Politecnico di Milano

A.Y. 2017/2018

Software Engineering 2: ***Travlendar+***

Implementation & Testing Document

Matteo Biasielli - Emilio Capo - Mattia Di Fatta

v. 1.0

Table of Contents

1. Introduction

1.1 Document purpose

This document focuses on the implementation of details the project Travlendar+. The application’s purpose is to support users in handling out one of the most difficult nowadays’ challenges: organization. No previous versions of this application were developed.
This document is meant to be a reference for any person who has an interest in the project. This includes, but is not limited to, development team members, stakeholders and end users.

1.2 Description of the problem *[reported from RASD document]*

The aim of the project is to create an all-in-one system that unites services that are nowadays offered by various different applications (e.g. Calendar, Travel Scheduler). In order to use Travlendar+, final users must be registered and logged in.
Users should be able to schedule their activities directly through the application and, by taking into account travelling times, constraints and preferences expressed by the user, Travlendar+ should:

- Identify the best mobility option;
- Support the user in buying public transport tickets, if necessary;
- Locate the nearest car or bike sharing, if they represent the best solution;
- Warn the user when a place can’t be reached in the available time.

In general, Travlendar+ should make it easier to organize complex schedules, by finding the best compromises between time optimization and the users’ needs and preferences.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

- **User**: actor that is using the application and may want to access all functionalities.
- **Application**: with the term application we are talking about the desktop version, the website and mobile version of the Travlendar+ system.
- **Scheduling**: action performed by a user that is adding a new activity to his personal calendar.
- **Flexible Activity**: An activity with starting and ending time larger than the duration.
- **Fixed Activity**: An activity with fixed starting and ending time.

- **Break**: Flexible Actvity;
- **Tag**: couple <keyword, address> defined by the user.

1.3.2 Acronyms

- **RASD:** Requirements Analysis and Specification Document
- **UI:** User Interface
- **API:** Application programming interface
- **UML**: Unified Modeling Language
- **GPS**: Global Positioning System

1.3.3 Abbreviations

- **[Gn]:** the n-th goal
- **[Rn]:** the n-th requirement
- **[NFRn]:** the n-th non-functional requirement
- **[An]:** the n-th assumption
- **[Cn]:** the n-th constraint
- **[Fn]:** the n-th functionality

2. Requirements mapping and Functionalities

2.1 Functionalities

In the allotted time for the implementation, we could develop the Server (which structure is discussed later in this document), the Windows Client and Android Client. Considering that **this is a prototype**, we focused on the basic features and what we achieved is that the functionalities we could provide are the following:

- **[F1] Login:** A Login system is available and it's mandatory to login into the system to use the application;
- **[F2] Registration:** Users that are not registered yet are able to register into the system. Note that for the moment the registration procedure asks nothing but username and password because those were the only vital data required to provide basic login functionalities. It can be easily extended;
- **[F3] View Calendar:** After the Login, on both clients it's possible for users to view their calendar with activities that were previously added;
- **[F4] Add a Fixed Activity:** After the Login , users can schedule fixed activities (see definition). After scheduling such an activity, they receive a response from the server containing the result status of the request (OK or an error code) and eventually a notification. There are many different notifications but it's worth it to focus the attention on a few of them:
  - A notification is received if the added activity does not make the calendar be inconsistent (no overlapping) but the user will not be allowed to be on time to the just added activity;
  - A notification is received if the added activity does not make the calendar be inconsistent (no overlapping) but the user will not be allowed to be on time to the activity that comes immediately after the just added one;
  - A notification is received if the added activity does not make the calendar be inconsistent (no overlapping) but the user will not be allowed to be on time to one activity and the above two cases are not verified.

- **[F5] Add a Flexible Activity (Break)**: After the Login , users can schedule flexible activities (see definition). After scheduling such an activity, they receive a response from the server containing the result status of the request (OK or an error code) and eventually a notification. There are many different notifications but it's worth it to focus the attention on one of them:
    - A notification is received if the added activity does not make the calendar be inconsistent (no overlapping) but the user will not be allowed to be on time to one activity and the above two cases are not verified.
- **[F6] Update an existing activity:** After the Login, users can update their previously added activities. They can modify any field of activities and they can also change it from fixed to flexible or vice versa. The answer/notification they receive is the same as for [F6] and [F5];
- **[F7] Delete an existing activity:** After the Login, users can delete their previously added activities.
- **[F8] Add a tag:** a tag is a tuple <Position, Address, text>. It represents an address that the user will refer to with a keyword text that he sets as well. Most likely there are some places that users will have to write down really often. The point of the tag system is to allow users to indicate those address by just selecting the tag from the tag list instead of writing down the whole address every time. After the Login, users can add new tags. We completely rely on Google Geocoding and Google Reverse Geocoding APIs for this service, to check that inserted addresses are valid.
- **[F8] Delete a tag:** a tag is a tuple <Position, Address, text>. It represents an address that the user will refer to with a keyword text that he sets as well. Most likely there are some places that users will have to write down really often. The point of the tag system is to allow users to indicate those address by just selecting the tag from the tag list instead of writing down the whole address every time. After the Login, users can delete already existing tags.
- **[F9] Preferences:** After the Login, users can indicate theur travel preferences. There are plenty of possibilities and each of them is always accepted. Though, is a user sets too strict preferences it won't be possible to find possible routes or to estimate the travel time for an activity in order to be allowed to send notifications to advice the user to get ready to go out. Should this happen, users will be informed.

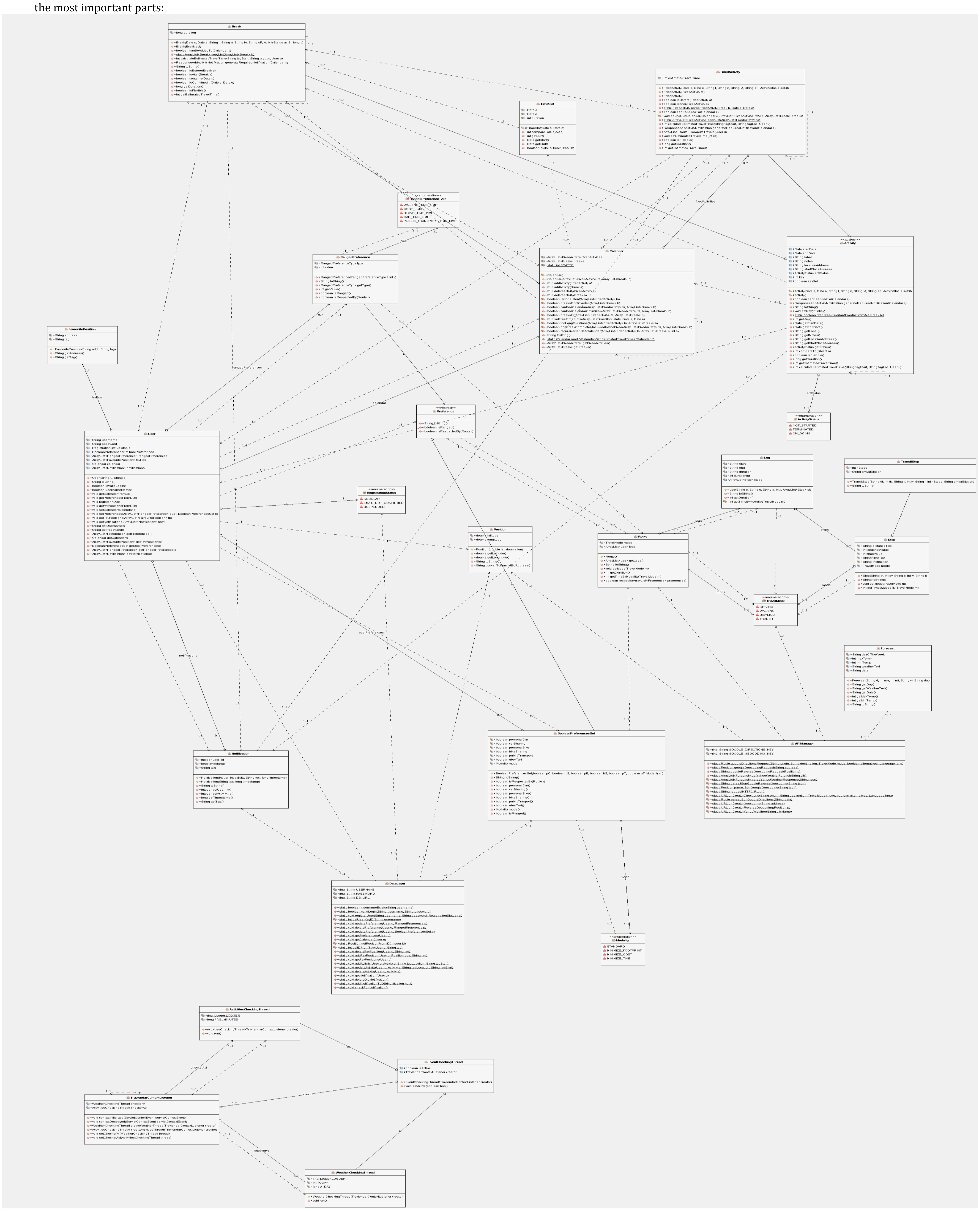
- **[F10] Notifications:** users are notified every day about the weather. Moreover, users are notified when they should get ready to leave for the next activity. Clients execute the notifications synchronization process every minute;
- **[F11] Travel:** Users can request to be shown a possible travel mean (or more than one) for the next activity. If this is not possible due to their too strict preferences, users are advised to change preferences and retry.

2.2 Requirements mapping *[Requirements reported from RASD]*

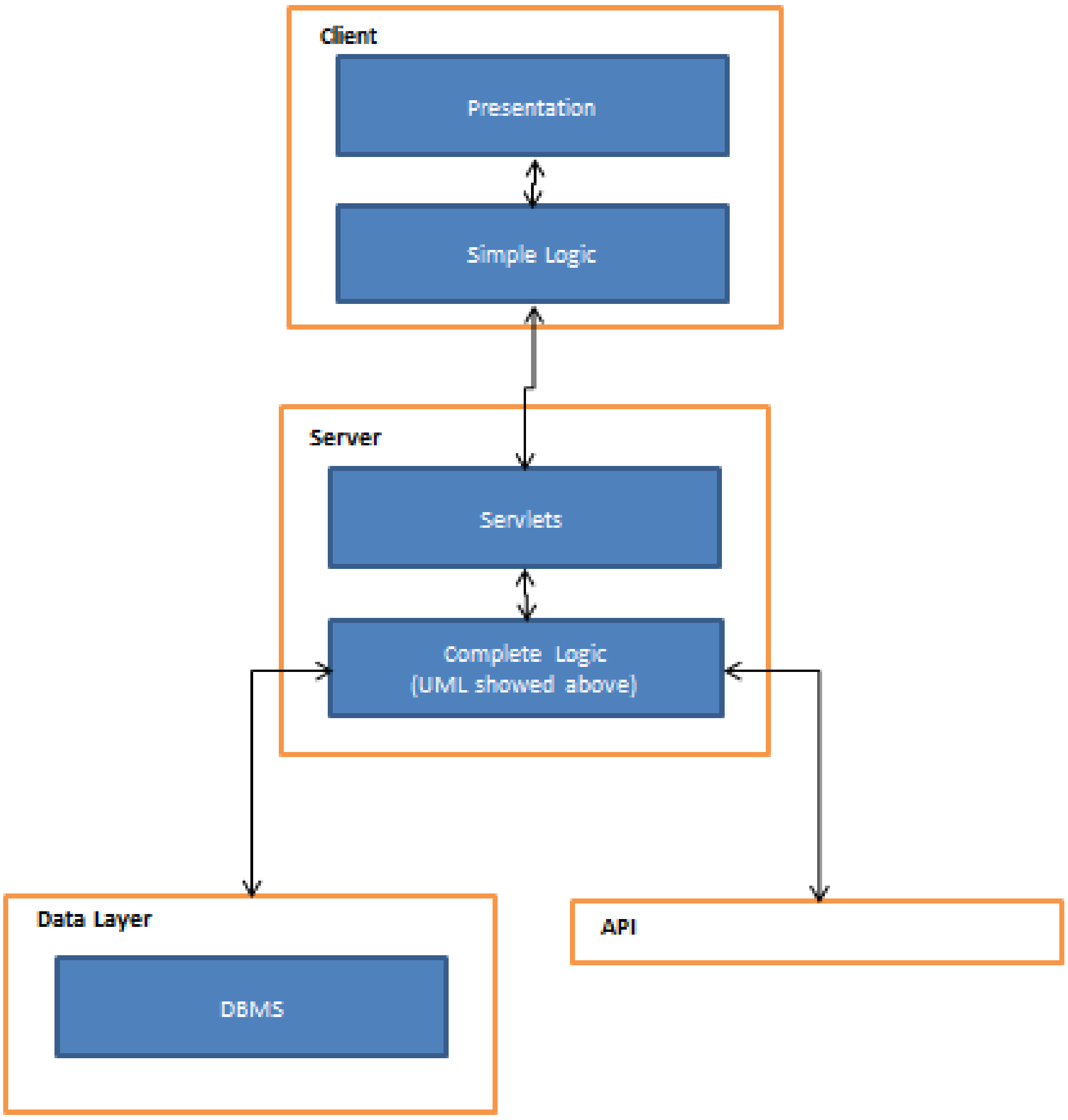
- **[R1]** Allow the users to manage already existing activities.
  This is achieved by [F3], [F6], [F7].
- **[R2]** Users should be able to log in to Travlendar+.
  This is achieved by [F1].
- **[R3]** Users should be able to register to Travlendar+.
  This is achieved by [F2].
- **[R5]** Users should be able to schedule new activities.
  This is achieved by [F4], [F5].
- **[R6]** Users should be able to set their own preferences that will be taken into account and will be applied to schedules every time this is possible and reasonable.
  This is achieved by [F9].
- **[R6.1]** *Specification*: The user can also set flexible activities (e.g. flexible lunch) , and, in particular, the modality “minimize carbon footprint” will be present.
  This is achieved by [F9], [F5].
- **[R8]** Users should be warned when they’re scheduling an activity that is not physically possible due to a lack of time or that overlaps with other activities.
  This is achieved by [F4], [F5].
- **[R9]** Mobility solutions involving car and bike sharing systems must be taken into account, when possible, and proposed to the user when they represent the optimal solution.
  This is achieved by [F11].
- **[R10]** Users should receive a notification (e.g. email, push notifications) a little before the time they have to leave to go to the next appointment.
  This is achieved by [F10].

3. Structure of the source code

We developed three independent projects: “Travlendar Server”, “Travlendar Desktop Client” and “Travlendar Android Client”. Here’s the UML diagram of the **server’s logic**, which is one of the most important parts:



Note that in the diagram we represented only classes and packages that are relevant to the real comprehension of the structure. This means that we removed Exceptions, Servlets and Response objects to keep only the logic in the diagram. The original diagram is way larger and a bit uncomprehensible, but for the sake of completeness it's available in the delivery folder.
The following simple schema, however, helps to understand the structure of the whole project and communications:



As said in the schema, the UML diagram showed above represents the most important part of the "Complete Logic" component. When the server receives a request, the receiving servlet just "delivers" to the request to the logic and then takes the answer and sends it back to the client.
The "Complete Logic" component is able to query the Data Layer and the external system's APIs through the classes "Data Layer" and "API Manager", that offer the necessary query methods and that are both showed in the UML diagram. Those two classes completely encapsulate the requests to the related

external systems so that the logic doesn't have to care about the connection details.
Due to the simplicity of the clients, their structure is not represented here.

4. Test Plan *[Made-up prior to the starting of the Implementation - Reported from DD]*

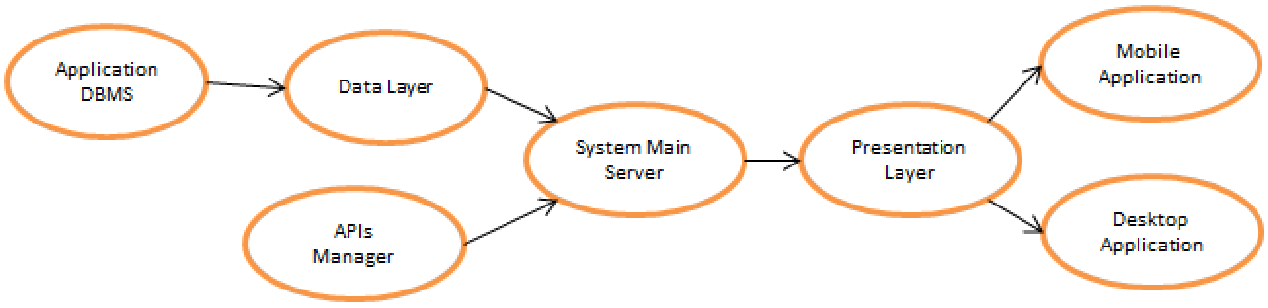
Considering the diagram showed in the **2.2 System Overview** section of the DD as reference, it's easy to notice which components (depend) on other components. There are components, anyway, that are independent. Considering also that there are no circular dependencies, we can conclude that we are able to implement components according to dependencies, that means that a component is implementable if and only if it does not depend on other components or its dependencies have already been implemented. Let's analyse dependencies:

- **Application DBMS** is independent;
- **Data Layer** depends on **Application DBMS**, since it has to build queries and send them to the **Application DBMS** component;
- **System Main Server** depends on the **Data Layer** and **APIs Manager**, since it needs both of them to perform its operations (controller);
- **APIs Manager** relies on the **Third Part Services APIs** in the sense that it can accomplish its role if and only if the **Third Part Services APIs** work fine. However, we don't need to implement the **Third Part Service APIs** because they have obviously been implemented already by the respective owners, so we can consider the **APIs Manager** independent;
- **Presentation Layer** relies on the **System Main Server**;
- **Mobile Application** relies on the **Presentation Layer**;
- **Desktop Application** relies on the **Presentation Layer**.

