Preliminary Prototypes of Integrated Solutions

Deliverable D4.2

Date: 30/09/2016

Author(s): João Barreto, Rodrigo Lourenço, Miguel Costa, Luís Veiga, Paulo Ferreira

INESC-ID
<table>
<thead>
<tr>
<th>Deliverable No.</th>
<th>D4.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workpackage No.</td>
<td>4</td>
</tr>
<tr>
<td>Workpackage Title</td>
<td>Overcome the ICT challenges</td>
</tr>
<tr>
<td>Task No.</td>
<td>4.5</td>
</tr>
<tr>
<td>Task Title</td>
<td>Task 4.5: Integrated solutions</td>
</tr>
<tr>
<td>Date of preparation of this version:</td>
<td>9/09/2016</td>
</tr>
<tr>
<td>Authors:</td>
<td>João Barreto, Rodrigo Lourenço, Miguel Costa, Paulo Ferreira</td>
</tr>
<tr>
<td>Status (F: final; D: draft; RD: revised draft):</td>
<td>D</td>
</tr>
<tr>
<td>File Name:</td>
<td>D4.2-Integrated prototypes</td>
</tr>
<tr>
<td>Version:</td>
<td>3</td>
</tr>
<tr>
<td>Task start date and duration</td>
<td>01/01/2016 – 30/09/2016</td>
</tr>
</tbody>
</table>

**Revision History**

<table>
<thead>
<tr>
<th>Version No.</th>
<th>Date</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9/9/2016</td>
<td>First draft</td>
</tr>
<tr>
<td>2</td>
<td>21/9/2016</td>
<td>First revision after leading reviewer’s and QAM’s reviews</td>
</tr>
<tr>
<td>3</td>
<td>28/9/2016</td>
<td>General revision</td>
</tr>
</tbody>
</table>

**Reviewers List**

<table>
<thead>
<tr>
<th>Name</th>
<th>Company</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan Christiaens</td>
<td>M21</td>
<td>16/9/2016</td>
</tr>
<tr>
<td>Giuseppe Liguori</td>
<td>SRM</td>
<td>22/9/2016</td>
</tr>
</tbody>
</table>
# TABLE OF CONTENTS

1. ABOUT TRACE ........................................................................................................4

2. ABOUT THIS DOCUMENT .......................................................................................5

3. APPROACH ...............................................................................................................6

4. OVERVIEW OF THE INTEGRATED PROTOTYPE ....................................................7

5. MAIN SOFTWARE MODULES IN THE INTEGRATED PROTOTYPE .............................8

5.1. Graph-based trajectory storage and analytics .....................................................8

5.1.1 Large-scale data storage and mining.................................................................8

5.1.2 Open and interoperable tracking ......................................................................10

5.2. Energy-efficient and modality-aware tracking ...................................................10

5.2.1 Speed-aware adaptive tracking ........................................................................11

5.2.2 Trajectory-driven modality recognition ..........................................................12

6. INTEGRATION TESTING ..........................................................................................13

7. FOR MORE INFORMATION: ....................................................................................16

8. ANNEX 1: APPLICATION PROGRAMMING INTERFACES ....................................17
1. About TRACE

The TRACE project will explore the potential of walking and cycling tracking services to promote walking and cycling mobility. We will focus on established walking and cycling promotion measures and thoroughly assess the potential of ICT based tracking services to overcome barriers to implementation and finding new factors driving the effectiveness of those measures.

Through specific research, the related ICT challenges like scheme dynamics, privacy, trust, low-cost, interoperability and flexibility will be tackled for each type of measure. It will be established measures to promote walking and cycling travel to workplace, shopping, school and leisure.

We will investigate both the ability that tracking tools may have to address traditional challenges of these measures and their potential to bring new features in the fields of awareness raising, financial/tax incentives, infrastructure planning and service concepts.

A common, flexible and open access tool will be developed to provide an ICT input and output platform that addresses the related ICT challenges. Over this platform it will be easy for anyone to build products based on tracking services tailored to the requirements of the specific measures. This project will develop and test a representative set of such products in real measures underway. These test cases will at the same time validate and provide additional inputs for the project’s research issues and trigger the widespread of tracking services to support walking and cycling measures in Europe.

