

**POLITECNICO**  
MILANO 1863



**Safe**Streets

# IMPLEMENTATION AND TEST DELIVERABLE

Version 1 – 12/01/2020

---

*Authors:*

Giulio A. Abbo 10538950

Gianmarco Accordi 10587213

Massimiliano Bonetti 10560496

*Professor:*

Elisabetta Di Nitto

*Academic year:*

2019 – 2020

## CONTENTS

---

Contents .....	2
1 Introduction .....	4
1.A Scope.....	4
2 Links to the source code and instructions .....	5
3 Implemented Requirements .....	6
4 Adopted Development Frameworks .....	7
4.A Programming Languages.....	7
4.B Middleware.....	7
4.C Framework.....	7
4.D API .....	7
4.E Software.....	7
5 Structure of the Source Code.....	9
5.A Client.....	9
5.B Application Server.....	9
5.C Web Server.....	10
6 Performed Testing.....	11
6.A For the Client.....	11
6.B For the Application Server .....	11
6.B.1 authorizationPack .....	11
6.B.2 data_analysis_manager.....	11
6.B.3 dataManagerAdapterPack.....	12
6.B.4 elaborationManagerTest .....	12
6.B.5 mapsserviceadapter.....	12
6.B.6 model.....	12
6.B.7 modelEntities .....	12
6.B.8 registrationManager.....	12
6.B.9 Dispatcher Test.....	12
7 Installation Instructions .....	13
7.A Set the database .....	13
7.B Run the Application Server .....	13
7.C Run the Web Server.....	16
7.D Run the Client on a Smartphone .....	16
7.E Run the Client on a Browser.....	16
7.F Run the Client on iOS.....	16
8 Effort spent .....	19

---

9	References.....	20
---	-----------------	----

# 1 INTRODUCTION

---

This document is about the implementation and the testing of the SafeStreets Application.

We have developed the application server, the web server, the mobile application for the smartphone, the web application for the browser and we have created a schema in a relational database.

## 1.A SCOPE

The scope of this document is to provide any information about the installation of the software, the instructions to run the source code, the structure of the source code, the implemented requirements, the adopted Frameworks and the test performed to validate the software.

## 2 LINKS TO THE SOURCE CODE AND INSTRUCTIONS

---

Instructions for the software to install:

*<https://github.com/gianfi12/AbboAccordiBonetti/tree/master/SoftwareToInstall/SoftwareToInstall.docx>*

Jars and other files:

*<https://github.com/gianfi12/AbboAccordiBonetti/tree/master/JARsAndOtherFiles>*

Source of the implementation of the project:

*<https://github.com/gianfi12/AbboAccordiBonetti/tree/master/Implementation>*

### 3 IMPLEMENTED REQUIREMENTS

---

We have implemented the “basic service” and the “advanced function 2” of the SafeStreets Project.

The following requirements have been implemented:

- R1: The reports about the violations are correctly stored.
- R2: The user can view the statistics calculated by the System with some exceptions<sup>1</sup>
  - R2.A: The vehicles that have committed the highest number of violations.
- R3: The Municipality can access only the data of the violations of its competence area.
- R5: The system must avoid the manipulation of the violations.
- R6: The system must be able to retrieve the position from the user or from the GPS
- R7: Only the Municipality can access the submitted parking violation of its competence area
- R8: The system must allow the User to take a picture or select one from the device.
- R9: The system accepts reports from the User.
- R10: The System must calculate some statistics
  - R10.A: The system must calculate the streets with the highest and the lowest number of violations.
  - R10.B: The system must calculate the effectiveness of the service.
  - R10.C: The system must calculate the vehicles (identified by the traffic plate) that have committed the highest number of violations.
  - R10.D: The system must calculate the most common violations of a given area
- R11: The municipality can view all the statistics calculated by the system.
- R14: The system must allow the user to perform the registration and the login.
- R15: The system must allow the Municipality to perform the registration and the login.
- R16: The system must ask the User the non-mandatory attributes of the report.
- R19: The System must communicate with the Maps Service.

We have not implemented these requirements because they regard the “advanced function 1”:

- R4: Violations registered by the Municipality can be retrieved by the system.
- R12: The system must suggest interventions to the Municipality.
  - R12.A: Inspect an area
  - R12.B: New cycle lane
  - R12.C: New sidewalk
  - R12.D: New pedestrian crossing
  - R12.E: New parking
  - R12.F: New speed detector

We have not implemented these requirements because they require a service from a third-party organization, which is usually for a fee:

- R17: The system must communicate with the Identity Verifier.
- R18: The system must communicate with the Plate Recognizer Service.