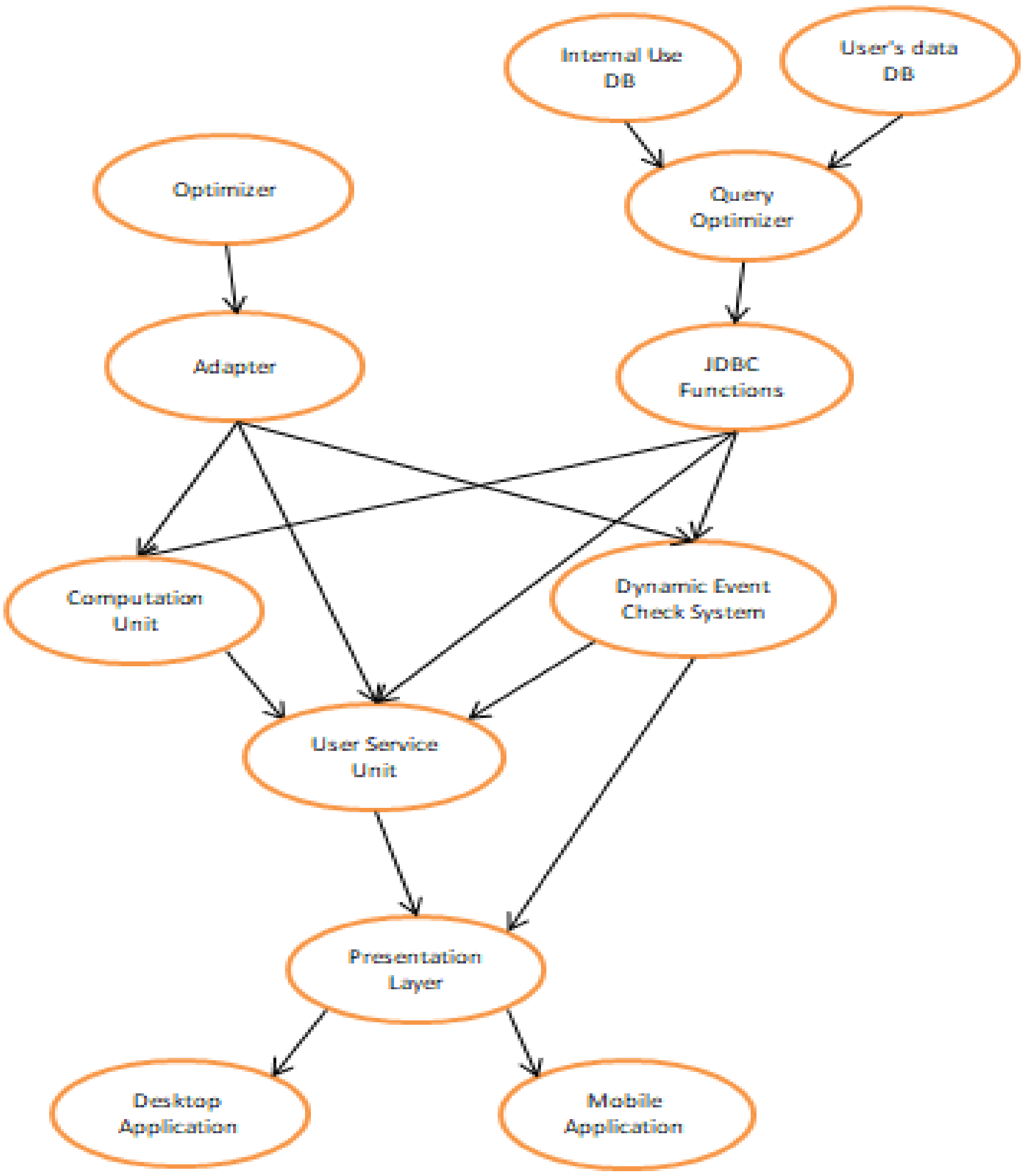
For example, a feasible developing order becomes: Application DBMS -> Data Layer -> APIs Manager -> System Main Server -> Presentation Layer -> Mobile Application -> Desktop Application.

Note that the browser is not represented here because it doesn't need to be developed.

As a matter of clarity, and to highlight other possible orders, we can represent dependencies as a graph, where A -> B if and only if B depends on A.



The same reasonment can be applied considering the Component view for each node of the graph. Expanding all nodes of the previous graph, we obtain a new and more detailed graph that expresses priorities, showed below.



Considering the way we designed the implementation process, it appears natural to proceed to the integration and test phases with this strategy:

- When the **Application Database** and the **Data Layer** have been completely projected, the **Data Layer** must be completely tested, together with all the queries it exposes to query the Database. Since the **Data Layer** implements all and only the queries that are useful to the system itself, this will also help to understand if the DB has been projected and implemented correctly;
- Similarly to the Data Layer, the **APIs Manager** exposes all the calls that the system needs to do to external systems. Provided that the Third Part Systems are reachable ,it needs to be tested completely, in order to be sure that the calls are formatted and implemented correctly and that the system receives the results it needs;
- Since the **System Main Server** is implemented after the **Data Layer** and the **APIs Manager** are fully implemented and tested (so we can say they are perfectly functional), it can be integrated with them from the beginning. Moreover, the **System Main Server** should be tested in all its functionalities.
- For the same reasons, the **Presentation Layer** can be integrated with the **System Main Server** immediately.
- When the clients (**Mobile Application** and **Desktop Application**) are developed, they just need to be integrated with the whole system, that is already implemented, integrated and tested and thus can be considered fully and perfectly functional.

5. Performed Tests and Analysis

5.1 Unit Tests

Among the classes of the Complete Logic showed in section 4, we performed unit tests where relevant. To be more precise, we tested the following classes:

- Calendar;
- FavouritePosition;
- RegistrationStatus;
- User;
- BooleanPreferencesSet;
- Modality;
- RangedPreference;
- RangedPreferenceType.

In particular, along with Calendar, we also tested Activity, FixedActivity and Break. Those classes include the most relevant and complicated part of the complete logic, because they contain methods necessary to exploit and verify all the conditions necessary to schedule, modify and remove activities. Furthermore, they're also responsible of generating the notifications of all the various cases that may happen when adding/updating an activity.

During the test process we were then able to locate bugs and, by observing execution time of tests, to find the most complex procedures and optimize them so that we can exploit all the computations in a fair amount of time, in order to avoid users having an endless wait just to schedule an activity.

Considering the percentage of tested code, the nature and the importance of the unit-tested modules , we're quite confident in saying that those stand-alone modules are bug-free.

5.2 Performances Evaluation

Considering that our server should be ready and able to satisfy requests from **several** users, it's understandable that **performances**, and in particular **response time**, represent a significant quality indicator for our application.

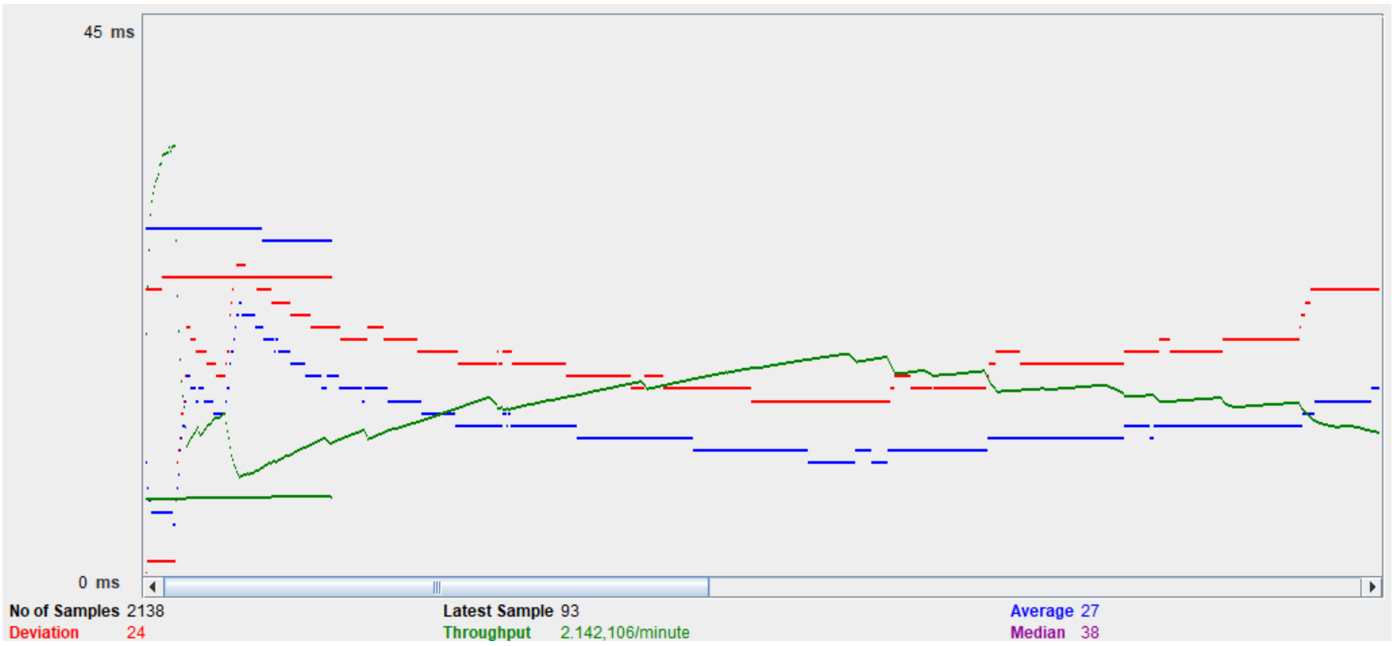
We performed the following tests:

- **Single Login Response Time**

  This is not extremely significant when evaluating performances of an application that should serve thousands of users at the same time, but it is a nice starting point and a good basis for the other tests because we can have a "measurement basis" thanks to this. A single login just requires a few simple queries to the database so thanks to this we know how long it takes to satisfy an "easy" request and we can compare this with the other results.

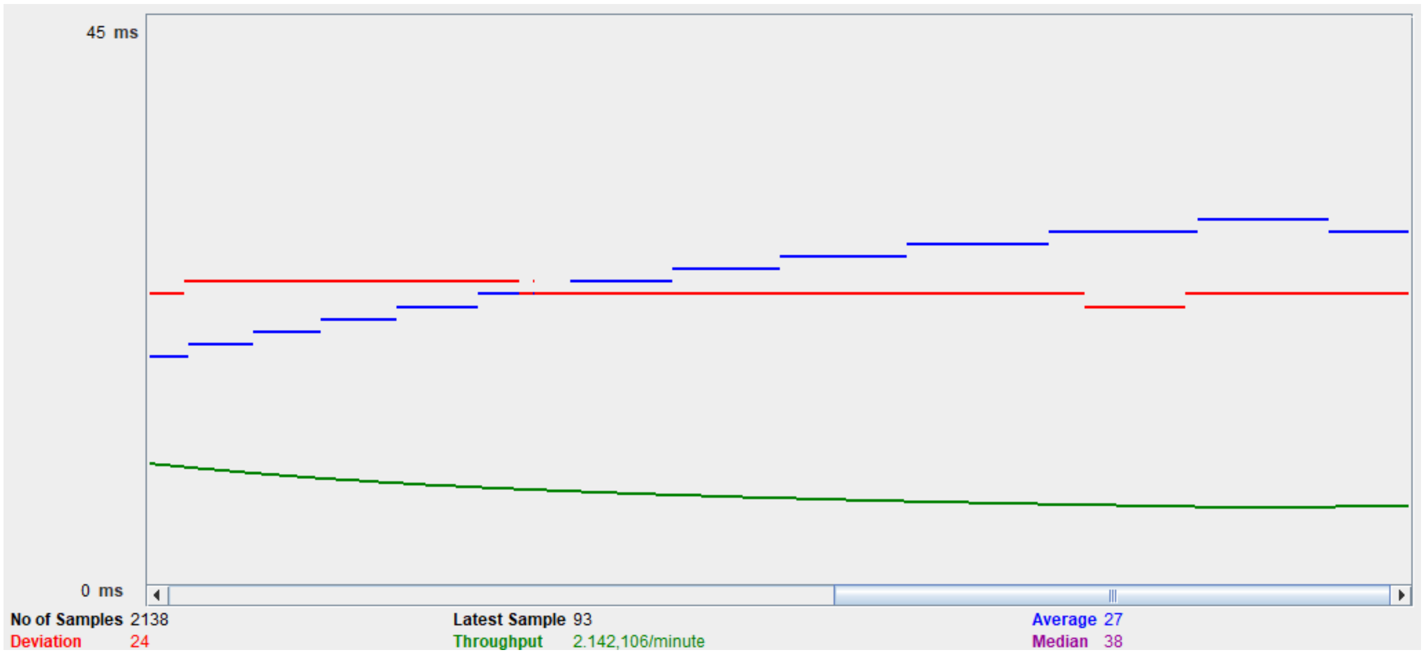
  In the graph showed below it's possible to observe the evolution of the average response time (blue) and the Throughput (green)

  Beginning of the test:



No of Samples 2138
Deviation 24
Latest Sample 93
Throughput 2.142,106/minute
Average 27
Median 38

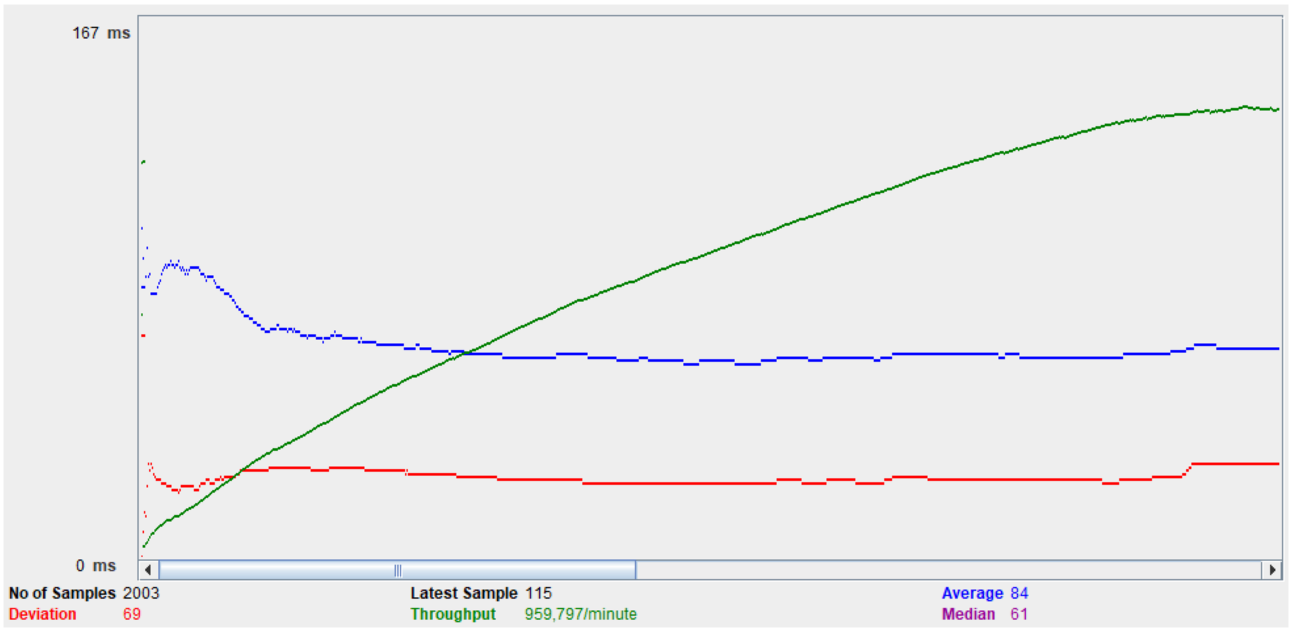
Asymptotically:



No of Samples 2138
Deviation 24
Latest Sample 93
Throughput 2.142,106/minute
Average 27
Median 38