Users, policy makers, walking and cycling practitioners and final users will be deeply involved in all stages of the project.
2. About this document

In WP4, tasks 4.2-4.4 addressed distinct ICT challenges in the context of a common architecture for tracking applications, defined in task 4.1. Each such task designed, developed, tested and evaluated different solutions. This process was strongly guided by assessment results described in deliverables D2.1-D2.4 and D3.1-D3.2, as well as by technical discussions that involved both the developer and pilot partners (namely, at the Lisbon kick-off meeting, June 2015, and Breda consortium meeting, April 2016).

In practice, the chosen solutions that emerged from tasks 4.2-4.4 may need to be combined in order to compose realistic systems that tackle multiple requirements arising from different research areas. That is the main motivation of task 4.5, whose focus was to study combinations of ICT solutions stemming from tasks 4.2-4.4 that are most representative of the main application groups that WP5 will target.

Integrating software modules that were designed and evaluated individually is not always trivial, as their composition into a larger software system may reveal incompatibilities or result pathological interferences that might strongly affect the effectiveness of each individual solution that is being combined.

This document describes a large prototype that was developed, tested and evaluated within task 4.5. This large prototype integrates the main solutions that resulted from tasks 4.2-4.4. The document not only describes the development process and the technical details of the integrated solution, but also documents the experimental testing and evaluation of such integrated prototype.

It should be noted that this generic architecture and the integrated prototype that instantiates such an architecture is not meant to be used as a real tool with real users. Their main purpose is to validate and evaluate a number of novel technical solutions for each individual module. A subset of such solutions can then be transferred, integrated and adapted into each tool that TRACE will develop.

The integrated prototype is available as a ready-to-run demonstrator. The end of this document provides a link to the code repository containing the integrated prototype.
3. Approach

This deliverable reflects the combined outcome of different work packages and deliverables. **A first stage** was based on the main assessment results, described in related deliverables from WP2 (D2.1-D2.4) and WP3 (D3.1-D3.2).

Departing from such conclusions, the consortium discussed and defined the most relevant research objectives that WP4 should target. This stage was also guided by contributions and feedback from both the ICT developer partners and the pilot partners (namely, at the Lisbon kick-off meeting, June 2015, and Breda consortium meeting, April 2016).

As expected, the decisions taken by the consortium implied relative detours from the research challenges and objectives that were originally defined in the tasks of WP4. More precisely, the assessment revealed that some ICT challenges which were only implicitly mentioned in the project proposal had, in reality, a much greater importance to the tools that TRACE is developing. The remainder of this chapter specifies such ICT challenges that were elected by the consortium as having highest-priority.

**As a second stage**, a number of research efforts addressed each ICT challenge. This stage received contributions from the partners with ICT development responsibilities. The standard approach in R&D projects was followed: a strong literature review studied the design space and existing techniques; a thorough design phase devised sketches of innovative solutions that tried to overcome the main limitations of current state-of-the-art; simple prototypes were developed, tested and experimentally evaluated in realistic settings and in a scientifically reproducible way in order to validate each solution. This was mostly an iterative process, where each iteration of the above procedure departed from previous solutions and extended them until satisfactory results were obtained. This second stage happened in parallel for each distinct task 4.2-4.4 in WP4. The solutions devised at each such task were mostly evaluated individually.

Finally, at a **final stage**, the best solutions resulting from tasks 4.2-4.4 were selected and integrated into a simple integrated prototype. This composed software module was then evaluated as an entire system. Again, the experimental methodology ensured realistic settings and relied on scientifically reproducible experiments to obtain its results.
4. Overview of the Integrated Prototype

The generic architecture of TRACE was proposed and defined in D4.1. It is depicted at a very high level in Figure 1. This generic architecture is intended to be a canonical reference on top of which the research efforts would be designed and evaluated. Conceptually, each tool resulting from the project can be instantiated as a subset of the components/functionality that this generic comprises.

