values are:<br><br>• 'TRANSIT,WALK' for public transport,<br>• 'BUSISH,WALK' for Bus only,<br>• 'TRAINISH,WALK' for Train only,<br>• 'WALK' for Walk only,<br>• 'BICYCLE' for Bicycle only,<br>• 'WALK,BICYCLE' for 'Rented or Public Bicycles,<br>• 'TRANSIT,BICYCLE' for 'Transit & Bicycle,<br>• 'TRANSIT,WALK,BICYCLE' for Transit & Rented Bicycle |

Results: The method will provide a set of trips that meet the user's preferences. The trips will be stored in the *Routes table* labeled as *to-be-decided*.

### 9.3.5 insertfeedback

Insert emotional feedback provided by the user into the *Feedback table*. Given the *from* and *to* stops, the method verifies if the specific leg exists in the database. If not, a new leg will inserted in the *Legs table*. This table will be filled with the data provided in the parameters.

Parameters

| Property | Description |
|---|---|
| iditinerary | The route id |
| idleg | The leg id |
| iduser | The user id |
| hour | The hour when the trip started |
| speed | Green, yellow or red. |
| seatsavailability | Green, yellow or red. |

Results: the table *Feedback table* is filled with the feedback. Also, a new leg is created if not available.

### 9.3.6 getdestinations

This method retrieves the user destinations order by the most recent. The method is intended to ease the selection of a destination by the user. The attribute *count* contains the occurrences of each destination.

| Field | Description |
|---|---|
| Iduser | The user'id |

http://193.144.226.70:8080/ita-ttnet-planner/planner-api/getdestinations?iduser=93

### 9.3.7 getstreets

This method retrieves all the streets of a specific city that contains a string of characters.

| Field | Description |
|---|---|
| routerID | city's name |
| street | A string containing the set of characters to search for |

http://193.144.226.70:8080/ita-ttnet-planner/planner-api/getstreets?routerId=paris
http://193.144.226.70:9091/ita-ttnet-planner/planner-api/getstreets?street=Jos&routerId=oslo

### 9.3.8 getStreetNumbers

The method returns the house numbers of a specific street

http://193.144.226.70:9091/ita-ttnet-planner/planner-api/getstreetnumbers?street=Calle%20de%20Bolonia

| Field | Description |
|---|---|
| routeID | city's name |
| street | The name of the street |

### 9.3.9 synchrotrips

The method synchronizes the trips of two users that want to reach the same destination at the same time. The algorithm calculates the trips for each user separately. Afterwards, the method checks if the last part of the trips (a set of legs) concurs and, in case of positive answer, delay or put forward the departure time of one user.

http://193.144.226.70:9090/ita-ttnet-planner/planner-api/synchrotrips?&iduser=itaplanner&token=0DGDZOP63YHHPGB&routerId=zaragoza&bannedPlaces=&mode=,TRANSIT,WALK&maxWalkDistance=50000&bannedStreets=,&toPlace=41.672526912910456,-0.8900410650512488&fromPlace=41.63270114851638,-0.9048039434692308,&toPlace2=&fromPlace2=41.63443321549116,-0.884633731677243&stairs=true&elevators=true&waitAtBeginningFactor=0.2&date=2014-10-16

The request for syncho is quite similar to calculate trips with a new field:

| Field | Description |
|---|---|
| fromPlace2 | The origin place of the second user |

### 9.3.10 sharedtrips

The method calculates the trips for two users that depart from the same origin at the same time. The algorithm calculates the trips for each user separately. Afterwards, the method checks if the initial part of the trips (a set of legs) concurs and, in case of positive answer, provide all the possible combinations.

http://193.144.226.70:9090/ita-ttnet-planner/planner-api/sharedtrips?&iduser=itaplanner&token=0DGDZOP63YHHPGB&routerId=zaragoza&bannedPlaces=&mode=,TRANSIT,WALK&maxWalkDistance=5000&bannedStreets=,&toPlace=41.646428

1055546,-0.9076578138612019&fromPlace=41.68196668902734,-0.8869940756105218,&toPlace2=41.63347096180583,-0.8825523374818319&fromPlace2=&stairs=true&elevators=true&waitAtBeginningFactor=0.2&date=2014-10-17

| Field | Description |
|---|---|
| toPlace2 | The destination place of the second user |

The XML response has the same format that synchro. It may happen that more than one solution can be found. It depends for the users to select the most suitable for them. The most disadvantaged user may see that his/her router is too long and may wish to select other solution.

