



D5.4 - Updated GOEASY data-oriented enablers and applications

Deliverable ID	D5.4
Deliverable Title	Updated GOEASY data-oriented enablers and applications
Work Package	WP5
Dissemination Level	PUBLIC
Version	1.0
Date	2019-07-25
Status	Final
Lead Editor	CNET
Main Contributors	Peter Rosengren, Leonardo Fringuelli

Published by the GOEASY Consortium

Document History

Version	Date	Author(s)	Description
0.1	2019-04-25	CNET	First Draft with TOC
0.2	2019-05-01	GAPES	Updated Entity – Relationship Diagram for ApesMobility
0.3	2019-05-10	CNET	Updated Entity – Relationship Diagram for AsthmaWatch
0.7	2019-05-27	GAPES	Review and suggestions
0.8	2019-06-01	CNET	Final to be peer reviewed
0.9	2019-06-20	GAPES	Peer review with comments
0.95	2019-06-20	BQ	Peer review with comments
1.0	2019-06-21	CNET	Final version after peer review

Table of Contents

Document History	2
Table of Contents	2
1 Introduction.....	3
1.1 1.1. Scope	3
1.2 1.2. Related documents	3
2 Updated domain data models	4
2.1 2.1 Updated domain data model for AsthmaWatch app.....	4
2.2 2.2 Domain data model for ApesMobility	4
3 Updated data-oriented enablers and applications.....	6
3.1 AsthmaWatch - Updated data-oriented enablers and application	6
3.2 ApesMobility - Updated data-oriented enablers and application	14
4 Conclusions.....	26
Acronyms	27
Appendix GOEASY Database	28

1 Introduction

This work package, i.e. WP5, is responsible for developing the two mass-market applications used to validate the GOEASY concept, namely ApesMobility and AsthmaWatch, as well as providing integration and testing for the whole project.

Within WP5, this task, i.e. T5.2, taking advantage of the infrastructure built on Task 4.1, is producing the algorithms both on a client and on a server side and, based on the acquired data, supports the operations of the two mass-market applications. This task will also produce the codebase of the two mass-market applications.

For what concerns the AsthmaWatch app, this task includes the development of the cloud-based services, to integrate data from the mobile air pollution sensors within the GOEASY infrastructure, and the development related to the adaptors for the Copernicus Earth Observation system to extract climate and weather forecasting data as a complement to the mobile sensing data gathered, to be used and integrated into the GOEASY platform and the AsthmaWatch app. The task also includes the development of the mobile AsthmaWatch app and its integration with the GOEASY cloud-based services. Details and codebase of the development for AsthmaWatch and above listed cloud services, are listed in a dedicated section of the present document.

For what concerns the app ApesMobility, this task includes the integration with the existing platform greenApes and possible mobility apps in the city of Torino. Details of the integrations are outlined in a dedicated document section.

This deliverable documents the software components developed so far in GOEASY.

1.1 1.1. Scope

Scope of this document is to present the updates regarding the GOEASY data-oriented enablers and applications that were presented in deliverable D5.2 “Initial GOEASY data-oriented enablers and applications”. The deliverable in this document will serve as the basis for D5.6, among others.

1.2 1.2. Related documents

ID	Title	Reference	Version	Date
[RD.1]	D2.1 - Vision Scenario and Use Case Definition		1.0	2018-02-27
[RD.2]	D2.2 - Initial GOEASY Platform and Pilots Reference Architecture		1.0	2018-05-31
[RD.3]	D4.1 - Cloud-based enablers for LBS provision		1.0	2018-10-31
[RD.4]	D5.2 - Initial GOEASY data-oriented enablers and applications		1.1	2019-02-07

2 Updated domain data models

2.1 2.1 Updated domain data model for AsthmaWatch app

As described in deliverable D4.1, the IoT generated data streams will be stored according to the OGC SensorThings data model. This will be complemented with a domain specific data model for each GOEASY application. Below is the updated data model for the AsthmaWatch use case. The appendix contains the corresponding database definitions.

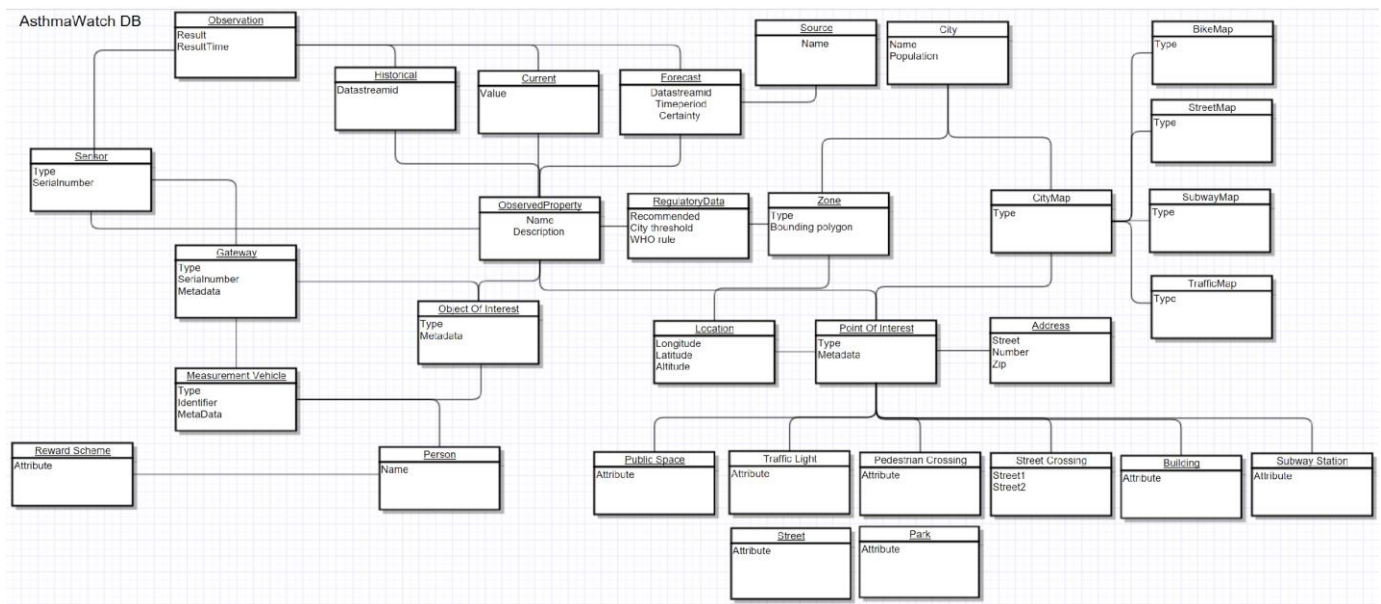


Figure 1 AsthmaWatch E-R Diagram

2.2 2.2 Domain data model for ApesMobility

Hereafter the initial Entity – Relationship Diagram (Chen Notation) for ApesMobility. This diagram is likely to be updated as the development progresses.