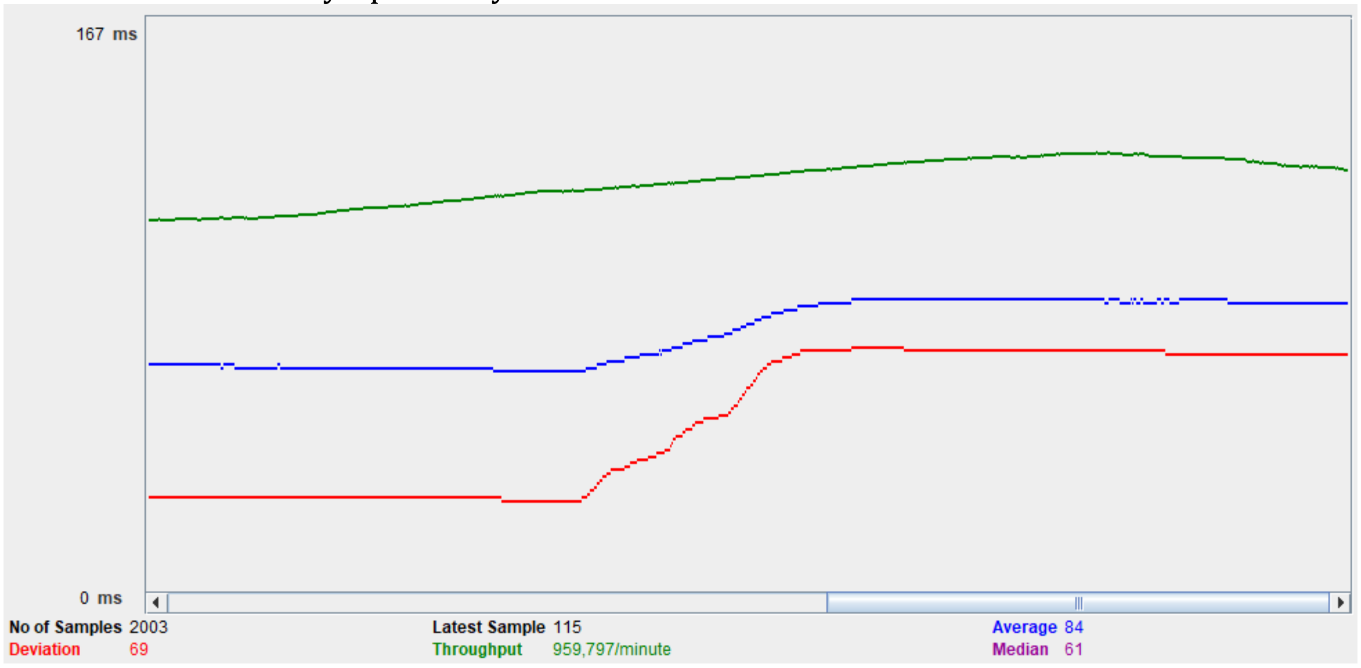
After stabilizing, the Throughput is such that a single login takes about 4.7 milliseconds.

- **Massive Login Response Time**
  To have a comparison with the previous test, we tried then to send 1 thousand of login requests per minute (with peaks at the beginning and at the end of the minute) and we had the results are shown below.



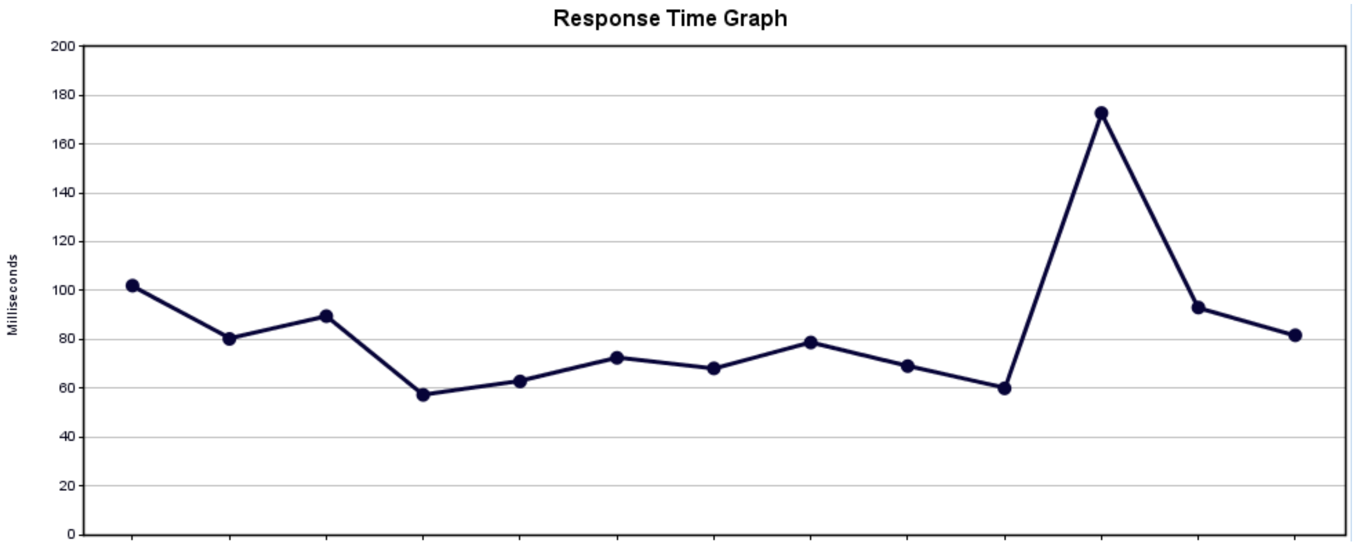
No of Samples 2003
Deviation 69
Latest Sample 115
Throughput 959,797/minute
Average 84
Median 61

Asymptotically:

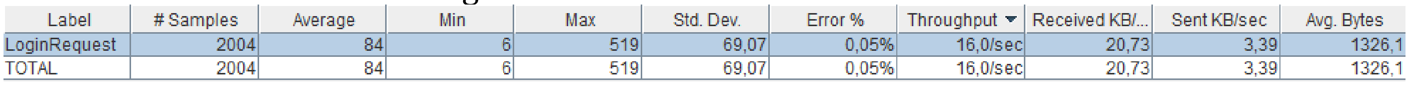


No of Samples 2003
Deviation 69
Latest Sample 115
Throughput 959,797/minute
Average 84
Median 61

Response Time Graph:



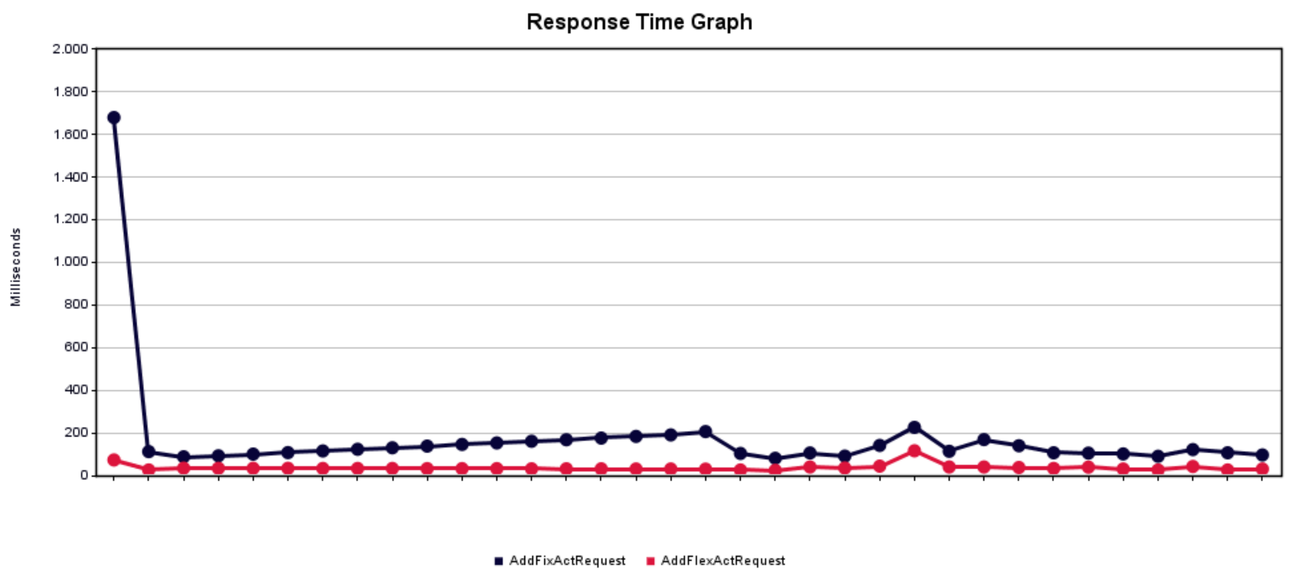
In the Graph, each step is 10 seconds. The peak is one of the moments almost all the requests have been sent together and it's observable that requests are still satisfied in an acceptable time.
Some more general statistics:



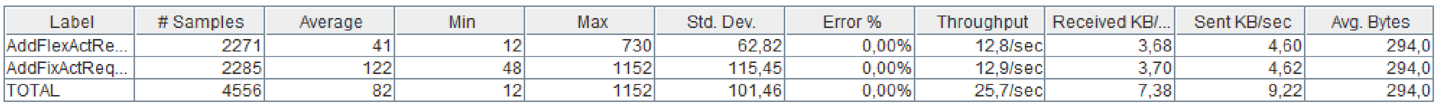
Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/...	Sent KB/sec	Avg. Bytes
LoginRequest	2004	84	6	519	69,07	0,05%	16,0/sec	20,73	3,39	1326,1
TOTAL	2004	84	6	519	69,07	0,05%	16,0/sec	20,73	3,39	1326,1

- **Multiple Activities Addition**

  The first tests on this didn't have good results when speaking performances and response time. However, they helped us to discover the points where our algorithms failed or took too long to compute the activity addition. After several optimizations we repeated the tests and we got to the following result.



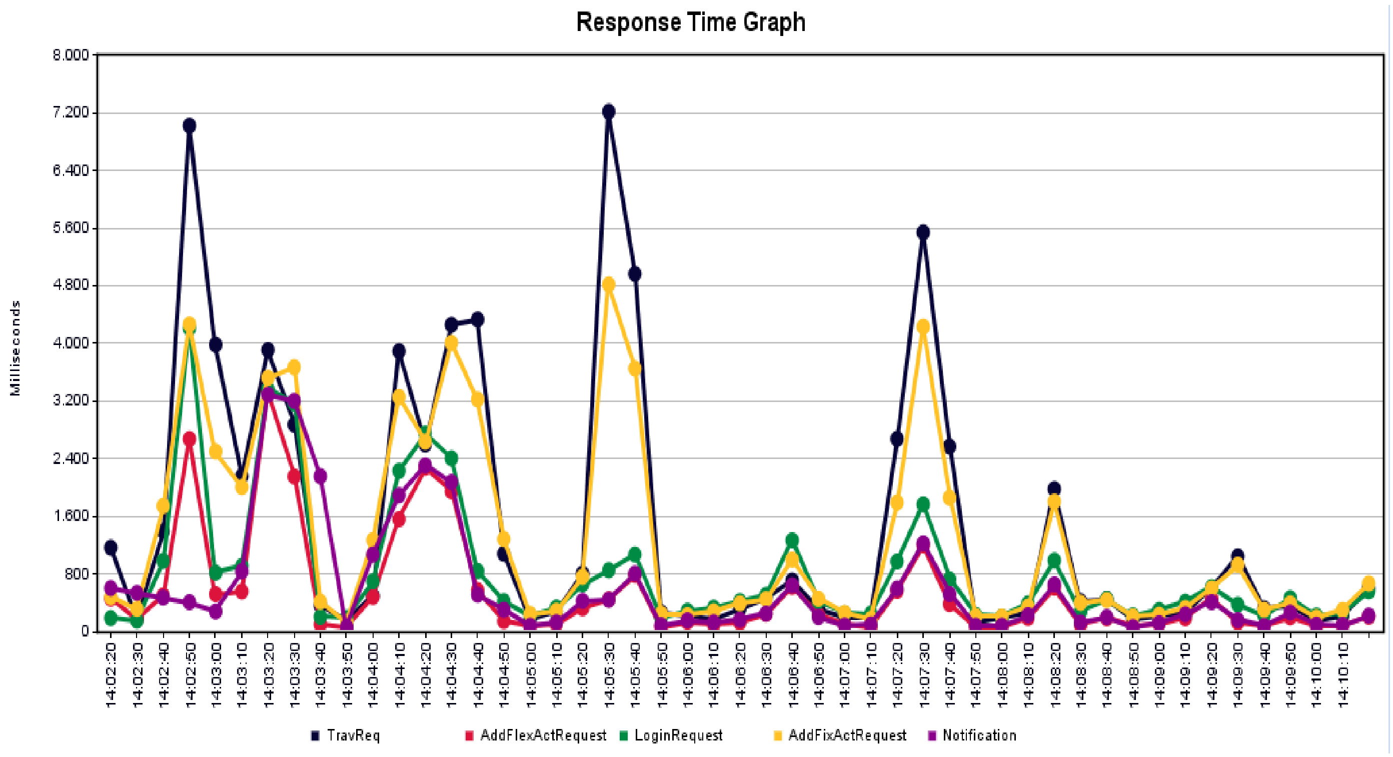
We requested 900 Fixed Activity insertion requests and 900 Breaks insertions per minute. All the requests were done the first time at the very beginning and then after that each thread repeated the request after a time determined according to a statistic distribution (Gaussian with average in 1 minute and 10 seconds of Variance).
It's possible to see in the graph showed above that when all the 1800 requests were sent together the response time was higher but the server could handle it anyway. After that, since not all the threads woke up again at the same moment, the response time was always acceptable.
Some more general statistics are provided below.



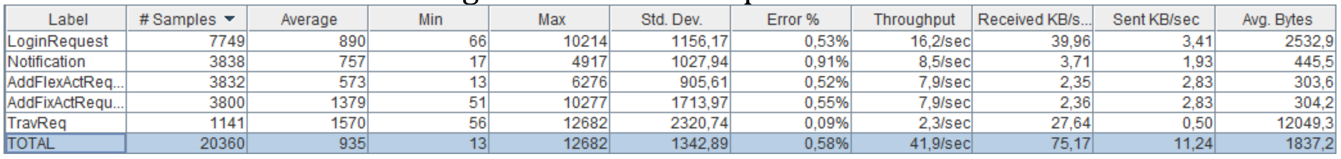
Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/...	Sent KB/sec	Avg. Bytes
AddFlexActRe...	2271	41	12	730	62,82	0,00%	12,8/sec	3,68	4,60	294,0
AddFixActReq...	2285	122	48	1152	115,45	0,00%	12,9/sec	3,70	4,62	294,0
TOTAL	4556	82	12	1152	101,46	0,00%	25,7/sec	7,38	9,22	294,0

- **All the kind of requests together**

  A significant test was also trying to send all the kind requests together**.** To be precise, we didn't send exactly all the kinds of requests but we have chosen the ones that are the most complex to satisfy.
  Some results are shown below.



The results we had this time are slightly different from the results we had on other tests. In this case, the server was has been really stressed up and overwhelmed by requests. It's easy to notice that the Travel requests and the requests of addition of a fixed activity are the ones that take longer time, and, most of the times, their response times are similar.
This is due to the fact that those requests require interaction with Google APIs so their response time is subject to other external factors and to the response time of the third party system. Apart from those peaks, even though the response times are higher than the ones we had when testing performances on single functionalities, they're still acceptable. Considering that tests have been executed on a machine that is not conceived to be a high-performances server, we believe that when our application server will run on a real server response times will be even lower.
Some more general details are provided below.



Label	# Samples ▼	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/s...	Sent KB/sec	Avg. Bytes
LoginRequest	7749	890	66	10214	1156,17	0,53%	16,2/sec	39,96	3,41	2532,9
Notification	3838	757	17	4917	1027,94	0,91%	8,5/sec	3,71	1,93	445,5
AddFlexActReq...	3832	573	13	6276	905,61	0,52%	7,9/sec	2,35	2,83	303,6
AddFixActRequ...	3800	1379	51	10277	1713,97	0,55%	7,9/sec	2,36	2,83	304,2
TravReq	1141	1570	56	12682	2320,74	0,09%	2,3/sec	27,64	0,50	12049,3
TOTAL	20360	935	13	12682	1342,89	0,58%	41,9/sec	75,17	11,24	1837,2

There's a really low error percentage because very few of the requests (order of magnitude of decades) have been refused for the same reason exposed above (the machine running the server was not a real server). We're not worried for that because we know the cause and this won't give rise to a problem.

All the requests have been performed in about 7 minutes.

6. External References

- API Yahoo! Weather: link.
- API Google Maps: link.
- API Google Geolocation: link.
- MySQL documentation: link.
- RASD document available in the delivery folder.
- DD document available in the delivery folder.
- Full code of the server and clients available on the repository an in the delivery folder.
- WAMP documentation: link.