The TRACEtracking module, present on TRACE users’ mobile devices, gathers tracking information and sends it to the TRACE’s storage system, namely the TRACEstore module. Essentially, TRACEtracking is a middleware that leverages the embedded sensors present on mobile devices to try to extrapolate users’ location, transportation modality, among other information.

TRACEstore is the storage module of TRACE. Through it, TRACE stores the users’ spatio-temporal trajectories, and their personal details along with the road network of cities.

Finally, there is the urban planner module. This module is able to provide high-level key performance indicators to urban planners. This information is anonymous and untraceable back to its original users.

Developing each module above entails overcoming a number of ICT challenges. The next section describes each such module, describing how it addresses the relevant ICT challenges.
5. Main software modules in the integrated prototype

The following subsections describe the main open software modules that were developed in the context of WP4 and then combined in the integrated prototype.

It is worth noting that it is out of the scope of this document to provide an in-depth specification and experimental evaluation of each module that is presented in the next subsections.

The documentation and source code of each module below is publicly available at the official website of TRACE (http://h2020-trace.eu/).

5.1. Graph-based trajectory storage and analytics

This module addresses two main challenges:

- Large-scale data storage and mining (Task 4.4)
  For larger-scale tracking initiatives, TRACEstore stores and supports complex analytics over a large set of spatio-temporal trajectories obtained from tracking applications and devices. Studying and devising efficient ways of storing and processing such large data sets was the main objective of Task 4.4.

- Open and interoperable tracking (Task 4.3)
  The growing popularity of tracking-based initiatives makes it increasingly frequent that distinct campaigns run in the same city, sometimes simultaneously. Rather than analysing data from each campaign individually, it is highly desirable if urban planners have the possibility of combining trajectory data sets from different sources. This can be achieved by using open and interoperable protocols and formats for the exchange of tracking information from trackers and TRACEstore. This is one of the main goals of Task 4.3.

Another relevant issue to be addressed was the need for TRACEstore to rely on open source map data, rather than on proprietary maps that may not always be available, affordable and/or accurate.

We next describe how each above challenge was addressed.

5.1.1 LARGE-SCALE DATA STORAGE AND MINING

Spatial-temporal networks are not easily modelled in the relational model that is found in mainstream databases. This is due to the fact that trajectories are usually modelled using graphs, composed of vertices (representing the intersections between roads), edges (represent the roads) and weights (representing the distance in kilometres or the travel time of that segment).

A representation extensively used on relational repositories to module spatial networks is as follows:

\[
\begin{align*}
\text{node} & : (\text{node_id}, \text{property}_1, \ldots, \text{property}_N) \\
\text{edge} & : (\text{edge_id}, \text{property}_1, \ldots, \text{property}_N, \text{orig}: \text{node}, \text{dest}: \text{node})
\end{align*}
\]
Node and edge are the name of two relationships, node_id, property1 … property, edge_id, orig and dest are attributes of a relationship, and :node is the type of information being stored in the attributes orig and dest. This solution has the disadvantage of requiring a join operation for each edge that is traversed. Thus, as the number of traversed edges increase, so does the computational and spatial complexity of the join operations being performed.

To overcome this crucial limitation, graph databases were proposed. Graph databases are optimized for graph-traversals, such as spatial network queries and pattern-matching on graphs. They consist of a set of vertices connected between themselves with edges. Each vertex can have one or more properties, and the queries are answered through an algorithm called graph-traversal.

As part of Task 4.4, we explored the use of graph databases for storing and processing spatio-temporal trajectories that were collected by TRACEtracker. The developed database stores the city’s road network and all its semantic data. Geographic locations, Pos, are stored as nodes. These nodes can then have multiple properties associated with them. Such properties allow nodes to specify their type, like interceptions, or points of interest, as well as store semantic data like IDs, names, addresses, etc.

Edges connect two nodes. Edges represent the road which physically connects the two points represented by the nodes. Just like nodes, edges can also have properties. Edges properties allow us to associate a specific road with characteristics such as road gradient, pavement condition, length, car traffic and other such aspects.