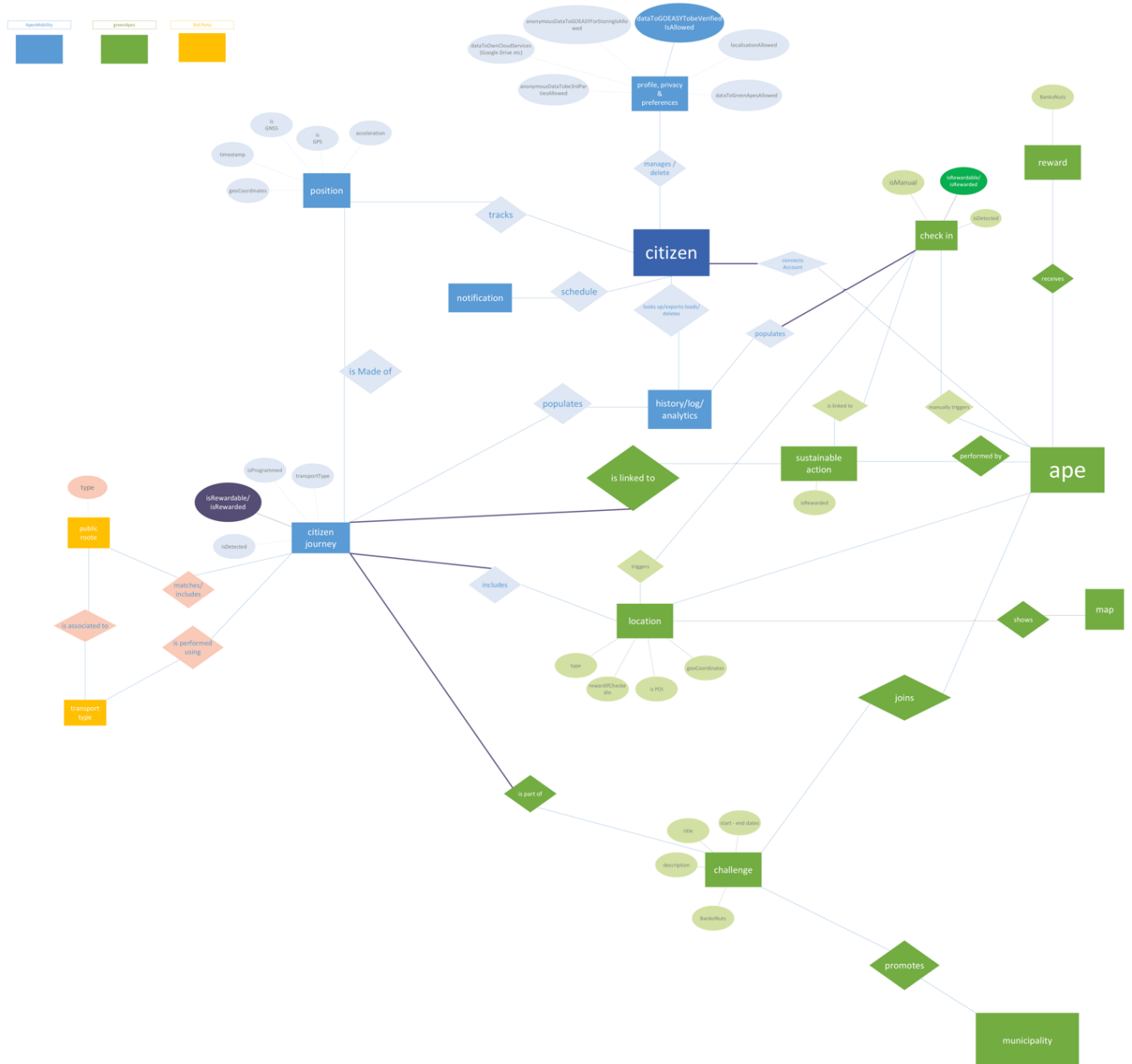


Figure 2 ApesMobility E-R Diagram

3 Updated data-oriented enablers and applications

3.1 AsthmaWatch - Updated data-oriented enablers and application

3.1.1 AsthmaWatch – Specification

Below is a listing of the user stories expressed for AsthmaWatch.

Reference	Story
GOEAS-16	As a user I want to know about the condition in an area to decide if I will go there or not.
GOEAS-17	As a user I want to know the alternative routes based on a specific criteria in order to avoid an asthma attack.
GOEAS-18	As a user I want to keep track of my health in order to react in time on changes.
GOEAS-19	As a user I want to inform specific persons about an asthma attack in order to get immediate help.
GOEAS-21	As a user I want to define rules for a specific criteria in order to be warned in relevant situations.
GOEAS-22	As a user I want to be notified when I am within a certain range of a dangerous spot.
GOEAS-23	As a user I want to be able to view information of a single measuring station to decide if I will go to this area or not.
GOEAS-24	As a user I want to be able to define a preferred route in order to get advised whether to take it or not.
GOEAS-25	As a user I want to be able to import a route from 3rd party app(s) in order to reduce redundant tasks (defining the same route in different applications).
GOEAS-26	As a user I want to get alternative routes suggested automatically if conditions along my preferred route exceed my rules in order to avoid an asthma attack.
GOEAS-27	As a user I want to be able to view details of any suggested route in order to avoid an asthma attack.
GOEAS-28	As a user I want to be able to export a picked route in order to start navigation with a 3rd party app.
GOEAS-29	As a user I want to be able to enter health information manually to keep track of my health.
GOEAS-30	As a user I want to be able to add health information automatically by a connected device to keep track of my health.
GOEAS-31	As a user I want to be able to enter health information to the application in order to keep track of my health.
GOEAS-32	As a user I want to get immediate feedback after adding health information in order to react in time on changes.
GOEAS-33	As a user I want to be able to view historical data in order to know about the development of my health.
GOEAS-35	As a user I want to control the map by known interaction patterns to get the required information as fast and easy as possible.
GOEAS-36	As a user I want to be able to specify multiple persons as emergency contacts to increase

	the probability to get help.
GOEAS-37	As a user I want to be able to activate an alarm in case of an asthma attack in order to get immediate help.
GOEAS-38	As a user I want to inform my emergency contacts about my position when I activated the alarm to get help as fast as possible.
GOEAS-39	As a user I want my phone to play an alerting sound when I activated the alarm to get immediate help by people around me.
GOEAS-40	As a user I want to view all outgoing calls in the app due to an activated alarm to know who was informed and who I should update about my situation now.
GOEAS-41	As a user I want to be able to cancel a call at any time to avoid unnecessary attempts of contacts.
GOEAS-42	As a user I want to be able to select text messages to send to all emergency contacts in case I cancelled a call in order to let them know about my current situation.
GOEAS-43	As a user I want to be able to create and update templates for text messages to inform people faster about my current situation.
GOEAS-44	As a user I want to be able to determine the order in which contacts are contacted to prioritize based on criteria.

3.1.2 Updated data-oriented enablers for the Application Scenarios - AsthmaWatch - Base code

3.1.2.1 Integration of GOEASY sensor data

The sensor data from IoT gateways will be transmitted as OGC (Open Geospatial Consortium) message and stored according to the OGC (Open Geospatial Consortium) SensorThings Data model. This model has been described in deliverable D4.1 “Cloud-based Enablers for LBS provision”. The domain data model is described in section 2 above.

3.1.2.2 Populating domain data

In order to populate the database with domain data an import module has been developed. It supports KML (Keyhole Markup Language). This is an XML-based format for representing layers of objects of interest that can be overlaid on a geographical map. Figure 3 below shows an example of such a file.

```
<Folder>
  <name>VASASTAN AREA -VASASTAN LAYER 1</name>
  <Placemark>
    <name>Point 2</name>
    <description>Drottninggatan_Tengergatan crossing street</description>
    <styleUrl>#icon-1899-0288D1</styleUrl>
    <Point>
      <coordinates>
        18.0566208,59.3386788,0
      </coordinates>
    </Point>
  </Placemark>
  <Placemark>
    <name>Point 3</name>
    <description>Holländargatan_Tegnergatan crossing street</description>
    <styleUrl>#icon-1899-0288D1</styleUrl>
    <Point>
      <coordinates>
        18.0580366,59.3390441,0
      </coordinates>
    </Point>
  </Placemark>
  <Placemark>
    <name>Point 4</name>
    <description>Saltmätargatan_Tegnergatan crossing street</description>
    <styleUrl>#icon-1899-0288D1</styleUrl>
    <Point>
      <coordinates>
        18.0587983,59.3392329,0
      </coordinates>
    </Point>
  </Placemark>
</Folder>
```

Figure 3 Example of a KML file with points of interest encoded

3.1.2.3 Integration of third-party sources

Obtaining data from third party sources is done in order to complement measurements made with the mobile air quality sensors and help validating the results. The data is retrieved from four sources:

- Swedish Environmental Protection Agency (Naturvårdsverket)
- World Air Quality Index website (WAQI)
- Swedish Meteorological and Hydrological Institute (SMHI)
- Copernicus Atmosphere Monitoring Service (CAMS).

Each of these services provide open access to different types of environmental data, where mainly the level of different pollutants is of interest for the AsthmaWatch use case.

Each of the sources provide an API for retrieving specific data which is utilized in the implemented service. All sources except CAMS depend solely on data from fixed measurement stations with a location and a set of sensors providing the information. Therefore, each of the station's measurements are obtained by specifying a unique identifier in a REST call made to the endpoint. The response from the services are JSON strings containing information such as pollutant type, pollution level, time, location - only to name a few.

Data from CAMS is defined as Gridded Binary (GRIB) files, a standard used in meteorology for storing historical and forecasted weather data. Provided in this service is historical and forecast data, which are mainly based on calculations and models of how the predicted state of the different pollutants will be. It is possible to retrieve weather data up to three days in the future. The main difference from the other sources is that data is not retrieved for a specific location. Instead, an area (grid) together with pollutant type and time for when the forecast should be valid. This results in a GRIB file where values together with locations within

the defined grid are defined in a checkerboard-like pattern for the specific time. The points are spaced by approximately one to two kilometers depending where on Earth the grid is defined.