### 9.3.11 confirmLegCompleted

Every time the app detects that the user has completed a leg, make a call to this method to update the leg on the planner side. The leg status is set to *complete*.

| Field | Description |
|---|---|
| Idleg | The leg's id |
| Iduser | The user'id |
| idtrip | The trip'id |

### 9.3.12 abortTrip

It is similar to make a call to the method *changeroutestatus* with a type = *aborted.*

| Field | Description |
|---|---|
| Iduser | The user'id |

| | |
|---|---|
| idtrip | The trip’id |

# 10 Tracker API

## 10.1 Introduction

**Interface usage:** Web, App

**Interface implementation responsibility:** TELLU

The SmartTracker server stores user profiles and other account data needed for tracking and system intelligence. Access to this data is provided by the SmartTracker Data API (short name Tracker API). Both viewing of data as well as creation and modification should is implemented by the web and mobile applications. The SmartTracker Data API is a standard HTTP REST API, supporting GET, POST, PUT and DELETE operations. Data is exchanged in JSON format.

Note that the documentation for the API is available online at the following URL:

https://telludoc.atlassian.net/wiki/display/SMARTTRACKER/Smarttracker+API+v3

Refer to this for the latest version, and for examples of the JSON data objects.

## 10.2 Account and authentication

All data access requires a SmartTracker account with the correct permissions. There are two important terms regarding a SmartTracker account. The account itself is referred to as a *customer* in SmartTracker terms. The data belongs to a specific customer, and will only be available to *users* tied to that customer. A *user* is what defines authentication data (username and password) and permissions. A customer account may have many users, with different permissions controlling what they can do in the account. In addition, a customer is owned by a service *provider*, and this is also specified in the API replies, but not important for our use in T&Tnet.

For the T&Tnet system, there is a T&Tnet customer in SmartTracker. We have chosen a single-customer approach because SmartTracker is a back-end in this project rather than being accessed directly and because this simplifies setup and access. It is also the best way to handle sharing of positions between friends, which although not done in the

final prototype, was part of the original specification for the Tracker server. Each user will have an *asset* object in this account, which represents a tracked entity with preferences and position, and a *device* object representing the mobile app as a source of sensor observations to be associated with the asset.

Each API request must include an authentication token, supplied as an HTTP header, which is tied to a user and is used in place of the username and password. A token may be time-limited or not. For the T&Tnet prototypes, we have created one user for the Web server and another user for the mobile app. Tokens with no time limitations have been issued for these users, and given to the Web server and included in the mobile app respectively.

For a commercial system, switching to an individual SmartTracker user for each T&Tnet user can be considered, to increase security. A SmartTracker user will then need to be created as part of each account creation process, and the token for the individual T&Tnet user, possibly time-limited, can be stored with the Web account.

## 10.3 URLs and requests

This section describes the format of API requests and replies. The relevant resources (data objects) are described in the next section.

### 10.3.1 URL

The URL consists of four parts: base (server address), customer, resource and id.

<base url>/<customer id>/<resource>/<resource id>

Retrieving only the root of the URL (without resource) will give an object describing what resources are available.

| Property | Description |
|---|---|
| providers | A list of service providers the user making the request can access. |
| customers | A list of customers the user making the request can access. |
| access | A list of available resources with a map per resource indicating |

| | |
|---|---|
| | what methods that is available and whether the client is allowed to perform them. |
| features | A list of suggestions to the client to enable or disable features in order to provide a simpler interface to the user. |
| user | An object containing the id of the user making the request. |
| provider | An object containing the id of the service provider of the customer in the request. |
| customer | An object containing the id of the customer in the request (or if the customer id was not included in the URL, the customer associated with the user). |
| time | The time the request was handled. |

### 10.3.2 Retrieving data

All data requests must be done with the HTTP method GET. All requests done on resources will have the same properties in the response.

| Property | Description |
|---|---|
| result | A list of resources matching the data request. This will always be a list, even if the client requests a resource with a specific id. |
| total | The total number of resources matching the data request. |
| offset | Marker used to paginate the data. |
| max | The maximum number of resources in each response. |
| user | An object containing the id of the user making the request. |
| provider | An object containing the id of the service provider of the customer in the request. |
| customer | The customer used as source of data in the request. |
| time | The time the request was handled. |

### 10.3.3 Filtering data requests

SmartTracker has a powerful filtering mechanism. Filters are added as parameters in the URL. Multiple filters can be added, but a mechanism can only be used once per

property (latitude:less=59 and latitude:greater=58 is possible, but name:contains=e and name:contains=m is not). All filters follow the same pattern.