However, the Tracking Database does more than just storing a mapping of the city’s roads. In order to fully achieve the requirements of TRACE, this database also stores TRACE users’ trajectories. Besides storing the road network, the database also associates users with locations and roads. Essentially what happens is that users are now paired with sessions that describe a particular spatio-temporal trajectory. The list of trajectories consists of a chronologically ordered set of nodes and edges, respectively locations and roads, taken by the user in one of his travels.

Each time a user starts travelling, this action initiates a new session. Sessions are also represented as nodes. Furthermore, each of these session nodes have a sessionID, date and time as properties. This allows sessions to be unique and perfectly pinpointed in terms of date and time. Some semantic data can also be associated with these sessions like for example, trip purpose, bringing some extra context into these sessions.

On top of this data representation, we developed the following main services:

- Map matching, relying on a parallelized version the Hidden Markov Model map matching algorithm\(^1\), to match the geographic coordinates of a submitted trajectories to logical vertices in our representation of each city’s map.
- Stay-point detection, using Li et al.’s algorithm\(^2\).


\(^2\) Quannan Li, Yu Zheng, Xing Xie, Yukun Chen, Wenyu Liu, and Wei-Ying Ma. 2008. Mining user similarity based on location history. In Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems (GIS '08). ACM, New York, NY, USA, , Article 34 , 10 pages. DOI=http://dx.doi.org/10.1145/1463434.1463477
• Key performance indicator calculation for urban planning, focusing on the indicators defined with highest priority in deliverable D3.1: volume of users; number of trips originated per zone (origin); number of trips ended per zone (destination); volume of users per origin-destination; speed average; distance average; and trip time average.

• Additional general-purpose queries such as:
  - User-based aggregation queries (e.g., calculating the total travelled distance of a user, as well as the travelled distance, and route taken, of a specific user session).
  - Geographical range queries (e.g. selecting nearby points of interest).

5.1.2 OPEN AND INTEROPERABLE TRACKING

For interoperability with different types of tracking applications, TRACEstore can receive spatio-temporal trajectories through a Web Service offering a RESTful application programming interface (API). The same applies to the multiple analytics services offered by TRACEstore (map matching, stay-point detection, key performance indicators and general-purpose queries).

The specification of the RESTful API is documented in Annex 1.

This choice enables the services of TRACEstore to be invokable by heterogeneous clients, neither placing any restriction on the programming language, runtime environment, operating system, nor hardware that the client application might be running.

In fact, the services of TRACEstore can be invoked by standard HTTP methods (e.g. PUT or POST) through an Uniform Resource Locator (URL) that identifies the service.

Another issue that was considered was the use of TRACEstore at distinct cities in Europe (and out of European Union). Instead of relying on proprietary map information, TRACEstore was extended with support for importing of maps from the open-source Open Street Maps (OSM) platform. This enables deploying TRACEstore at virtually any region in the world, as long as OSM has up-to-date maps of that region.

5.2. Energy-efficient and modality-aware tracking

Following the assessment obtained from WP2, the consortium agreed on two main requirements to tackle regarding tracking (in the context of Task 4.2 – Secure and reliable tracking):

Energy-efficiency tracking

This requirement reflects the conclusions from the user and stakeholder surveys in D2.1, which showed that battery consumption was among the main concerns for users and stakeholders when considering whether or not they would accept to adopt tracking-based applications.

---

Ability to accurately infer the transport modality associated with each trajectory

This stems from the need to segment tracked trajectories according to the different means of transport for an adequate analysis for urban planning. This was also a common requirement to each tracking application in TRACE, which intrinsically rely on knowledge about the current transport modality.

Unfortunately, the native support for tracking and modality recognition that is offered by the native services of mainstream mobile operating systems (e.g. Android or iOS) has important limitations.

In the following, we detail the solutions that were developed in line with each requirement above.

5.2.1 SPEED-AWARE ADAPTIVE TRACKING

Adequate energy-efficient solutions need to cope with a delicate trade-off with accuracy: while it is desirable that tracking minimized energy consumption, it should simultaneously fulfil the accuracy requirements that the applications demand. This trade-off is far from trivial to accomplish.