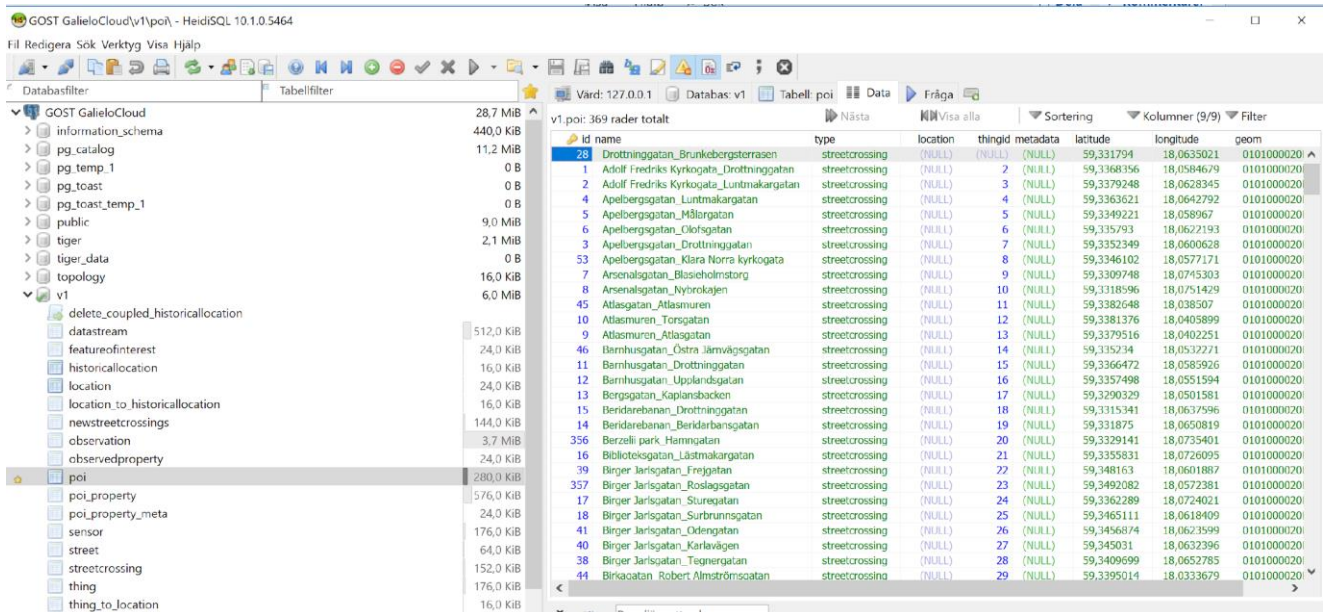
Common between all data-sources is how the information gets formatted before being sent for storing. The format used follows the OGC SensorThings format, a standard used for defining and connecting IoT devices to the web. Using a JSON to define the structure of a message, they contain fields to define the identifier of the device, time of the phenomena, time of creating the message, coordinates, value and many others. An example of the format used can be found in Figure 4.

For transmitting messages from the service to a storage solution, the lightweight MQTT protocol is used. JSON messages, illustrated in Figure 4, are sent as payload to the message broker defined at the Edge Connectivity Manager. Measurement stations, real or virtual (CAMS), have set topics which they post to which they initially receive by announcing themselves to the GOST service. If there exists a data stream with the provided name, the message

```
{
  "resultTime": "2018-11-22T10:08:45",
  "Datastream": {
    "@iot.id": 209
  },
  "phenomenonTime": "2018-11-22T10:06:33",
  "result": {
    "valueType": "NO2",
    "Position": {
      "type": "Point",
      "coordinate": [
        59.3993652, 18.0343221
      ]
    },
  },
  "response": {
    "value": 6,53
  }
}
```

3.1.2.4 Database Structure

The database has been implemented using Postgres see



id	name	type	location	thingid	metadata	latitude	longitude	geom
28	Drottninggatan_Brunkebergsterrassen	streetcrossing	(NULL)	(NULL)	(NULL)	59,331794	18,0635021	0101000020
1	Adolf Fredriks Kyrkogata_Drottninggatan	streetcrossing	(NULL)	2	(NULL)	59,3368156	18,0584679	0101000020
2	Adolf Fredriks Kyrkogata_Luntmakargatan	streetcrossing	(NULL)	3	(NULL)	59,3379248	18,0628345	0101000020
4	Apelbergsgatan_Luntmakargatan	streetcrossing	(NULL)	4	(NULL)	59,3363621	18,0642792	0101000020
5	Apelbergsgatan_Mälargatan	streetcrossing	(NULL)	5	(NULL)	59,3349221	18,058967	0101000020
6	Apelbergsgatan_Olofsgatan	streetcrossing	(NULL)	6	(NULL)	59,335793	18,0622193	0101000020
3	Apelbergsgatan_Drottninggatan	streetcrossing	(NULL)	7	(NULL)	59,3352349	18,0600628	0101000020
53	Apelbergsgatan_Klara Norra kyrkogata	streetcrossing	(NULL)	8	(NULL)	59,3346102	18,0577171	0101000020
7	Arnsdalsgatan_Bluesholmstorg	streetcrossing	(NULL)	9	(NULL)	59,3309748	18,0745303	0101000020
8	Arnsdalsgatan_Nyrovskajen	streetcrossing	(NULL)	10	(NULL)	59,3318596	18,0751429	0101000020
45	Atlasgatan_Atlasmuren	streetcrossing	(NULL)	11	(NULL)	59,3382648	18,038507	0101000020
10	Atlasmuren_Torsgatan	streetcrossing	(NULL)	12	(NULL)	59,3381376	18,0405899	0101000020
9	Atlasmuren_Atlasgatan	streetcrossing	(NULL)	13	(NULL)	59,3379516	18,0402251	0101000020
46	Barnhusgatan_Östra Jämsvågsgatan	streetcrossing	(NULL)	14	(NULL)	59,335234	18,0532271	0101000020
11	Barnhusgatan_Drottninggatan	streetcrossing	(NULL)	15	(NULL)	59,3366472	18,0585926	0101000020
12	Barnhusgatan_Upplandsgatan	streetcrossing	(NULL)	16	(NULL)	59,3357498	18,0551594	0101000020
13	Bergsgatan_Kaplanbacken	streetcrossing	(NULL)	17	(NULL)	59,3290329	18,0501581	0101000020
15	Beridarebanan_Drottninggatan	streetcrossing	(NULL)	18	(NULL)	59,3315341	18,0637596	0101000020
14	Beridarebanan_Beridarbansgatan	streetcrossing	(NULL)	19	(NULL)	59,331875	18,0650819	0101000020
356	Berzelli park_Hamngatan	streetcrossing	(NULL)	20	(NULL)	59,3329141	18,0735401	0101000020
39	Biblioteksgatan_Lästmakargatan	streetcrossing	(NULL)	21	(NULL)	59,3355831	18,0726095	0101000020
357	Birger Jarlsatan_Freigatan	streetcrossing	(NULL)	22	(NULL)	59,348163	18,0601887	0101000020
17	Birger Jarlsatan_Roslagsgatan	streetcrossing	(NULL)	23	(NULL)	59,3492082	18,0572381	0101000020
18	Birger Jarlsatan_Suregatan	streetcrossing	(NULL)	24	(NULL)	59,3362289	18,0724021	0101000020
41	Birger Jarlsatan_Surbrunnsgatan	streetcrossing	(NULL)	25	(NULL)	59,3465111	18,0618409	0101000020
40	Birger Jarlsatan_Karlavägen	streetcrossing	(NULL)	26	(NULL)	59,3456874	18,0623599	0101000020
38	Birger Jarlsatan_Tognergatan	streetcrossing	(NULL)	27	(NULL)	59,345031	18,0632396	0101000020
44	Birgagatan_Robert Almströmsgatan	streetcrossing	(NULL)	28	(NULL)	59,3409699	18,0652785	0101000020
29			(NULL)	29	(NULL)	59,3395014	18,0333679	0101000020

Figure 5 The physical database structure for GOEASY AsthmaWatch

3.1.2.5 Common Retrieval API

Both sensor data and third-party information source data are made available for developers through a standardized OGC (Open Geospatial Consortium) SensorThings API. This has been described in deliverable D4.1 “Cloud-based Enablers for LBS provision”.

Here are some examples of the query API:

- [http://g.galileocloud.goeasyproject.eu:8080 /#/Things\(id\)?\\$expand=Datastreams](http://g.galileocloud.goeasyproject.eu:8080 /#/Things(id)?$expand=Datastreams)
 - Retrieve the data streams for a given thing
- [http://galileocloud.goeasyproject.eu:8080 /#/Datastreams\(id\)/Observations](http://galileocloud.goeasyproject.eu:8080 /#/Datastreams(id)/Observations)
 - Retrieve all observations in a certain datastream.
- <http://galileocloud.goeasyproject.eu:8080 /#/Sensors>
 - List all sensors
- [http://galileocloud.goeasyproject.eu:8080 /#/Observations?\\$top=3&\\$orderby=phenomenonTime desc](http://galileocloud.goeasyproject.eu:8080 /#/Observations?$top=3&$orderby=phenomenonTime desc)
 - Returns the first three Observation entries after sorted by the phenomenonTime property in descending order.

3.1.3 Application AsthmaWatch

The AsthmaWatch app, see Figure 6, aims at providing users with relevant and accurate air quality information in their daily activities. Some screenshots from an early prototype app is shown below.