7. Tools and Frameworks Used

- WAMP Server to host the database and keep it physically separated from the rest.
- Eclipse 8.2 as IDE.
- Glassfish 4.1.1 to run the application server.
- IntelliJ as IDE.
- JMeter to do performances tests.
- JUnit to do unit tests.
- Mockito together with JUnit for tests.
- Arquillian to perform integration testing.
- GitHub Desktop and Git to publish and version our code.

8. Installation Instructions

8.1 Desktop Client

No installation is required for the Desktop Client since it’s been developed purely with JavaFX. Just make sure you have an internet connection and run the “travlendardesktopclient.jar” file that is available in the delivery folder (bin) of the repository.

Note that the “travlendardesktopclient.jar” file goes together with the “lib” folder, if you separate them, unexpected errors will occur.

8.2 Android Client

Locate the “travlendarandroidclient.apk” file in the delivery folder (bin) and transfer it to the phone. Alternatively, you can download it directly on the phone from here: link .

Note that it requires Android 5.0 or higher.

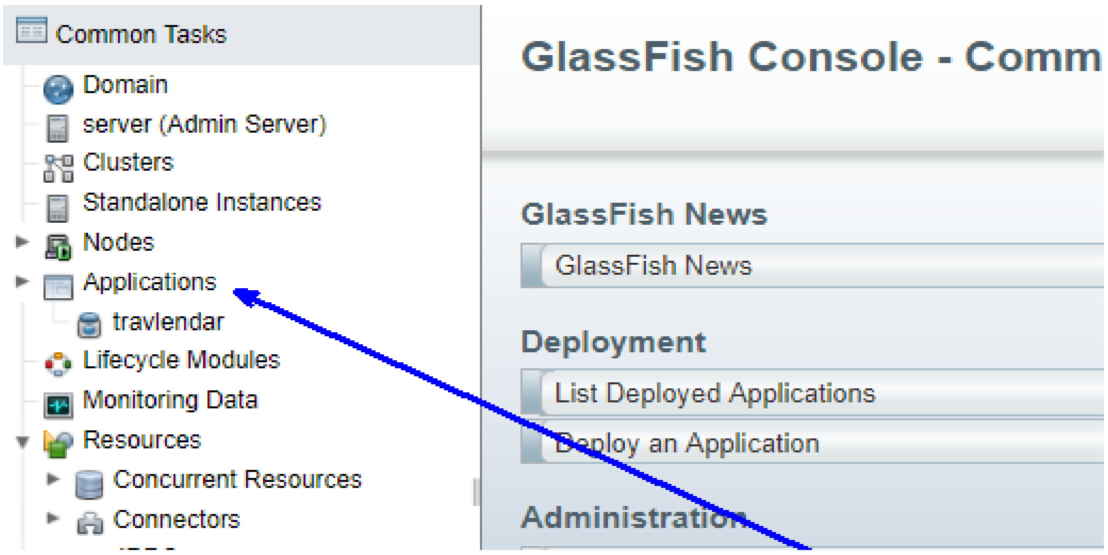
8.3 Server

8.3.1 Application Server

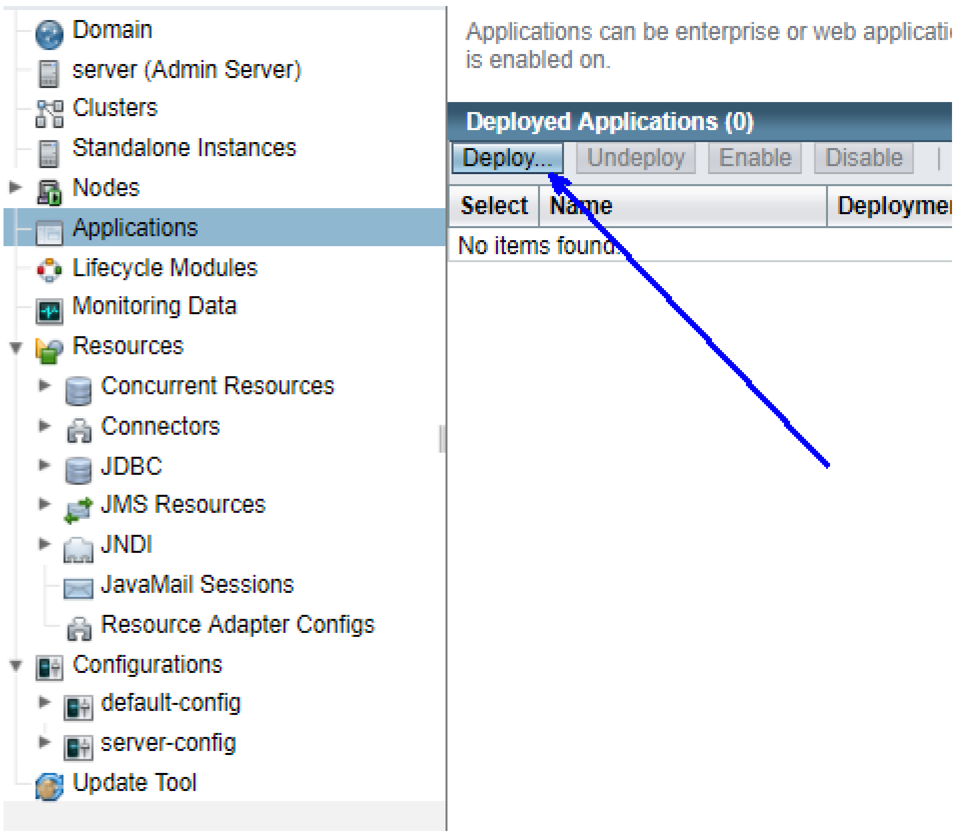
Since we used GlassFish 4.1.1, it is now required to run the application server. GlassFish is automatically installed by installing NetBeans 8.2 IDE or any other IDE that supports JavaEE. If not, you can download it from here: https://javaee.github.io/glassfish/download

After Installing GlassFish 4.1.1:

- Open the glassfish admin console (after starting GlassFish, it’s generally available at http://localhost:4848/);
- Locate the file “travlendar.war” that is available in the delivery folder of the repository (bin) ;
- In the Glassfish Admin Console, click on “Application” from the left-sidebar menu



- Click on "Deploy"

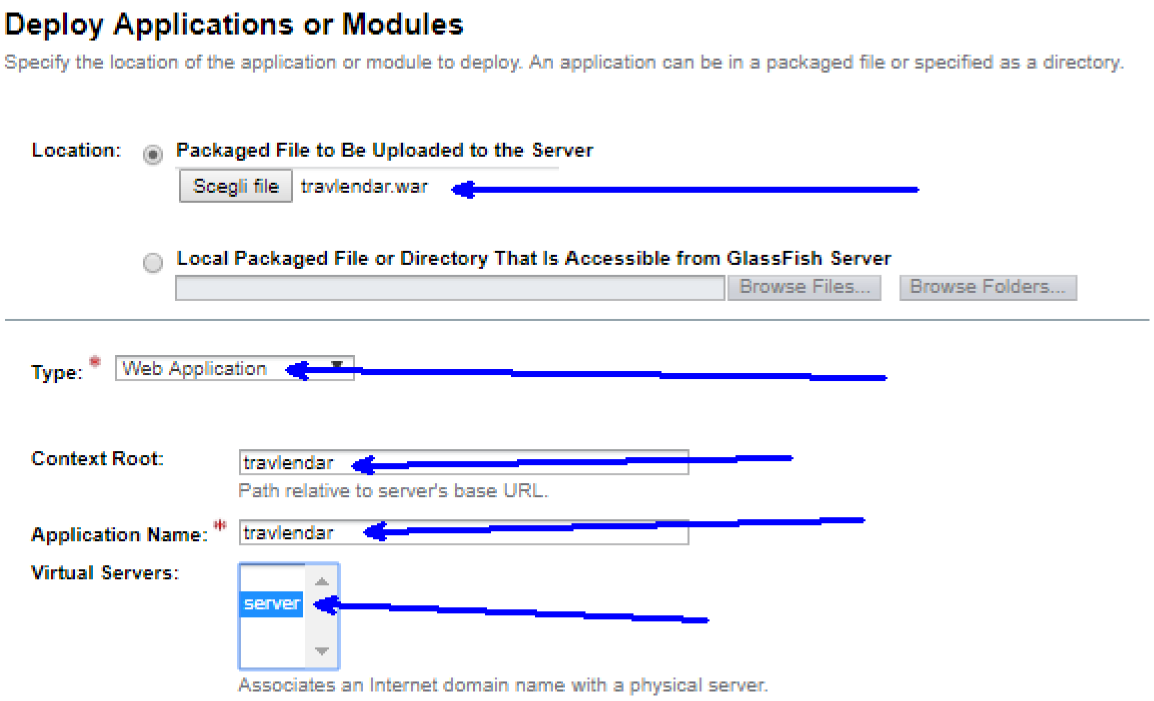


Applications can be enterprise or web applicati
is enabled on.

Deployed Applications (0)

Select	Name	Deployme
No items found.		

- On the field "Packaged File To Be Uploaded To The Server", select the "travlendar.war" file located before and complete as follows:



Deploy Applications or Modules

Specify the location of the application or module to deploy. An application can be in a packaged file or specified as a directory.

Location: (●) Packaged File to Be Uploaded to the Server
Scegli file travlendar.war
( ) Local Packaged File or Directory That Is Accessible from GlassFish Server

Type: * Web Application

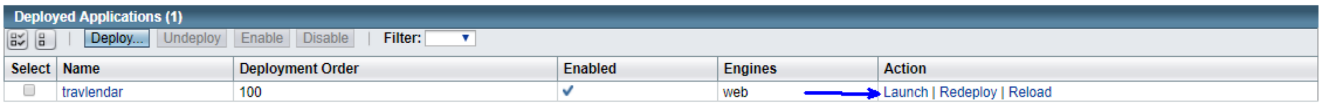
Context Root: travlendar
Path relative to server's base URL.

Application Name: * travlendar

Virtual Servers: server
Associates an Internet domain name with a physical server.

Leave blank all the other fields and click "Ok" in the bottom right corner of the page;

- Now click "Launch"



Deployed Applications (1)

Select	Name	Deployment Order	Enabled	Engines	Action
☐	travlendar	100	✓	web	Launch | Redeploy | Reload

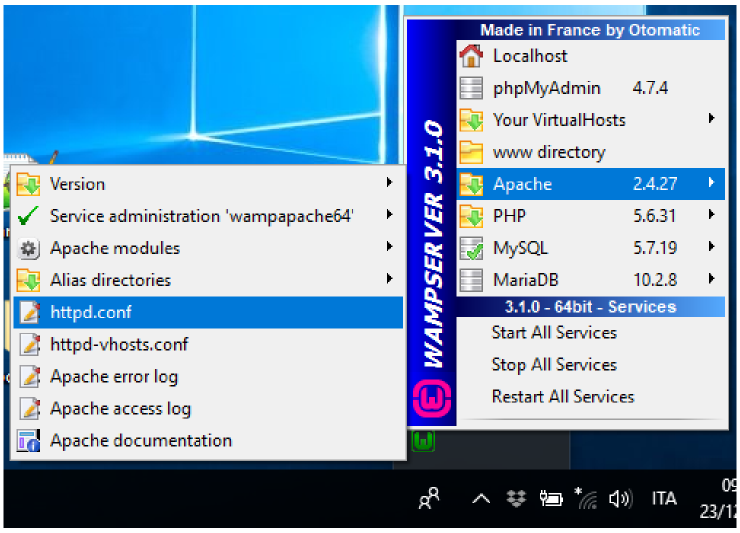
- If the procedure has been followed correctly, by accessing http://127.0.0.1:8080/travlendar/ you should obtain the following page:

TODO write content

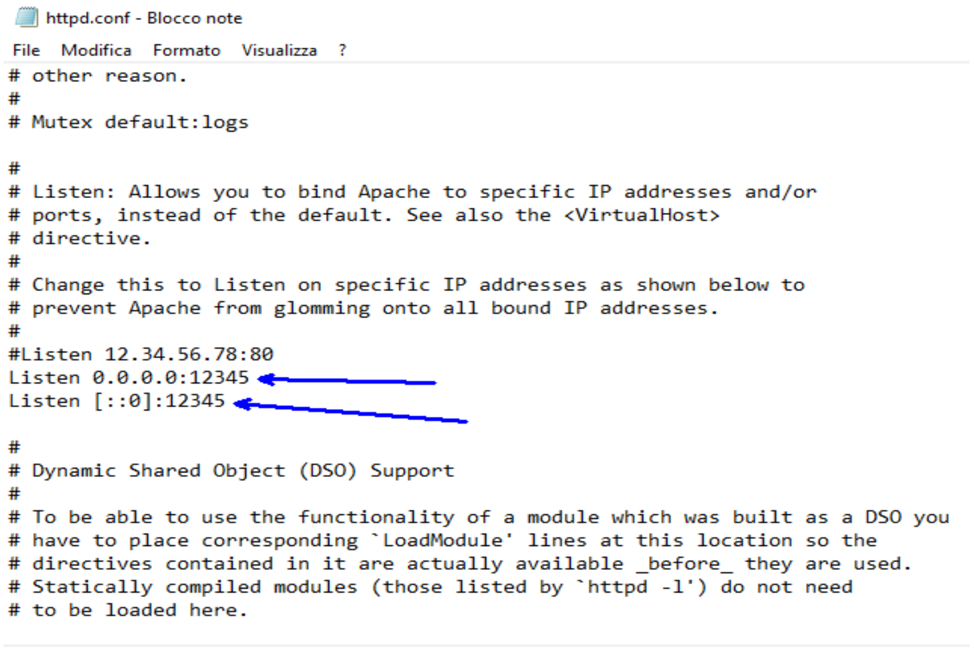
8.3.2 DBMS

To host the DataBase, we user WAMP 3.1.0 (or 3.0.6), you can download it from here:
http://www.wampserver.com/en/download-wampserver-64bits/
If you can, download the x64 version, otherwise x32 will work fine a anyway.
After installing it:

- **[Optional, but adviced]** By opening the menu from the tray icon, open the file httpd.conf



Change the listening port to 12345. This will affect ONLY the access to the administration port vie browser.

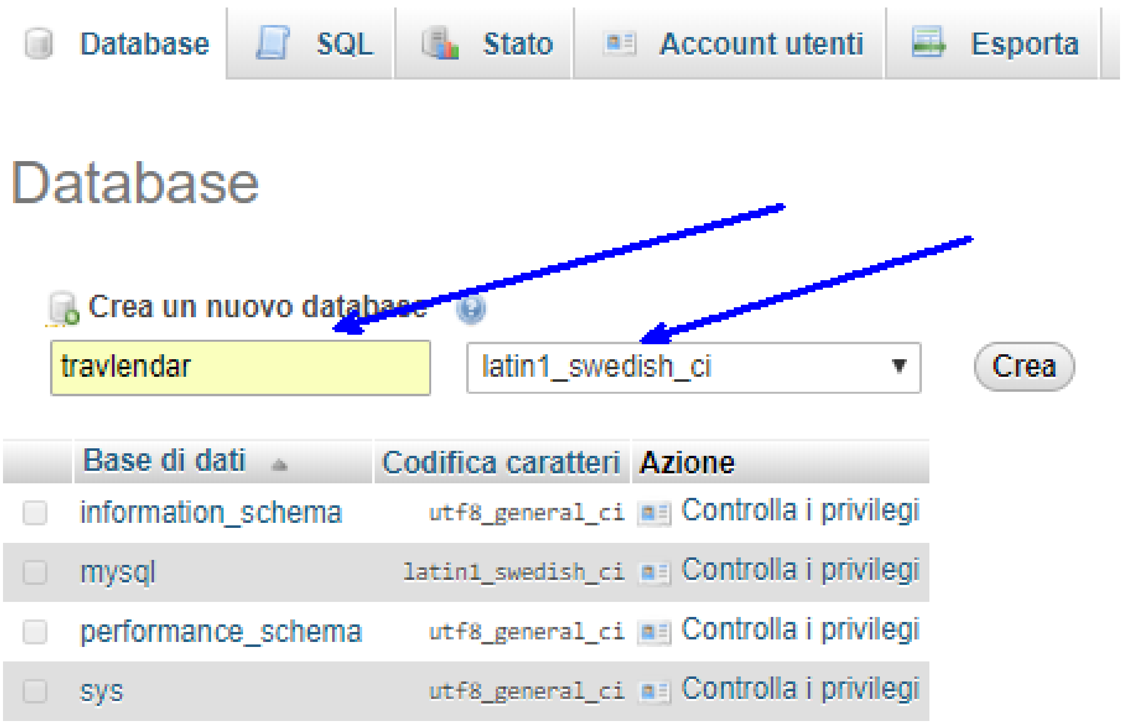


```
# other reason.
#
# Mutex default:logs

#
# Listen: Allows you to bind Apache to specific IP addresses and/or
# ports, instead of the default. See also the <VirtualHost>
# directive.
#
# Change this to Listen on specific IP addresses as shown below to
# prevent Apache from glomming onto all bound IP addresses.
#
#Listen 12.34.56.78:80
Listen 0.0.0.0:12345
Listen [::0]:12345

#
# Dynamic Shared Object (DSO) Support
#
# To be able to use the functionality of a module which was built as a DSO you
# have to place corresponding `LoadModule' lines at this location so the
# directives contained in it are actually available _before_ they are used.
# Statically compiled modules (those listed by `httpd -l') do not need
# to be loaded here.
```

- Click "start all services" on the tray icon and open the browser, then access http://localhost:12345/phpmyadmin/
- If you're asked a password for the user "root", well there's no default password, just go ahead.
- In the "Database", create a new database called "travlendar". Be careful about the name and spell it correctly since the server by default accesses a database with this name.

