

DELIVERABLE REPORT

D5.2

“User Profiling and Personalization”

collaborative project

MASELTOV

Mobile Assistance for Social Inclusion and Empowerment of Immigrants with Persuasive Learning Technologies and Social Network Services

Grant Agreement No. 288587 / ICT for Inclusion

project co-funded by the
European Commission















Information Society and Media Directorate-General
Information and Communication Technologies
Seventh Framework Programme (2007-2013)

Due date of deliverable:	31 December, 2013 (month 24)
Actual submission date:	07 August 2014 (Revision 1)
Start date of project:	Jan 1, 2012
Duration:	36 months

Work package	WP5 – PERSONALIZATION AND RECOMMENDATION
Task	T5.2 – User Profiling and Personalization
Lead contractor for this deliverable	AIT
Editor	Sofoklis Efremidis
Authors	Sofoklis Efremidis, Iakovos Georgiou, Ioannis Christou
Quality reviewer	Sara Wickert, Patrick Luley, Stefan Ladstätter

Project co-funded by the European Commission within the Seventh Framework Programme (2007–2013)		
Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

© MASELTOV - for details see MASELTOV Consortium Agreement

partner	organisation	ctry
01 	JOANNEUM RESEARCH FORSCHUNGSGESELLSCHAFT MBH	AT
02 	CURE – CENTER FOR USABILITY RESEARCH AND ENGINEERING	AT
03 	RESEARCH AND EDUCATION LABORATORY IN INFORMATION TECHNOLOGIES	EL
04 	UNDACIO PER A LA UNIVERSITAT OBERTA DE CATALUNYA	ES
05 	THE OPEN UNIVERSITY	UK
06 	COVENTRY UNIVERSITY	UK
07 	CESKE VYSOKE UCENI TECHNICKE V PRAZE	CZ
08 	FH JOANNEUM GESELLSCHAFT M.B.H.	AT
09 	TELECOM ITALIA S.p.A	IT
10 	FLUIDTIME DATA SERVICES GMBH	AT
11 	BUSUU ONLINE S.L	ES
12 	FUNDACION DESARROLLO SOSTENIDO	ES
13 	VEREIN DANAIDA	AT
14 	THE MIGRANTS' RESOURCE CENTRE	UK

CONTENT

1. Executive Summary	5
2. Overview	6
3. User Data and Preferences.....	11
3.1 User Information and Preferences	11
4. Events and Notifications	16
4.1 Multi-sensory context awareness	16
4.2 Event manipulation	23
4.3 Structure of Events	23
4.4 Event and notification log.....	24
4.5 Recommendations log	25
5. User Profile Interfaces	26
6. Security and Privacy Issues	27
7. User Profile Design and Prototype Implementation.....	28
7.1 MApp Activities Interconnection.....	29
7.2 Client GUI.....	30
7.2.1 Login, Registration and Language Selection	30
7.2.2 User Preferences.....	33
7.3 Client API (Content Provider)	36
7.3.1 AITUserProfileProvider	36
7.3.1.1 Is User Logged In?	37
7.3.1.2 Get Current Language.....	37
7.3.1.3 Get All Fields	37
7.3.1.4 Get One Field by Id	38
7.3.1.5 Send An Event.....	38
7.3.1.6 Update Fields	39
7.3.2 Content Resolver.....	40
7.3.3 MANIFEST.XML.....	40
7.3.4 Code Examples	40
7.4 Back-end Management GUI	44
7.4.1 Compatibility.....	44
7.4.2 Available Operations	45

7.4.2.1	Login Page	45
7.4.2.2	Dashboard	45
7.4.2.3	Messages	46
7.4.3	Personal Data Fields	47
7.4.3.1	Editing and Deleting Fields	47
7.4.3.2	Adding a New Field	51
7.4.4	User Editable Fields	51
7.4.4.1	Field Description.....	51
7.5	Back-end Applications Programming Interface (API)	52
7.5.1	Data format.....	52
7.5.2	User Profile Web service	53
7.5.2.1	User Data Collection.....	53
7.5.2.2	Create New User.....	56
7.5.2.3	Update User's Preferences.....	58
7.5.2.4	User Profile Fields Collection.....	60
7.5.2.5	User Profile Fields.....	66
7.5.2.6	Languages	71
7.5.2.7	Event	75
7.5.3	Error Reporting.....	77
7.5.3.1	Error Structure Result	77
7.6	User Data Security	79
7.7	Engineering Issues	80
7.8	Recommender	81
8.	Database Schema.....	82

1. EXECUTIVE SUMMARY

This document reports on the architecture and design of the user profile, which is the central component of the overall MASELTOV platform as it maintains user data, preferences, and the user context as it is captured through events and notifications coming from a number of smartphone sensors and Mapp applications, in an attempt to support personalized services. The document presents the internal structure and interactions of the User Profile with the other components of the platform, and the interfaces it offers to other applications for communicating events and notifications and manipulating the user profile data and preferences. Moreover, the back end management interface of the user profile is presented, which allows the formulation of the structures that store user data and preferences, and the API the back end component offers. Finally, a prototype implementation of the User Profile is presented.

2. OVERVIEW

This document is a report on the User Profile, which is a central component of the overall MASELTOV platform. It provides details on its architecture, its interrelations with other components, in particular with the recommender, as well as its relations to and interactions with the rest of the MASELTOV applications and services. Moreover, it presents design alternatives and design decisions pursued for its structure and functionalities.

The User Profile maintains information about personal data and user preferences, as well as collected knowledge on users' usage behavior, progress and user context recognitions that facilitate the personalization of used services, like triggering of personalized recommendations on MASELTOV functionalities, suggesting the use of targeted services and providing personalized assistance. Based on information contained in the user profile, the resulting targeted services and generated personalized recommendations are expected to enhance the overall user experience and speed up the social inclusion of the immigrant users. In addition, feedback and progress indicators that are maintained by the User Profile allow the monitoring of the user satisfaction for the offered services and the progress they make towards set targets, enhancing thus the level of personalization of the offered services.

The overall architecture of the User Profile is shown in Figure 1. The figure depicts the client and the server sub-components of the User Profile, which are in direct interaction with each other. The client component provides three interfaces: (a) an API to which other MApp applications can send their events and notifications, (b) a GUI for the user to set and update user data and preferences, and (c) an API for other MApp applications to query and update user data and preferences and usage data. The server component provides a GUI for managing the User Profile and defining the structure of the user data and preferences fields that are maintained by it. The server component of the User Profile makes use of a back end data store for persistently maintaining profile data, user preferences, and events. Details of the aforementioned interfaces are given in subsequent sections.

It should be noted that the design of the User Profile and the Recommender provides for buffering of events and delayed transmission when there is no connection between the smartphone and the backend components. One advantage of local storage of events is that they can be directly used by lightweight recommenders that run on the smartphone. Current announcements of Jess 8.0 (herzberg.ca.sandia.gov) and DROOLS (drools.jboss.org) are beta versions and rather unstable. When rule engines for Android become available, it will be an easy task to have fast lightweight recommendations produced by the smartphone components only.

In addition to the users' data and preferences, the User Profile maintains a log of events and notifications that capture the user context. Such data may pertain, for example, to the current user location, or user actions, like use of MApp applications, browsing for specific keywords, or searching through the Social Radar for assistance. The combined information of the user context and the user data and preferences forms the basis for the generation of targeted recommendations by the recommender.

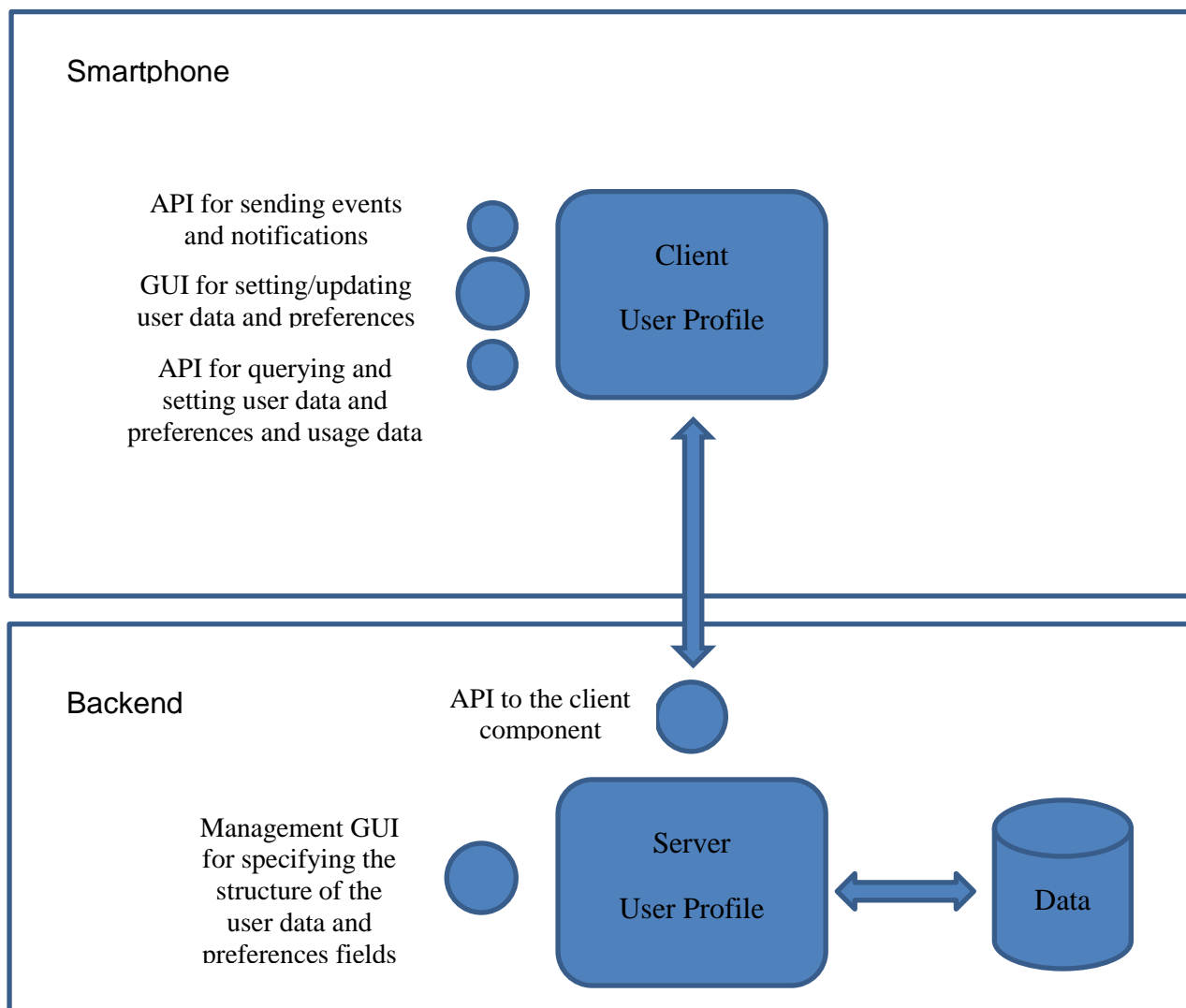


Figure 1: Architecture of the User Profile.

The User Profile is complemented by the Recommender System whose task is to issue personalized recommendations to its users based on the events and notifications that are collected from MApp applications as well as the declared user preferences and user data. The information that is carried through the events and notifications are used to formulate the required level of user's context awareness, and, allow the recommender to issue targeted recommendations based on a set of rules. Therefore, the functioning of the recommender depends on (a) events received from MApp applications, which are application dependent and reflect the current user activity and context, (b) notifications received from MApp services, like capturing and reporting the current user location, (c) the information contained in the user profile, like user preferences and progress indicators, and (d) a set of defined rules, which may also be specialized to specific user groups. (a) and (b) is dynamic information that captures the user context, whereas (c) is static information as has been declared by the user. Static and dynamic information is stored in the back end database for the recommender to use. The structure of the database is given in Deliverable D5.1. Finally the rules that drive the recommender reside in special purpose files outside the back end database. Examples of recommender rules are given in subsequent chapters.

The architecture of the User Profile and its relation to the recommender system is shown in Figure 2. The figure shows that the User Profile is the central component of the overall architecture and the single point of interfacing with the rest of the MASELTOV applications and services. The client User Profile component forwards the events and notifications it receives from other MApp applications to the back end User Profile, which eventually logs them in the back end database. In the sequel the recommender queries the database for events that will trigger recommendations to be presented to the user.

By making the user profile a central element of the architecture, it becomes possible to easily disable the recommender system without affecting the smooth operation of the rest of the components, while events and notifications can still be logged for off line processing.

The architecture that is presented in Figure 2 shows the two platforms on which the components of the User Profile and the Recommender system run, namely the mobile client and the back end server. The architecture shows a lightweight rule based recommender system (shown shaded in the figure) that runs on the smartphone and a fully-fledged rule based recommender system that runs on the backend server. The reasoning behind this approach is to allow the lightweight recommender provide its services even in the absence of any connectivity, allowing it to issue quick and lightweight recommendations. On the other hand the backend recommender will be able to issue more specialized recommendations by making use of a richer set of more powerful rules. As shown in the architecture of Figure 2, a set of rules are used to drive the recommender. The rules are actually programs in the specialized language of the rule based engine and they reside in files that are kept separate from the back end database. The rules have the form precondition \rightarrow action, where precondition is a predicate over events collected from various Mapp applications as well as user data and preferences as specified by the user. The collected events form the dynamic part of the user profile. They come from a number of Mapp applications (e.g., POI, text lens, game, social radar, etc.) or sensors (e.g., current user location) and they actually capture the user activity and state. In this sense they carry all information that is pertinent to the user context. This user contextual information along with the declared user preferences forms the basis for the recommendations that are produced to the user.

It should be noted that the architecture of Figure 2 provides hooks for future extensions regarding the lightweight recommender that runs on the client side. As noted before, such technologies are being developed and current implementations are rather unstable or rely on non-standard ways of integrating packages to Android. For example DROOLS on Android relies on AWT classes and needs to be packaged in a non-standard way, breaking compatibility with non-rooted Adndroid devices. Therefore in this document we keep the architecture of Figure 2 as a reference for future implementations and we actually ignore the shaded component of it.

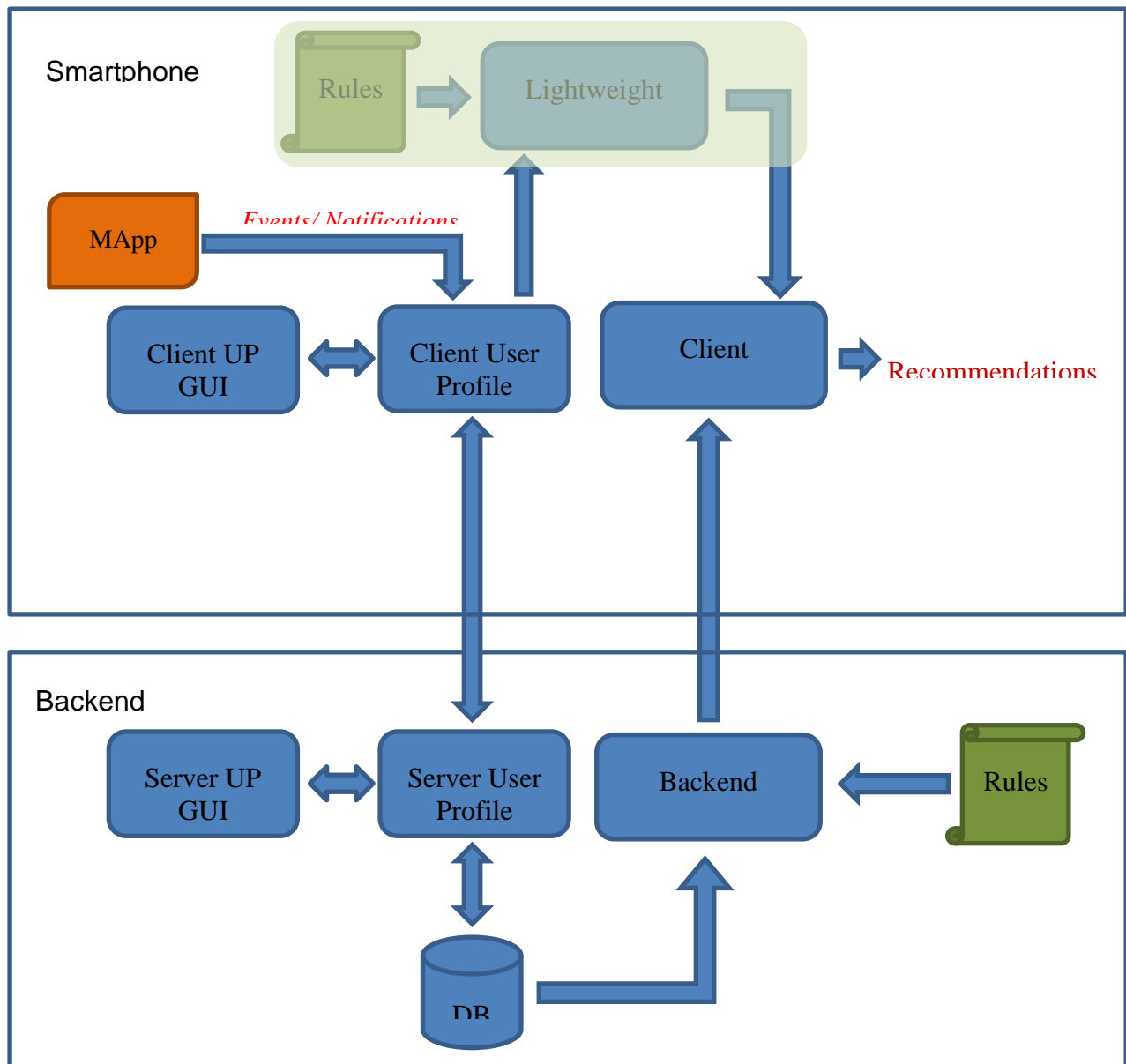


Figure 2: Reference Architecture of the User Profile and the Recommender System.