<property name>:<filtering mechanism>=<filter value>

| Filtering mechanisms | Description |
|---|---|
| equals | Usable on most data types. |
| contains | Usable on string data types and some more complex types. |
| less | Usable on number and date data types. |
| greater | Usable on number and date data types. |

| Example | Limit request to resources with ... |
|---|---|
| name:equals=Demo | name equal to Demo. |
| name:contains=em | the text "em" somewhere in the name. |
| latitude:less=59 | latitude less than 59. |
| longitude:greater=11 | longitude greater than 11. |
| timestamp:greater=2013-04-18T00:00:00.0 | timestamp after April 18. 2013. |
| timestamp:less=2013-04-20T00:00:00.0 | timestamp before April 20. 2013. |

### 10.3.4 Data content

When requesting data, not all data is included due to performance and bandwidth reasons. When querying a list of data, only id and name is included by default. When querying a single resource, all immediate properties are included (without any recursion). Complex objects will (usually) include an id and name. This behaviour can be overridden by adding a parameter to the URL named select. Select accepts a list of property names separated by the character +. It also has two special values, star ( * ) and at ( @ ). Star includes all properties and all subproperties. At includes all properties but only the minimum of subproperties (id and name).

| Example | |
|---|---|
| select=* | All properties of the resource, and all subproperties |
| select=@ | All properties of the resource, but minimum of data for subproperties |
| select=lastValidPosition+type | Only lastValidPosition and type properties. Type (a complex type) will only have id and name. |
| select=lastValidPosition+type.icon | Only lastValidPosition and type properties. Type will now also have icon as well as id and name. |
| select=type.* | Only type. All properties of type will be included. |
| select=positionProvider.@ | Only positionProvider. The immediate properties of positionProvider is included. |

### 10.3.5 Submitting data

Adding a resource must be done with HTTP method POST, without a resource id in the object or in the URL. Resource objects are wrapped in a list to allow creating more than one object in the same request.

```
POST <base>/<customer id>/<resource>
```

Updating an object must be done with HTTP method PUT, with a resource id in the URL. In both cases the resource must a JSON object inside a JSON list in the request payload. See each resource section for more information about which properties that are required and valid values. The resource object is wrapped in a list to be consistent with creating an object. If a property is omitted then it will not be changed on the server.

```
PUT <base>/<customer id>/<resource>/<resource id>
```

### 10.3.6 Deleting data

Deleting data must be done with HTTP method DELETE with a resource id in the URL. The response if successful is an empty GET response (with HTTP code 200).

```
DELETE <base>/<customer id>/<resource>/<resource id>
```