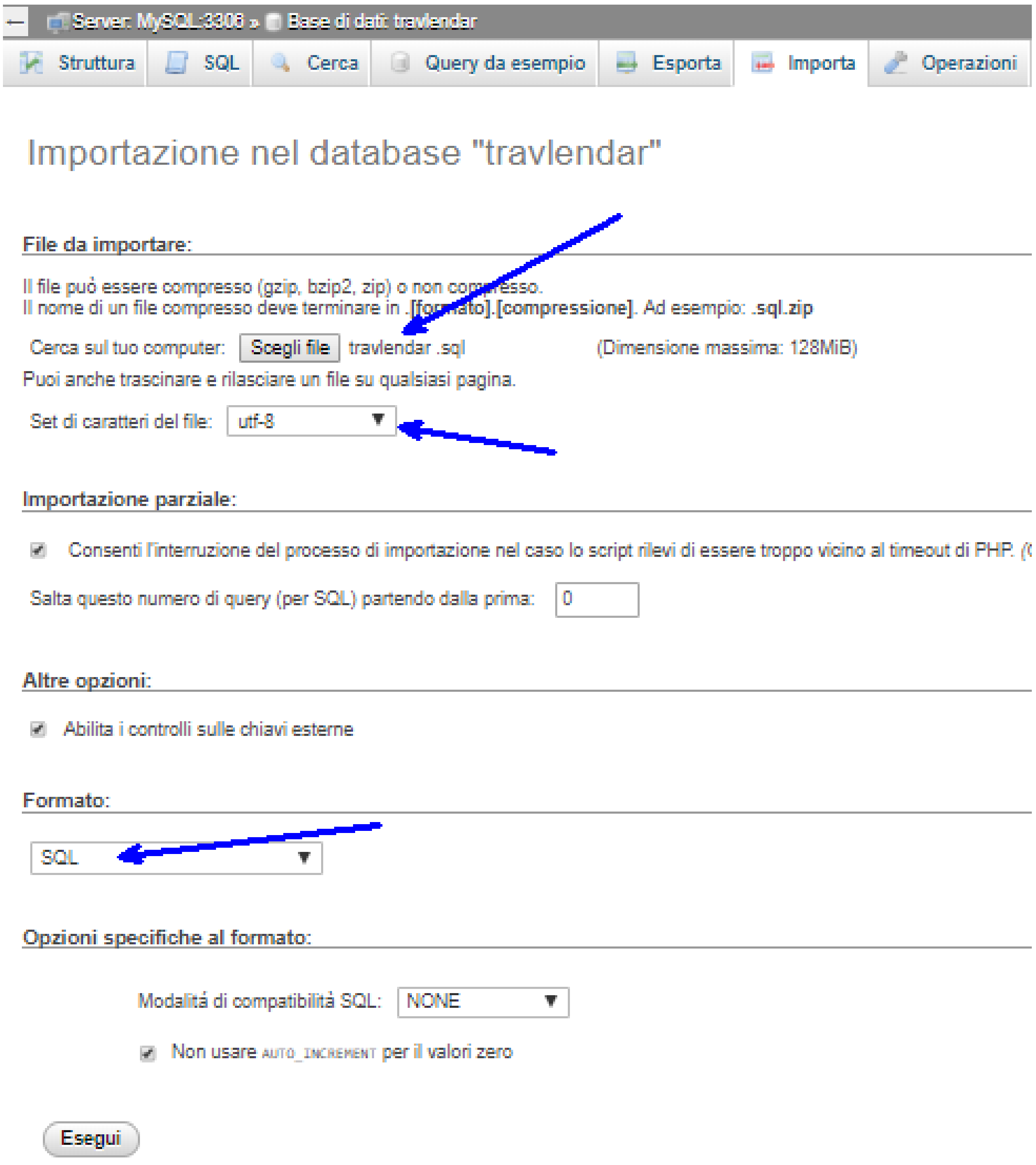


travlendar

latin1_swedish_ci

	Base di dati	Codifica caratteri	Azione
☐	information_schema	utf8_general_ci	Controlla i privilegi
☐	mysql	latin1_swedish_ci	Controlla i privilegi
☐	performance_schema	utf8_general_ci	Controlla i privilegi
☐	sys	utf8_general_ci	Controlla i privilegi

- Select the travlendar database and open the "import" ("importa") section. As showed in the picture below, select the travlendar.sql file available in the delivery folder and click on the "esegui" button.


Server: MySQL:3306 » Base di dati: travlendar

Struttura | SQL | Cerca | Query da esempio | Esporta | Importa | Operazioni

Importazione nel database "travlendar"

File da importare:

Il file può essere compresso (gzip, bzip2, zip) o non compresso.
Il nome di un file compresso deve terminare in **.[formato].[compressione]**. Ad esempio: **.sql.zip**

Cerca sul tuo computer: Scegli file travlendar .sql (Dimensione massima: 128MiB)

Puoi anche trascinare e rilasciare un file su qualsiasi pagina.

Set di caratteri del file: utf-8

Importazione parziale:

✓ Consenti l'interruzione del processo di importazione nel caso lo script rilevi di essere troppo vicino al timeout di PHP. (

Salta questo numero di query (per SQL) partendo dalla prima: 0

Altre opzioni:

✓ Abilita i controlli sulle chiavi esterne

Formato:

SQL

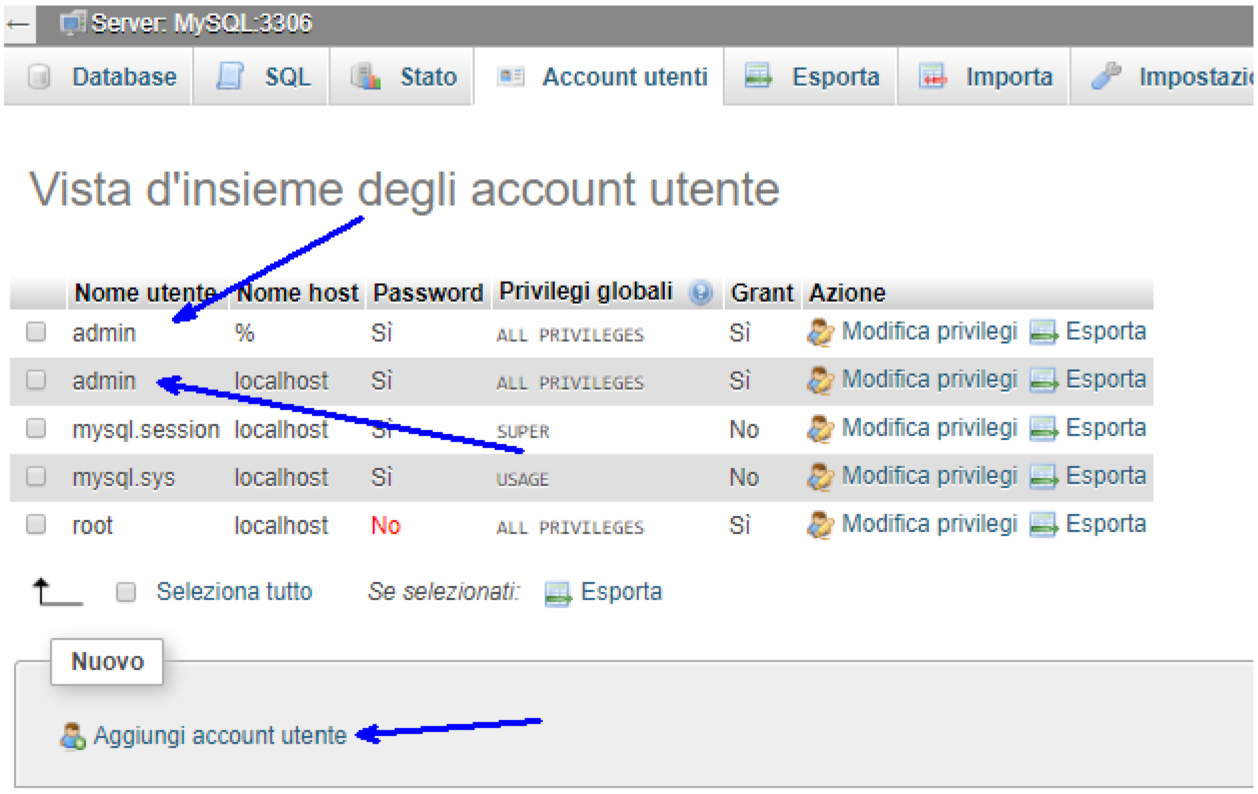
Opzioni specifiche al formato:

Modalità di compatibilità SQL: NONE

✓ Non usare AUTO_INCREMENT per il valori zero

Esegui

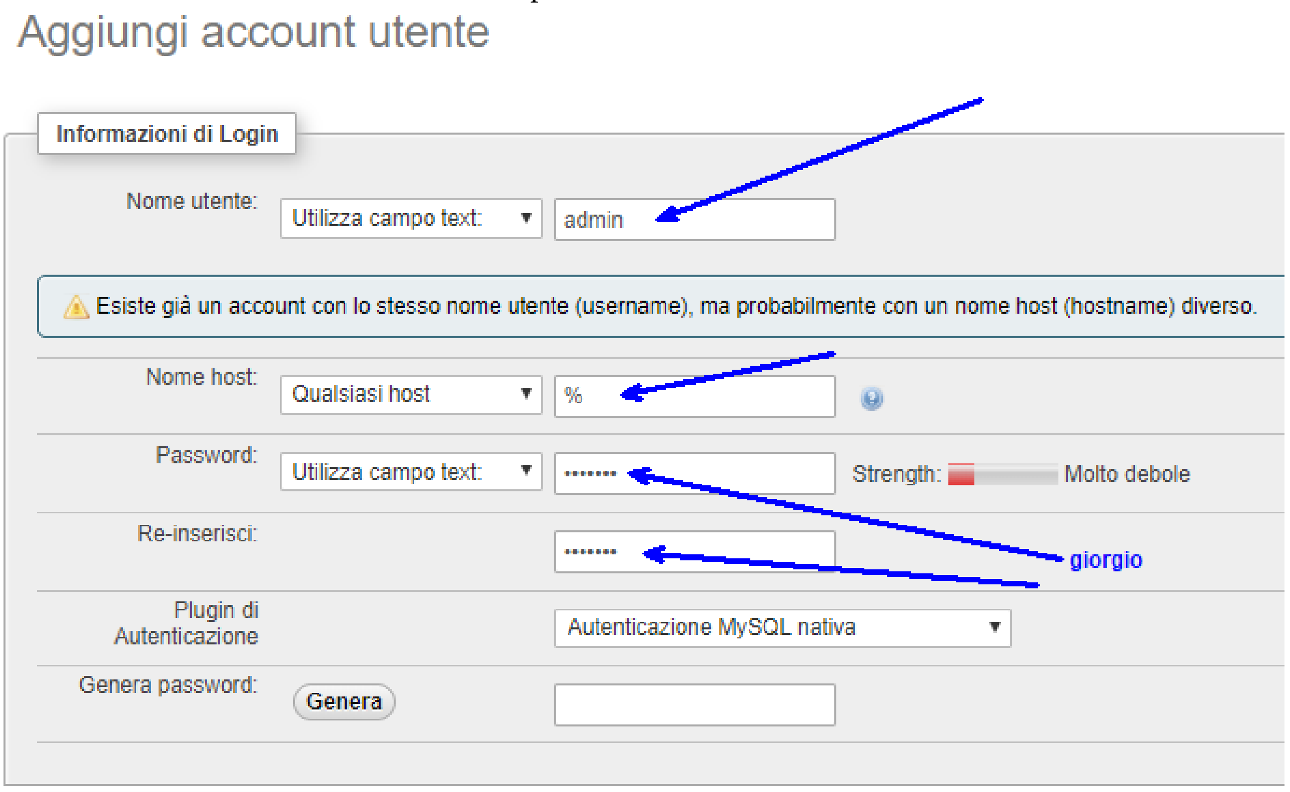
- Head back to http://localhost:12345/phpmyadmin/index.php
- Our server, by default, accesses the DB with a user that has username=”admin” and password=”giorgio” (don’t question why, we didn’t question ourselves).



Vista d'insieme degli account utente

	Nome utente	Nome host	Password	Privilegi globali	Grant	Azione
☐	admin	%	Sì	ALL PRIVILEGES	Sì	Modifica privilegi Esporta
☐	admin	localhost	Sì	ALL PRIVILEGES	Sì	Modifica privilegi Esporta
☐	mysql.session	localhost	Sì	SUPER	No	Modifica privilegi Esporta
☐	mysql.sys	localhost	Sì	USAGE	No	Modifica privilegi Esporta
☐	root	localhost	No	ALL PRIVILEGES	Sì	Modifica privilegi Esporta

To create them, click on “Aggiungi account Utente” and compile as follows:


Aggiungi account utente

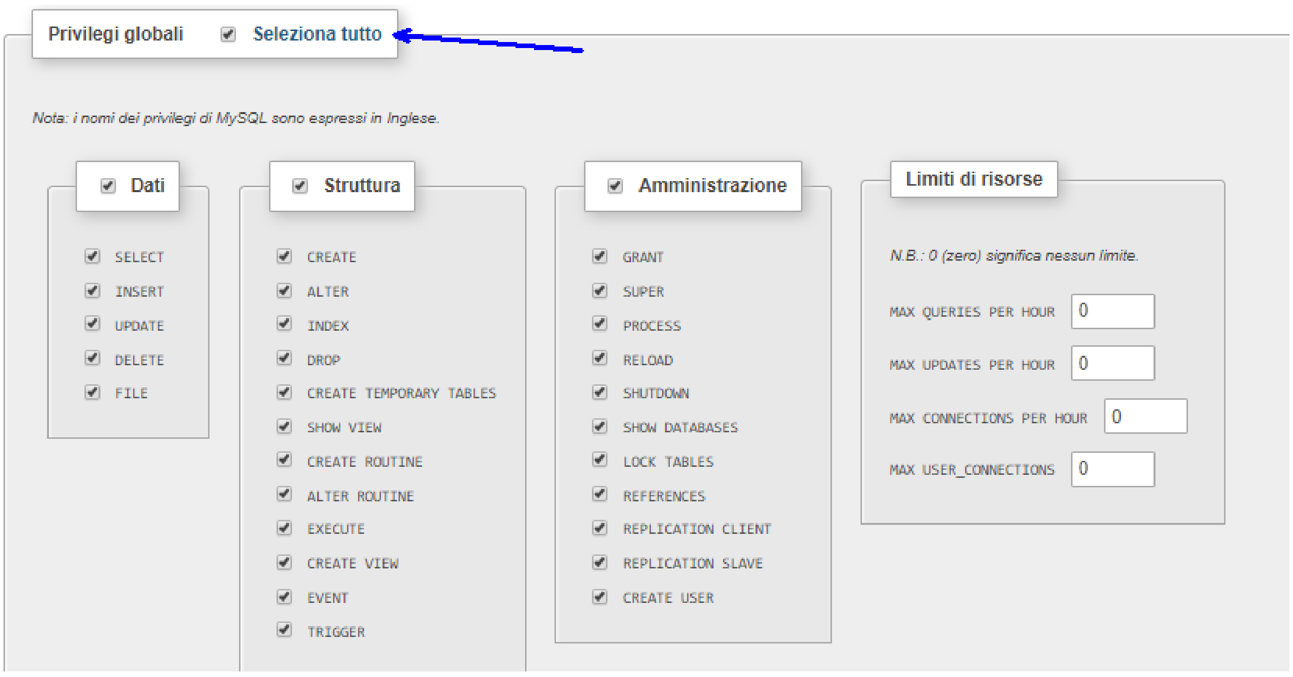
Informazioni di Login

Nome utente:	Utilizza campo text:	admin

Esiste già un account con lo stesso nome utente (username), ma probabilmente con un nome host (hostname) diverso.

Nome host:	Qualsiasi host	%
Password:	Utilizza campo text:	••••••• Strength: Molto debole
Re-inserisci:		•••••••
Plugin di Autenticazione		Autenticazione MySQL nativa
Genera password:	Genera	

Plus, give the user “All privileges”.


Privilegi globali ✓ Seleziona tutto

Nota: i nomi dei privilegi di MySQL sono espressi in Inglese.