Extensions of the reference architecture of Figure 2 can easily be envisaged. For example, the present architecture employs a rule based system as the basis of the recommender. For the purposes of the prototype implementation rules are defined for driving the recommender to generate personalized recommendations to the user. The rules may take into account the user activity as well as the user data and preferences to specify what recommendations to be issued. This approach is sufficient for the targeted level of personalization. A future extension may also employ a statistical recommender, which may work in parallel with the one user in this architecture. A statistical recommender is based on general patterns that are detected from the behaviour of users. They typically build a model from the past activities of a user of a group of users. They then use the constructed model to issue recommendations. For example based on the user behavioural data it may detect that most female immigrant users from Colombia in the age group 20 – 25 are inclined towards taking French classes. Therefore, when a user is detected to fall within this group, the recommender will issue a

recommendation for taking French classes. The available dataset is compiled from the activities of a group of users and is analysed by considering vectors in multidimensional spaces. In the previous example such a vector is defined across the dimensions (sex, country of origin, age group). By collecting and analysing a large number of such vectors a statistical recommender may produce recommendations that reflect the tendencies, or what is highly probable to be case (taking French courses in the previous example). A statistical recommender produces recommendations that have an associated level of probability. The accuracy of the statistical recommender depends on the available dataset as well as the algorithms used for analysing it. It is apparent that the functionality of a statistical recommender depends on the availability of a data set (or training set) from which a model is created to be used in subsequent recommendations. It also turns out that the size of the data set affects the quality and accuracy of the produced recommendations; larger data sets will typically result into more accurate recommendations.

The two major approaches for statistical recommenders are based on collaborative filtering and content based filtering. Collaborative filtering systems recommend items to the user based on past ratings of other users. Content-based recommending systems recommended items to the user based on similar items the user has liked in the past. Statistical recommenders are not in the scope of the design and prototype implementation of WP5 but they can be a useful complement to the rule based one for future extensions of the prototype.

The following sections present design details of the User Profile, as well the recommender system, their interfaces with the rest of the MASELTOV components, and the structure of events and notifications that are communicated to them.

3. USER DATA AND PREFERENCES

It was mentioned in the previous section that the User Profile is a central MASELTOV component and contains among others

- information about the user and user preferences,
- information about user behavior, i.e., a log of events and notifications that are issued by other MApp applications and services, including progress and usage events
- a log of the recommendations that are issued by the recommender system

The information pertaining to a user that is kept with the User Profile comprises two parts, a static and a dynamic one. The static part contains information about the user and his/her preferences, while the dynamic part contains all information about the user context and state as is captured by the events and notifications that are generated by Mapp applications and services. This section elaborates on the static part of the User Profile, namely, user data and preferences.

3.1 USER INFORMATION AND PREFERENCES

The information about the user and user preferences that is contained in the User Profile has a hierarchical structure and comprises two parts, a static and a dynamic one.

The static part contains data that are rather invariant, like user attributes and preferences. These data are kept in a hierarchical structure and are typically entered by the user through the client User Profile GUI. Once this information is entered by the user, it is likely that will not change frequently. Nevertheless, static data can be updated by the user or Mapp applications. Taking into account the sensitivity of personal data and the particularities of the target user group of immigrants who are expected to be mostly reluctant to provide such data, the User Profile has been designed to make most of the fields of the static data be optional. It is expected that reluctant immigrant users will feel more comfortable to provide a minimal set of mandatory fields than providing long lists of personal data, an activity which would rather deter them from using the MASELTOV platform.

The following list summarizes tentative contents of the static part of the User Profile as well as its structure. Items that are marked with (*) are mandatory, the rest are optional. The actual structure of the static part of the User Profile, with the exception of the mandatory fields, is not fixed but it can be defined and arranged by the provider of the MASELTOV service. Therefore what is shown below is a tentative structure. Different deployments may opt for additional fields or even leave out fields from those shown below.

- Personal data
 - Username (*): a user chosen name to refer to him/her. The username has no association whatsoever to the actual identity of the user. It is used to refer to a user in certain applications, like the MASELTOV forum.
 - Email address (*): a user chosen email address. The email address serves as a cyber-identity of the user and, similar to the username, it bears no association to the actual user identity.

- Password (*): a user chosen password for logging in and accessing MApp services.
 - Name: the actual name of the user. As it may reveal their actual identity, it remains an optional field. MApp applications may use its value to generate personalized messages, for instance “Jack tomorrow at 21:00 a free concert is organized in which your favorite ethnic music will be performed”.
 - Birthdate: Its value may be used to classify the user into an age group and so issue more targeted recommendations.
 - Gender: Its value may be used to classify the user into a gender group and issue targeted recommendations. For example, recommending organizations, such as the Moroccan Women’s Centre, which focus on supporting women.
 - Nationality: Its value may be used to classify the user into a nationality group and so issue more targeted recommendations. For example the issuance of a recommendation to Nigerian immigrants for the possibility of legal support regarding their residence status.
 - Education Level: It can have any of the values: *None, Primary School, Secondary School, College, University*. Its value can be used for issuing recommendations, like the organization of introductory courses for immigrants with *None* education level.
 - Current job: Its value can be used for issuing recommendations like the organization of a seminar for immigrants currently working as gardeners.
 - Years in country: Its value can be used to classify the user into groups according to the length of time they have been in the country, so as recommendations may be destined to newly arrived immigrants, others to older ones.
 - Language and skills levels. Its value can be used to issue personalized recommendations for language learning courses. Typical skill levels are *introduction, intermediate, advanced*.
 - City (*): Its value can be used to customize the recommendations to the city the immigrant lives in. The field is mandatory as it is used to initialize wiki content and navigation services.
 - Religion: Its value can be used to classify the user into religion groups and for issuing personalized recommendations for the different groups. Users select their preferred language upon registration and they may change the language whenever they feel like it from the User Preferences.
- Preferences
 - Preferred Language of Communication: English, French, Spanish, and Arabic are possible options. Users select their preferred language upon registration and they may change the language whenever they feel like it from the User Preferences.
 - Preferred cuisine. Any, Mexican, Vegetarian, Italian are possible options.
 - Entertainment/hobbies

- Games
 - Adventure
 - Skill/competence
- Arts
 - Music
 - Theater
 - Dance
 - Literature
 - Painting
- Recreational
 - Jogging
 - Cycling
 - Cinema
- Cooking

The list above contains indicative fields of the static part of the User Profile. As noted above the list is neither exhaustive nor final. With the exception of the four mandatory fields, the actual content and structure of the static part of the User Profile is rather flexible as new fields and nesting structures can be defined. In this way the static data can be tuned to the specifications of NGOs', which have a detailed view of the appropriateness of different fields for the immigrant groups they target and can come up with specific suggestions for them.

The user data and preferences are stored in the back end database along with the events and notifications that are produced by various Mapp applications and services. All together form the basis for the recommendations that are produced by the rule based recommender. As it is noted above, most of the fields of user related information are optional, the intention being that immigrant users be rather encouraged to use the user profile and recommender systems of MASELTOV instead of being skeptical about providing personal information and preferences, which they may consider sensitive and confidential. It should be noted that from a technical perspective, the more information and more details about user preferences the recommender system has, the better the personalization level and the quality of the recommendations it will issue. In particular, in the absence of any user preferences the recommender can only issue generic recommendations, which will not be targeted to a specific user. On the other hand, as trust is gained in the MASELTOV applications and services, including the recommender system, additional information may be provided by immigrant users for their personal data and preferences, which will improve the quality level of the recommender's responses.

The static part of the profile can be flexibly defined and tuned to the needs of the specific service provider by using the provided management GUI. As parts of the profile (like the hobbies) may be open-ended the structure allows for definition of new items as well as specializations of existing ones. The User Profile provides an interface for querying the fields of the static part of the user data and preferences as is detailed in subsequent sections.

The following DTD specifies the structure of a sample user profile as shown in the previous section, both its static and its dynamic parts. Elements that are tagged with a ? are optional, in accordance with the informal list above.

```
<!ELEMENT UProfile (StaticPart, DynamicPart)>

<!ELEMENT StaticPart (PersonalData, Preferences?)>

<!ELEMENT PersonalData
  (Username, email, Name?, Birthdate?, Sex?,
   Nationality?, EducationLevel?,
   CurrentJob?, YearsInCountry?,
   PreferredLanguage?, LanguageSkills?,
   City, Religion?)>
<!ELEMENT Username (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Birthdate (#PCDATA)>
<!ELEMENT Sex (#PCDATA)>
<!ELEMENT Nationality (#PCDATA)>
<!ELEMENT EducationLevel (#PCDATA)>
<!ELEMENT CurrentJob (#PCDATA)>
<!ELEMENT YearsInCountry (#PCDATA)>
<!ELEMENT PreferredLanguage (#PCDATA)>
<!ELEMENT LanguageSkills (LanguageSkill)+>
<!ELEMENT LanguageSkill (Language,Level)>
<!ELEMENT Language (#PCDATA)>
<!ELEMENT Level (#PCDATA)>
<!ELEMENT City (#PCDATA)>
<!ELEMENT Religion (#PCDATA)>

<!ELEMENT Preferences
  (preferredCuisine,EntertainmentHobbies)>
<!ELEMENT PreferredCuisine (#PCDATA)>

<!ELEMENT EntertainmentHobbies
  (Games,Arts,Recreational,Cooking)>

<!ELEMENT Games (Adventure,Skill,Lottery)>
<!ELEMENT Adventure (#PCDATA)>
<!ELEMENT Skill (#PCDATA)>
<!ELEMENT Lottery (#PCDATA)>

<!ELEMENT Arts (Music,Theater,Dance,Literature,Painting)>
<!ELEMENT Music (#PCDATA)>
<!ELEMENT Theater (#PCDATA)>
<!ELEMENT Dance (#PCDATA)>
<!ELEMENT Literature (#PCDATA)>
<!ELEMENT Painting (#PCDATA)>
```

```
<!ELEMENT Recreational (Jogging,Cycling,Sailing)>
<!ELEMENT Jogging (#PCDATA)>
<!ELEMENT Cycling (#PCDATA)>
<!ELEMENT Cinema (#PCDATA)>

<!ELEMENT Cooking (#PCDATA)>

<!ELEMENT DynamicPart (DynamicData*)>

<!ELEMENT DynamicData (User,Event)>
<!ELEMENT Keyword (#PCDATA)>
<!ELEMENT Event ...>
```

4. EVENTS AND NOTIFICATIONS

It was noted in a previous section that events and notifications form the dynamic part of the User Profile. The dynamic information is automatically collected and reflects the behavior, the context and the state of the user. Dynamic data are generated because of user activities and are collected from smartphone sensors or MApp applications that are used by the user.

4.1 MULTI-SENSORY CONTEXT AWARENESS

Sophisticated smartphones that are already available on the market contain a number of sensors that can sense the user's environment. Examples are GPS receiver, camera, proximity sensors, accelerometers, etc. In the context of MASELTOV two sensors are employed for capturing the user's context, namely the GPS receiver and the camera. The user's context is communicated to the User Profile through events, which are stored in the back end database, but more importantly, they are used for generating recommendations to the user.

The GPS receiver can be queried through a system call for the current geographical coordinates of the user and also the user's movement. The raw data that are retrieved from the GPS receiver are converted into events by a special purpose developed service, which generates at regular intervals events that contain the user's position. The frequency of the generated events is a configurable parameter allowing either fine grain granularity of the user's position at the expense of extra space to store the corresponding events or coarse grained granularity of user's position. The generated events capture the user's context (position) and state (user is still, walking, travelling, etc.) and are used for generating recommendations of nearby POIs (for example). The generated recommendations may be further refined by taking into consideration preferences that have been declared by the user in his/her profile.

Another smartphone sensor that is used in the context of MASELTOV is the camera. The raw images that are captured by it are processed by the TextLens application to recognize words. In the sequel the TextLens sends an event to the User Profile, which contains the text that has been recognized. The event may subsequently fire a rule for the generation of a recommendation concerning health issues if, for example, the recognized text contains a word about measles.

Besides the hardware sensors the MASELTOV platform comprises a number of applications that produce their own events. The Language Learning Mapp for example generates events as the user goes through the different modules or sections of learning material, or when a language qualification test is successfully passed. The forum Mapp generates its own events, and the same holds for the social radar and other applications. Similarly to the previous cases, the generated events capture the user's context and form the basis for targeted recommendations. In a more abstract setting the various Mapp applications may be considered as sensors that capture different aspects of the user's context and communicate it to the User Profile through events.

Table 1: List of MApp Events.

Event id	Mapp Component Name	Source	When event is issued	Values of <key, value> pairs
Usage Events				
eu1	Language learning	LanguageLearning	At stop	("duration", <duration>)
eu2	TextLens	TextLens	At stop	("duration", <duration>)
eu3	User Profile	User Profile	At stop	("duration", <duration>)
eu4	Recommendations	Recommendations	At stop	("duration", <duration>)
eu5	Info	fluinfo	At stop	("duration", <duration>)
eu6	Augmented reality	maseltov-arnav	At stop	("duration", <duration>)
eu7	Help Radar	GeoRadar	At stop	("duration", <duration>)
eu8	Navigation	flunav	At stop	("duration", <duration>)
eu9	Places of Interest	flupoi	At stop	("duration", <duration>)

eu10	Serious Game	SeriousGame	At stop	("duration", <duration>)
eu12		Social Forum	At stop	("duration", <duration>)
Progress Events				
ep1				
ep2	Language learning	Language learning	User completes an activity in a language lesson	("publication", <string: publication code>), ("lesson", <string: lesson id>), ("task", <string: task id>), ("progress", <number: percentage>)
ep3	Language learning	Language learning	User completes a test at the end of a language lesson	("publication", <string: publication code>), ("lesson", <string: lesson id>), ("score", <number: percentage>)
ep4	Language learning	Language learning	User rates a statement about their language lesson with a value of 1-5	("publication", <string: publication code>), ("lesson", <string: lesson id>), ("statement", <string: statement id>), ("rating", <number: integer 1..5>)
ep5	Serious Game	SeriousGame.CollectedExceptions	Currency collected	("coins" <number: integer>)
ep6	Serious Game	SeriousGame.ActivityCompleted	User completes an activity in a scenario and receives a journal update	("theme" <string: theme name>)

ep7	Goal Setting	Goal Setting	User within 7 days of deadline	
Interesting Things				
e1	User Profile	GPS tracking	Every 1 minute provided user has moved	("Longitude", <long>), ("Latitude", <lat>)
e2		Language learning	Upon change of learning level (CEFR: A1-A2-B1)	("Language", <lang>), ("Course", <course>), ("Level", <new level>)
e3		Wiki search	Upon pressing of "search" button	("keywords", <keyword list>)
e4	Text Lens	TextLens	Between text detection and text translation (in case user corrects the detected text, event will carry corrected text)	("detectedText", <user corrected text in the image>)
e5		MaseltovContext. ModeOfTransportation	As soon as the mode of transportation changed	("type", <String>), ("confidence", <int>)
e6		MaseltovContext. ActivitySummary	Sent at the end of the day	("total_distance", <long>), ("distance_walking", <long>), ("distance_driving", <long>), ("distance_biking", <long>), ("distance_unknown", <long>), ("time_moving", <long>),