This research effort in WP4 produced a novel speed-aware positioning technique. The novel technique dynamically adapts the behaviour of a hybrid positioning service (based on a combination of GPS and other techniques) such that the measured trajectory ensures a minimal distance, $d$, between points while minimizing energy consumption.

Intuitively, the speed-aware adaptation technique relies on periodic measurements of the user’s instantaneous speed to infer (with high probability) when the next spatial coordinate should be obtained. Ideally, this informed guess should postpone the next measurement to the latest moment such that the distance between the next coordinate and the previously obtained one is at most $d$ meters. Therefore, this adaptive scheme is able to considerably reduce the frequency of positioning measurements (and, hence, the energy spent with those measurements) when the user is temporarily moving at low speeds.

Putting the above insight into practice is, however, a challenging task for a number of reasons. Firstly, since the user’s speed is not always constant, measuring his/her past speed values is not always an accurate prediction of his/her speed in the near future. Secondly, measuring speed with GPS is also an energy-consuming action; as such, its frequency should be minimized. Thirdly, operating systems like Android and iOS place considerable restrictions on the ability of an application to measure instantaneous speed while an application is running in background mode; this is an important obstacle, since the tools that TRACE will develop will run in this mode for most of their lifecycle (e.g. during each trip). Hence, a combination of different techniques had to be devised in order to cope with the above restrictions.

To overcome the above challenges, the speed-aware that WP4 designed relies on a sliding window of the last $k$ instantaneous speeds that were obtained measured in the last $k$ GPS location requests that the application performed. At each stage, the average speed of the last $k$ measurements determines the time interval until the next spatial coordinate is requested from the operating system’s positioning service.

The above method to infer the user’s speed in the near future is, of course, fallible (as the user’s speed is not always constant). Hence, the proposed technique follows a best-effort approach, as it does not always guarantee that the measured spatial trajectory ensures that each pair of points is within $d$ meters. However, a careful choice of parameters can ensure that the technique is well behaved with a high probability.

The most critical parameter is $k$. It can be empirically defined so as to be the as small as possible while ensures that the tracked trajectory reaches close the precision of at least $d$ meters between pairs of coordinates. Since different means of transport may have different optimal values for $k$, we allow distinct value of $k$ to be considered depending on the current transport modality.
When compared to other solutions to the adaptive tracking problem that have been proposed in literature, the solution devised in WP4 has the novelty of running even when the application is in background state, both on Android and iOS (today’s most adopted operating systems for smartphones).

5.2.2 TRAJECTORY-DRIVEN MODALITY RECOGNITION

Many tracking applications need to associate a trajectory with the corresponding transport modality. In particular, this is a crucial requirement for all the tools that TRACE is developing.

Current mainstream operating systems (such as Android and iOS) already offer powerful human activity recognition services. These services rely on sophisticated inference methods based on commodity sensors such as the accelerometer. Among other uses, such activity recognition services may be used to determine, with relative precision, the current transport modality of the user carrying the smartphone.

Still, the nature of these activity recognizers is prone to outliers – measurements that make an incorrect guess about the current activity – due to unavoidable external factors. Following our initial experiments with state-of-the-art activity recognizers, we observed that, during an entire trajectory, the activity recognizer was normally able to correctly infer the transport modality; however, a minority of outlier measurements was also found.

Therefore, from a sample of modality results obtained from the activity recognizer service during a given trajectory, we need to be able to identify the outliers and elect the correct transport modality that was actually used during the entire trajectory. However, the answer to this problem is not trivial, as some trajectories may be multi-modal – at some points, the user changes from one modality to another. Hence, simple methods such as selecting the mean or median modality guess within the whole sample will not always return the appropriate result.

Our contribution regarding this goal produced an innovative support for long-term modality detection, which filters intermittent fluctuations in detected modalities. It follows a state-machine approach for a smart outlier detection and elimination. Intuitively, it can be seen as an abstract machine that, at each moment, can be in one of a pre-defined set of states. Periodically, the local modality recognition service is invoked and returns its guess of the current transport modality. That answer dictates a transition from the current state of the machine to another one.