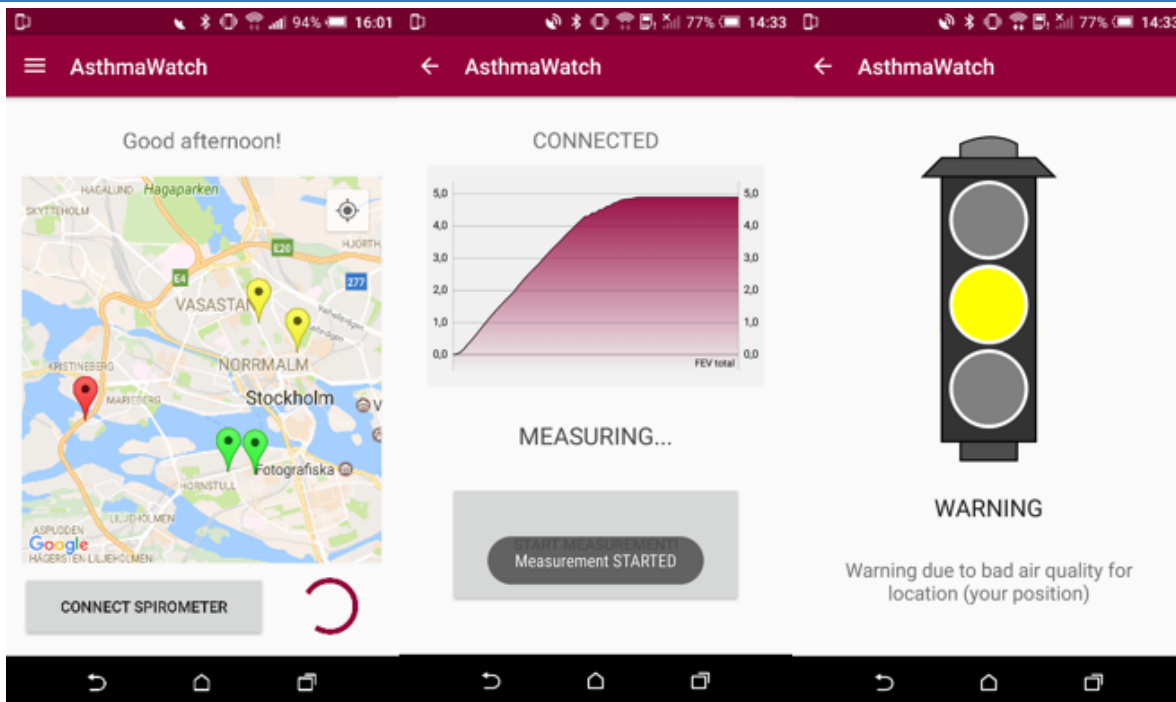


Figure 6 AsthmaWatch - End User Screens

The AsthmaWatch application consists of a smart phone user app and a GOEASY cloud backend which has been configured for the AsthmaWatch requirements. Specific micro services will be developed and instantiated, see Figure 7 below for the GOEASY Microservice architecture. This is explained in detail in deliverable D4.1 “Cloud-based enablers for LBS provision”.

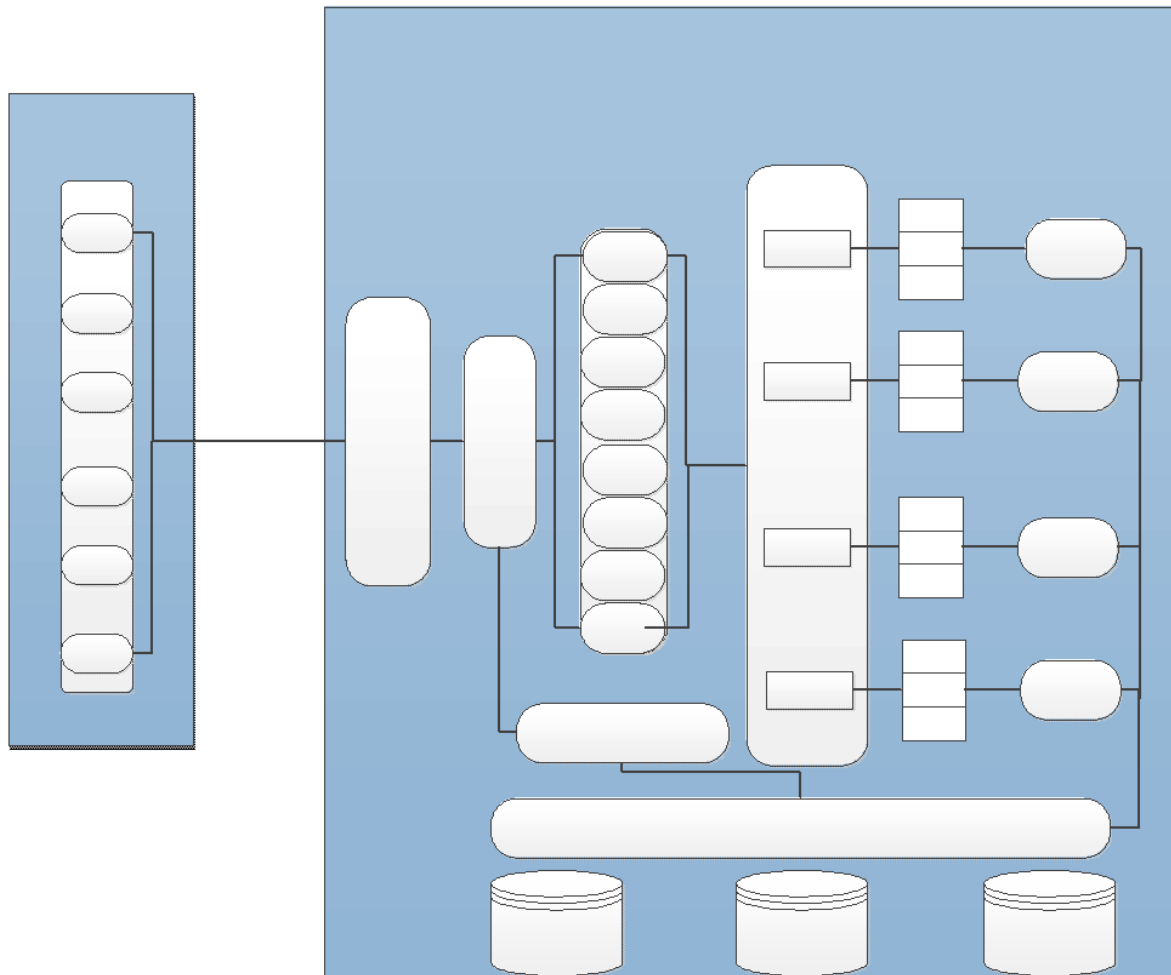


Figure 7 GOEASY Microservices architecture

3.1.3.1 AsthmaWatch Data Collection



Figure 8 Some photos from first deployment of GOEASY Application with IoT Gateway equipped with Galileo chip and air quality sensors for data collection.

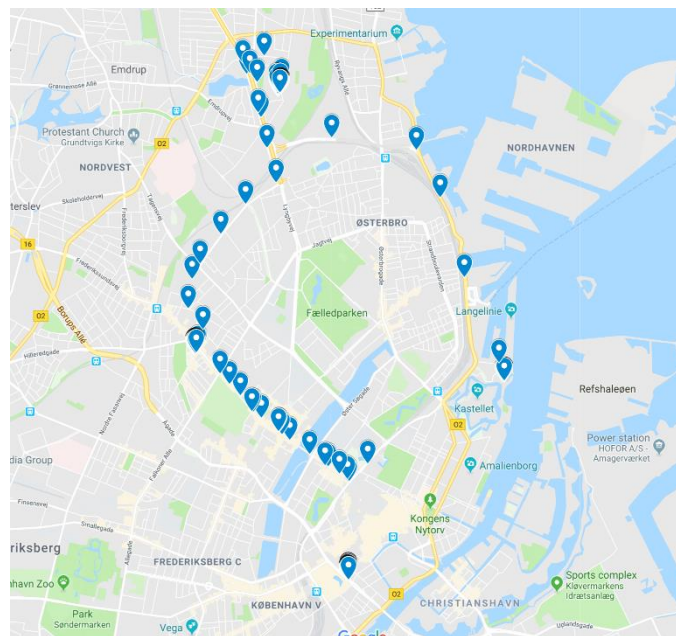


Figure 9 Map of electrical bike being used for collecting air quality data.