				("time_still", <long>), ("time_walking", <long>), ("time_driving", <long>), ("time_biking", <long>), ("time_unknown", <long>)
e7		MaseltovContext. Interests	Sent at the end of the day	("interests", <String keyword list>), ("weights", <Integer weight list>)
e8		MaseltovContext. PlaceEntry	Sent if a person stays at least 5 minutes at one location.	("longitude", <double>), ("latitude", <double>)
e9		MaseltovContext. PlaceExit	Sent if a person leaves a location. Only occurs after MaseltovContext. PlaceEntry	("longitude", <double>), ("latitude", <double>)
e10		MaseltovContext. PlaceHistory	Sent at the end of the day	("coordinates", <Double coordinates list>), ("type", <String types list>), ("ts_entry", <Long ts_entry list>), ("ts_exit", <Long ts_exit list>), ("osm_id", <Integer id list>)
e11		MaseltovContext. PlacesOfInterest	Sent at the end of the day	("coordinates", <Double coordinates list>), ("type", <String>), ("visit_duration", <long duration list>), ("visit_count", <Integer visit_count list>), ("osm_id", <Integer id list>)
e12		MaseltovContext. SocialInteraction	Sent at the end of the day	(Only for illustration purposes in JSON) "contact_data": [{ "contact_id": "int", "calls" : ["call" : { "ts": "long", "duration" : "int", "type" : "String"}, ...],

				"messages" : [{"message" : {"ts": "long", "type" : "String"}, ...}], {...}]
e13	Help Radar	GeoRadar. signupVolunteer	Sent when the user signs up himself/herself as volunteer	("username",<String>), ("knowledges",<String>), ("languages",<String>), ("status",<String>)
e14	Help Radar	Georadar. removeVolunteer	Sent when the user remove himself/herself as volunteer	("volunteerUsername",<String>)
e15	Help Radar	Georadar. statusVolunteer	Sent when the volunteer status changed	("volunteerUsername",<String>), ("status",<String>)
e16	Help Radar	GeoRadar. contactVolunteer	Sent when the user contacts a volunteer	("username",<String>), ("volunteerUsername",<String>), ("reqKnowledges",<String>), ("reqLanguage",<String>)
e17	Help Radar	GeoRadar. ratingAssistance	Sent when the user rates an assistance	("username",<String>), ("volunteerUsername",<String>), ("rating",<String>)
e18	Forum	Social Forum.sendPost	Message (i.e. post) posted	("username",<String>)
e19	Forum	Social Forum.sendReply	Replying to a message (i.e.	("username",<String>)

			post) another has posted	
e20	Places of Interest	flupoi.PoiSearch	search keywords	("keyword",<String>)
e21	Info	fluinfo.CategoryTitle	viewed categories	("categoryTitle",<String>)
e22	Info	fluinfo.ArticleTitle	viewed article	("articleTitle",<String>)
e23	Navigation	flunav.RouteStart	start/destination	{ "pointAs": "start", "title": "<title>", "latlng": "<latitude>:<longitude>:WGS84:", "type": "<type>" }
e24	Navigation	flunav.RouteEnd	start/destination	{ "pointAs": "end", "title": "<title>", "latlng": "<latitude>:<longitude>:WGS84:", "type": "<type>" }

Table 1 gives a comprehensive list of all events defined for the MASELTOV platform and for each one the source component and the data that are carried with the event. Events are classified into three categories:

- Usage events are generated at the start, stop, suspension, resumption of applications. They are used for generating statistics for the relative usage of various applications.
- Progress events are generated by specific applications, like the language learning, to indicate user progress through the use of the application. They are used to monitor user progress.
- Interesting things are generated when an action of interest takes place, like new user position, new text recognized by the Text Lens, etc. These events are referenced in the recommendation rules that are used by the recommender to produce targeted personalized recommendations as presented in Deliverable D5.4.2 “Recommendation Services”.

It is evident that events that capture the user context, as explained previously, may carry sensitive user related information, like the current user’s position. Collection and use of such data may inconvenience the user, therefore, as a step towards making the user feel comfortable with the collected contextual information, he/she is given control over which dynamic data he/she agrees to be collected. For example the user may disable the collection of positional data through the GPS receiver. A number of switches in the User Profile allow the user control over which of a number of dynamic data are allowed to be collected and further processed. A step towards maintaining user privacy is the design decision to free the dynamic profile data from any information that could reveal the user identity. In particular, no IMEI number or device MAC data are recorded with the collected dynamic data.

4.2 EVENT MANIPULATION

Events are produced by Mapp applications and communicated to the User Profile for further storage and processing. If the user maintains a connection to the backend server, dynamic event data are continuously transmitted to it. Otherwise, if no connection is available, they are stored temporarily in the smartphone and then flushed to the backend server next time when connection is established.

Details of the client User Profile and the server User Profile APIs are presented in later sections. Events are replayed to the back end database where they are stored but otherwise are not meant to be manipulated by other MApp applications. In particular they cannot be queried or even modified; they are rather used solely by back end applications and services that can interpret the user’s context and provide thus personalized services to him/her. Examples of such services are the Recommender service, and services that provide usability statistics for MApp applications.

4.3 STRUCTURE OF EVENTS

It was noted before that an event carries user contextual information. As a number of different contextual parameters have to be captured, its structure must be as generic as possible. Therefore, an event contains the following fields.

1. its source identification, i.e., a unique id of the component and the action that has produced the event (for example `flunav.RouteStart`, `fluinfo.ArticleTitle`, `GeoRadar.contactVolunteer`, etc.)
2. a timestamp, indicating the time the event has been produced, and
3. a collection of `<key, value>` pairs that specify the information that is carried along with the event.

More concretely an event has the following structure (using Java class definitions).

```
String source;  
GregorianCalendar timestamp;  
HashMap<String, Object> info;
```

Each MApp application has its own unique source string, for example, events generated by the personalized learning application have as value of the source field “Learning”. When the mixed reality game reaches a point where a point of interest is shown in the background of a scene, it will send an event that has the value “MixedRealityGame” for the source field and {“ImmigrationOfficeBuilding”, `<location>`}} for the info field. Example usage of the interface is demonstrated in D4.1.2.

Each event is associated to the user whose actions result in the generation of the event. The user is not part of the event structure but his/her id is used when invoking the API for transmitting an event. Therefore invocations to the API have the form of (*user*, *event*), where *user* is the currently logged user. The client User Profile always knows who the user is after the login process.

Various MApp components are further structured as a set of interacting subcomponents. For example, the context module has various sub components such as “TransporationMode”, “SocialInteraction”, “PointsNearby”, etc. As a result the naming of the source field of an event follows a parallel nesting structure, like `<ComponentName>.<Submodule>`; e.g. “ContextModule.TransportationMode” to reflect the (sub) component it comes from.

4.4 EVENT AND NOTIFICATION LOG

Part of the information that is maintained by the User Profile is a log of the events and notifications that are communicated to it by other MApp applications (which forms the basis for the dynamic part of the User Profile). The log may be further used by statistical based recommender systems as noted in a previous section, which may provide recommendations based on behavioral patterns of a user or groups of users. It should be noted that a MApp application generates a number of events during each run, including events for its own activation and deactivation. Therefore the User Profile provides a uniform approach for collecting events either for their statistical processing or for calculating MApp application usage data.

Deliverable D5.4.1 “Recommendation Services” gives several examples of events that are generated by other MApp applications and the way they are used to produce personalized recommendations. Deliverable D5.4.2 will provide a systematic and categorized list of all

events to be produced by other MApp applications and the recommendations that are to be produced from them.

4.5 RECOMMENDATIONS LOG

In addition to the events, the recommendations issued by the recommender system are also logged in the back end server. When a recommendation is issued by the recommender system it is propagated both to the MASELTOV notification application for displaying it in the user's screen and the user profile component, which logs it in the back end database. In particular the Recommender sends its recommendations directly to the Android notification bar using the Android API. The notification bar displays pending recommendations, which can either be ignored or picked up by users. By monitoring this behaviour the recommender can learn which kind of recommendations are of interest for the user. Most of the recommendations need a direct link to a MApp service or tool to point the user directly to the service/info he/she needs.

5. USER PROFILE INTERFACES

The User Profile provides a number of interfaces to MApp applications, users, and administrators as summarized below.

- A user GUI which runs on the smartphone and allows the user to initialize and update their profile data and personal preferences by giving values to a number of fields that have been specified by the Service Provider through the back end management GUI.
- An API for other MApp applications to communicate events and notifications to it, and query and update the fields of the profile data and preferences. The interface (Android Content Provider) specifies a number of methods for
 - Logging an event e for user u ,
`logEvent(User u , Event e)`
 - Querying the value of field f of user u ,
`getFieldValue(User u , Field f)`
 - Setting the value of field f to v for user u ,
`setFieldValue(User u , Field f , Value v)`
- A management web-based GUI for defining the structure of the static part of the User Profile. The GUI allows the addition of new fields as well as editing and modifications of existing ones. The type of each field can be specified.
- A back end server API that allows the client User Profile component to communicate with the back end server. Communication between the client and the server User Profile is done with either XML or JSON messages.

The design and details of a prototype implementation of the above interfaces are given in subsequent sections of the document.

6. SECURITY AND PRIVACY ISSUES

The User Profile and the recommender system make use of sensitive personal and behavioral data in order to provide their services. As a result a number of security and privacy issues are raised, which must be treated in a responsible manner. The design approaches taken that address these issues are presented in the following.

As a first step, the MASELTOV platform remains ignorant of the real user identity. Even though the email field of the profile is mandatory and serves as a unique id for the user, it actually provides no direct pointer to the users' identity. Moreover the validity of the email address is never checked by the User Profile. For example if a user creates an account `abcd@somewhere.com` no one can make an association of the user's identity and the email account that is registered to the User Profile. Even if the user's identity is known to the email provider, this information will not (and should not) be available to the MASELTOV platform. Therefore users are essentially treated as anonymous, and the mail is used for referring to them and allowing them to have a means for logging in and using the MASELTOV applications.

Second, dynamically collected data are disassociated from the particular user they come from, since no user id (username/email, phone IMEI, MAC address, etc.) is maintained along with them. All collected user data may reference the user's id, so even if these data are compromised not even the registered user email address can be revealed. The only link between the email and the user id is maintained in a separate table of the user profile, which may be forced to reside in secure storage (for example tamper proof device) so as this information never gets revealed.

Moreover the User Profile gives the option to the user to selectively turn on and off the collection of behavioral as a step to put the user in control of what information they agree to be recorded about their activities. New users may opt for switching off all behavioral data collection as they may not trust the platform and may suspect possible misuse of such data. It is obvious that turning off the collection of behavioral data will impact the generation of personalized recommendations.

As a final step towards guarding the security of collected user related data and preferences an actual implementation may store these data in an encrypted form. This ensures that they are rendered useless to anyone except the MASELTOV platform itself.

Section 7.6 gives the low level details of the security features that have been implemented in the prototype.

7. USER PROFILE DESIGN AND PROTOTYPE IMPLEMENTATION

This section presents the details of the User Profile design and a prototype implementation of it. In particular, it presents

1. The structure and interfaces of the client component. In particular
 - a. The workflow of activities a user has to go through for registering and logging in for using the MASELTOV platform and the resulting interactions with other activities (MApp applications).
 - b. The client GUI that allows the user editing of profile data and user preferences
 - c. The client API that allows other Mapp applications to interact with the client User Profile component
2. The structure and interfaces of the server component. In particular
 - a. The backend GUI, which allows management and specification of the structure of the User Profile.
 - b. The server API specification for the communication of the client User Profile the with the back end server
 - c. The structure of the back end database.

In order to use the MASELTOV application the user must login with a valid MASELTOV account. The first screen the user sees upon starting the MASELTOV application is a register/login screen. MASELTOV users may select the Profile button from the dashboard screen in order change their preferences and see progress indicators or application usage statistics. Additionally, anonymous use of the MASELTOV services is also supported.

When registering, users are asked to provide a minimal set of mandatory information for the purpose of identifying them and allow also the correct functioning of some MApp applications. Thus, users are asked to enter an e-mail address, a password, a username of their choice and the user's city. The e-mail serves as a user identifier for the platform, even though it may refer to a non-existent service provider. For the MASELTOV platform the e-mail is treated as a string identifier and actually reveals no information about the identity of the user. The username is a user chosen identifier that allows him/her to be referenced in the MASELTOV forum, and again, it bears no relation to the actual user identity. The e-mail and username fields (both mandatory) are necessary to register a user and allow him/her to hold an account to the MASELTOV platform. Finally, the name of the user's home city is mandatory, and is used for initializing other MApp applications like the POI.

Users are able, and are encouraged, to complete a number of other fields (optional) about them and their preferences. These optional fields will help the recommender to generate personalized and targeted recommendations.

Figure 3 shows the User Profile components and the way they are related. The back end User Profile runs on a Tomcat web server and provides two interfaces one for the client User Profile and one for administration purposes. The whole architecture is distributed and communication between the components takes places through the Internet. The interface to the mobile client (client User Profile) is defined as an API; communication between the client

and server components is done with either XML or JSON strings. In addition, the server implements a GUI for administrative purposes. Through the latter interface, a Service Provider may define the structure of the user preferences fields that best suits the target group of immigrants that wants to address, and after taking input from the pertinent NGOs.

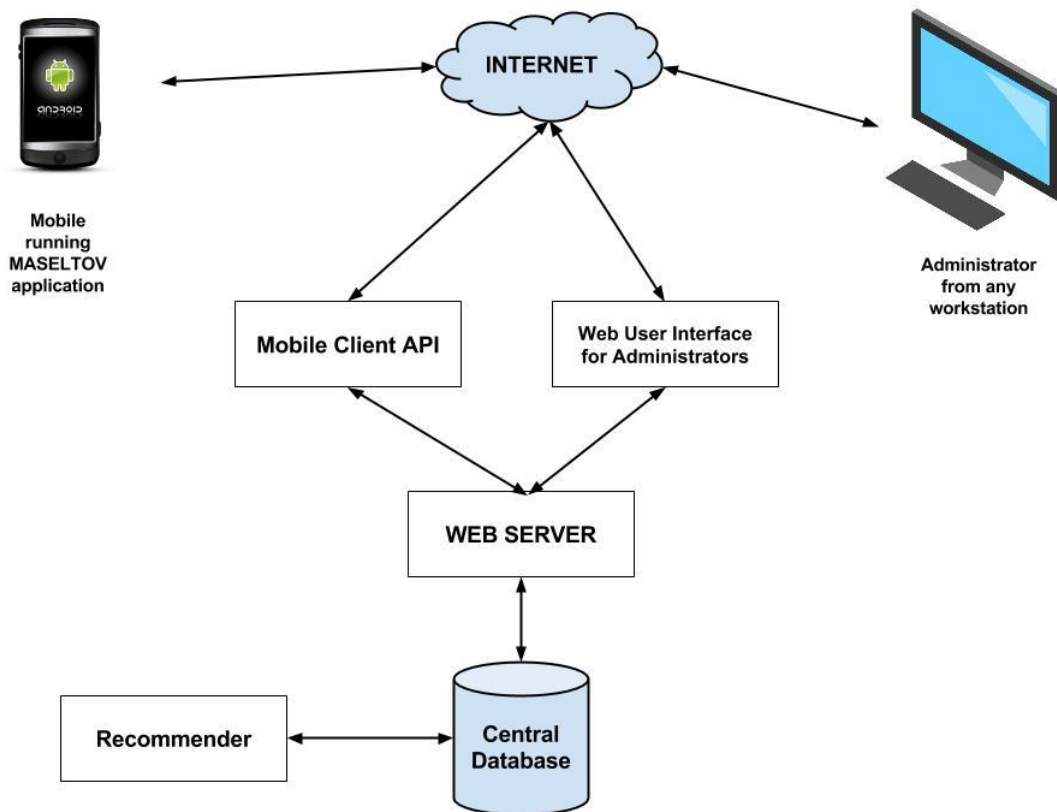


Figure 3: Inter-relationship Diagram.

7.1 MAPP ACTIVITIES INTERCONNECTION

The workflow diagram of Figure 4 shows the interconnections between the User Profile activities and the other MApp applications. The first time the app launches the MApp must call the AITUserProfileProvider. After a number of activities are invoked (as described in Figure 4) the User Profile will return to the activity that first called it.