This state-machine abstraction allows application programmers to define, according to each application’s semantics, rich sets of rules to associate trajectories with specific transport modalities.

As a proof of concept, we used this state-machine approach to encode the semantics that will be used by the cycle-to-shop tool of TRACE to determine whether a given multi-modal trajectory (i.e., a trajectory which can include distinct transport modalities at its inner points) is eligible as a “cycle-to-shop” trajectory.5 We have empirically confirmed that this state-machine is able to adequately infer correct guesses, even when the activity recognition service of the smartphone occasionally provides incorrect guesses at some points within the trajectory.

5 However, it should be noted that this trajectory-driven modality recognizer can be used by any other tool in TRACE.
6. Integration testing

As planned for task 4.5, we assembled an integrated prototype that composes together all the modules described in the previous sections. The main goal of this was to test the interfaces between components against the original design and specifications of each individual module.

The integration process was useful in exposing minor incompatibilities and interferences, which were iteratively corrected. At its final iteration, the integrated prototype comprised the following concrete components:

- A simple tracker client application, designed to run on Android smartphones. This client is continuously tracking the spatio-temporal trajectories of the device, as well as the associated modes of transport, and periodically sending such traces to the back-end TRACEstore server. This tracking employs the speed-aware adaptive tracking and the trajectory-driven modality recognition techniques described in Section 5.
  The client application was tested using different smartphones, namely Nexus 5, Nexus 5X and Nexus 6.
- An instance of TRACEstore developed on top of the Titan Distributed Graph Database. This instance ran on a back-end server machine provided both the support for storage of spatio-temporal trajectories and the analytics services described in Section 5.
  The server ran on a Intel(R) Core(TM) i5 CPU 760 at 2.80GHz, with 4GB RAM, with Linux 3.19.0-64-generic x86_64 as its operating system.
- A simple interactive console application, running locally on the server machine, which invokes the different analytics services offered by TRACEstore. This console represented a simplified version of an urban planning module.

The methodology followed a progressive approach, in which we started by integrating a small set of components, which were tested together, and gradually added more components to the integrated prototype. Figures 2 to 4 depict screenshots of different aspects of the full-fledged prototype.

Testing was based on different sets of test batteries that exercised different features of the full solution, both from the tracker client’s side and from the urban planning console’s side. Testing included both controlled micro-benchmarking (based on simulated inputs to the tracker client) and realistic on-street experiments. The latter were performed by members of the INESC ID team on selected paths of Lisbon (Portugal).

This experimental procedure enabled us to confirm that all the modules are effectively compatible and composable with each other. When employed in the integrated prototype, every module retained its correct behavior.

Hence, we conclude that all the individual modules are compatible and integrated with no undesirable interferences with negative implications on their behaviour.

---

http://titan.thinkaurelius.com/
Figure 2 – Tracking client application, presenting a recently tracked trajectory and the associated mode of transport (as detected by the modality recognizer).

Figure 3 – Examples of spatio-temporal trajectories collected with the tracker prototype, relying on the speed-aware adaptive tracking technique, using different values of the d parameter (top-left: d=5m; top-right: d=20m; bottom-left: d=30m; bottom-right: d=60m). Increasing values of d imply improved battery savings.
Figure 4 – Console for invoking analytics services on TRACEstore.
7. For more information:

- Project coordinator at INESC ID: Paulo Ferreira: paulo.ferreira@inesc-id.pt
- Project communication manager at Polis: Giacomo Lozzi: glozzi@polisnetwork.eu
- Project website: www.h2020-trace.eu
- Twitter: @TRACE_project
- TRACE LinkedIn Group: TRACE Project
8. Annex 1: Application Programming Interfaces

1. TRACE Tracker API

The Tracker Module Library was designed to enable applications to leverage the Android devices' tracking capabilities. In particular, this library enables the applications to keep track of the user's location and current transportation modality. In this section we detail how this library may be used to its fullest.