## 10.4 Resources

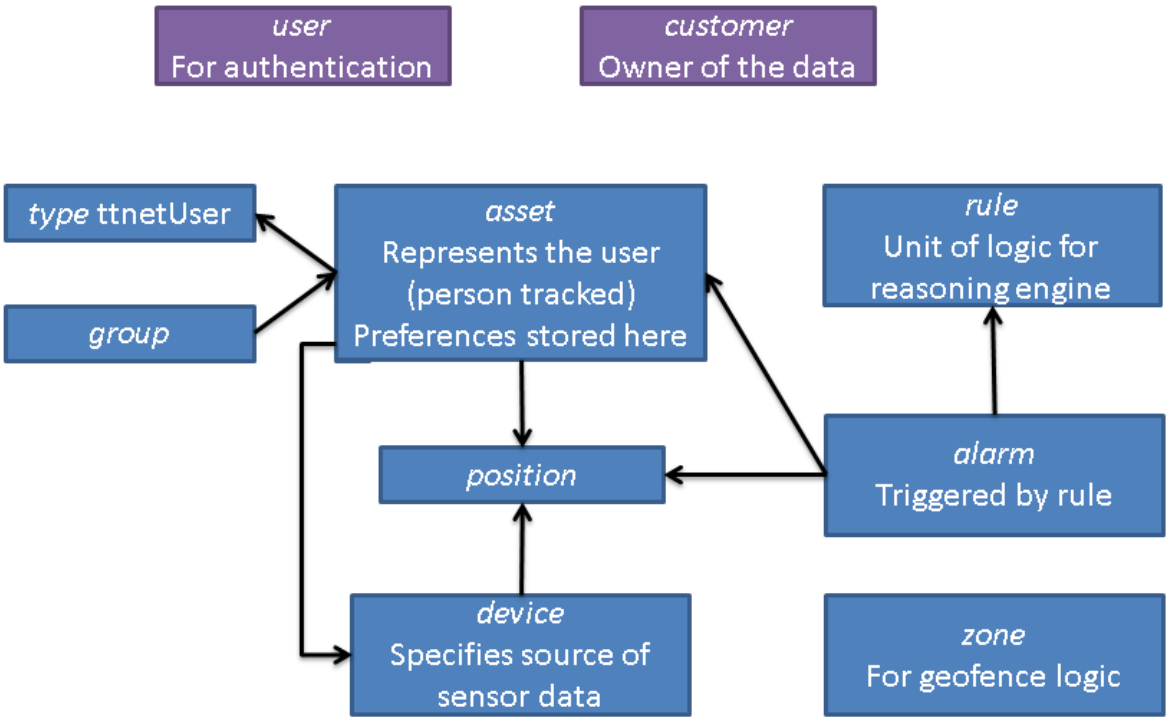


**Figure 11: SmartTracker domain objects**

This section describes the SmartTracker resource types relevant to T&Tnet. The resources are listed in alphabetical order, with relevant properties listed for each type. The figure above gives an overview of the resources, with the main relationships. The central resource type is *asset*. An asset represents an entity which is tracked, and is the primary object for the reasoning engine. For T&Tnet, there will be one asset representing each user.

### 10.4.1 Alarm

An alarm is a notification that requires the attention of a user, usually generated by the reasoning engine based on some rule.

| Property | Type | Optional | More info | Filtering |
|---|---|---|---|---|
| name | string | - | | - |
| owner | customer | - | | - |

| | | | | |
|---|---|---|---|---|
| dateCreated | date | - | | equals, greater, less |
| lastUpdated | date | - | | |
| comment | string | - | | equals, contains |
| ackNeeded | boolean | - | | equals |
| logLevel | integer | - | Degree of severity of alarm. 0 is most severe, -20 least. -20 should be without immediate notification. | equals, greater, less |
| asset | asset | - | Asset associated with the alarm. | equals (id of asset) |
| ackedBy | user | - | | |
| rule | rule | - | | |
| trigger | position | - | The observation triggering the rule/alarm, if available. | |
| position | position | - | The position of the related asset when the alarm was created. | |
| zone | zone | - | Zone relevant to the triggering of this alarm. | - |

### 10.4.2 Asset

The asset represents the person tracked by the system (the primary user). Most important to the T&Tnet system is the properties field, where user preferences will be stored.

| Property | Type | Optional | More info | Filtering |
|---|---|---|---|---|
| name | string | NO | | equals, contains |
| description | string | YES | | equals, contains |
| owner | customer | - | | - |
| lastValidPosition | position | - | The most recent observation with a valid position received by the position provider of the | - |

| | | | | |
|---|---|---|---|---|
| | | | asset. | |
| lastPosition | position | - | The most recent observation received by the position provider of the asset. If the position is valid this will be the same as lastValidPosition. | - |
| groups | list of group | YES | When creating or updating only the id of the group will be used. | - |
| icon | string | - | This icon is the icon set by the asset's type. | - |
| image | string | - | | - |
| type | type | YES | When creating or updating only the id of the type will be used. | equals (id of type) |
| tracked | boolean | YES | If enabled all observations received by the position provider will be stored. Cannot be set to true if the trackMode is "never", cannot be set to false if the trackMode is "always". | - |
| trackMode | string | YES | "always" will always store observations received. Rules cannot change whether or not the asset is tracked.<br>"never" will never store observations received permanently. Rules cannot change whether or not the asset is tracked.<br>"manual" depend on the tracked property to determine if observations are stored. Rules can change whether or not the asset is tracked. | - |
| properties | list of objects | YES | Each property object has a number of fields. “name” and “value” are the primary ones for reading, with the possible property names being based on the configured properties in the asset's type. For writing, we need to specify “typePropertyIdentifier”, which is a unique internal id for the property. | contains |

| | | | | |
|---|---|---|---|---|
| alarms | list of alarm | - | The five most recent, unacknowledged alarms. Useful for creating lightweight clients. | - |
| positionProvider | tag or device | YES | When creating or updating only the id of the object will be used. The server will first attempt to find a device matching the id and if not found a tag. | equals (id of positionProvider) |