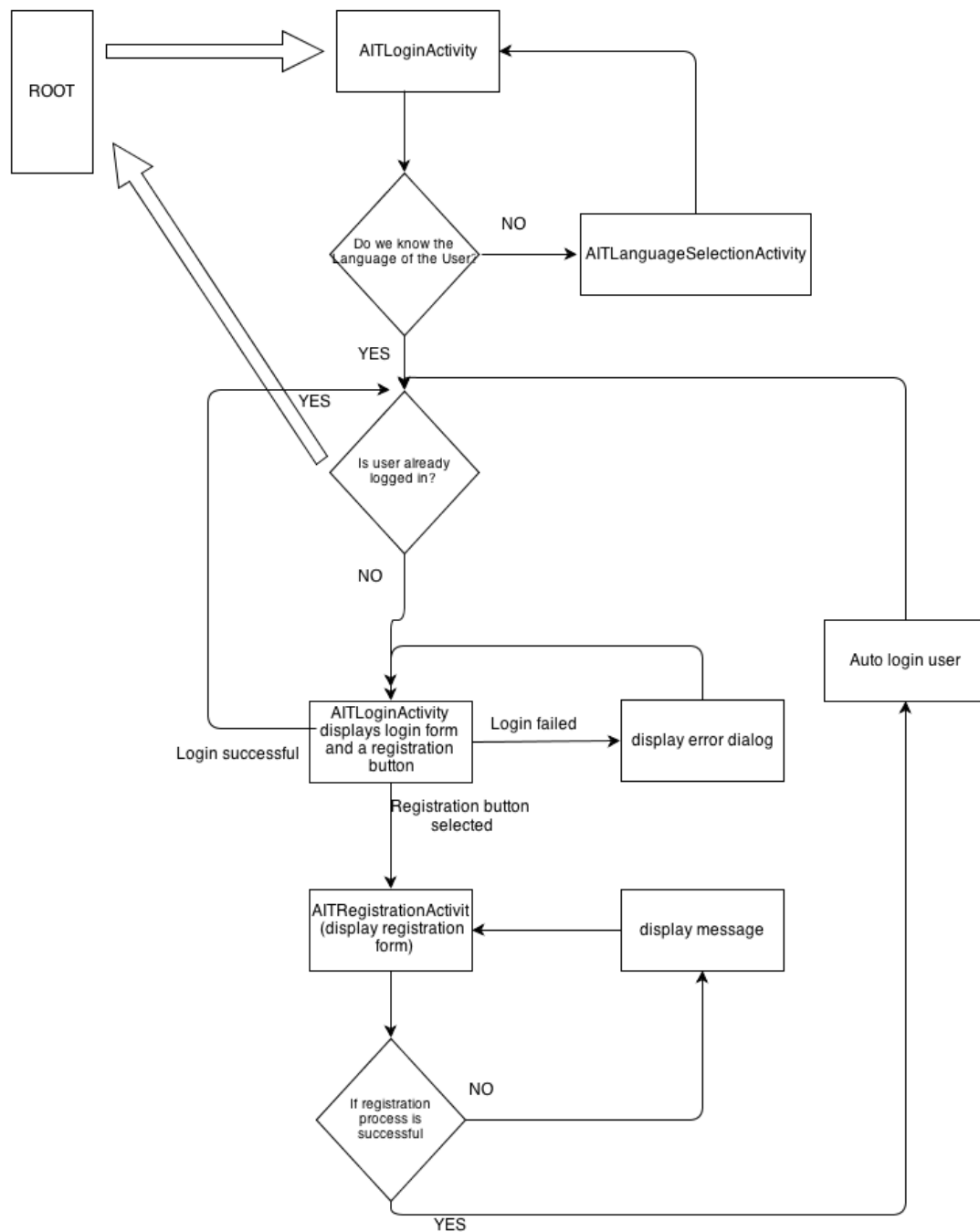


Figure 4: Activities Interconnection.

7.2 CLIENT GUI

7.2.1 LOGIN, REGISTRATION AND LANGUAGE SELECTION

The login form is shown in Figure 5 and is displayed after the AITLoginActivity is invoked. The app automatically responds with the AITLanguageSelectionActivity that enables the user to select his preferred language for the application. The user has the option of selecting the

language to be used by the application. The result of language selection is that all headers and captions as well as all informative and error messages will be shown in the selected language.

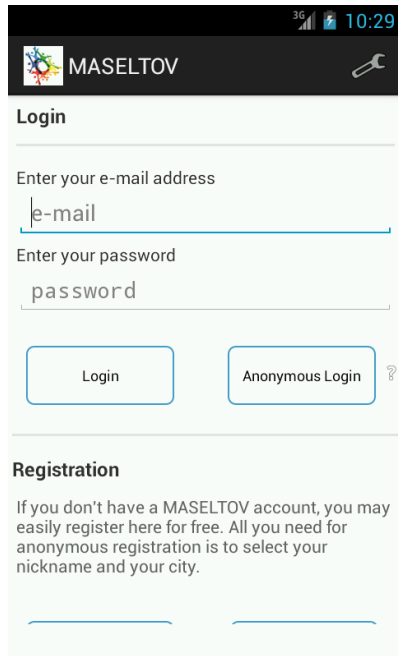


Figure 5: Login Screen.

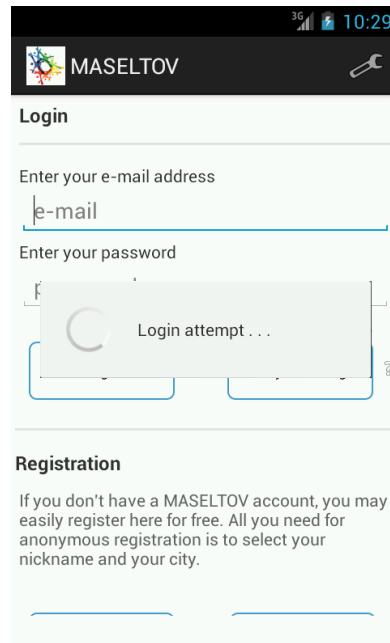


Figure 6: Loader during login process.

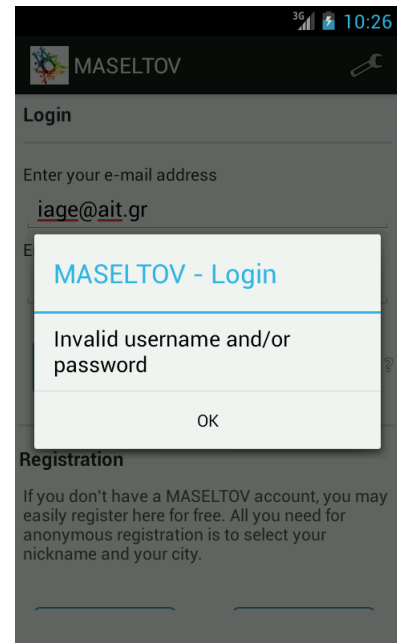


Figure 7: Login Failed Dialog.

Anonymous login is supported by the platform allowing users to login without registering an email address. Instead the client User Profile uses their device id for creating and filling in details about the user identity.

As long as the login is successful the client application brings all the user preferences and corresponding values into the memory (Figure 6) for easy access and use during the user's session. In case of failure of logging in (Figure 7) the user is prompted again.

In the sequel a scrollable list of available languages is shown (Figure 8) and once the user selects a language, the application records the selection and moves to the next screen. The available languages are presented in their own language so that they can be spotted easily by the user. As shown in Figure 8 right to left languages (like Arabic) are supported as well. If the application does not have any language information it retrieves the information from the central database using the API. After the user's language selection the application displays everything in the selected language.

The next step for the client User Profile application is to retrieve the user preferences from the server. First, it builds in the background a list of the fields and the type of data for each one, based on the server's response. Later on (during login) the application fills the user's selected values in each one of these fields.

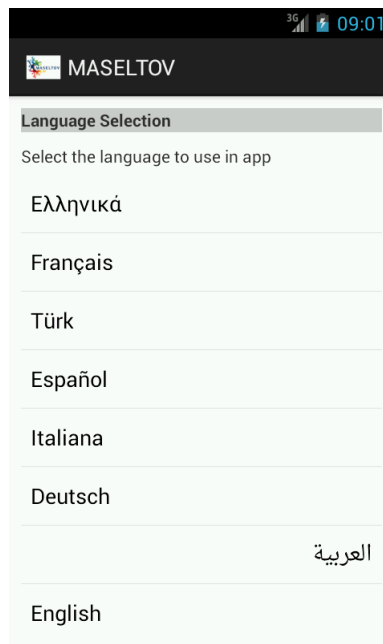


Figure 8: Language Selection Screen.

As soon as the user selects the preferred language the application shows him/her the login form and expects the user to enter an e-mail and password for logging in to his/her account. After a successful login the application returns to the parent activity (i.e., the Mapp dashboard).

For users that do not already have an account, the client application presents them an option to register. The Registration button on the bottom of the login screen brings the user to the AITRegistrationActivity that displays a registration form (Figure 9).

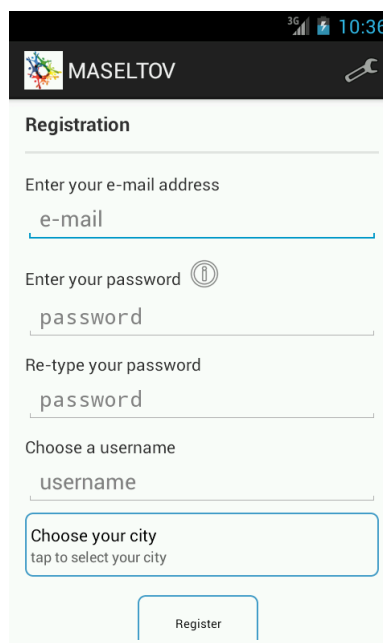


Figure 9: Registration Screen.

For registering, a user enters a valid e-mail address, a password, a username of his/her choice and selects his city. Based on the provided values the User Profile creates a fresh profile entry for the user and records it in the database.

As noted above, anonymous registration is also supported (at the login screen) that allows users to create an account without providing any e-mail address. In this case the user enters his/her username and selects the city (Figure 10). The User Profile then creates a fake e-mail address based on the device id and generates a password for the user that allows him/her to anonymously login into the platform. The drawback of making use of anonymous registration is that the account can only be accessed from the device that is created.

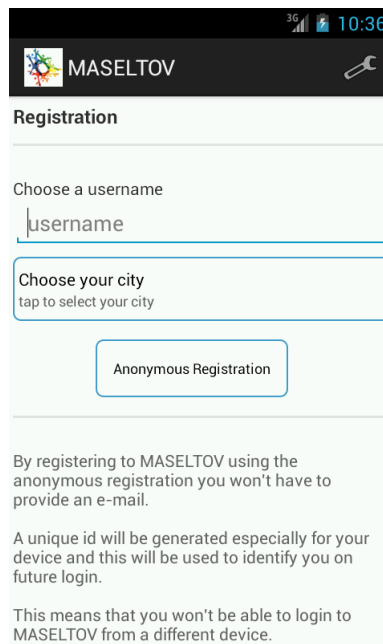


Figure 10: Anonymous Registration Screen.

After a successful login, the application returns to the parent activity (the one that called the AITLoginActivity).

7.2.2 USER PREFERENCES

User preferences can be easily accessed through an icon on the Mapp dashboard.

The preferences that are presented to the user are the ones that have been defined from the Service Provider through the back end server management GUI. An indicative screen that contains the list of all mandatory and optional fields is shown in Figure 11. As is shown in that figure, the username and the user's e-mail are non-editable fields. Moreover, each field shows its own caption while the value set by the user for that field is shown on the following line.

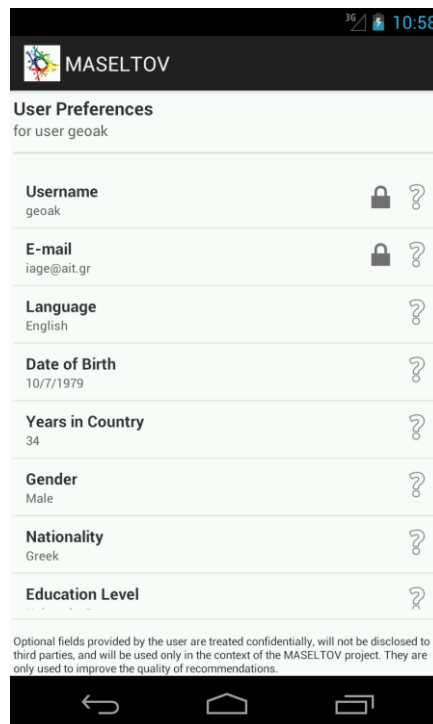


Figure 11: User Preferences Screen (values not entered).

Initially the values of the preferences fields are set to null and the user is presented with a message “*Tap to enter Gender*” or “*Tap to enter Date of birth*”. When the user taps on a list item a popup dialog is shown that requests user input depending on the type that has been declared the field. The different possibilities are:

- Single Selection Dialog (Figure 12)
- Limited Numbers Dialog (Figure 13)
- Numbers Dialog (Figure 14)
- Text Dialog (Figure 15)
- Date Selection Dialog (Figure 16)
- Multi Selection Dialog (Figure 17)
- Informative Dialog (Figure 18)

When a value has been set by the user the field shows the selected value on the second line (Figure 19).

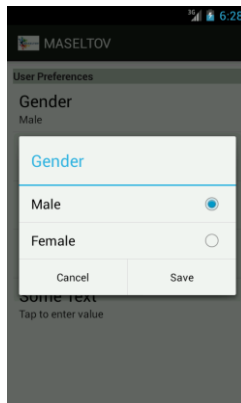


Figure 12: Single Selection Dialog.

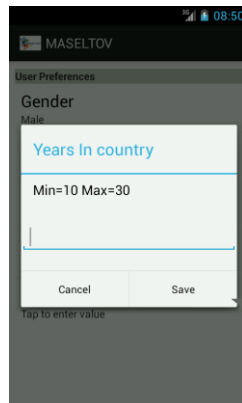


Figure 13: Limited Number Dialog.

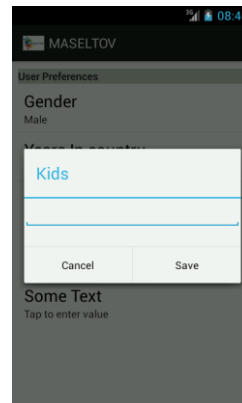


Figure 14: Number Dialog.

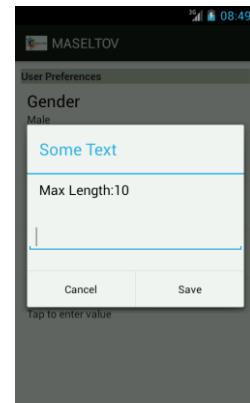


Figure 15: Text Dialog.

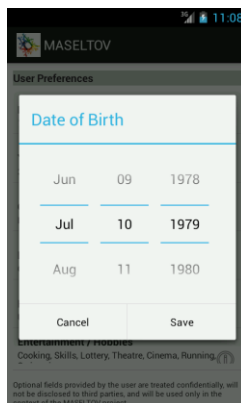


Figure 16: Date Selection Dialog.

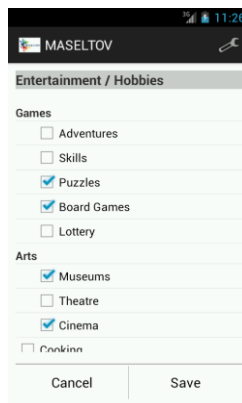


Figure 17: Multi Selection Dialog.



Figure 18: Informative Dialog.

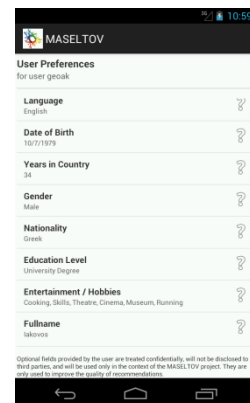


Figure 19: User Preferences Screen (data entered).

As soon as the user selects a new value for a field, this is stored locally on the client User Profile on the smartphone. It is actually transmitted to the backend server when the Preferences screen is gone (this is the case when the activity goes on pause either because the user hits the back button or the phone rings) or the application is forced to close.

Field Information

Each field is accompanied by an information icon that the user can tap to get information about possible usage the platform may make for this field. The following figures (Figure 20 to Figure 23) show a demo information text for the field Date of Birth. The text that is shown is actually the one entered by the Service Provider through the back end management GUI. Different texts for different languages must be provided and the User Profile will choose the appropriate one for the language selected by the user to show. If no text is available for a specific language the info icon is automatically hidden.



Figure 20: Field Information (in English).

Figure 21: Field Information (in Greek).

Figure 22: Field Information (in Arabic).

Figure 23: Field Information (in German).

Read only Fields

Some fields of the user preferences are not allowed to be modified by the user, i.e., they are read-only as far as the user is concerned. The values of these fields may not be allowed to change at all (for example the user's e-mail) or may be allowed to be changed only programmatically (for example the learning level for a language of interest the user wants to pursue may be changed by the recommender once an exam is successfully passed). The Service Provider may specify through the back end management GUI which of fields should be read only for the user. Read only fields are shown with a lock which means that the user can see their value but not change it.

Configuration switches

A number of settings are available for the user to switch on and off specific functionalities, for example the GPS Tracking option can be set to off and the application will stop collecting the user's position (an option that is required for the User.Location Recommendations).

7.3 CLIENT API (CONTENT PROVIDER)

A Content Provider has been implemented within the User Profile to allow other applications to get and send data to the User Profile for the currently logged user, like retrieving preferences, sending events and so on. The following sections give the specifications of the Content Provider.

7.3.1 AITUSERPROFILEPROVIDER

The user provider can be accessed using a content resolver; possible queries are shown in the following subsections.