3.1.3.2 AsthmaWatch Instantiation within GOEASY

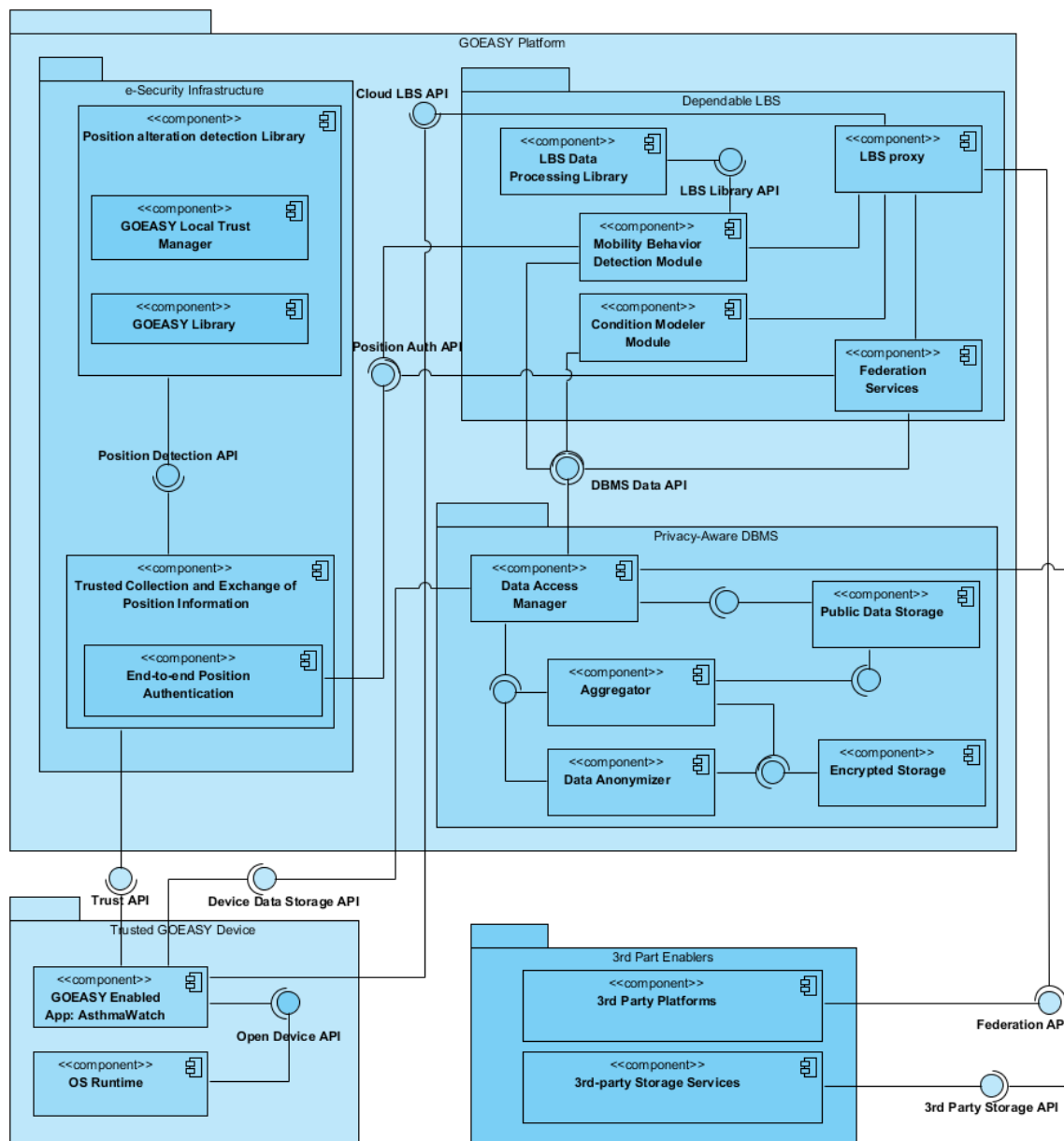


Figure 10 The instantiation of the GOEASY architecture to support AsthmaWatch

3.2 ApesMobility - Updated data-oriented enablers and application

3.2.1 ApesMobility - Specification

Requirements for ApesMobility app have been outlined as User Stories on the Jira GOEASY repository and referenced with the tag “ApesMobility”.

The main purpose of the app is to take advantage of location-based services linked to GNSS data as measured on the Android smartphones with the goal of certifying user behaviour related to sustainable mobility patterns. As such key stories are:

- “GOEAS-56: As a citizen, I want to be able to trigger detection of mobility behavior manually to share my activity immediately.” This story is linked to one narrated from a developer perspective: “GOEAS-54: As a SW application developer, I want to be able to detect specific mobility behavior of users

without a data connection in order to reduce data and battery consumption of the mobile device". When accessing the app, the user is prompted with the possibility of tracking with GNSS sensor. As further option the user has the possibility of scheduling notifications that will act as a reminder about the activation / deactivation of the tracking functionality at specific times of the day. There is also the possibility to repeat the notifications on a daily basis or Mon to Fri. Since Android 8.0, the possibility of activating GNSS through background services has been disabled, hence this implementation of a notification system.

- "GOEAS-13: As a citizen, I want to get my location certified to get additional rewards."
- "GOEAS-10: As a citizen, I want to get my activity certified to get additional rewards." This story has an alter ego on the developer side that is "GOEAS-53: As a SW application developer, I want to certify specific mobility behaviour of users in order to reward them for sustainable behaviour."
- "GOEAS-9: As a citizen, I want to be able to confirm my certified activity to be additionally rewarded."
- "GOEAS-57: As a citizen, I want to be notified about a detected, certified activity in order to not miss additional rewards."

Privacy and profile management.

- "GOEAS-65: As a citizen, I want to be able to delete identifiable location data to protect my privacy."
- "GOEAS-67: As a citizen, I want to be able to manage permissions of my data in order to protect my privacy". -> this story is linked to the following -> GOEAS-48: As a SW application developer, I want to be able to aggregate data to hide details that endanger personal identifiable information.
- "GOEAS-62: As a citizen, I want to be able to en-/disable location tracking to protect my privacy."
- "GOEAS-72: As a citizen, I want to have a profile of my gathered data in order to help me changing my behaviour."

Some of the Jira stories are or will be implemented as part of the GOEASY project in the greenApes platform. These are the following:

- "GOEAS-61: As a citizen, I want to subscribe to specific areas or topics only in order to get relevant information only."
- "GOEAS-60: As a citizen, I want to be notified about new challenges in order to improve my sustainable behaviour."
- "GOEAS-55: As a citizen, I want to be able to share my certified activities in order to motivate fellow citizens."
- "GOEAS-15: As a citizen, I want to join challenges to get motivated and to support the community."

Stories that are more specific to the AsthmaWatch scenario but that can find also an implementation, maybe in a second step, on the ApesMobility app are:

- "GOEAS-70: As a citizen, I want to know which route is the best/ healthiest one in order to not further pollute specific areas."

Some stories are more related to the technical architecture and the way apps exchange data with each other.

Third party apps will be able to send location-based data to the GOEASY platform to exploit the location-based services brokered by GOEASY. The story that generically mentions this feature is the following:

- "GOEAS-47: As a SW application developer, I want to transmit data to the GOEASY platform to make it available for location-based services." This is the story that generically sets the case for third party apps

In the final version of the app there might be a functionality related to looking up on a list or on a map POI or Events with which the user can interact by checking in and with the goal of getting rewarded (being certified and / or by sharing).

- GOEAS-59: As a citizen, I want to see additional information about registered locations/ events in order to decide whether to go there or not.

- GOEAS-58: As a citizen, I want to be able to find registered POIs and challenges in order to collect additional rewards by check-in (and share).

At present there is only one story related to City of Torino or other organisations and this is related to the capability of involving citizens/users in challenges to inspire them on sustainability themes:

- “GOEAS-14: As an organization, I want to offer challenges for users in order to familiarize them with sustainable topics.”

We should probably deepen further use cases related to the organisations and for instance would like to measure data and KPI on population or on the initiatives.

3.2.2 Data-oriented enablers for the Application Scenarios - ApesMobility

The main purpose of the GOEASY platform is to act as a hub between apps and location-based services while providing an additional layer of security to GNSS. ApesMobility is one of the apps that will exploit these features. As seen in the previous paragraph, user's journey composed by the tracked locations will be certified within the ApesMobility app. The certification of the location is possible through data-oriented enablers that could differ depending on the geography of the users, because algorithms or services that works best in the context of a specific city might not have the same efficacy in a different location. The flexibility of the GOEASY platform to connect to different LBSs will give the added value to app developers and users of using the ones that best fit their needs. ApesMobility pilot will be held in Torino, so a selection process is ongoing that aims to find the LBS that will give the best accuracy while certifying the transport system in the context of the Torino urban mobility. In the final version of this document, details of the chosen LBS for the app ApesMobility will be provided, including methods for invoking such services.

3.2.3 Application - ApesMobility

3.2.3.1 Application - ApesMobility - Architecture within the GOEASY framework

In figure 11 the Instantiation Architecture for the ApesMobility app is represented. Locally on the device the app integrates a GOEASY library to authenticate the smartphone identity for a specific session with the GOEASY platform and in particular with the Key-Cloak service.

Once, received the token from the Key Cloak authentication service, the app authenticates the GNNS Message and sends the data to the GOEASY platform that will:

- Anonymises the positions and saves them in a database
- Reroutes to 3rd party solutions to validate the mobility behaviour of the users. This data flow between ApesMobility and the 3rd party component requires a component on the GOEASY platform acting as a proxy and routing the mobility data to the 3rd party module managing the feedback that it provides, giving it back to the app itself.

As shown in **Figure 11**, ApesMobility can be connected to the app greenApes if this is also installed on the Android device. This will allow ApesMobility to use greenApes's different functionalities and to reward users for their sustainable behaviour. Some functionalities are being developed in greenApes exclusively for the GOEASY project to engage the users in missions or challenges that may include checking-in to green venues in the city. As such the greenApes app will require on one hand to access the greenApes platform back-end APIs and on the other to use the local GOEASY platform libraries to authenticate the position.