✓ Dati
- ✓ SELECT
- ✓ INSERT
- ✓ UPDATE
- ✓ DELETE
- ✓ FILE

✓ Struttura
- ✓ CREATE
- ✓ ALTER
- ✓ INDEX
- ✓ DROP
- ✓ CREATE TEMPORARY TABLES
- ✓ SHOW VIEW
- ✓ CREATE ROUTINE
- ✓ ALTER ROUTINE
- ✓ EXECUTE
- ✓ CREATE VIEW
- ✓ EVENT
- ✓ TRIGGER

✓ Amministrazione
- ✓ GRANT
- ✓ SUPER
- ✓ PROCESS
- ✓ RELOAD
- ✓ SHUTDOWN
- ✓ SHOW DATABASES
- ✓ LOCK TABLES
- ✓ REFERENCES
- ✓ REPLICATION CLIENT
- ✓ REPLICATION SLAVE
- ✓ CREATE USER

Limiti di risorse

N.B.: 0 (zero) significa nessun limite.

MAX QUERIES PER HOUR	0
MAX UPDATES PER HOUR	0
MAX CONNECTIONS PER HOUR	0
MAX USER_CONNECTIONS	0

Click on “Esegui”.

Repeat the procedure exactly the same way, but selecting “localhost” in the “Nome host” (“hostname”) field.

- *Note that by default the server accesses the database if is on the same machine (that is to say, it looks for a MySQL database at 127.0.0.1). This means that that WAMP should be installed on the same machine that will run the application server. Since a remote administration console for the server is still not available, if you do/did any different, you’ll have to change the ip address in the class DataLayer of the server and recompile it.*
- If the whole procedure has been followed correctly, when using any of the clients we developed everything will work correctly. Otherwise, any request done from a client to the server will result in a “Connection Error” as response.

9. Source Code

The source code is available in the three projects that are located in "DeliveryFolder". Namely:

- The "travlendar" folder contains the server. We worked on it using both NetBeans and IntelliJ, so it should be easy to import in on both IDEs;
- The "travlendardesktopclient" folder contains the desktop client. It is a simple JavaFX application;
- The "travlendarandroidclient" folder contains the application for Android. It has been developed with Android Studio, which is based on IntelliJ.

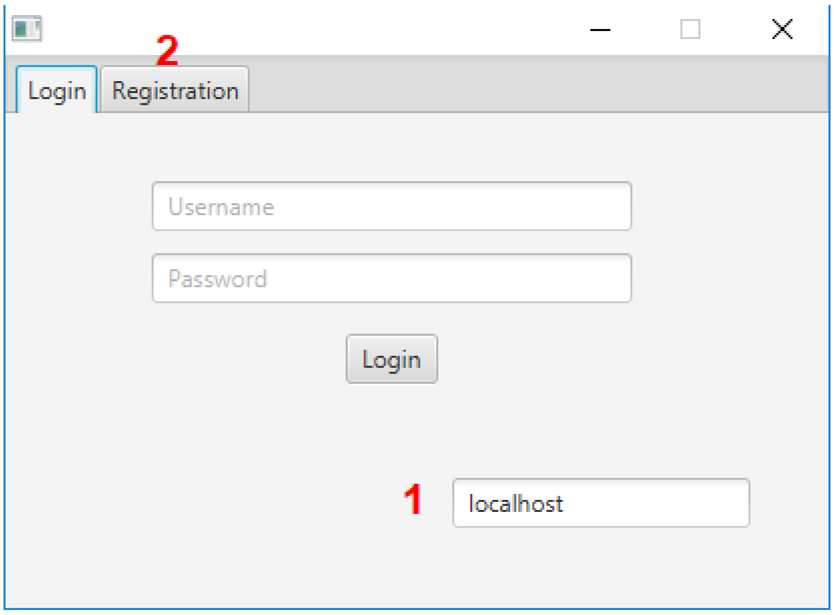
All the projects listed above require some dependencies to work properly (and to compile). They're all available in the "Dependency" folder, so if they're not imported automatically, please import them into the project libraries manually.

The code has been commented for a really high percentage.

10. How clients work

10.1. Desktop Client

In this section **the non-obvious functionalities** of the desktop client will be explained.

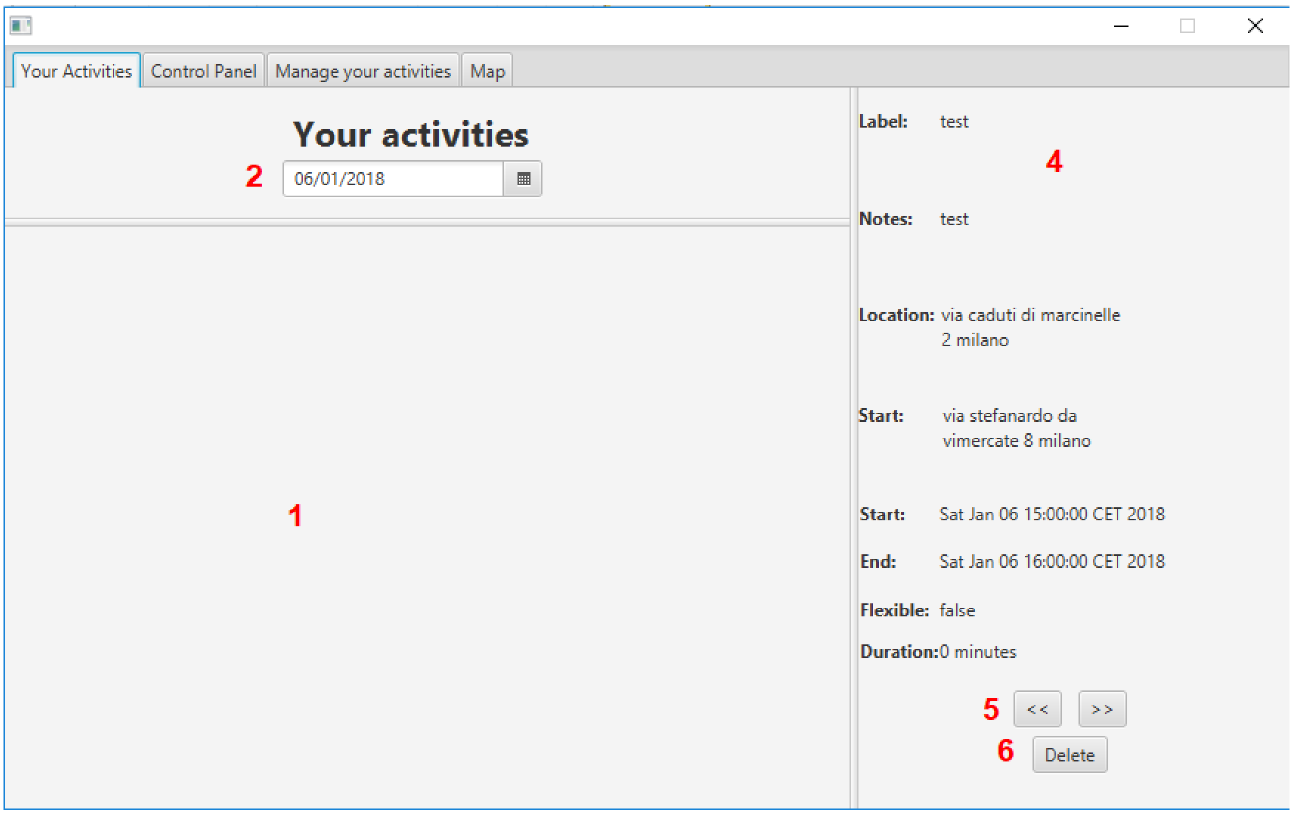


In the **login screen**, showed above, it's worth it to notice that:

1. this field represents the is IP address (x.y.w.z or localhost) of the server. Since this is a prototype we need this field because we don't have a server with a fixed IP address.
2. switch to registration screen.

The **registration screen** works in the exact same way.

After setting the address in the login or registration screen it will be kept for the entire time the application is being used.



Label: test

Notes: test

Location: via caduti di marcinelle 2 milano

Start: via stefanardo da vimercate 8 milano

Start: Sat Jan 06 15:00:00 CET 2018

End: Sat Jan 06 16:00:00 CET 2018

Flexible: false

Duration:0 minutes

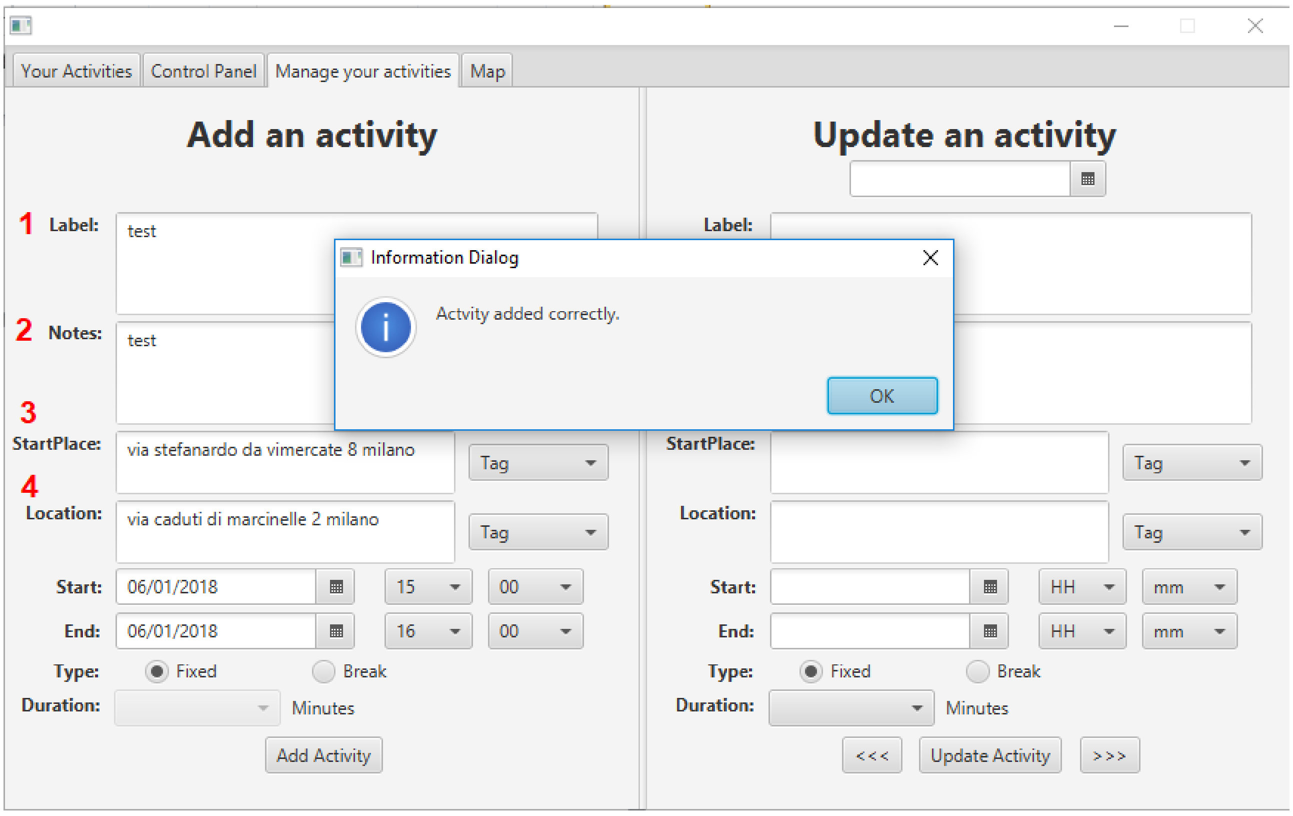
The tab **"Your Activities"** of the main screen is showed immediately after login:

1. This area will contain a user friendly view of the activities of the selected day. Since this is a prototype, it is now unused.
2. By selecting a day, the activities scheduled for that day will be showed in area on the right.

4. In this area activities for the selected day will be showed.

5. Those "<<" and ">>" buttons can be used to navigate among activities of the same day.

6. The “Delete” button can be used to delete from the calendar the activity that is actually being showed.



Label: test

Notes: test

StartPlace: via stefanardo da vimercate 8 milano

Location: via caduti di marcinelle 2 milano

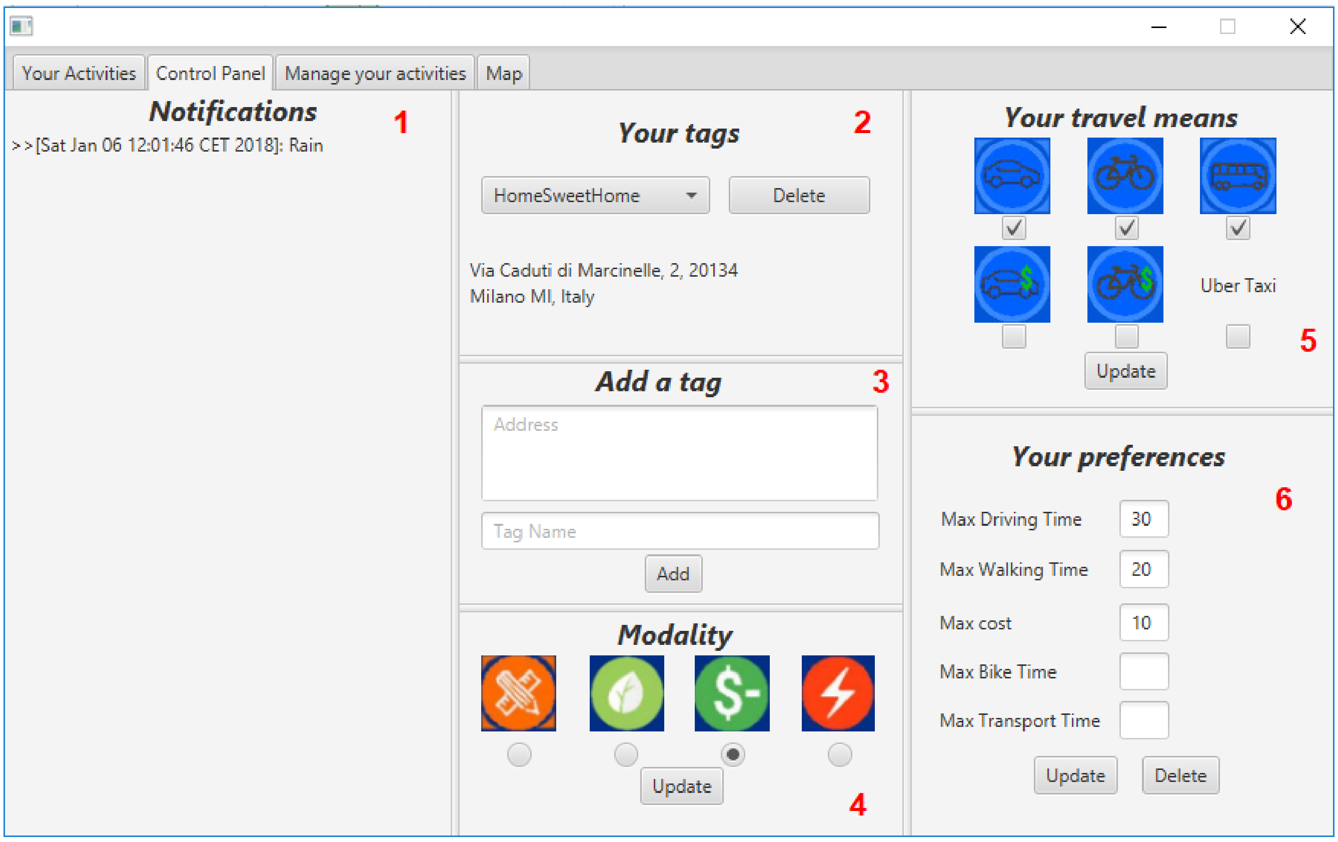
Start: 06/01/2018 15 00

End: 06/01/2018 16 00

Activty added correctly.

In the “Manage your activities” tab, showed above:

1. **Label** field is **a single word** that represents the activity.
2. **Notes** field can be empty or up to 200 letters.
3. **StartPlace** field represents the place from which the user is supposed to leave to go to the real location of the activity.
4. **Location** field is the real location of the activity.