7.3.1.1 IS USER LOGGED IN?

Purpose	Returns the list of user's id and user's username for the currently logged user
Type	query
URI	content://com.ait.userprofile.AITUserProfileProvider/isUserLoggedIn
Returns	<ol style="list-style-type: none"> 1. A Cursor with one row of 'id', 'username' which represent the user's id, and the user's username for the logged in user 2. <i>null</i> if the user is not logged in; one must call the AITUserProfileProvider as described in 7.1 MApp Activities Interconnection sections earlier in this document

7.3.1.2 GET CURRENT LANGUAGE

Purpose	Returns the user's current language
Type	query
URI	content://com.ait.userprofile.AITUserProfileProvider/currentLanguage
Returns	<ol style="list-style-type: none"> 1. A Cursor with one row of 'lang', which represents the user's selection for language 2. <i>null</i> if the user is not logged in; one must call the AITUserProfileProvider as described in 7.1 MApp Activities Interconnection sections earlier in this document

7.3.1.3 GET ALL FIELDS

Purpose	Returns a list of (id, name, value) tuples of all fields for the currently logged user
Type	query
URI	content://com.ait.userprofile.AITUserProfileProvider/allFields
Returns	<ol style="list-style-type: none"> 1. A Cursor with a list of 'id', 'name', 'value', 'userEditable' which represent <ul style="list-style-type: none"> • 'id': the field's id,

	<ul style="list-style-type: none"> • ‘name’: the field’s name, • ‘value’: user’s selected value. This is either the actual value (for integers and free text) or a JSONObject that always has a ‘value’ which is what you use to display, and an ‘id’ to use if you need to update the user’s selected ‘value’ in his profile, • ‘userEditable’: zero (0) if the field is not user editable and one (1) if the field is user editable. <ol style="list-style-type: none"> 2. <i>null</i> if the user is not logged in; you must call the AITUserProfileProvider as described in 7.1 MApp Activities Interconnection sections earlier in this document 3. <i>IllegalArgumentException</i> if the URI is not valid
--	---

7.3.1.4 GET ONE FIELD BY ID

Purpose	Returns the id, name, and current user’s value for a specific field given the field id
Type	query
URI	content://com.ait.userprofile.AITUserProfileProvider/field/#
Returns	<ol style="list-style-type: none"> 1. A Cursor with a single row of ‘id’, ‘name’, ‘value’, ‘userEditable’ which represent: <ul style="list-style-type: none"> • ‘id’: the field’s id, • ‘name’: the field’s name, • ‘value’: user’s selected value. This is either the actual value (for integers and free text) or a JSONObject that always have a ‘value’ which is what you use to display, and an ‘id’ to use if you need to update the user’s selected ‘value’ in his profile, • ‘userEditable’: zero (0) if the field is not user editable and one (1) if the field is user editable. 2. <i>null</i> if the user is not logged in; the AITUserProfileProvider must be called as described in 7.1 MApp Activities Interconnection sections earlier in this document 3. <i>NullPointerException</i> if the given field id in the URI is not matched with a field in user preferences 4. <i>IllegalArgumentException</i> if the URI is not valid

7.3.1.5 SEND AN EVENT

Purpose	Sends a new event
---------	-------------------

Type	insert
URI	content://com.ait.userprofile.AITUserProfileProvider/insert/event
Returns	content://com.ait.userprofile.AITUserProfileProvider/insert/event/# where # is the number of the row in the temporary database stored in device

7.3.1.6 UPDATE FIELDS

Purpose	Updates the value of a field within the user's preferences. This is only available to those fields that are marked by the administrator as non-user editable fields.
Type	update
URI	content://com.ait.userprofile.AITUserProfileProvider/update/field/#/*
Returns	either zero (0) for no changes, or one (1) if the changes were successful

Table 2 below shows the format of the returned data for each of the supported data types.

Table 2: Response Data for Update Fields request.

Text	String as entered by the user
Number or Limited Number	Integer as entered by the user
Date	A number in YYYYMMDD format (for DATE) or YYYYMMDDHHMMSS format (for DATETIME)
Enumeration	String as displayed in the selection dialog
Multi-level Boolean Enumeration	String of a JSONArray, e.g., [{"id": "12", "value": "Swimming"}, {"id": "8", "value": "Cooking", "Cinema"}] or

	[{"value": "23"}]
Multi-level Integer Enumeration	String of a JSONArray, e.g., [{"id": "12", "value": "Reading", "rate": "2"}, {"id": "8", "value": "Writing", "rate": "4"}]

It should be noted that *value* in the table above must be URL encoded using `URLEncoder.encode(value, "UTF-8")`.

7.3.2 CONTENT RESOLVER

An example of a content resolver code for any application that would like to access the user's preferences through the `AITUserProfileProvider` is shown in the following subsections.

7.3.3 MANIFEST.XML

The following *users-permission* tag must be added in the `manifest.xml` file inside the *application* tag.

```
<uses-permission android:name="com.ait.userprofile.AITUserProfileProvider"
/>
```

7.3.4 CODE EXAMPLES

This section gives some code examples that demonstrate the User Profile interfaces and the way they can be used from other Mapp applications.

```
//check if the user is logged in

Uri uri =
Uri.parse("content://com.ait.userprofile.AITUserProfileProvider/isUserLoggedIn");

String[] projection = new String[] {"id", "username"};

Cursor userPreferences = getContentResolver().query(uri, projection, null, null, null);

if (userPreferences!=null){

    if (userPreferences.getCount()>0){

        int indexId = userPreferences.getColumnIndex("id");

        int indexName = userPreferences.getColumnIndex("username");

        userPreferences.moveToFirst();

        do {    //actually there is only one row in cursor

            //you can get now and use the user's id and user's username
```



```
String id = userPreferences.getString(indexId);

String name = userPreferences.getString(indexName);

} while (userPreferences.moveToNext());

}

}

else {

    //user is not logged in; do what you have to do

}

//get the user's current application language

Uri uri =
Uri.parse("content://com.ait.userprofile.AITUserProfileProvider/currentLanguage");

String[] projection = new String[] {"lang"};

Cursor userPreferences = getContentResolver().query(uri, projection, null, null, null);

if (userPreferences!=null){

    if (userPreferences.getCount()>0){

        int indexLanguage = userPreferences.getColumnIndex("lang");

        userPreferences.moveToFirst();

        do { //actually there is only one row in cursor

            //you can get now and use the user's language selection

            String language = userPreferences.getString(indexLanguage);

        } while (userPreferences.moveToNext());

    }

}

else {

    //user is not logged in; do what you have to do

}

//get all fields in available in user profile

Uri uri = Uri.parse("content:// com.ait.userprofile.AITUserProfileProvider/allFields");

//or get the values and options of a specific field in the user profile

Uri uri = Uri.parse("content://com.ait.userprofile.AITUserProfileProvider/field/" +
```

```
fieldId.getText());

//after either of these parse you may query the Content Provider

String[] projection    = new String[] { "id", "name", "value" };

try{

    Cursor userPreferences = getContentResolver().query(uri, projection, null, null,
null);

    if (userPreferences!=null){

        if (userPreferences.getCount()>0){

            int indexId = userPreferences.getColumnIndex("id");

            int indexName = userPreferences.getColumnIndex("name");

            int indexValue = userPreferences.getColumnIndex("value");

            userPreferences.moveToFirst();

            do {

                //variables that contain the data

                //retrieved and can be used for your purpose

                //id, name, value

                String id    = userPreferences.getString(indexId);

                String name = userPreferences.getString(indexName);

                String value = userPreferences.getString(indexValue);

            } while (userPreferences.moveToNext());

        }

    }

    else {

        //user is not logged in; do what you have to do

    }

}

catch (NullPointerException e){

    Log.w(getString(R.string.app_name), e.getMessage());

    Toast.makeText(getApplicationContext(), "Field not found. Try another one!",
```

```
Toast.LENGTH_SHORT).show();

}

catch(IllegalArgumentException e){

    Log.w(getString(R.string.app_name), e.getMessage());

    Toast.makeText(getApplicationContext(), "Content URI is invalid",
        Toast.LENGTH_SHORT).show();

}

//send an event

Uri uri = Uri.parse("content://com.ait.userprofile.AITUserProfileProvider/insert/event");

String[] projection    = new String[] {};

try{

    //create your data to send

    //a source, a timestamp (YYYYMMDDHHMMSS) and the info

    ContentValues values = new ContentValues();

    values.put("source", "Test.Source");

    values.put("timestamp", "20130927163105");

    values.put("info", "test_info");

    getContentResolver().insert(uri, values);

}

catch(IllegalArgumentException e){

    Log.w(getString(R.string.app_name), e.getMessage());

    Toast.makeText(getApplicationContext(), "Content URI is invalid",
        Toast.LENGTH_SHORT).show();

}

//update the value of a field in user preferences

//following uri sets the value 2 to the field with id 6

Uri uri =
Uri.parse("content://com.ait.userprofile.AITUserProfileProvider/update/field/6/2");

int i = getContentResolver().update(uri, new ContentValues(), "", new String[]{});

if (i>0)

{

    //value changed
```

```

}

else

{

    //value failed to change

}

//Another Example:

//following uri sets the value `Mobile Developer` to the filed with id 20 (Current Job)

String value = URLEncoder.encode("Current Job", "utf-8"); //value should be now
Current+Job in order to safely passed to the url

Uri uri =
Uri.parse("content://com.ait.userprofile.AITUserProfileProvider/update/field/20/"+value);

int i = getContentResolver().update(uri, new ContentValues(), "", new String[]{});

if (i>0)

{

    //value changed

}

else

{

    //value failed to change

}

```

7.4 BACK-END MANAGEMENT GUI

The server backend is accessible only to registered and authorized users. Registered users can access the backend from any tested browser, using their e-mail and password for authenticating themselves.

7.4.1 COMPATIBILITY

The functionality of the backend has been tested and verified for the most popular browsers as shown in Table 3.

Table 3: Tested browsers.

Chrome	Firefox	Internet Explorer	Safari	Opera
30 (T)	22.0 (T)	10 (T)	6 (NT)	15.0 (T)
28 (T)	21.0 (T)	9 (NT)	5.1.7 (T)	

27 (T)	8 (T)
T – Tested	NT – Not tested yet
	NW – Not Working

7.4.2 AVAILABLE OPERATIONS

7.4.2.1 LOGIN PAGE

The login page is displayed (Figure 24) when the user is not logged in or the session is reset. To get access to the MApp backend GUI the user must enter a valid e-mail and password.



Figure 24: Login Page.

Passwords must be five (5) to fifteen (15) characters long, and valid characters for password are listed in Table 4.

Table 4: Allowed characters for passwords.

a-z, A-Z	Latin both lower and upper case letters
0-9	Numbers
!	Exclamation mark
@	At sign
#	Hash sign
\$	Dollar sign
%	Percentage sign
^	Carret sign
.	Dot

7.4.2.2 DASHBOARD

After a successful log in the system presents to the user the dashboard screen (Figure 25). Currently the dashboard displays a frameset with user information (username, full name, last login data and ip), and a frameset with a link to personal data fields list.

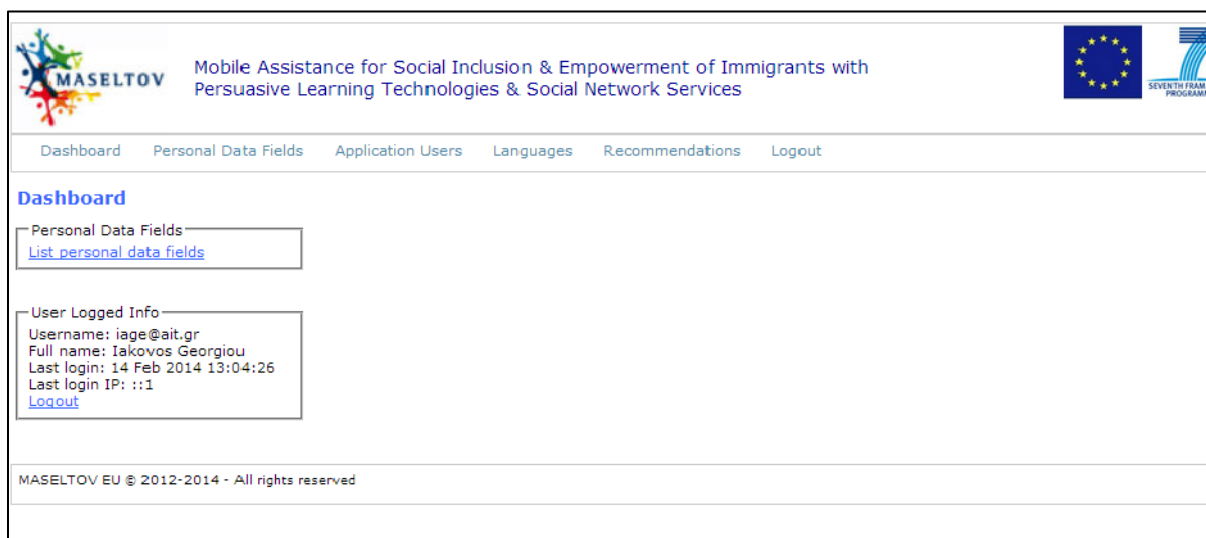


Figure 25: Dashboard Screen

7.4.2.3 MESSAGES

The backend application produces messages to inform the user for:

- i. Successful outcome of operations
- ii. Failed operations
- iii. Information about user actions

These messages are displayed on the top of the browser's document area in a light yellow background balloon (examples shown at Figure 26).



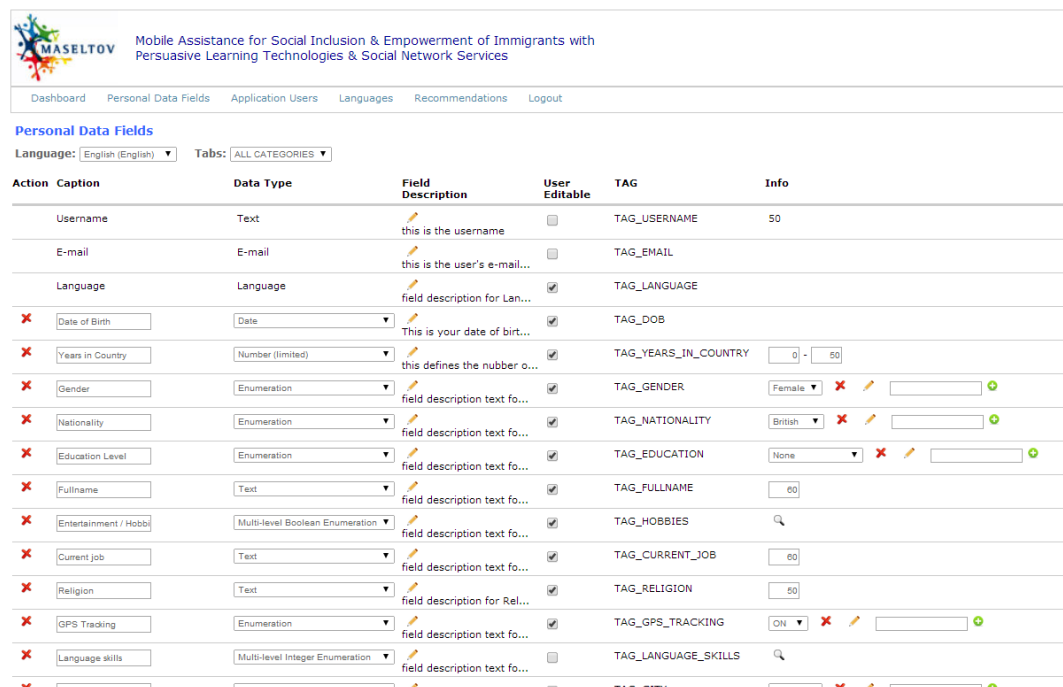
Figure 26: User Profile messages.

7.4.3 PERSONAL DATA FIELDS

Personal Data Fields can be edited (and possibly deleted) by the Service Provider through the back end Management GUI, by choosing the option ‘*Personal Data Fields*’ (Figure 27). In the displayed list of fields, the mandatory ones cannot be edited; otherwise typical editing functionalities are available for non-mandatory fields.

The various fields of the user preferences structure can be displayed in a number of supported languages. The structure itself of the user preferences fields (i.e., number of fields and their data types) remains the same in all languages.

By selecting a language from the drop down selection the captions of the fields are automatically changed to the ones for the selected language. When a new field is added through the back end administrative GUI its caption has to be provided in all supported languages. This is done in a stepwise manner by selecting each language in sequence and supplying the field’s caption in that language. The end result that is presented to the user is a caption to the language of their preference, for example the caption ‘Date of Birth’ will be displayed appropriately in the language the user has selected.