1.1. Main Tracking Features

In this section, we briefly describe the tracker library main features:

- Location tracking
- Modality/Activity tracking using the Modality Recognizer Library
- Persistent storage of tracked trajectories
- Real-time adjustable and togglable removal of outlier locations
- Configurable tracking

1.2. Tracker library API

In this section, we detail the tracker library’s API.

static <Tracker> getInstance(Context context, Messenger messenger)
Fetches an instance of the tracker singleton.

```java
<void> teardown()
```

This method should be invoked when the Tracker is no longer required for the foreseeable future, e.g. before the application closes. The method will terminate any pending connections and open resources.

```java
<void> startTracking()
```

Initiates the location and activity tracking endeavors, therefore, initiating a new track, which is identifiable by an unique identifier.

```java
<String> stopTracking()
```

Stops the location and activity tracking modules. Additionally, it returns the local identifier of the traced route.

```java
<TraceLocation> getLastLocation()
```

Request the most current location.

```java
<void> updateTrackingProfile(ConfigurationProfile profile)
```

Updates the tracking profile settings. These define the sampling rates used, how outliers are identified, among other information, and are described in detail further ahead.

```java
<ConfigurationProfile> getCurrentTrackingProfile()
```

Fetches the currently enforced tracking profile.
List<TrackSummary> getAllTracedTracks()

Fetches the list of all stored tracks. The tracks are represented by a TrackSummary that contains only top-level information, such as the route identifier, elapsed time, traveled distance.

.Track> getTracedTrack(String sessionId)

Fetches the track identified by the specified route identifier. The Track contains not only top-level information, but also all the traced locations and corresponding transportation modalities.

<void> deleteTracedTrack(String sessionId)

Permanently removes the specified track, identified by the provided route identifier, from the mobile device. Already uploaded tracks will remain stored in the servers.

1.3. Configuration Parameters

The configuration of the tracking parameters can be divided into two groups: i) location and ii) activity tracking.

For location tracking these are the most relevant tracking parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interval</td>
<td>The sampling rate employed for location tracking.</td>
<td>3.5s</td>
</tr>
<tr>
<td>Fast Interval</td>
<td>The fastest acceptable interval for location updates.</td>
<td>1.5s</td>
</tr>
<tr>
<td>Priority</td>
<td>Allows to prioritize either accuracy or power saving. Four different modes are available, ranging from high accuracy to no location at all, as to minimize the energy consumption.</td>
<td>High Accuracy</td>
</tr>
<tr>
<td>Displacement Threshold</td>
<td>Minimum distance that should be travelled for a new location to be acceptable.</td>
<td>2m</td>
</tr>
<tr>
<td>Minimum Accuracy</td>
<td>Each location is associated with an accuracy, i.e. an error radius in meters. This parameter specifies the minimum acceptable accuracy.</td>
<td>40m</td>
</tr>
<tr>
<td>Maximum</td>
<td>Each location is associated with a GPS measured instantaneous speed.</td>
<td>200Km/h</td>
</tr>
</tbody>
</table>
**Speed** parameter specifies the maximum acceptable speed.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable Outlier Filtering</td>
<td>Specifies if the Tracker service should remove outlier location or not.</td>
<td>True</td>
</tr>
</tbody>
</table>

For activity tracking, these are the most relevant tracking parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interval</td>
<td>The sampling rate employed for activity tracking</td>
<td>3.5s</td>
</tr>
<tr>
<td>Minimum Confidence</td>
<td>Each tracked activity is associated with a confidence level, ranging from 0% to 100%. This parameter specifies the minimum acceptable confidence.</td>
<td>75%</td>
</tr>
</tbody>
</table>
2. Trace Store API

The Trace Store API is a REST API which allows for communication over HTTP.
The API can be accessed at: <host>:8080/trace/ where <host> is the public IP address of the machine the service is running on.
A simple example, in order to verify the availability of the service, can be found at: http://<host>:8080/trace/tracker/test

2.1. Main Trace Store Features

In this section, we briefly describe Trace Store main features:

- User registration
- User login and logout
- Request of new tracking sessions
- Submission of a new route
- Request of user sessions
- Request of user sessions and dates
- Request of a previously submitted route

2.2. User Registration

**URL:** http://<host>:8080/trace/tracker/register

**Description:** Allows the registration of a new Trace User.

**Params:**

- (JSON) name: Name of the user
- (JSON) username: Username for the account
- (JSON) email: Email address of said user
- (JSON) password: Password with lowercase, uppercase, numbers and special characters such as #
- (JSON) confirm: Confirmation of the password
- (JSON) phone: Phone number, with or without the country code
- (JSON) address: Street Address of the user.