### 10.4.3 Device

A device specifies a source of sensor data, and is assigned to an asset to provide sensor data for that asset. In T&Tnet this is the app running on the user's phone, although adding additional sensor devices will also be possible.

| Property | Type | Optional | More info | Filtering |
|---|---|---|---|---|
| name | string | | | |
| description | string | | | |
| owner | customer | | | |
| lastValidPosition | position | | | |
| lastPosition | position | | | |
| sensorDeviceType | string | | | |
| active | boolean | | | |
| uuid | string | | | |
| primaryProperties | object | | | |
| commandProperties | object | | | |
| additionalProperties | object | | | |

### 10.4.4 Group

Assets can belong to groups, which may be useful for group logic (trigger rules for all assets in a group).

| Property | Type | Optional | More info | Filtering |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| name | string | NO | Any non-empty string. Cannot be the same as any existing group inside the customer. | equals, contains |
| description | string | YES | | equals, contains |
| owner | customer | - | | |
| assets | list of assets | YES | | |

### 10.4.5 Position

The position of an asset (usually reported by a sensor device). We will mainly encounter it as part of the asset data.

| Property | Type | Optional | More info | Filtering |
|---|---|---|---|---|
| asset | asset | - | | equals |
| valid | boolean | - | | equals |
| latitude | double | - | | equals, greater, less |
| longitude | double | - | | equals, greater, less |
| accuracy | integer | - | Estimated accuracy in meters | equals, greater, less |
| speed | integer | - | Speed as reported by device in meter per second. | equals, greater, less |
| address | string | | | - |
| timestamp | date | | | equals, greater, less |
| properties | object | | | - |
| events | list of string | | | - |

### 10.4.6 Rule

A rule is a configurable unit of logic for the reasoning engine. The active set of rules defines the service behavior. The data available through the API is mainly for viewing and changing rule states (turn on and off).

| Property | Type | Optional | More info | Filtering |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| name | string | - | | equals, contains |
| description | string | - | | equals, contains |
| owner | customer | - | | - |
| status | string | YES | "active", "inactive", "unknown", "stopped", "failed", "requiresConfiguration". When changing the status, only "active" or "inactive" are valid inputs. | - |

### 10.4.7 Type

Assets can be typed, with the type specifying what properties an asset has. T&Tnet assets have their own type, specifying the user preferences.

| Property | Type | Optional | More info | Filtering |
|---|---|---|---|---|
| name | string | NO | Any non-empty string. Cannot be the same as any existing type inside the customer. | equals, contains |
| description | string | YES | Any string. | |
| owner | customer | - | | |
| properties | object | YES | Object where each key is a different property name. Property value is an object with at least two entries, type and dataType. If the type is a list it must also include a list of string called items. See example. | |

### 10.4.8 Zone

Zones are used to define location-specific logic such as geofence (trigger a rule on entering or leaving an area).

| Property | Type | Optional | More info | Filtering |
|---|---|---|---|---|
| name | string | NO | Any non-empty string. Cannot be the same as any existing zone inside the customer. | equals, contains |
| description | string | YES | Any string. | equals, contains |
| owner | customer | - | | |

| | | | | |
|---|---|---|---|---|
| position | latlon | - | An object with two entries, latitude and longitude. | |
| singleLevel | boolean | YES | When checking if an asset is inside, do they need to be on the same floor. | |
| floor | integer | YES | Any number. | |
| textual | string | YES | Any string. | |
| address | string | YES | Any string. | |
| points | list of latlon | NO | List of objects with two entries, latitude and longitude. Must have at least 3 objects. | |

# 11 Conclusion

This deliverable has presented the software architecture of the T&Tnet system, which is the result of the initial analysis and iterative users' evaluations, the respective technology and expertise of the different partners and the ongoing work in the project. It constitutes a very important document for the implementation of the two prototype iterations, as our system is comprised of sub-systems built by different partners in different parts of Europe. The API specifications are vital to the integration of the sub-systems, and have been successfully implemented and used in the working system.