Action	Caption	Data Type	Field Description	User Editable	TAG	Info
	Username	Text	this is the username	<input type="checkbox"/>	TAG_USERNAME	50
	E-mail	E-mail	this is the user's e-mail...	<input type="checkbox"/>	TAG_EMAIL	
	Language	Language	field description for Lan...	<input checked="" type="checkbox"/>	TAG_LANGUAGE	
X	Date of Birth	Date	This is your date of birth...	<input checked="" type="checkbox"/>	TAG_DOB	
X	Years in Country	Number (limited)	this defines the nubber o...	<input checked="" type="checkbox"/>	TAG_YEARS_IN_COUNTRY	0 - 50
X	Gender	Enumeration	field description text fo...	<input checked="" type="checkbox"/>	TAG_GENDER	Female X
X	Nationality	Enumeration	field description text fo...	<input checked="" type="checkbox"/>	TAG_NATIONALITY	British X
X	Education Level	Enumeration	field description text fo...	<input checked="" type="checkbox"/>	TAG_EDUCATION	None X
X	Fullname	Text	field description text fo...	<input checked="" type="checkbox"/>	TAG_FULLNAME	60
X	Entertainment / Hobbies	Multi-level Boolean Enumeration	field description text fo...	<input checked="" type="checkbox"/>	TAG_HOBBIES	
X	Current job	Text	field description text fo...	<input checked="" type="checkbox"/>	TAG_CURRENT_JOB	60
X	Religion	Text	field description for Rel...	<input checked="" type="checkbox"/>	TAG_RELIGION	50
X	GPS Tracking	Enumeration	field description text fo...	<input checked="" type="checkbox"/>	TAG_GPS_TRACKING	ON X
X	Language skills	Multi-level Integer Enumeration	field description text fo...	<input type="checkbox"/>	TAG_LANGUAGE_SKILLS	
X				<input type="checkbox"/>	TAG_CITY	

Figure 27: Personal Data Fields.

7.4.3.1 EDITING AND DELETING FIELDS

The user preferences structure GUI allows for a number of fields editing functionalities. A field can be deleted by pressing the red X of the row that corresponds to it. Alternatively, its name can be edited to one that reflects its intended use by typing directly to the field name box (Figure 28).

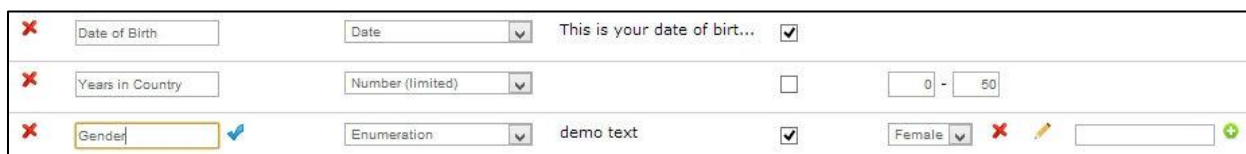


Figure 28: Edit field's caption.

In addition to the field's name, the type of the field can be specified by selecting of the supported types from the drop down list. The available data types are shown in Table 5.

Table 5: Available data types for fields.

Date	For fields of type <i>Date</i>
E-mail	for fields of type <i>e-mail</i>
Number	For fields of type number
Number (limited)	For fields of type number in specific range (lower and upper value applies)
Text	For fields of type text
Enumeration	For fields of type enumeration (values have to be specified)
Multi-level Boolean Enumeration	For fields of type enumeration with two values, each one having two states: checked and unchecked
Multi-level Integer Enumeration	For fields of type enumeration with an integer value, each one having an integer as its value

According to the selected data type the forth column changes to display the appropriate information for the specific data type. Details for the various data types are given in the sequel.

Date

A field of type *date* does not require any additional information (Figure 29)



Figure 29: The date data type.

E-mail

A field of type *e-mail* does not require any additional information (Figure 30)



Figure 30: The E-mail data type.

Number

A field of type *number* does not require any additional information (Figure 31)



Figure 31: The number data type.

Number (limited)

A field of type *limited number* requires two numeric values, namely the lower bound and the upper bound of the range (Figure 32).



Figure 32: The limited number data type.

Text

A field of type *text* requires one numeric value (greater than or equal to one) that specifies the number of allowed characters the text may have (Figure 33).



Figure 33: The text data type.

Enumeration

A field of type *enumeration* requires the list of values of the enumeration. The Management GUI allows editing of the values of the enumeration, which are displayed in a drop down list. A value can be removed by selecting it from the drop down list and pressing the red X symbol. Moreover, a new value can be introduced by typing it in the provided text box and pressing the green + symbol next to it (Figure 34).

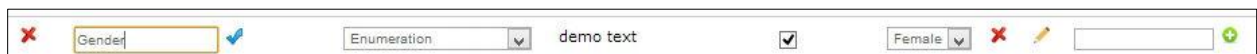


Figure 34: The enumeration data type.

Multi-level Boolean Enumeration and Multi-level Integer Enumeration

A field of this type contains a number of options categorized in a tree view up to two levels (Figure 35 and Figure 36). By clicking the magnifier icon a new window is shown that displays the current values of the field. By clicking the plus (+) and minus (-) signs the list expands and collapses respectively (Figure 37).



Figure 35: The multi-level Boolean enumeration type.



Figure 36: The multi-level integer enumeration type.

A green plus icon allows the user to add a new option under the option the icon resides. A prompt window appears for entering the name of the new option. If the option does not exist then the list is refreshed and shows the new option in place.

A light yellow pencil allows editing the name of an option by supplying the new name in a prompt window. Again, if the option's name does not exist in the tree the list refreshes automatically.

Finally a red X can be used to delete options provided that have no other subheadings under them. By clicking the delete icon the option is removed and the list refreshes once again.

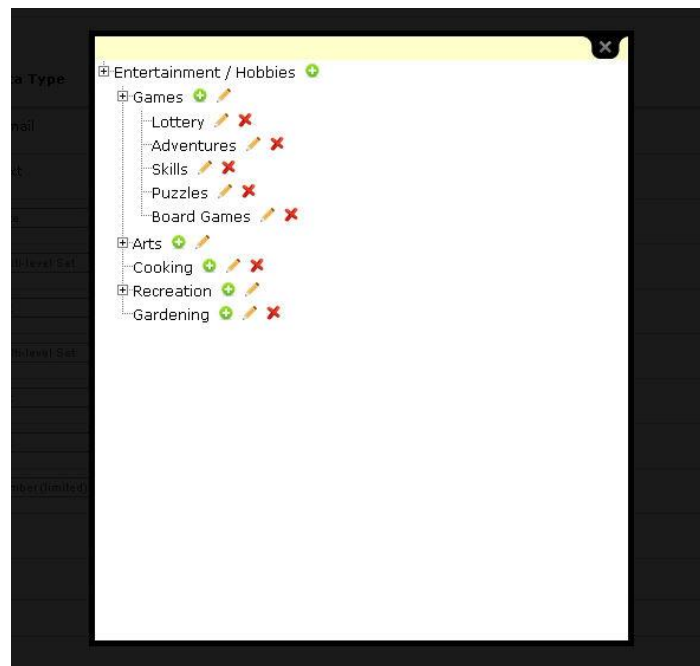


Figure 37: All options available for the field in a tree view.

At the top of the editing window, a yellowish strip shows messages for the outcome of the requested operations; such as error/success messages (Figure 37).



Figure 38: Error / Success Messages.

7.4.3.2 ADDING A NEW FIELD

A new field may be added by clicking on the green plus icon on the left bottom of the list; in this case a new line is inserted. The caption of the newly added field reads *New field* (Figure 39), its data type is preselected to be *text* and the default value of maximum characters is set to 50. The caption must be edited before the system allows addition of another field.

	<input type="text" value="New field"/>	<input type="text" value="Text"/>	<input checked="" type="checkbox"/>	<input type="text" value="50"/>
				

Figure 39: Adding a new field.

7.4.4 USER EDITABLE FIELDS

The User Editable column determines if the field is a read only field or not. When clicked, the specific field can be edited by the user, i.e., the user can set a value for the field. If unchecked, the user can only see the value of the field but cannot change it.

7.4.4.1 FIELD DESCRIPTION

The field description column shows the information text that explains to the user the purpose of the field and how the MASELTOV platform uses the provided information, in an attempt to let the user understand why he should provide the requested information and as a result

increase the level of trust of the user to the platform. The description is language specific, so the admin must provide a description for each field and each language.

7.5 BACK-END APPLICATIONS PROGRAMMING INTERFACE (API)

The server User Profile component provides an Application Programming Interface (API) to allow the mobile application communicate with the back end and the central database. A technical description for the available requests is shown in the sequel.

Figure 40 shows the API's flow.

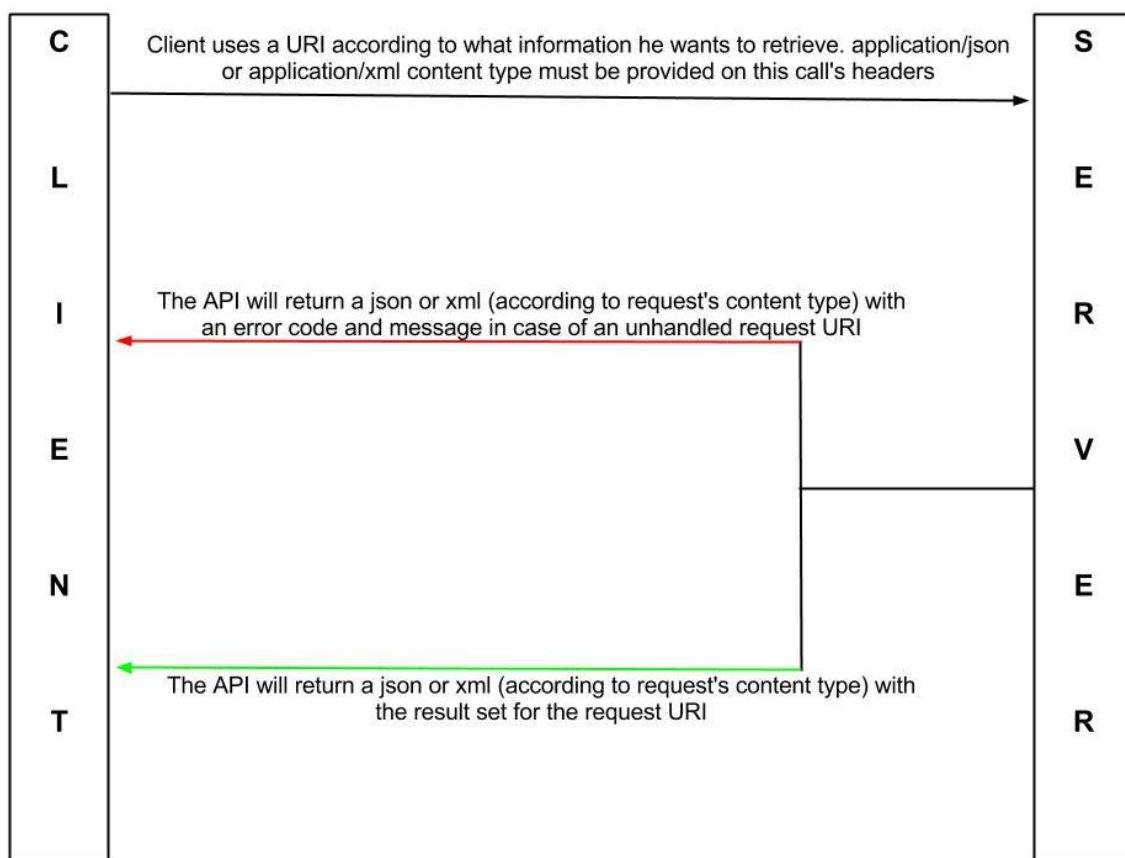


Figure 40: API Flow.

7.5.1 DATA FORMAT

The API accepts requests in both XML and JSON formats and recognizes the request by the detected content type. Content type *application/xml* is used for XML requests and *application/json* for JSON requests. The API responds in the same content type for each response.

7.5.2 USER PROFILE WEB SERVICE

The back end User Profile API accepts requests at the public URI <http://maseltov.ait.gr/maseltov/wservice>. The following table summarizes the available resources.

Table 6: Web Service Resources.

Resource	Description
User Data Collection	Collects all user information Media types: application/xml, application/json
Create New User	Attempts to create a new user. Media types: application/xml, application/json
Update User's Preferences	Attempts to update the user's preferences. Media types: application/xml, application/json
User Profile Fields Collection	Contains the available fields currently available in the user profile Media types: application/xml, application/json
User Profile Fields	This resource contains the data structure of the User Profile specific field Media types: application/xml, application/json
Languages	Returns a list of available languages Media types: application/xml, application/json
Event	Adds an event into the database as received from the device Media types: application/xml, application/json

The following sections give the details of the available resources.

7.5.2.1 USER DATA COLLECTION

URI	Description
/wservice/user/{encrypted_user_email}/{encrypted_user_password}	If user successfully authenticated it returns all the available data for this user.

Following the *user* path, two additional elements have to be included, namely the user's email (encrypted) and the user's password (hashed). The user's e-mail is encrypted using a two-way AES encryption with a fixed key. The password is one-way hashed using MD5.

The response comes in a single *user* element that contains a number of subelements. The response always contains an *id* element that can be either *null* or a number.

If the *id* is *null* then the server could not find the user with the requested credentials. In this case a *reason* element is also included in the response that contains a special tag for the reason the user could not be found. *WRONG_CREDENTIALS* is currently the only reason that the API detects.

If the *id* is a number then this is the user's unique id; in this case the following elements will be also present in the response

email: the user's e-mail

username: the user's username

preferences: is a set of (id, value) pairs. The id is the id of the user preferences (it can be found using the *upfields/short* or *upfields/long* and can be used in *upfield* and *userup* calls). The value field holds either a value (i.e., a date) or an id of the value (for enumeration or multi-level enumeration data types each option has an id, that is mentioned in *upfields/long*). [an upfield is an update field of the update query the device calls when the user makes a change into his user preferences].

Examples (XML)
<pre> <result> <user> <id> </id> < reason>WRONG_CREDENTIALS </reason> </user> </result> </pre>
<pre> <result> <user> <id>12</id> <email>myname@server.com</email> <username>myusername</username> <preferences> <id>4</id> <value>19790710</value> </preferences> <preferences> <id>6</id> <value>3</value> </preferences> </user> </result> </pre>
DTD
<pre> <!ELEMENT result (user)> <!ELEMENT user (id, username?, email?, preferences?, reason?)> <!ELEMENT id (#PCDATA)> <!ELEMENT username (#PCDATA)> <!ELEMENT email (#PCDATA)> </pre>

```
<!ELEMENT preferences (id, value)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT value (#PCDATA)>

<!ELEMENT reason (#PCDATA)>
```

XML Schema

```
<?xml version="1.0"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="result">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="user" >
        <xs:complexType>
          <xs:sequence>
            <xs:element name="id" type="xs:integer" use="required"/>
            <xs:element name="username" type="xs:string"/>
            <xs:element name="email" type="xs:string"/>
            <xs:element name="preferences">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="id" type="xs:integer"/>
                  <xs:element name="value" type="xs:string"/>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

Examples (JSON)

```
{
  "result": {
    "user": {
      "id": null,
      "reason": "WRONG_CREDENTIALS"
    }
  }
}
```

```
{
  "result": {
    "user": {
      "id": 12,
      "email": "myname@server.com",
      "username": "myusername",
      "preferences": [
```

<pre> { { { "id":4, "value": "19790710" }, { "id":6, "value": "3" } } } </pre>
JSON-Schema
<pre> { "result":{ "user":{ "id":"number", "email":"string", "username":"string", "preferences":[{ "id":"number", "value":"string" }] } } } </pre>

7.5.2.2 CREATE NEW USER

URI	Description
/wservice/newuser/{encrypted_user_email}/{encrypted_user_password}/{nickname}	Returns all available fields of a user profile.

Following the *newuser* path, three additional elements have to be included; the user's email (using a two-way encryption; so it can be decrypted and stored into the central database) the user's password hashed using MD5, and the preferred username.

The response comes in a single *user* element that contains a number of subelements. The response always contains an *id* element that can be either *null* or a number.

If the *id* is *null* then the API could not create the user with the requested credentials. In this case a *reason* element is also present that would have a special tag for the reason the user could not be found. Table 7 contains all the supported tags.

Table 7: New User Response Tags.

TAG	Description
-----	-------------

EMAIL_EXISTS	There is another user in the database using this e-mail address
USERNAME_EXISTS	There is another user in the database using this username
CREATE_USER_FAILED	Any other reason

If the *id* is a number then this is the new user's unique id and the following elements will be also present:

- *email*: the user's e-mail
- *username*: the user's username

Examples (XML)
<pre><result> <user> <id> </id> < reason>NICKNAME_EXISTS </reason> </user> </result></pre>
<pre><result> <user> <id>20</id> <email>myname@server.com</email> <username>myusername</username> </user> </result></pre>
DTD
<pre><!ELEMENT result (user)> <!ELEMENT user (id, username?, email?, reason?)> <!ELEMENT id (#PCDATA)> <!ELEMENT username (#PCDATA)> <!ELEMENT email (#PCDATA)> <!ELEMENT reason (#PCDATA)></pre>
XML Schema
<pre><?xml version="1.0"?> <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"> <xs:element name="result"> <xs:complexType> <xs:sequence> <xs:element name="user" > <xs:complexType> <xs:sequence> <xs:element name="id" type="xs:integer" use="required"/> <xs:element name="username" type="xs:string"/></pre>

```
<xs:element name="email" type="xs:string"/>
<xs:element name="reason" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

Examples (JSON)

```
{
  "result": {
    "user": {
      "id": null,
      "reason": "EMAIL_EXISTS"
    }
  }
}
```

```
{
  "result": {
    "user": {
      "id": 20,
      "email": "myname@server.com",
      "username": "myusername"
    }
  }
}
```

JSON-Schema

```
{
  "result": {
    "user": {
      "id": [null, "number"],
      "email": "string",
      "username": "string"
    },
    "required": [id]
  }
}
```

7.5.2.3 UPDATE USER'S PREFERENCES

URI	Description
/wservice/userup/{encrypted_user_email}/{encrypted_user_password}?{list_of_fields_to_update}	Updates one or more of the user's preferences selected values.