**Output:**

- (JSON) success: true|false

**Details:**

- method: 'POST'
- datatype: 'json'
- url: 'http://<host>:8080/trace/tracker/register'
- contentType: 'application/json'
2.3. User Login

**URL:** http://<host>:8080/trace/auth/login  
**Description:** Enables the user to login, either with a TRACE account (option #1) or a Google account (option #2).

- **Params:**  
  - (JSON) username (option #1)  
  - (JSON) password (option #1)  
  - (JSON) idToken (option #2)

- **Output:**  
  - (JSON) success: true|false  
  - (JSON) token: JSON Web Token (JWT)

**Details:**
- **method:** 'POST'  
- **datatype:** 'json'  
- **url:** 'http://<host>:8080/trace/auth/login'  
- **contentType:** 'application/x-www-form-urlencoded'

**User Logout**

**URL:** http://<host>:8080/trace/auth/logout  
**Description:** Enables the user to logout revoking his JWT Token.

- **Params:**  
  - (JSON) token: JSON Web Token

- **Output:**  
  - (JSON) success: true|false

**Details:**
- **header:** 'Authorization', 'Bearer ' + token  
- **method:** 'POST'  
- **datatype:** 'json'  
- **url:** 'http://<host>:8080/trace/auth/logout'  
- **contentType:** ''

2.4. Open Tracking Session

**URL:** http://<host>:8080/trace/auth/session/open  
**Description:** Enables a tracking application to request a unique sessionld.

- **Params:**  
  - (JSON) token: JSON Web Token

- **Output:**  
  - (JSON) success: true|false  
  - (JSON) session: sessionld

**Details:**
2.5. Submit Route

**URL:** http://<host>:8080/trace/tracker/put/track/

**Description:** Enables a tracking application to report a whole tracked route.

**Params:**
- (JSON) token: JSON Web Token
- (JSON) session: The corresponding sessionId
- (JSON) track: A list of locations that compose the taken route

**Output:**
- (JSON) success: true|false

**Details:**
- header: 'Authorization', 'Bearer ' + token
- method: 'POST'
- datatype: 'json'
- url: 'http://<host>:8080/trace/tracker/put/track'
- contentType: 'application/json'

2.6. Get User Sessions

**URL:** http://<host>:8080/trace/tracker/sessions

**Description:** Allows the user to request for a list of all his sessions.

**Params:**
- (JSON) token: JSON Web Token

**Output:**
- (JSON) sessions: A list of all the user's sessions.

**Details:**
- header: 'Authorization', 'Bearer ' + token
- method: 'POST'
- datatype: 'json'
- url: 'http://<host>:8080/trace/tracker/sessions'
- contentType: ''

2.7. Get User Sessions And Dates

**URL:** http://<host>:8080/trace/tracker/sessionsAndDates
**Description:** Allows the user to request for a list of all his sessions and corresponding dates.

Params:
- (JSON) token: JSON Web Token

Output:
- (JSON) sessionsAndDates: A list of all the user's sessions and corresponding dates

Details:
- header: 'Authorization', 'Bearer ' + token
- method: 'POST'
- datatype: 'json'
- url: 'http://<host>:8080/trace/tracker/sessionsAndDates'
- contentType: ''

### 2.8. Get Route by Session

**URL:** http://<host>:8080/trace/tracker/route

**Description:** Allows the user to request for the route of a specific session.

Params:
- (JSON) token: JSON Web Token
- (JSON) session: The session to be used for the route lookup

Output:
- (JSON) route: A list of all the user's tracked locations during the specified session

Details:
- header: 'Authorization', 'Bearer ' + token
- method: 'POST'
- datatype: 'json'
- url: 'http://<host>:8080/trace/tracker/route'
- contentType: 'application/json'