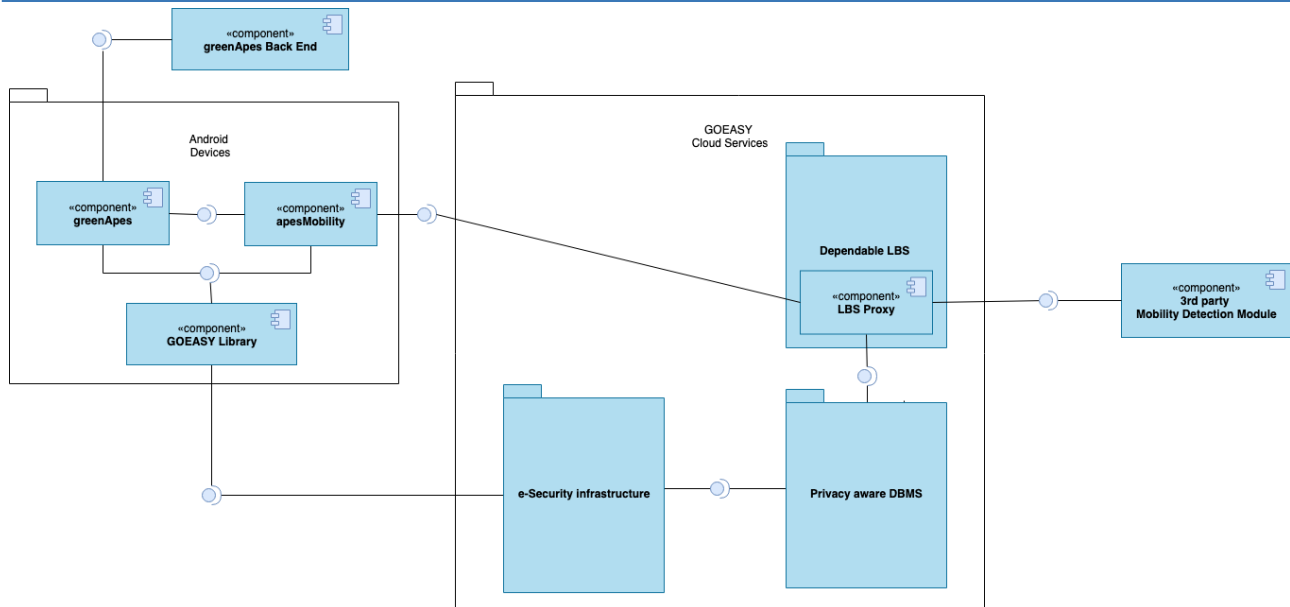


Figure 11 ApesMobility Instantiation Architecture

3.2.3.2 Updated Application - ApesMobility - Programming Language and Code organization

ApesMobility is an Android application and is being developed in Kotlin language implementing an MVP architecture. Kotlin is a cross-platform language designed to be concise and fully intraoperative with Java and JVM. Officially supported for Android since October 2017, it's now Google's preferred language for app development. This brand new technology offers some advantages with respect to the issues that developing such an app would submit.

In Java, access to a null value will result in an exception (and crash in worst case) and due to data types involved in this project (sensors, GPS, GNSS, network), we need to consider something to limit or avoid completely this type of problems.

With Kotlin, that gets its name from an island near St. Petersburg, we can have references that can hold a null value and eliminate the possibility of an exception. Below we have shown a comparison between Java and Kotlin.

Java code example:

```
String a = "apesmobility";
a = null;
print(a); // → Cause an NPE exception
```

Kotlin code example:

```
var a: String? = "apesmobility"
b = null // Ok
print(b) // → Ok
```

The choice of MVP (Model View Presenter) as pattern and code organization offers some clear benefits. For instance, the traditional Android Activities (what you see on screen) contain both UI and data access mechanism on the same file. This type of architecture may generate two kind of problems:

- change only to UI or data mechanism can "break up" the Activity
- difficulty to create a suite of unit tests

MPV architecture adds a new layer (the Presenter) between Model and View.

The Presenter will be the mediator that retrieves data from the model and return them to the view and defines what happens to the view on user interaction.

Separating the app in well-defined layers, though not straightforward in its implementation, can help avoid code degrading and help have an application maintainable and extendable.

In addition to MVP, we decided to organize code in packages layered by features to get benefits such as:

- Good modularity
- Easy navigation
- Possibility to work in team by feature

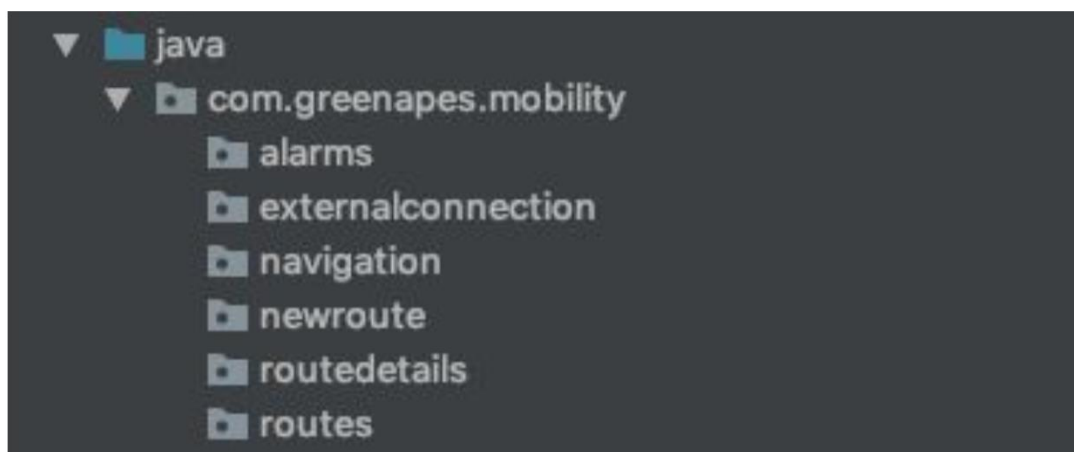


Figure 12 ApesMobility – Package code organization

3.2.3.3 Application - ApesMobility - Screen Apps

In its final version, this paragraph will contain references to the code developed for the app apesMobility. Being this the updated issue, it presents the status at the current stage of the project. The wireframes showed in the initial version have been transformed in mock-ups that in a typical “agile” approach undergo continuous revisions as long as the process evolves. This means that even at this stage when the app is actually being developed, they have to be considered nor definitive nor complete, even if they are capable of conveying a solid picture of the main interactions within the app. Not all the wireframes have been mocked up yet, but those who are being developed at present.

Below we can see a few examples. Using the app, user is encouraged to adopt sustainable transport habits, measured through the location-based services leveraged by the Galileo GNSS signal and the GOEASY platform. A “gamification” mechanism will be deployed, that consists in progressing through different levels gaining points increased by tracking sustainable trips. This will keep in good health an avatar that will assume different shapes on each level.

The app does not require for user to log-in, but will have an on-boarding flow which will prompt the user with some options that include, but are not limited to, the consent given to the app for using the phone location-based services or the consent to let the positions recorded by the app be verified by the GOEASY platform cloud services. These steps are not described by the current mock-ups.

Once the on-boarding process is completed, the user lands on the main page of the app (see Fig. 13).



Figure 13 ApesMobility - Main page

The main screen shows at the bottom a button to start / pause / stop recording the geographical coordinates, while in the centre, it shows the avatar corresponding to the level the user is.

In the very first stage after being released, the users will be asked to track a number of sessions to help the algorithm hone his predictive capability. Once they have completed a number of journeys, the recorded tracks will let the users be rewarded for helping the system work better. For this reason, users can monitor how many sessions have been tracked in this specific phase, through a kind of progress bar and numeric indicator.

When starting a new tracking session, the screen changes to inform the users that the system is recording the journey (see picture 14).

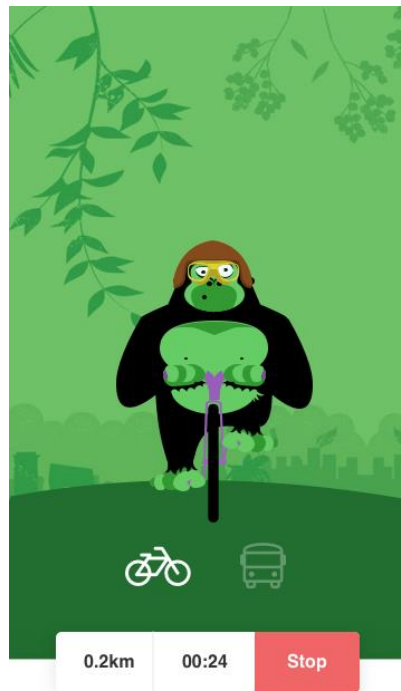


Figure 14 ApesMobility – Tracking a journey

ApesMobility is an Android app and, as such, it uses patterns of navigation typical for the platform. On the left corner, the hamburger menu opens a sliding menu (Fig. 15) that allows reaching all the main pages/sections of the app.

3.2.3.4

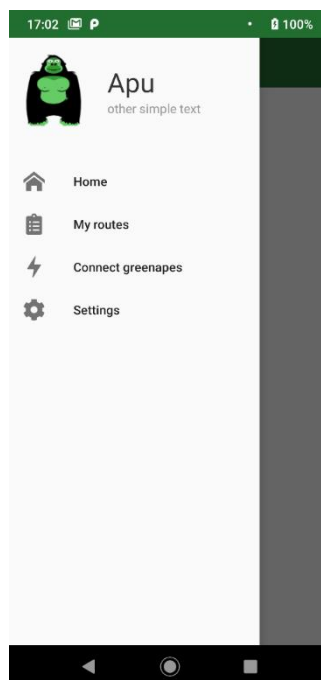


Figure 15 ApesMobility – Hamburger menu

The second icon in the menu leads users to the My Routes page (the name is temporary, see Fig. 16) where they may find a list of the recorded journeys with the associated main meta-data.