The URL must contain after the *user* path three additional elements; the user's email (encrypted), the user's password hashed using MD5, and a number of query parameters starting with a question mark (?). The fields are separated with ampersands (&). Each value is defined like *id{field_id}={value}* (e.g., *id12=8* which denotes the field with id 12 to be set with value of options with id 8 which could be equal to Female).

The response comes in a single *user* element that contains a single element *id* that should be the user's id.

If the user name and password are not correct for the user the id is *null* and a *reason* element is also present that would have a special tag for the reason the user could not be found. Table 8 contains the possible reason tags currently supported.

Table 8: Update User's Preference Response Tag.

TAG	Description
<i>WRONG_CREDENTIALS</i>	User name and password did not match a user in database

Examples (XML)

```
<result>
  <user>
    <id> </id>
    < reason> WRONG_CREDENTIALS </reason>
  </user>
</result>
```

```
<result>
  <user>
    <id>20</id>
  </user>
</result>
```

DTD

```
<!ELEMENT result (user)>
<!ELEMENT user (id, reason?)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT reason (#PCDATA)>
```

XML Schema

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="result">
    <xs:complexType>
      <xs:sequence>
```

```
<xs:element name="user" >
  <xs:complexType>
    <xs:sequence>
      <xs:element name="id" type="xs:integer" use="required"/>
      <xs:element name="reason" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>

</xs:schema>
```

Examples (JSON)

```
{
  "result": {
    "user": {
      "id": null,
      "reason": " WRONG_CREDENTIALS "
    }
  }
}
```

```
{
  "result": {
    "user": {
      "id": 20,
    }
  }
}
```

JSON-Schema

```
{
  "result": {
    "user": {
      "id": [null, "number"],
      "reason": "string",
      "required": ["id"]
    }
  }
}
```

7.5.2.4 USER PROFILE FIELDS COLLECTION

URI	Description
/wservice/upfields/{language_code}/short or /wservice/upfields/{language_code}/long	This URI returns all the available fields a user profile may have using GET. The name of the fields and their options are delivered in the requested language (by the language code, i.e., en, el, de, etc.).

The response comes in a single *fields* element that contains a number of *field* elements.

Each *field* element contains the attribute *id* which is the unique id of the field.

Each *field* element contains a *caption* element which holds the caption of the field.

The *short* version contains only id and caption while the *long* version contains all data from the fields.

Examples (XML)

```
<result>
  <fields>
    <field>
      <id>1</id>
      <caption>Nickname</caption>
    </field>
    <field>
      <id>2</id>
      <caption>E-mail</caption>
    </field>
    <field>
      <id>3</id>
      <caption>Gender</caption>
    </field>
  </fields>
</result>
```

DTD

```
<!ELEMENT result (fields)>
<!ELEMENT fields (field)>
<!ELEMENT field (id, caption)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT caption (#PCDATA)>
```

XML Schema

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="result">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="fields" >
          <xs:complexType>
            <xs:sequence>
              <xs:element name="id" type="xs:integer" use="required"/>
              <xs:element name="caption" type="xs:string"
use="required"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
```

```
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

Examples (JSON)

```
{
  "result": {
    "fields": {
      "field": [
        {
          "id": 1,
          "caption": "Nickname"
        },
        {
          "id": 2,
          "caption": "E-mail"
        },
        {
          "id": 3,
          "caption": "Gender"
        }
      ]
    }
  }
}
```

JSON-Schema

```
{
  "result": {
    "fields": [
      {
        "id": "number",
        "caption": "string"
      },
      {
        "id": "caption",
        "caption": "string"
      }
    ],
    "required": ["id", "caption"]
  }
}
```

Example for *long* version (JSON)

```
{
  "result": {
    "fields": {
      "field": [
        {
          "id": 1,
          "caption": "Nickname",
          "mandatory": "1",
          "userEditable": 0,
          "type": "text",
          "description": "text describing how the field's data
are used",
          "info": {
```

```

        "type": "text",
        "value": "50"
    },
    {
        "id": 2,
        "caption": "E-mail",
        "mandatory": "1",
        "userEditable": 0,
        "type": "email",
        "description": "text describing how the field's data
are used",
        "info": {
            "type": "email",
            "value": ""
        }
    },
    {
        "id": 4,
        "caption": "Date of Birth",
        "mandatory": "0",
        "userEditable": 1,
        "type": "date",
        "description": "text describing how the field's data
are used",
        "info": {
            "type": "date",
            "value": ""
        }
    },
    {
        "id": 5,
        "caption": "Years in Country",
        "mandatory": "0",
        "userEditable": 1,
        "type": "limit",
        "description": "text describing how the field's data
are used",
        "info": {
            "type": "limit",
            "lower": 0,
            "upper": 50
        }
    },
    {
        "id": 6,
        "caption": "Gender",
        "mandatory": "0",
        "description": "text describing how the field's data
are used",
        "type": "enum",
        "options": [
            {
                "id": 2,
                "name": "Female"
            },
            {
                "id": 18,
                "name": "Male"
            }
        ]
    },
    {
        "id": 7,
        "caption": "Nationality",

```

```

        "mandatory": "0",
        "userEditable": 1,
        "type": "enum",
        "description": "text describing how the field's data
are used",
        "options": [
            {
                "id": 3,
                "name": "British"
            },
            {
                "id": 4,
                "name": "German"
            },
            {
                "id": 5,
                "name": "Greek"
            },
            {
                "id": 6,
                "name": "Italian"
            }
        ]
    },
    {
        "id": 8,
        "caption": "Education",
        "mandatory": "0",
        "userEditable": 1,
        "type": "enum",
        "description": "text describing how the field's data
are used",
        "options": [
            {
                "id": 7,
                "name": "None"
            },
            {
                "id": 8,
                "name": "Primary School"
            },
            {
                "id": 9,
                "name": "University Degree"
            },
            {
                "id": 10,
                "name": "High School"
            }
        ]
    },
    {
        "id": 17,
        "caption": "Entertainment / Hobbies",
        "mandatory": "0",
        "userEditable": 1,
        "type": "mset",
        "description": "text describing how the field's data
are used",
        "options": [
            {
                "name": "Cooking",
                "children": null,
                "id": 14
            },
        ],
    },

```



```

        {
            "name": "Games",
            "children": [
                {
                    "name": "Adventure",
                    "children": null,
                    "id": 19
                },
                {
                    "name": "Skills",
                    "children": null,
                    "id": 20
                },
                {
                    "name": "Lottery",
                    "children": null,
                    "id": 21
                }
            ],
            "id": 15
        },
        {
            "name": "Arts",
            "children": [
                {
                    "name": "Theatre",
                    "children": null,
                    "id": 22
                },
                {
                    "name": "Cinema",
                    "children": null,
                    "id": 23
                },
                {
                    "name": "Music",
                    "children": null,
                    "id": 24
                }
            ],
            "id": 16
        },
        {
            "name": "Recreation",
            "children": [
                {
                    "name": "Cycling",
                    "children": null,
                    "id": 26
                },
                {
                    "name": "Running",
                    "children": null,
                    "id": 27
                },
                {
                    "name": "Swimming",
                    "children": null,
                    "id": 28
                }
            ],
            "id": 25
        }
    ],
    },

```

```
{
  "id": 19,
  "caption": "Fullname",
  "mandatory": "0",
  "userEditable": 1,
  "type": "text",
  "description": "text describing how the field's data
are used",
  "info": {
    "type": "text",
    "value": "60"
  }
}
```

JSON-Schema

```
{
  "result": {
    "fields": {
      "field": [
        {
          "id": "number",
          "caption": "string",
          "mandatory": "number",
          "userEditable": "number",
          "type": "string",
          "description": "string",
          "info": {
            "type": "string",
            "value": "number"
          },
          "required": ["type", "value"]
        },
        {
          "id": "string",
          "caption": "string",
          "mandatory": "number",
          "userEditable": "number",
          "type": "string",
          "description": "string",
          "info": {
            "type": "string",
            "value": "number"
          },
          "required": ["type", "value"]
        }
      ],
      "required": ["id", "caption",
"mandatory", "userEditable", "type", "description", "info"]
    }
  }
}
```

7.5.2.5 USER PROFILE FIELDS

URI	Description
/wservice/upfield/{language_code}/{field-id}	Returns the data structure of the requested field (by unique field id) in User Profile using GET.

The response comes in a single *field* element that contains the elements describing the data of the field. Table 9 contains the available types.

Table 9: User Profile Fields Types.

id	Integer (mandatory) The unique id of the field
----	---

caption	String (mandatory) The caption of the field												
datatype	<p>Enumerator (mandatory) Describes the data type of the field Values: text, num, limit, date, email, enum</p> <table> <tr> <td>text</td><td>This is used for text inputs. This data type can have a character limit; by default is set to 50 characters</td></tr> <tr> <td>num</td><td>This is used for numeric (integer) inputs.</td></tr> <tr> <td>limit</td><td>This is used for numeric (integer) inputs with a set lower and upper limit for its values. By default lower limit is set to zero (0) and upper limit is set to 50.</td></tr> <tr> <td>date</td><td>This is used for date inputs. Date must be provided in YYYYMMDD format.</td></tr> <tr> <td>email</td><td>This is used for e-mail address inputs.</td></tr> <tr> <td>enum</td><td>This is used for a set of options.</td></tr> </table>	text	This is used for text inputs. This data type can have a character limit; by default is set to 50 characters	num	This is used for numeric (integer) inputs.	limit	This is used for numeric (integer) inputs with a set lower and upper limit for its values. By default lower limit is set to zero (0) and upper limit is set to 50.	date	This is used for date inputs. Date must be provided in YYYYMMDD format.	email	This is used for e-mail address inputs.	enum	This is used for a set of options.
text	This is used for text inputs. This data type can have a character limit; by default is set to 50 characters												
num	This is used for numeric (integer) inputs.												
limit	This is used for numeric (integer) inputs with a set lower and upper limit for its values. By default lower limit is set to zero (0) and upper limit is set to 50.												
date	This is used for date inputs. Date must be provided in YYYYMMDD format.												
email	This is used for e-mail address inputs.												
enum	This is used for a set of options.												
maxlength	Integer (optional) The number of characters Appears only to <i>text</i> data types												
mandatory	Boolean (0 or 1) (mandatory) Zero (0) if the field is optional; one (1) if the field is mandatory												
userEditable	Boolean (0 or 1) (mandatory) Zero (0) if the user is not allowed to change the value of this field; one (1) if the user is allowed to change the value of this field												
lowerlimit	Integer (optional) The lower limit of the accepted value Appears only to <i>limit</i> data types												
upperlimit	Integer (optional) The upper limit of the accepted value Appears only to <i>limit</i> data types												
description	String (mandatory) the description entered by the administrator describing the usage of the field, explaining how this data provided by the user are used in order to provide better service to the user.												
options	Is a superset of a list of options available for <i>enum</i> data type fields. Each <i>options</i> element may contain a number of <i>option</i> elements.												

Each *option* element contains the id and a value/text of the option.

Examples (XML)

/wservice/upfield/en/1

```
<result>
  <field>
    <id>1</id>
    <caption>Nickname</caption>
    <datatype>text</datatype>
    <maxlength>50</maxlength>
    <mandatory>1</mandatory>
    <description>text describing how the
filed's data are used</description>
  </field>
</result>
```

/wservice/upfield/en/3

```
<result>
  <field>
    <id>6</id>
    <caption>Gender</caption>
    <datatype>enum</datatype>
    <mandatory>0</mandatory>
    <description>text describing how the
filed's data are used</description>
    <options>
      <id>2</id>
      <name>Female</name>
    </options>
    <options>
      <id>18</id>
      <name>Male</name>
    </options>
  </field>
</result>
```

/wservice/upfield/en/4

```
<result>
  <field>
    <id>4</id>
    <caption>Years in Country</caption>
    <description>text describing how the
field's data are used</description>
    <datatype>limit</datatype>
    <lowerlimit>0</lowerlimit>
    <upperlimit>50</upperlimit>
    <mandatory>0</mandatory>
  </field>
</result>
```

DTD

```
<!ELEMENT result (field)>

<!ELEMENT field (id, caption, datatype, description, lowerlimit?,
upperlimit?, mandatory, options?)>

<!ELEMENT id (#PCDATA)>
<!ELEMENT caption (#PCDATA)>
<!ELEMENT datatype (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT lowerlimit (#PCDATA)>
<!ELEMENT upperlimit (#PCDATA)>
<!ELEMENT mandatory (#PCDATA)>
<!ELEMENT options (id, name)>

<!ELEMENT id (#PCDATA)>
<!ELEMENT name (#PCDATA)>
```

XML Schema

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="result">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="field" >
        <xs:complexType>
          <xs:sequence>
            <xs:element name="id" type="xs:integer" use="required"/>
            <xs:element name="caption" type="xs:string"
use="required"/>
            <xs:element name="datatype" type="xs:string"
use="required"/>
            <xs:element name="description" type="xs:string"
use="required"/>
            <xs:element name="lowerlimit" type="xs:string"/>
            <xs:element name="upperlimit" type="xs:string"/>
            <xs:element name="mandatory" type="xs:integer"
use="required"/>
            <xs:element name="options">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="id" type="xs:integer"/>
                  <xs:element name="name" type="xs:string"/>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```
</xs:schema>
```

Examples (JSON)

```
/wservice/upfield/1
```

```
{
  "result": {
    "field": {
      "id": "1",
      "caption": "Nickname",
      "datatype": "text",
      "maxlength": "50",
      "mandatory": "1"
      "description": " text describing how the field's data are used
    "
    }
  }
}
```

```
/wservice/upfield/3
```

```
{
  "result": {
    "field": {
      "id": 6,
      "caption": "Gender",
      "datatype": "enum",
      "mandatory": 0,
      "description": " text describing how the field's data are used
    "
      "options": [
        {
          "id": 2,
          "name": "Female"
        },
        {
          "id": 18,
          "name": "Male"
        }
      ]
    }
  }
}
```

```
/wservice/upfield/4
```

```
{
  "result": {
    "field": {
      "id": "4",
      "caption": "Years in Country",
      "datatype": "limit",
      "description": " text describing how the field's data
are used "
      "lowerlimit": "0",
      "upperlimit": "50",
      "mandatory": "0"
    }
  }
}
```

}
JSON-Schema
<pre>{ "result":{ "id": "number", "caption": "string", "datatype": "string", "description": "string", "lowerlimit": "number", "upperlimit": "number", "mandatory": "number", "options":[{ "id": "number", "name": "string" }] }, "required": ["id", "caption", "datatype", "description", "mandatory"] }</pre>

7.5.2.6 LANGUAGES

URI	Description
/wservice/langs	This URI returns all the available languages.

The URL must contain only the path *langs*.

The response comes in a single array that contains a number of sets. Each set contains the elements as shown in Table 10.