>>[Sat Jan 06 12:01:46 CET 2018]: Rain

HomeSweetHome

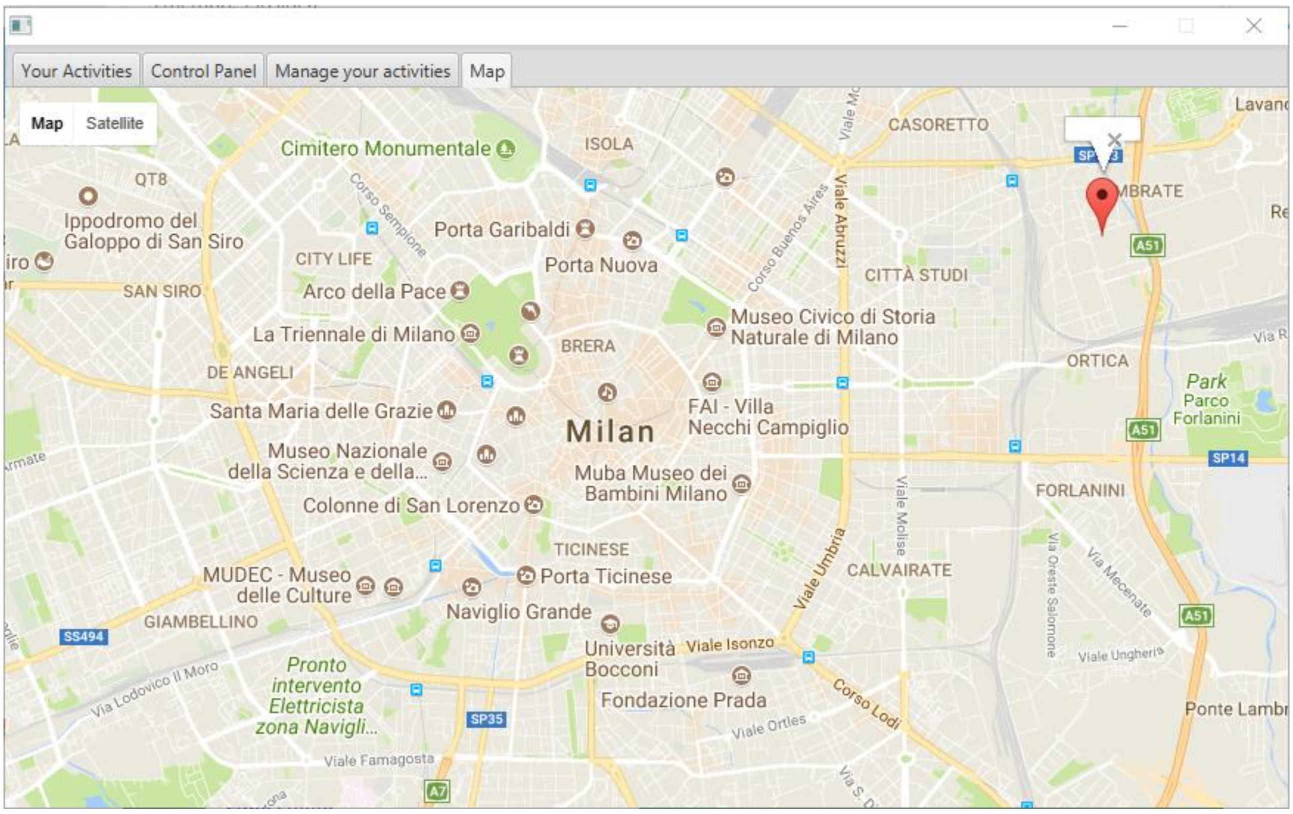
Via Caduti di Marcinelle, 2, 20134 Milano MI, Italy

Max Driving Time	30
Max Walking Time	20
Max cost	10
Max Bike Time	
Max Transport Time	

In the **"Control Panel"** tab:

1. The notifications area will show notifications about weather and the notifications that tell the user when it's time to leave for a specific activity. Notifications are updated every minute.
2. This area is used to show already existing tags (see definition of tag in section 1.3.1) with their corresponding addresses and to delete them.
3. This section is used to create new tags. Note that the "Tag Name" field has to be a single word.
4. Starting from the left, first modality is "standard", second modality is "minimize carbon footprint", third modality is "Minimize cost" and the fourth modality is "Minimize time". As for now, only the first modality is implemented.
5. This section is used to select the travel means the user wants to use. In order (left to right, top to bottom): car, bike, public transport, car sharing, bike sharing, Uber/Taxi.
6. This kind of preference is expressed in minutes. An empty field means that no preference has been

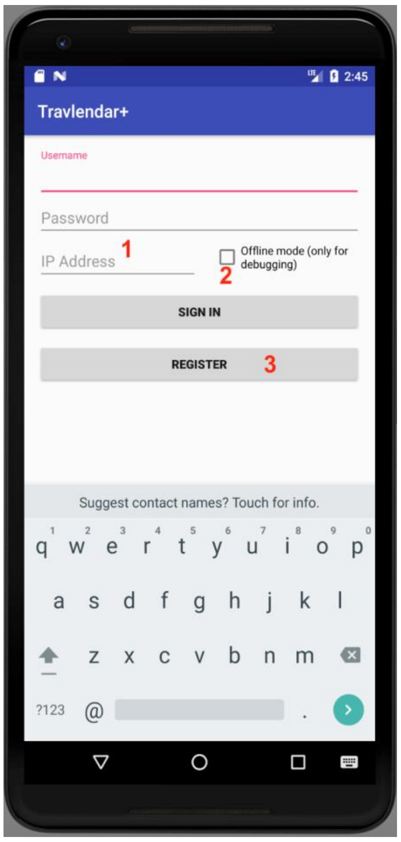
expressed **(so it's different from 0!)**. To delete an existing preference, empty the corresponding field and then click on the "Delete" button.



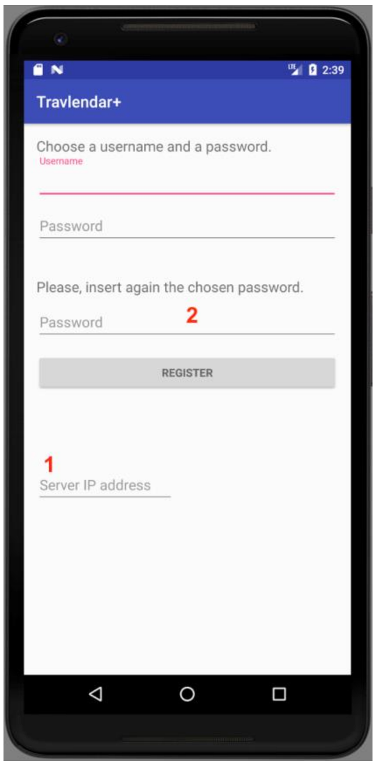
The "Map" tab, for now, is a simple map. We're planning to show user's tags on the map and to include a search bar to allow users to look for specific places.

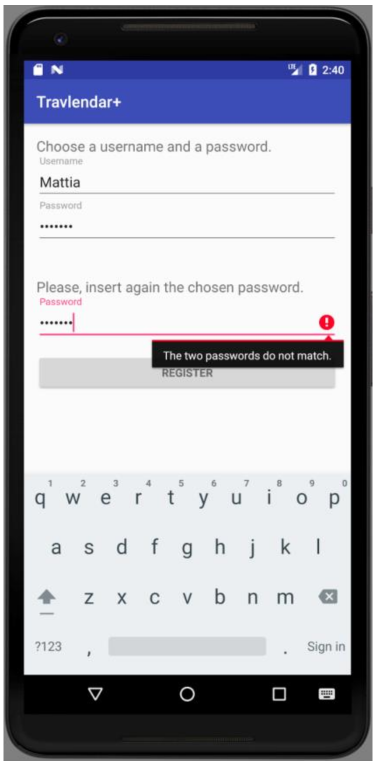
10.2. Android Client

As previously, here we present the **non-obvious functionalities** and **design choices** of the mobile client, in this case an Android application released as an APK (Android Package Kit) file.

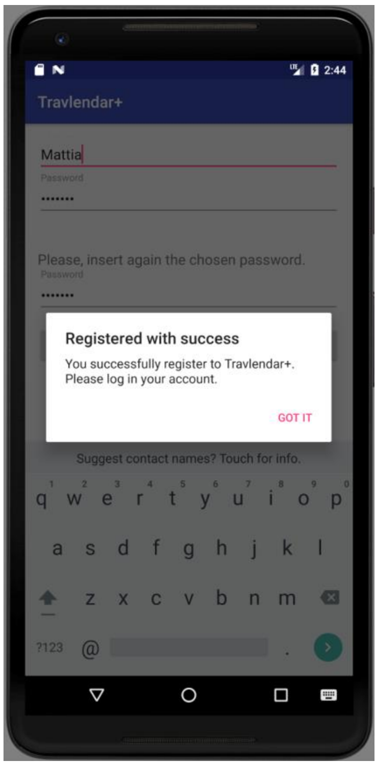

2:45
Travlendar+
Username
Password
IP Address
1
Offline mode (only for debugging)
2
SIGN IN
REGISTER
3
Suggest contact names? Touch for info.

The first screen (aka activity in Android) to be shown, after a short loading one, is the **login activity** shown in the second screenshot above. It contains the trivial fields for username and password, as the field for the server IP address (1). This is necessary in this prototype since we don't have a server with its fixed IP address. The activity also contains a check-box (2), thought to be used for offline debugging of layout rendering, and a button (3) in order to call the **registration activity**.


2:39
Travlendar+
Choose a username and a password.
Username
Password
Please, insert again the chosen password.
Password
2
REGISTER
1
Server IP address


2:40
Travlendar+
Choose a username and a password.
Username
Mattia
Password
Please, insert again the chosen password.
Password
REGISTER
Sign in

The two passwords do not match.

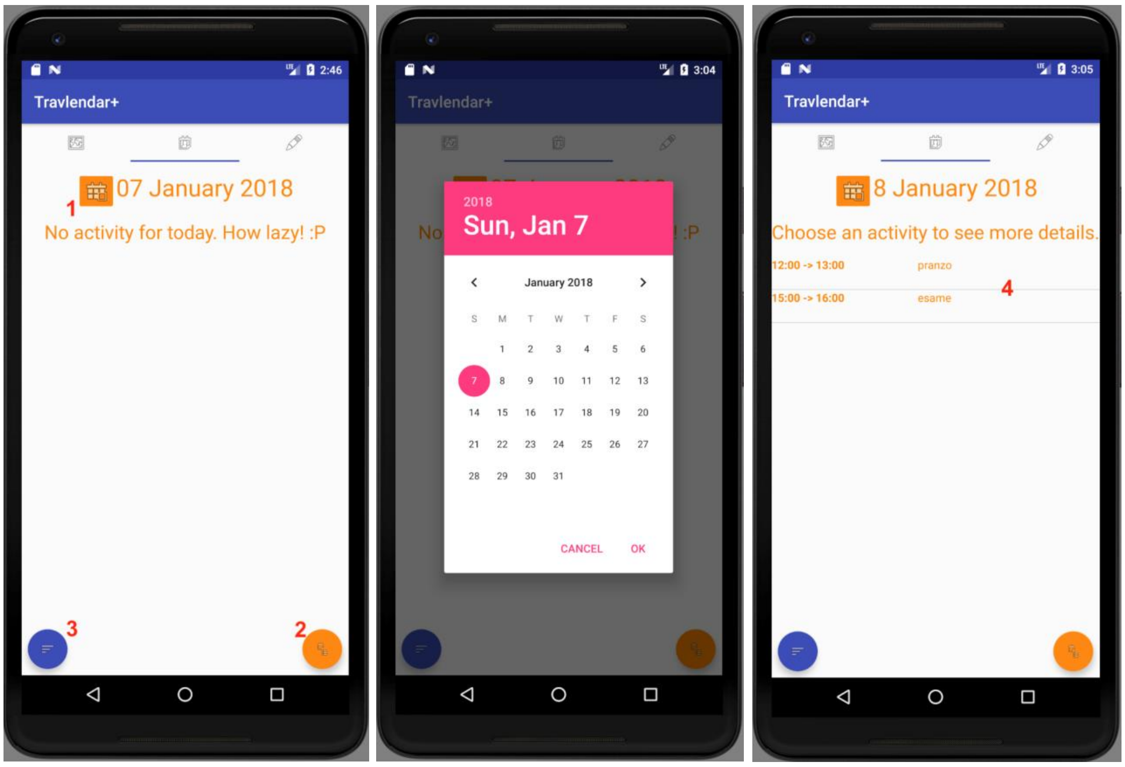

2:44
Travlendar+
Mattia
Password
Please, insert again the chosen password.
Password
GOT IT
Suggest contact names? Touch for info.

Registered with success

You successfully register to Travlendar+.
Please log in your account.

The **registration activity** above too contains a field (1) for the IP address of the server for technical reasons (as pointed previously). We decided to implement the registration routine with trivial username and password fields and one more field (2) in which the user is asked to confirm the chosen password. If the user successfully creates a new account on the server, a dialog pops-up notifying the action and the user is forwarded back to the **login activity**.

Once the user in logged in, a tab-structured view is shown. Below the three tabs are presented.



07 January 2018

No activity for today. How lazy! :P

Sun, Jan 7

January 2018

8 January 2018

Choose an activity to see more details.

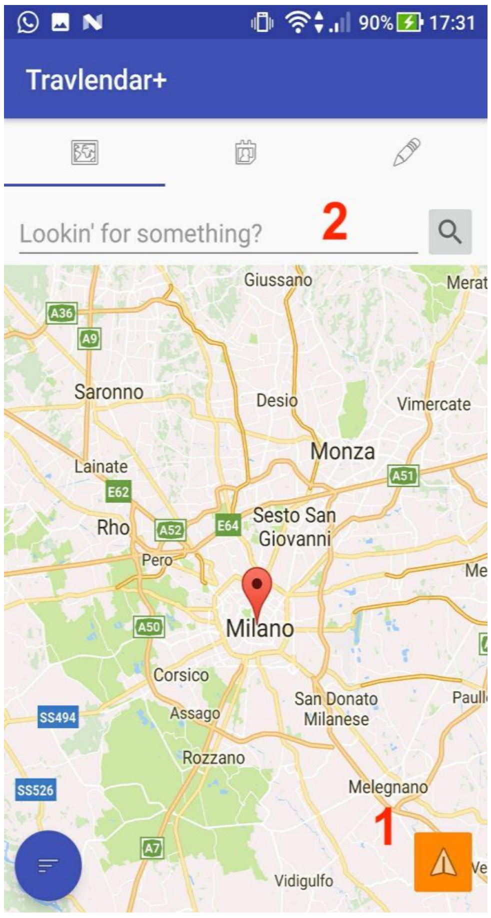
12:00 -> 13:00 pranzo

15:00 -> 16:00 esame

The **calendar tab** shown above contains three buttons:

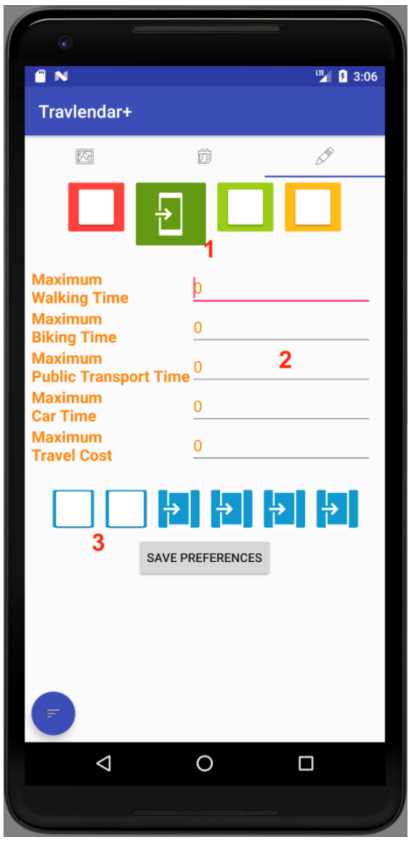
- a image-button (1) which displays a date picker (i.e. a calendar). If a day is chosen, a list (4) of the scheduled activities of the user for that day is displayed. If an activity row is clicked, the **update activity** (discussed below) is called.
- a floating-button (2) to add a new activity (see **new activity screen**) on the bottom right edge of the display
- a floating-button (3) to access the **settings activity** where the user can check, delete and add his tags (i.e. his favorite places), and read some FAQs about the application.

By a complete scroll on the left from the **calendar tab**, the user reaches the **map tab**.