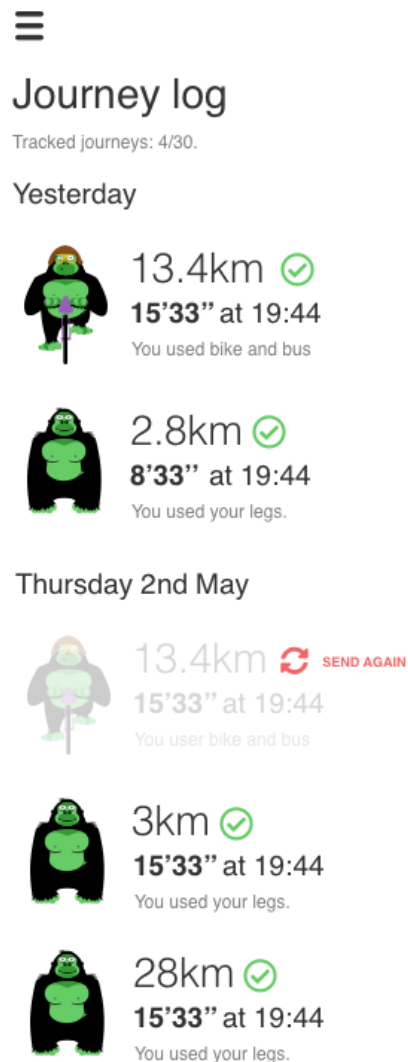


Figure 16 ApesMobility – Routes page

The central part of the page shows the sessions that have been recorded in a range of time (that can be navigated by scrolling the app vertically) and for each the main info such as duration, time and date, whether this session have been validated by the GOEASY services and the result of the validation. If the result of the validation is successful, then there is an indication of the transport system used in the session and any health points related to it. If the session has not been validated, then the user has an icon that allow the submission of the session to the GOEASY platform for validation.

Tapping a journey, a user is able to reach the Route details page (see Fig. 17) that shows more details for that specific session. In particular, sessions with a mixed range of transport systems used show the breakdown of any leg of the journey. Here the session can be deleted by the user.

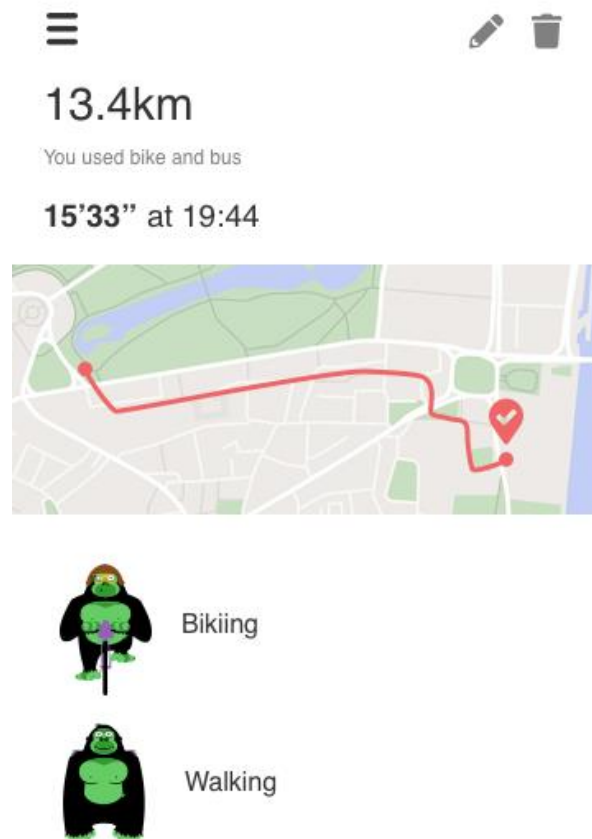


Figure 17 ApesMobility - Routes details

Another link in the hamburger menu takes users to the “Connect greenApes” functionality that lets them connect a greenApes account. This will give the chance to communicate the events related the sustainable journeys had by the users to the greenApes platform and as such to be rewarded for them. In the platform there will be also functionalities related to checking in to POIs or joining challenges linked to sustainable actions.

Through the hamburger users can reach the Settings page (see Fig. 18), a composite page that provides the user with many interaction possibilities.

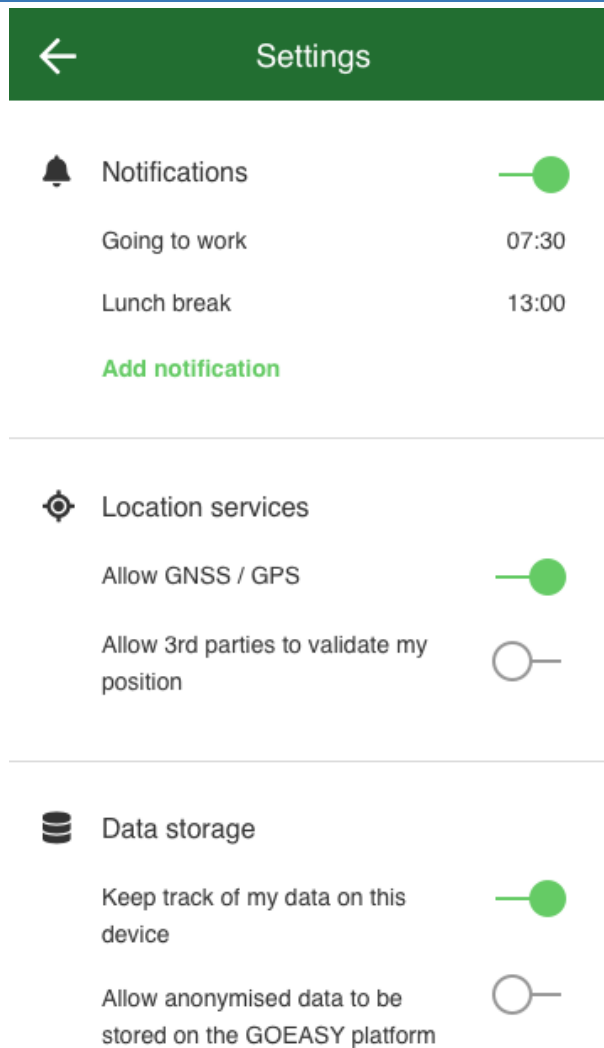


Figure 18 ApesMobility – Settings

The “Notifications” section allows the user to schedule in advance notifications the will remind him/her to activate the GNSS tracking feature in specific also recurring times/days (see Figure 19).

←

Save

Choose title

🕒

Set time

When

☐ Only today
 ☐ Every day
 ☐ Mon - Fri
 ☒ Only on

L

M

M

G

V

S

D

Figure 19 ApesMobility – Notifications

A “Privacy / Consents” section enables the user to fully manage his consents in terms of data privacy protection and to enable / disable location-based services. Here the user is able to erase any data stored locally.

In future versions of the app the user will be given the feature of exporting his data to a personal cloud-based storage service such as Google Drive for then being able to load again back the data onto the app.

Finally, the hamburger menu gives access to a page that shows analytics related to the user transport habits and to gamification elements in the app. This page hasn’t been mocked up yet, for this reason a wireframe is shown below (see fig. 20).

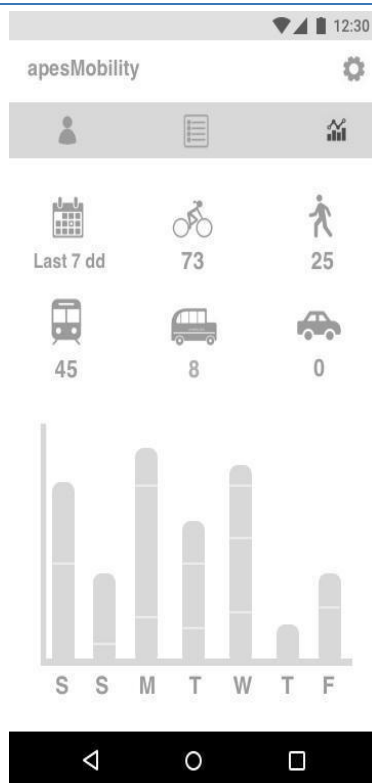


Figure 20 ApesMobility –Analytics Wireframe

4 Conclusions

This document is dedicated to describe/provide the technical details of the first 2 apps implementing use cases that are fully leveraged by the GOEASY platform services. They rely on a heavy use of cloud-based algorithms and services that are or directly part of the GOEASY platform (AsthmaWatch) or provided by 3rd parties (ApesMobility). In this second case, the platform acts as hub/broker connecting the apps and location-based services.