Table 10: User Profile Languages Elements.

id	Integer The unique id of the language
caption	String The caption of the language in the current language
english_caption	String The caption of the language in English
code	String The code is the ISO 639-1 Code for the language i.e., en for English, de for German, el for Greek
isactive	Boolean (0 or 1) Zero (0) if the language is disabled; one (1) if the language is

	enabled
lang_direction	ENUM The lang_direction can be LTR for Left-to-right or RTL for Right-to-Left Values: LRT, RTL

Examples (XML)

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
  <language>
    <id>6</id>
    <caption>Deutsch</caption>
    <english_caption>German</english_caption>
    <code>de</code>
    <isactive>1</isactive>
    <lang_direction>LTR</lang_direction>
  </language>
  <language>
    <id>1</id>
    <caption>English</caption>
    <english_caption>English</english_caption>
    <code>en</code>
    <isactive>1</isactive>
    <lang_direction>LTR</lang_direction>
  </language>
  <language>
    <id>4</id>
    <caption>Español</caption>
    <english_caption>Spanish</english_caption>
    <code>es</code>
    <isactive>1</isactive>
    <lang_direction>LTR</lang_direction>
  </language>
  <language>
    <id>3</id>
    <caption>Français</caption>
    <english_caption>French</english_caption>
    <code>fr</code>
    <isactive>1</isactive>
    <lang_direction>LTR</lang_direction>
  </language>
  <language>
    <id>5</id>
    <caption>Italiana</caption>
    <english_caption>Italian</english_caption>
    <code>it</code>
    <isactive>1</isactive>
    <lang_direction>LTR</lang_direction>
  </language>
  <language>
    <id>7</id>
```



```

    <caption>Türk</caption>
    <english_caption>Turkish</english_caption>
    <code>tr</code>
    <isactive>1</isactive>
    <lang_direction>LTR</lang_direction>
  </language>
  <language>
    <id>2</id>
    <caption>Ελληνικά</caption>
    <english_caption>Greek</english_caption>
    <code>el</code>
    <isactive>1</isactive>
    <lang_direction>LTR</lang_direction>
  </language>
  <language>
    <id>8</id>
    <caption>بـيـة رـعـل</noitpac/>
    <english_caption>Arabic</english_caption>
    <code>ar</code>
    <isactive>1</isactive>
    <lang_direction>RTL</lang_direction>
  </language>
</result>

```

DTD

```

<!ELEMENT result (language)>

<!ELEMENT language (id, caption, english_caption, code, isactive,
lang_direction)>

<!ELEMENT id (#PCDATA)>
<!ELEMENT caption (#PCDATA)>
<!ELEMENT english_caption (#PCDATA)>
<!ELEMENT code (#PCDATA)>
<!ELEMENT isactive (#PCDATA)>
<!ELEMENT lang_direction (#PCDATA)>

```

XML Schema

```

<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="result">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="language" >
          <xs:complexType>
            <xs:sequence>
              <xs:element name="id" type="xs:integer" use="required"/>
              <xs:element name="caption" type="xs:string"
use="required"/>
              <xs:element name="english_caption" type="xs:string"
use="required"/>
              <xs:element name="code" type="xs:string" use="required"/>

```

```
        <xs:element name="is_active" type="xs:string"
use="required"/>
        <xs:element name="lang_direction" type="xs:string"
use="required"/>
    </xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>

</xs:schema>
```

Examples (JSON)

```
[
  {
    "id": "6",
    "caption": "Deutsch",
    "english_caption": "German",
    "code": "de",
    "isactive": "1",
    "lang_direction": "LTR"
  },
  {
    "id": "1",
    "caption": "English",
    "english_caption": "English",
    "code": "en",
    "isactive": "1",
    "lang_direction": "LTR"
  },
  {
    "id": "4",
    "caption": "Español",
    "english_caption": "Spanish",
    "code": "es",
    "isactive": "1",
    "lang_direction": "LTR"
  },
  {
    "id": "3",
    "caption": "Français",
    "english_caption": "French",
    "code": "fr",
    "isactive": "1",
    "lang_direction": "LTR"
  },
  {
    "id": "5",
    "caption": "Italiana",
    "english_caption": "Italian",
    "code": "it",
    "isactive": "1",
    "lang_direction": "LTR"
  },
  {
    "id": "7",
```

<pre> "caption": "Türk", "english_caption": "Turkish", "code": "tr", "isactive": "1", "lang_direction": "LTR" }, { "id": "2", "caption": "Ελληνικά", "english_caption": "Greek", "code": "el", "isactive": "1", "lang_direction": "LTR" }, { "id": "8", "caption": "اڤرعرع", "english_caption": "Arabic", "code": "ar", "isactive": "1", "lang_direction": "RTL" }] </pre>
<p>JSON-Schema</p> <pre> [{ "id": "number", "caption": "string", "english_caption": "string", "code": "string", "isactive": "number", "lang_direction": "string" }, "required": ["id", "caption", "english_caption", "code", "isactive", "lang_description"]] </pre>

7.5.2.7 EVENT

URI	Description
/wservice/event/{user-email}/{encrypted_user_password}	This URI is used to send an event received from the User Profile in the device.

The URL must contain after event two paths; the first one is the user's email and the second is the user's password encoded using MD5.

The URL must be followed by the GET parameters as shown in Table 11.

Table 11: New Event URL Parameters.

id	Integer
----	---------

	The unique id of the mobile database table id for this event. This will be returned in the JSON or XML of the response for reference
source	String The source identification string for the event
timestamp	String The timestamp of the event in YYYYMMDDHHMMSS format
info	String JSON This contains the info in JSON format

The response comes in a *result* element that contains either ok or error and the *reference* element contains the id passed by the application to the API on URI. This reference id can be used to identify the response.

Examples (XML)

```
<response>
<result>ok</result>
<reference>12</reference>
</response>
```

```
<response>
<result>error</result>
<reference>12</reference>
</response>
```

DTD

```
<!ELEMENT response (result, reference)>

<!ELEMENT result (#PCDATA)>
<!ELEMENT reference (#PCDATA)>
```

XML Schema

```
<?xml version="1.0"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="response">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="result" type="xs:string" use="required"/>
        <xs:element name="reference" type="xs:integer" use="required"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

Examples (JSON)

```
{
  "result": "ok",
  "reference": "12"
}
```

```
{
  "result": "error"
  "reference": "12"
}
```

JSON-Schema

```
{
  "result": "string",
  "reference": "number"
},
"required": ["result", "reference"]
```

7.5.3 ERROR REPORTING

7.5.3.1 ERROR STRUCTURE RESULT

The system will report errors using the following structure. The response might be in JSON or XML format. This depends on the format that the request came into the system.

1. If the request header contained content-type *application/json* then the response will be in JSON
2. If the request header contained content-type *application/xml* then the response will be in XML
3. If the request header contained a non-supporting content type then the response will be in the default content type which is XML

All results from the API contain a top element *result*. In case of an error the *result* element should contain an *error* element.

Examples (XML)

```
<result>
  <error>
    <code>901</code>
    <value>Invalid Content Type</value>
  </error>
</result>
```

DTD

```
<!ELEMENT result (error)>

<!ELEMENT error (code, value)>
<!ELEMENT code (#PCDATA)>
<!ELEMENT value (#PCDATA)>
```

XML Schema

```
<?xml version="1.0"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="result">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="error" >
          <xs:complexType>
            <xs:sequence>
              <xs:element name="code" type="xs:integer" use="required"/>
              <xs:element name="value" type="xs:string" use="required"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Examples (JSON)

```
{
  "result": {
    "error": {
      "code": 101,
      "value": "Invalid action"
    }
  }
}
```

JSON-Schema

```
{
  "result": {
    "code": "number",
    "value": "string"
  },
  "required": ["code", "value"]
}
```

Each error contains a numeric error code in *code* and a text description in *value* element.

A complete error table that contains a list of all the error codes the API may return, along with a description of each error is shown in Table 12.

Table 12: Table of Error Codes and Description.

Code	Description
101	<i>Invalid action</i> The error occurs when the requested URI contains an unknown action

102	<i>No action specified</i> The error occurs when no action is specified at the URI
103	<i>Invalid data for this action</i> The error occurs when the requested URI contains less or more data for the specific action
104	<i>Invalid data format for one or more action parameters</i> The error occurs when for a specific action the parameter passed is in wrong format; i.e., if called <some action>/param1 and <some action> requires the first parameter(param1) to be a numeric.
201	<i>User Profile Field ID is invalid</i> The error occurs when the requested id in the URI is invalid; the user profile field with such an id, does not exist
202	<i>No User Profile Fields exist</i> The error occurs when no User Profile Fields exist.
203	<i>User Profile Field with specific ID does not exist</i> The error occurs when the requested id does not match any User Profile Field
901	<i>Invalid Content Type</i> The request contains a content type header that is not supported
902	<i>Configuration Error</i> A configuration file contains an error or the configuration file does exist
999	<i>Unknown Error</i> All unclassified errors

7.6 USER DATA SECURITY

The client application uses the API in order to retrieve or save data into the main database. For a number of actions the API requires an authentication check. This is done using the appropriate API path. For example to login a user and get the user's preferences one needs to access the URI:

http://maseltov.ait.gr/wservice/user/encoded_user_email/md5pass

where encoded_user_email is the encoded user's username (e-mail) and md5pass is the md5 hash of the user's password.

The user's password is never transmitted in plain text; only the md5 hash that is applied to it is sent. On the other hand to protect the user's account from unauthorized access, the data that is stored for the user's password in the back end database is the md5 result of the md5 hash of the plain text password. In this way the user's password is protected both during its transmission as well as during its storage. So, even if someone gets access to the database he/she won't be able to get plain text password or even the md5 hash of it, since the transmitted md5 is further hashed and stored as encoded scrambled string. Of course if an

adversary manages to tap over the communications between the mobile client application and the server and captures the transmitted md5 of the user's password, then he/she can impersonate the user and get access to their profile. Stronger authentication mechanisms, like Lamport hashes, can certainly be employed but the current prototype implementation of the User Profile does not pursue this direction any further.

As an example the plain text password *qwerty* will never be saved in the device or transmitted in clear through the network. However the md5 hash of *qwerty* which is *d8578edf8458ce6fbc5bb76a58c5ca4* will be stored in the device and will be transmitted to the API. As noted above, this is not actually the string that is stored in the back end database; instead the md5 of it (*ae6c6f4563f63e356e6decffee08520e*) is stored.

Upon successful login the application holds in the handset's memory the following information:

- The user's Id
- The user's email (email)
- The user's password (md5 hashed)
- The user's username
- The user's preferences

These values are stored in a SharedPreferences string using the *MODE_PRIVATE* parameter so that only the client application can access them. This information can be removed from the device by clearing data from the Android settings (Settings→Applications→MASELTOV).

In addition to the user password, the user email (e-mail) is also transmitted encoded (one way symmetric encryption) during login or other authentication API requests. During the new user registration process the application sends the username to the API using a two-way encryption.

Information about a user (for example if he/she is logged in or not), or a user's preferences can be retrieved through a Content Provider which is described in the next section.

7.7 ENGINEERING ISSUES

The current prototype implementation has been developed so as to scale well with increasing number of users. Even though the project trials are designed to involve a few tenths of users, largest user populations should be supported as well.

Therefore, taking into account that a significant number of users may use the platform simultaneously, there is a need to have a backend service that will be able to serve a number of requests and handle appropriate execution (php scripts) in a scalable way. For the purposes of the prototype the backend server used has an XAMPP v1.8.2 environment installed which includes Apache HTTP Server 2.4.4, PHP 5.4.16 and MySQL 5.0.10. Both the web server and the database server are multithreaded, which allows multiple users to access and use the services of the prototype.

7.8 RECOMMENDER

The Recommender was introduced in Figure 2; its objective is to generate targeted and personalized recommendations based on the user context, as it has been captured by a number of sensors and Mapp applications and communicate to the User Profile through events, and the declared preferences that are also stored with the User Profile. Details of the Recommender are given in Deliverables D5.4.1 and D5.4.2 “Recommendation Services”. A prototype implementation of the Recommender is available and is based on the DROOLS rule engine. It integrates smoothly with the User Profile and the rest of the MApp applications.

A Recommendations component is available from the MASELTOV Dashboard (at the client side) where the user may find all the recommendations that the system produced for his profile (Figure 41). The recommendations may be starred for future reference, deleted or folded as read or unread. Where relevant, some recommendations are clickable, providing the user an easy and direct way to move to a certain URI on a browser or to another Mapp component (for example taking the user to a Learning Language Lesson). Thus recommendations become active objects as opposed to mere passive strings presented to the user.

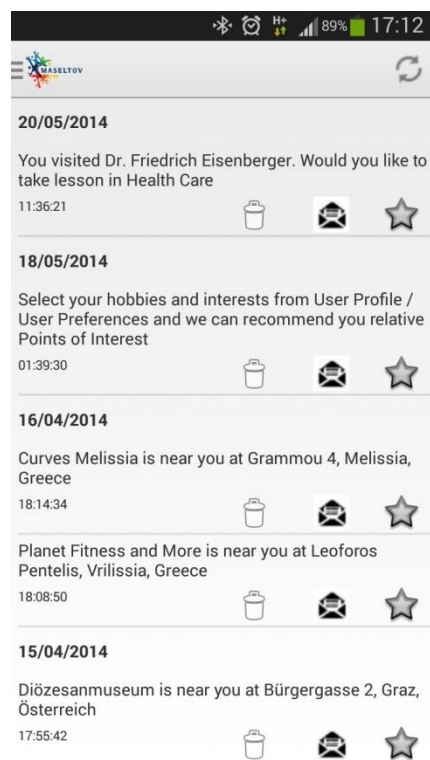


Figure 41: Recommendations Component.

As noted above the back end recommender is based on the DROOLS rule engine, which implements the Rete algorithm. It is driven by a set of rules, the user preferences, and the user context as captured by events that are generated by Mapp applications.

8. DATABASE SCHEMA

Figure 42 depicts an EER (Extended Entity Relation) diagram of the back end Database the User Profile uses. The central tables of the schema are *users* (which holds user related information) and *personaldatafields* (which holds the user preferences data). Table *events* contains information about events that are generated by the various Mapp applications. Details of the database as well as the user's context abstractions are presented in Deliverable D5.1 "Content Abstractions and Databases". The same document gives the details of how preferences fields are communicated to the client User Profile component, how these data and the generated events are associated with a user, and how the user profile data, preferences and events are stored for subsequent querying from the recommender.

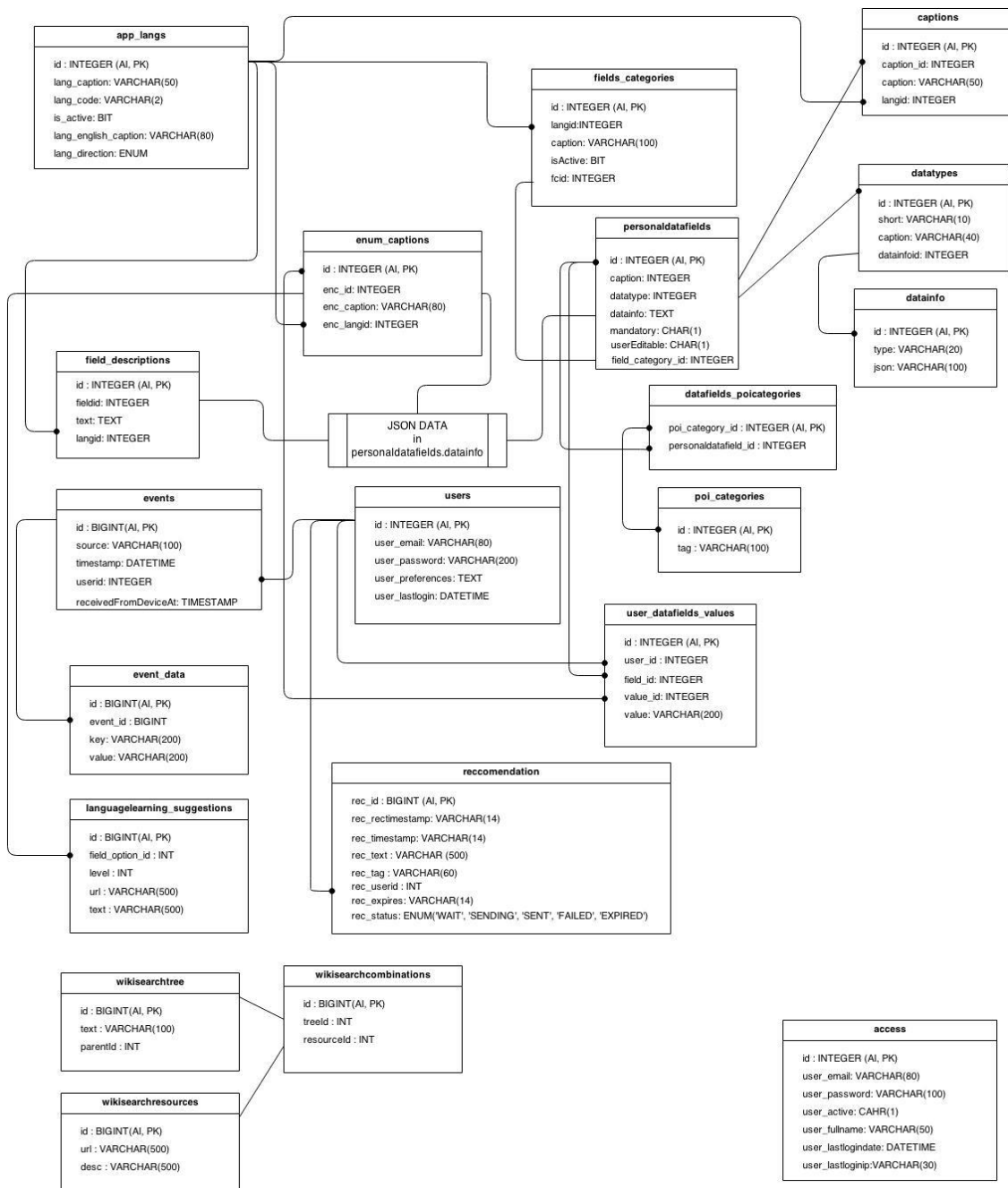


Figure 42: Database schema.