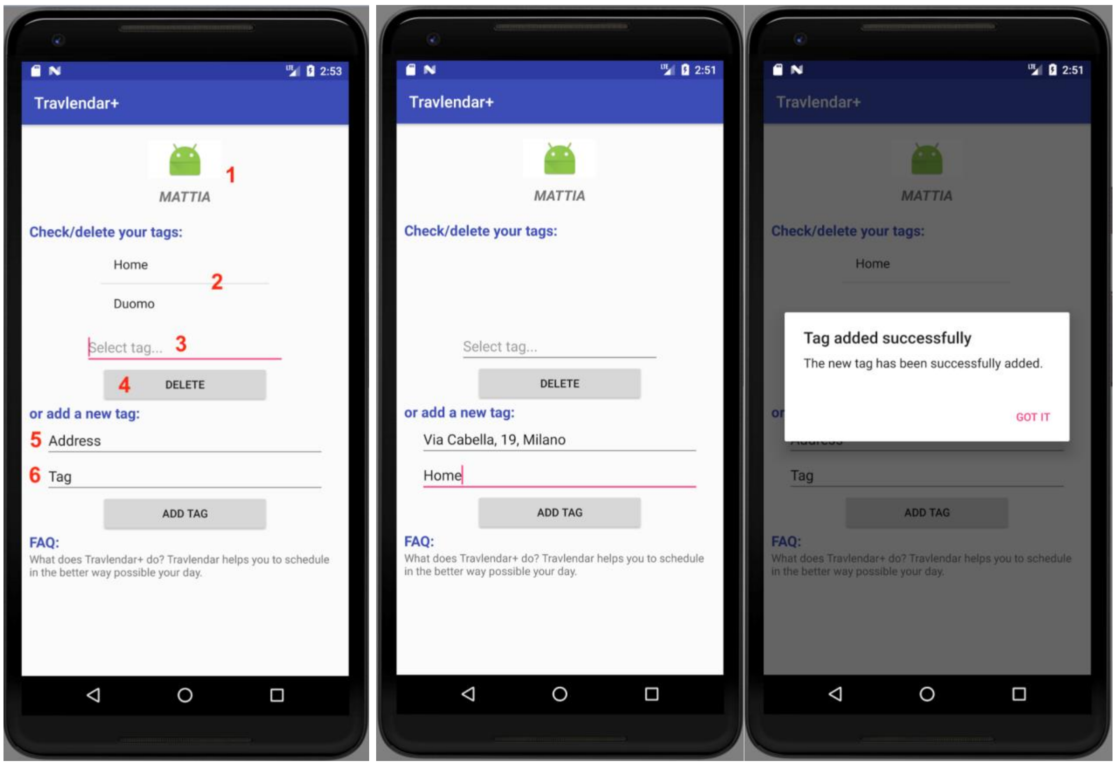
This tab displays a map provided by Google Maps APIs, a floating button (1) on the right bottom part which calls the **indications activity** (presented afterwards) and a search bar (2) to search location in the map, by means of Google Maps APIs support.

By a complete scroll on the right from the **calendar tab,** the user reaches the **preferences tab.**

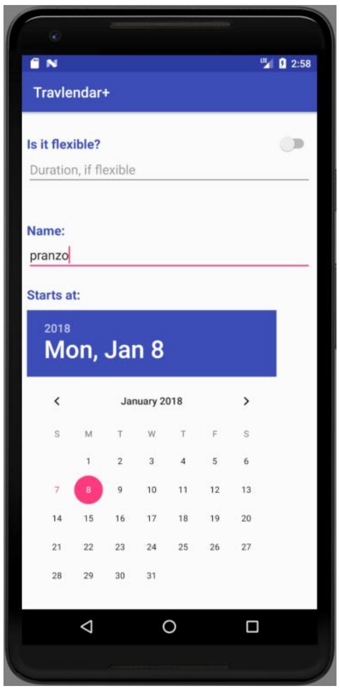


Maximum Walking Time	0
Maximum Biking Time	0
Maximum Public Transport Time	0
Maximum Car Time	0
Maximum Travel Cost	0

Here, the user can choose one of the four available mode (1) offered by the application (from left to right: minimize time, minimize cost, minimize carbon-footprint and customizable mode) and modify his travel preferences, such as max time with each means (2) and travel means to be consider for the route (3). Notice that the left bottom floating-button for settings remains visible in each tab.



The **settings activity** is accessed by means of the bottom left floating-button displayed in each initial tab. This activity contains a simple profile icon and the username of the current logged-in user (1). Below, there is a list view (2) where the user can check his tags. The text field (3) and the button (4) are intended to be used to select (by means of a dialog) and delete a tag. The activity also allows the user to add a new tag providing an address (5) and a name (6) for the new tag. The result of deletion/addition is reported to the user by means of a dialog (an example in shown in the screenshot above).



Is it flexible?

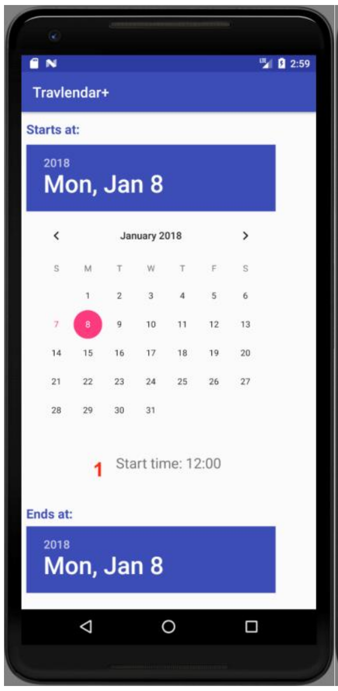
Duration, if flexible

Name:

pranzo

Starts at:

2018
Mon, Jan 8



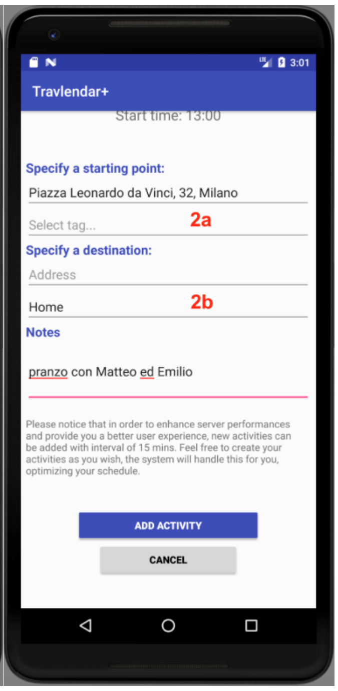
Starts at:

2018
Mon, Jan 8

Start time: 12:00

Ends at:

2018
Mon, Jan 8



Start time: 13:00

Specify a starting point:

Piazza Leonardo da Vinci, 32, Milano

Select tag...

Specify a destination:

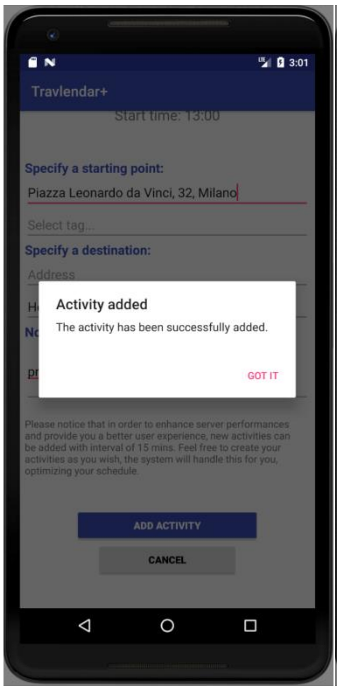
Address

Home

Notes

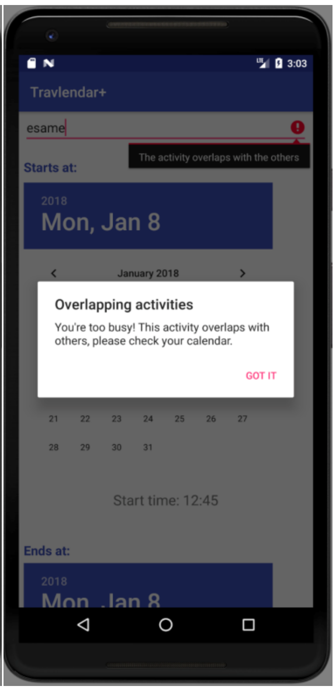
pranzo con Matteo ed Emilio

Please notice that in order to enhance server performances and provide you a better user experience, new activities can be added with interval of 15 mins. Feel free to create your activities as you wish, the system will handle this for you, optimizing your schedule.



Activity added

The activity has been successfully added.



esame

The activity overlaps with the others

Starts at:

2018
Mon, Jan 8

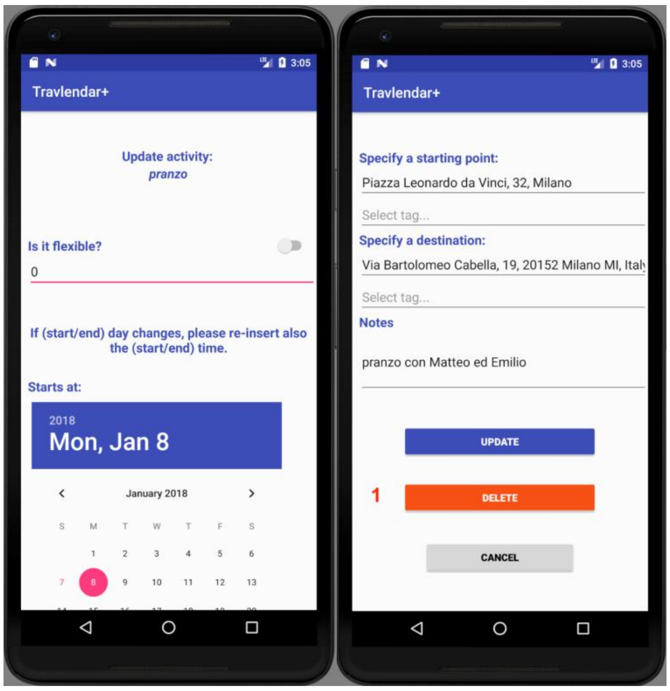
Overlapping activities

You're too busy! This activity overlaps with others, please check your calendar.

Start time: 12:45

Ends at:

The **new activity screen** is called with the bottom right button in the calendar tab. This screen is structured as a vertical scroll-view and contains all the fields needed to the user to add a new activity in his calendar. Here, we highlight just the behaviour of time picker text views (1) and tag choice text fields (2a, 2b). The text views labelled as "Select time…" allow the user to select start and end time for the activity, by means of a time picker dialog. The tag choice text fields allow the user to select one of his favorite places, through a dialog.


Travlendar+

Update activity:
pranzo

Is it flexible?
0

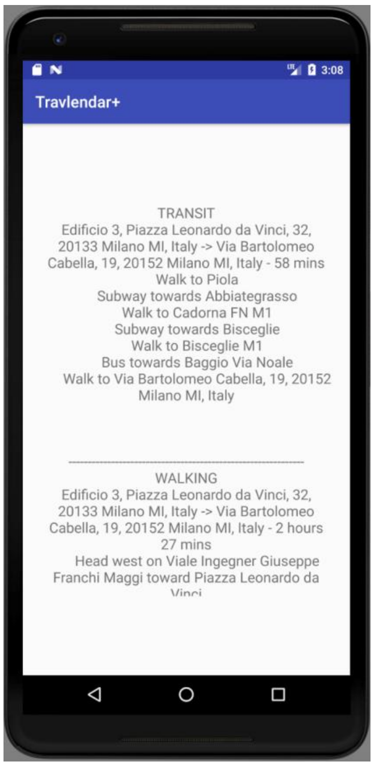
If (start/end) day changes, please re-insert also the (start/end) time.

Starts at:
2018
Mon, Jan 8

Travlendar+

Specify a starting point:
Piazza Leonardo da Vinci, 32, Milano
Select tag...
Specify a destination:
Via Bartolomeo Cabella, 19, 20152 Milano MI, Italy
Select tag...
Notes
pranzo con Matteo ed Emilio

The **update activity** can be called by clicking on an activity in the calendar tab list view. It works as the **new activity screen** with the addition of allowing the user to delete the selected activity from his calendar by means of the “Delete” button (1).


Travlendar+

TRANSIT
Edificio 3, Piazza Leonardo da Vinci, 32, 20133 Milano MI, Italy -> Via Bartolomeo Cabella, 19, 20152 Milano MI, Italy - 58 mins
Walk to Piola
Subway towards Abbiategrasso
Walk to Cadorna FN M1
Subway towards Bisceglie
Walk to Bisceglie M1
Bus towards Baggio Via Noale
Walk to Via Bartolomeo Cabella, 19, 20152 Milano MI, Italy

WALKING
Edificio 3, Piazza Leonardo da Vinci, 32, 20133 Milano MI, Italy -> Via Bartolomeo Cabella, 19, 20152 Milano MI, Italy - 2 hours 27 mins
Head west on Viale Ingegner Giuseppe Franchi Maggi toward Piazza Leonardo da Vinci

The **indications activity** is accessed by means on the bottom right floating button in the map tab. This screen displays the indications for the most imminent activity in the user's calendar. The screen contains drive, walking, bicycle and public-transportation indications for the user.

11. Contact us

In order to speed up the operations required to complete the assigned tasks, in case you have any problems while following the provided installation instructions or while importing the projects to inspect the source code, feel free to contact us at:

- Matteo Biasielli: matteo.biasielli@mail.polimi.it
  Mobile Phone/WhatsApp: +39 3333687411

- Emilio Capo: emilio.capo@mail.polimi.it
  Mobile Phone/WhatsApp: +39 3348361831

- Mattia Di Fatta: mattia.difatta@mail.polimi.it
  Mobile Phone/WhatsApp: +39 3406118677

12. Effort Spent

Matteo Biasielli, matr. 893590

Section(s)	**Number of hours**
18-nov-17 Group Review	4
18-nov-17 Login & Registration & Delete Tag	4
19-nov-17 Delete Tag & Add Tag	1,5
19-nov-17 AddActivity	3,5
21-nov-17 TravelServlet	3
24-nov-17 Desktop Client	5
25-nov-17 Desktop Client	3
26-nov-17 Desktop Client	3
28-nov-17 Group meeting	2
30-nov-17 repository management and Desktop Client	5
02-dic-17 Group meeting	5
05-dic-17 Map implementation on windows client	
06-dic-17 Group meeting	2
06-dic-17 Testing	2
07-dic-17 Testing	2
12-dic-17 Testing	4
13-dic-17 Group meeting	2.5
13-dic-17 Overlapping check optimization	3
14-dic-17 Implementation Document	1
15-dic-17 Implementation Document	3
16-dic-17 Implementation Document	1
20-dic-17 Implementation Document	2
20-dic-17 Group Meeting	2
23-dic-17 Implementation Document	1
27-dic-17 Testing & Document	4
01-jan-18 Implementation Document	1
01-jan-18 Commenting code	2
06-jan-18 Implementation Document	3
07-jan-18 Implementation Document & Debugging	6.5
TOTAL:	81

Mattia Di Fatta, matr. 893608

Section(s)	**Number of hours**
18-nov-17 Group Review	4
19-nov-17 Delete activity + errors	3
20-nov-17 Get calendar + errors	2.5
21-nov-17 Update Activity + errors	3
22-nov-17 started Retrieve Notifications	1
23-nov-17 Retrieve Notifications query	2
24-nov-17 started Retrieve Notifications thread and servlets	3
25-nov-17 ended Retrieve Notifications servlets	2
26-nov-17 thread Retrieve Notification	1.5
28-nov-17 Group meeting	2
02- dic -17 Group meeting	5
3-dec-17 register screen	1.5
6-dec-17 register screen + side menu	2.5
06- dic -17 Group meeting	2
9-dec-17 new fixed activity screen	2.5
10-dec-17 new fixed activity screen	2
13- dic -17 Group meeting	2.5
13-dec-17 new activity screen	3.5
14-dec-17 problem fixing and pop-ups implementation	2
15-dec-17 settings screen	2.5
20-dec-17 settings screen	2.5
21-dec-17 settings screen	2
22-dec-17 settings screen + beginning directions screen	3.5
27-dec-17 general debugging	4.5
28-dec-17 general debugging	4.5
30-dec-17 update/delete activity screen	2
31-dec-17 update/delete activity screen + bug fixing	3.5
1-jan-18 bug fixing	3
2-jan-18 bug fixing	3
4-jan-18 bug fixing	2.5
5-jan-18 bug fixing	4
6-jan-18 bug fixing and last minute improvements	1.5
07-jan-18 Implementation Document	2.5
TOTAL:	89

Emilio Capo, matr. 899842

Section(s)	**Number of hours**
18-nov-17 Group Review	4
20-nov-17 Get Preferences	1
21-nov-17 Update Boolean Preferences Servlet	1
22-nov-17 Bugfix + Delete Ranged Preferences Servlet	1
23-nov-17 Delete Ranged Preferences Servlet	1
24-nov-17 Update Ranged Preferences Servlet	1
26-nov-17 Servlet Testing & Debugging	1
27-nov-17 Android Studio setup for Mobile Client	1.5
28-nov-17 Group Meeting	2
29-nov-17 Mobile Client	4
30-nov-17 Mobile Client	2
02- dic -17 Group Meeting	5
03- dic -17 Mobile Client	3
06- dic -17 Mobile Client	2.5
06- dic -17 Group Meeting	2
10- dic -17 Mobile Client	3
11- dic -17 Mobile Client	1.5
13- dic -17 Mobile Client	1
13- dic -17 Group Meeting	2.5
14-dic-17 Mobile Client	1
19-dic-17 Mobile Client	2
20-dic-17 Group Meeting	2
23-dic-17 Mobile Client	4
27-dic-17 Mobile Client	5.5
29-dic-17 Mobile Client	4.5
30-dic-17 Mobile Client	5
2-gen-18 Mobile Client	6
3-gen-18 Mobile Client	1
4-gen-18 Mobile Client	3
5-gen-18 Mobile Client	4
6-gen-18 Mobile Client	7
7-gen-18 Mobile Client Finalization	9
TOTAL:	94