Acronyms

Acronym	Explanation
LBS	Location Based Services
DBMS	Data Base Management System
OS	Operating System
API	Application Programming Interface
SASL	Simple Authentication and Security Layer
TLS	Transport Layer Security
PII	Personal Identifiable Information
IOT	Internet of Things
GEP	GOEASY Platform
CAPs	Collective Awareness Platforms
GNSS	Global Navigation Satellite System
JVM	Java Virtual Machine
ER	Entity Relationship
OGC	Open Geospatial Consortium
REST	Representational State Transfer
JSON	JavaScript Object Notation
KML	Keyhole Markup Language
LBS	Location Based Service
XML	Extensible Markup Language
POI	Point of Interest

Appendix GOEASY Database

```

CREATE TABLE IF NOT EXISTS "datastream" (
    "id" BIGINT NOT NULL DEFAULT nextval('datastream_id_seq'::regclass) COMMENT E",
    "name" CHARACTER VARYING(255) NULL DEFAULT NULL COMMENT E",
    "description" CHARACTER VARYING(500) NULL DEFAULT NULL COMMENT E",
    "unitofmeasurement" JSONB NULL DEFAULT NULL COMMENT E",
    "observationtype" INTEGER NULL DEFAULT NULL COMMENT E",
    "observedarea" GEOMETRY(GEOMETRY,4326) NULL DEFAULT NULL COMMENT E",
    "phenomenontime" TSTZRANGE NULL DEFAULT NULL COMMENT E",
    "resulttime" TSTZRANGE NULL DEFAULT NULL COMMENT E",
    "thing_id" BIGINT NULL DEFAULT NULL COMMENT E",
    "sensor_id" BIGINT NULL DEFAULT NULL COMMENT E",
    "observedproperty_id" BIGINT NULL DEFAULT NULL COMMENT E",
    KEY ("thing_id"),
    KEY ("sensor_id"),
    KEY ("observedproperty_id"),
    PRIMARY KEY ("id")
);

CREATE TABLE IF NOT EXISTS "featureofinterest" (
    "id" BIGINT NOT NULL DEFAULT nextval('featureofinterest_id_seq'::regclass) COMMENT E",
    "name" CHARACTER VARYING(255) NULL DEFAULT NULL COMMENT E",
    "description" CHARACTER VARYING(500) NULL DEFAULT NULL COMMENT E",
    "encodingtype" INTEGER NULL DEFAULT NULL COMMENT E",
    "geojson" JSONB NULL DEFAULT NULL COMMENT E",
    "feature" GEOMETRY(GEOMETRY,4326) NULL DEFAULT NULL COMMENT E",
    "original_location_id" BIGINT NULL DEFAULT NULL COMMENT E",
    PRIMARY KEY ("id")
);

CREATE TABLE IF NOT EXISTS "historicallocation" (
    "id" BIGINT NOT NULL DEFAULT nextval('historicallocation_id_seq'::regclass) COMMENT E",
    "thing_id" BIGINT NULL DEFAULT NULL COMMENT E",
    "time" TIMESTAMP WITH TIME ZONE NULL DEFAULT NULL COMMENT E",
    KEY ("thing_id"),
    PRIMARY KEY ("id")
);

CREATE TABLE IF NOT EXISTS "location" (
    "id" BIGINT NOT NULL DEFAULT nextval('location_id_seq'::regclass) COMMENT E",
    "name" CHARACTER VARYING(255) NULL DEFAULT NULL COMMENT E",
    "description" CHARACTER VARYING(500) NULL DEFAULT NULL COMMENT E",
    "encodingtype" INTEGER NULL DEFAULT NULL COMMENT E",
    "geojson" JSONB NULL DEFAULT NULL COMMENT E",
    "location" GEOMETRY(GEOMETRY,4326) NULL DEFAULT NULL COMMENT E",
    PRIMARY KEY ("id")
);

CREATE TABLE IF NOT EXISTS "location_to_historicallocation" (
    "location_id" BIGINT NULL DEFAULT NULL COMMENT E",
    "historicallocation_id" BIGINT NULL DEFAULT NULL COMMENT E",
    KEY ("historicallocation_id"),
    KEY ("location_id")
);

CREATE TABLE IF NOT EXISTS "observation" (
    "id" BIGINT NOT NULL DEFAULT nextval('observation_id_seq'::regclass) COMMENT E",
    "data" JSONB NULL DEFAULT NULL COMMENT E",
    "stream_id" BIGINT NULL DEFAULT NULL COMMENT E",
    "featureofinterest_id" BIGINT NULL DEFAULT NULL COMMENT E",
    KEY ("stream_id","id"),
    KEY ("id"),
    KEY ("featureofinterest_id")
);

CREATE TABLE IF NOT EXISTS "observedproperty" (
    "id" BIGINT NOT NULL DEFAULT nextval('observedproperty_id_seq'::regclass) COMMENT E",
    "name" CHARACTER VARYING(120) NULL DEFAULT NULL COMMENT E",
    "definition" CHARACTER VARYING(255) NULL DEFAULT NULL COMMENT E",

```

```

        "description" CHARACTER VARYING(500) NULL DEFAULT NULL COMMENT E",
        PRIMARY KEY ("id")
    );
CREATE TABLE IF NOT EXISTS "poi" (
    "id" INTEGER NOT NULL DEFAULT nextval('poi_id_seq'::regclass) COMMENT E",
    "name" CHARACTER VARYING(300) NOT NULL COMMENT E",
    "type" CHARACTER VARYING(50) NULL DEFAULT NULL COMMENT E",
    "location" INTEGER NULL DEFAULT NULL COMMENT E",
    "thingid" INTEGER NULL DEFAULT NULL COMMENT E",
    "metadata" JSON NULL DEFAULT NULL COMMENT E",
    "latitude" TEXT NULL DEFAULT NULL COMMENT E",
    "longitude" TEXT NULL DEFAULT NULL COMMENT E",
    "geom" TEXT NULL DEFAULT NULL COMMENT E",
    PRIMARY KEY ("id")
);
CREATE TABLE IF NOT EXISTS "poi_property" (
    "id" INTEGER NOT NULL DEFAULT nextval('"POI_measurement_id_seq"'::regclass) COMMENT E",
    "poiid" INTEGER NOT NULL COMMENT E",
    "property" CHARACTER VARYING(100) NOT NULL COMMENT E",
    "currentvalue" REAL NULL DEFAULT NULL COMMENT E",
    "historicalvaluestream" INTEGER NULL DEFAULT NULL COMMENT E",
    "timestamp" TIMESTAMP WITHOUT TIME ZONE NULL DEFAULT NULL COMMENT E",
    "forecastvaluestream" INTEGER NULL DEFAULT NULL COMMENT E",
    "observedpropertyid" BIGINT NULL DEFAULT NULL COMMENT E",
    "poi_property_metaid" INTEGER NULL DEFAULT NULL COMMENT E",
    PRIMARY KEY ("id")
);
CREATE TABLE IF NOT EXISTS "poi_property_meta" (
    "id" INTEGER NOT NULL DEFAULT nextval('poi_property_meta_id_seq'::regclass) COMMENT E",
    "name" CHARACTER VARYING(50) NOT NULL COMMENT E",
    "unitofmeasurement" JSON NULL DEFAULT NULL COMMENT E",
    "valuerules" JSON NULL DEFAULT NULL COMMENT E",
    "description" TEXT NULL DEFAULT NULL COMMENT E",
    PRIMARY KEY ("id")
);
CREATE TABLE IF NOT EXISTS "sensor" (
    "id" BIGINT NOT NULL DEFAULT nextval('sensor_id_seq'::regclass) COMMENT E",
    "name" CHARACTER VARYING(255) NULL DEFAULT NULL COMMENT E",
    "description" CHARACTER VARYING(500) NULL DEFAULT NULL COMMENT E",
    "encodingtype" INTEGER NULL DEFAULT NULL COMMENT E",
    "metadata" TEXT NULL DEFAULT NULL COMMENT E",
    PRIMARY KEY ("id")
);
CREATE TABLE IF NOT EXISTS "street" (
    "id" INTEGER NOT NULL DEFAULT nextval('street_id_seq'::regclass) COMMENT E",
    "name" CHARACTER VARYING(200) NOT NULL COMMENT E",
    "length" INTEGER NULL DEFAULT NULL COMMENT E",
    "start" POINT NULL DEFAULT NULL COMMENT E",
    "end" POINT NULL DEFAULT NULL COMMENT E",
    "metadata" JSON NULL DEFAULT NULL COMMENT E",
    "poiid" INTEGER NULL DEFAULT NULL COMMENT E",
    PRIMARY KEY ("id")
);

CREATE TABLE IF NOT EXISTS "streetcrossing" (
    "id" INTEGER NOT NULL DEFAULT nextval('streetcrossing_id_seq'::regclass) COMMENT E",
    "name" CHARACTER VARYING(300) NULL DEFAULT NULL COMMENT E",
    "street1" INTEGER NOT NULL COMMENT E",
    "street2" INTEGER NOT NULL COMMENT E",
    "metadata" JSON NULL DEFAULT NULL COMMENT E",
    "poiid" INTEGER NULL DEFAULT NULL COMMENT E",
    PRIMARY KEY ("id")
);

```

```
CREATE TABLE IF NOT EXISTS "thing" (
    "id" BIGINT NOT NULL DEFAULT nextval('thing_id_seq'::regclass) COMMENT E",
    "name" CHARACTER VARYING(255) NULL DEFAULT NULL COMMENT E",
    "description" CHARACTER VARYING(500) NULL DEFAULT NULL COMMENT E",
    "properties" JSONB NULL DEFAULT NULL COMMENT E",
    PRIMARY KEY ("id")
);
CREATE TABLE IF NOT EXISTS "thing_to_location" (
    "thing_id" BIGINT NULL DEFAULT NULL COMMENT E",
    "location_id" BIGINT NULL DEFAULT NULL COMMENT E",
    KEY ("thing_id"),
    KEY ("location_id")
);
```