The following requirement has been implemented partially:

- R13: The system accepts only reports with a valid plate number and position.

In particular the system retrieves the position of the device in which the report has been made, but the plate number is not verified because the Plate Recognizer Service was not provided.

---

<sup>1</sup> The user can not view the statistics specified.

## 4 ADOPTED DEVELOPMENT FRAMEWORKS

---

### 4.A PROGRAMMING LANGUAGES

We have adopted “Dart” for the client because:

- it is familiar and easy to learn for developers with backgrounds in Java,
- it allows to build app for every smartphone and for web applications through flutter,
- it is free and open source,
- it is object-oriented, so it allows to create reusable code and modular programs.

We have adopted “Java” for the server, which has several advantages:

- it is one of the most popular programming languages,
- it is platform-independent,
- it is well-known by the developers of this project,
- it is object-oriented, so it slows to create reusable code and modular programs.

### 4.B MIDDLEWARE

We have adopted “JEE” because:

- it uses the programming language “Java”,
- it allows to manage the component of the system in an efficient way,
- it allows to communicate with the database through the JPA.

### 4.C FRAMEWORK

We have adopted “Flutter” because:

- it allows to build apps for every smartphone and for web applications,
- it is free and open source,
- it uses the programming language “Dart”.

We have used “Apache Maven” which allows to manage the external dependencies in an easier way.

We have used “Hibernate” which allows the mapping between an object-oriented domain model and a relational database. It was chosen because it is open source, easy to use and it simplifies the communication with the DBMS.

For testing we have used “JUnit” because it supports Java and it is supported by the major IDEs. We have also used “Mockito” to create the stubs for the not-yet implemented classes. It was chosen because it is easy to use, free and it has a lot of functionalities for testing.

### 4.D API

As already said in the Design Document we have used the APIs of Google Maps because it is a service of very good quality and it is used every day by a lot of people. It has a lot of experience and it is supported by one the largest companies on the world, Google. Furthermore, Google Maps is easy to use, and has a good user interface, which fits with the design of the SafeStreets’ client applications.

### 4.E SOFTWARE

We have used “GlassFish” because:

- it is open source,
- it supports the framework “JEE”.

As already said in the Design Document we have used MySQL server for various reasons:

- it is an open source DBMS,
- it is present from the 1995,
- it is adopted by a lot of organizations and private users and it is currently supported.

We have also used Android Studio and IntelliJ IDEA to develop the project because they can boost the productivity, they have a lot of plugins, they support the above frameworks and programming languages and they are well supported.



## 5 STRUCTURE OF THE SOURCE CODE

---

The source code is present in the following directory:

<https://github.com/gianfi12/AbboAccordiBonetti/tree/master/Implementation>

Inside it there are these directories:

- `safe_streets_client`, for the client,
- `SafeStreetsRPC`, for the application server,
- `SafeStreets`, for the web server.

### 5.A CLIENT

In Dart, each file can contain multiple classes, and works as a java package. The main components of a Flutter applications are objects (classes) called widgets.

The source code is in `/Implementation/safe_streets_client/lib`. In `/main.dart` the app is created and launched; the files starting with “handler” contain code that does not directly create widgets (interaction with the backend, the device, and similar); those that start with “widget” contain instructions on how to create the components of the apps and on their behaviour.

The file `/Implementation/safe_streets_client/pubspec.yaml` contains the dependencies for this Flutter project. In `/Implementation/safe_streets_client/assets` there are the images and data used in the apps.

Flutter framework has some bugs and is missing some functionalities, for this reason some workarounds were used: `/Implementation/safe_streets_client/lib/workarounds` contains some files that are used with conditional imports in the rest of the source; the map is implemented in HTML and JavaScript and its source is in `/Implementation/safe_streets_client/assets/code.html` and `/Implementation/safe_streets_client/web/assets/code.html`

### 5.B APPLICATION SERVER

In

<https://github.com/gianfi12/AbboAccordiBonetti/tree/master/Implementation/SafeStreetsRPC/SafeStreetsSOAP/pom.xml>

there is the file `pom.xml` which specifies the maven dependencies, maven was chosen in order to manage easier the external dependencies.

The file

<https://github.com/gianfi12/AbboAccordiBonetti/tree/master/Implementation/SafeStreetsRPC/SafeStreetsSOAP/src/main/resources/META-INF/persistence.xml>

specifies the connection with the DBMS.

The source code is in

<https://github.com/gianfi12/AbboAccordiBonetti/tree/master/Implementation/SafeStreetsRPC/SafeStreetsSOAP/src/main/java/com/SafeStreets>

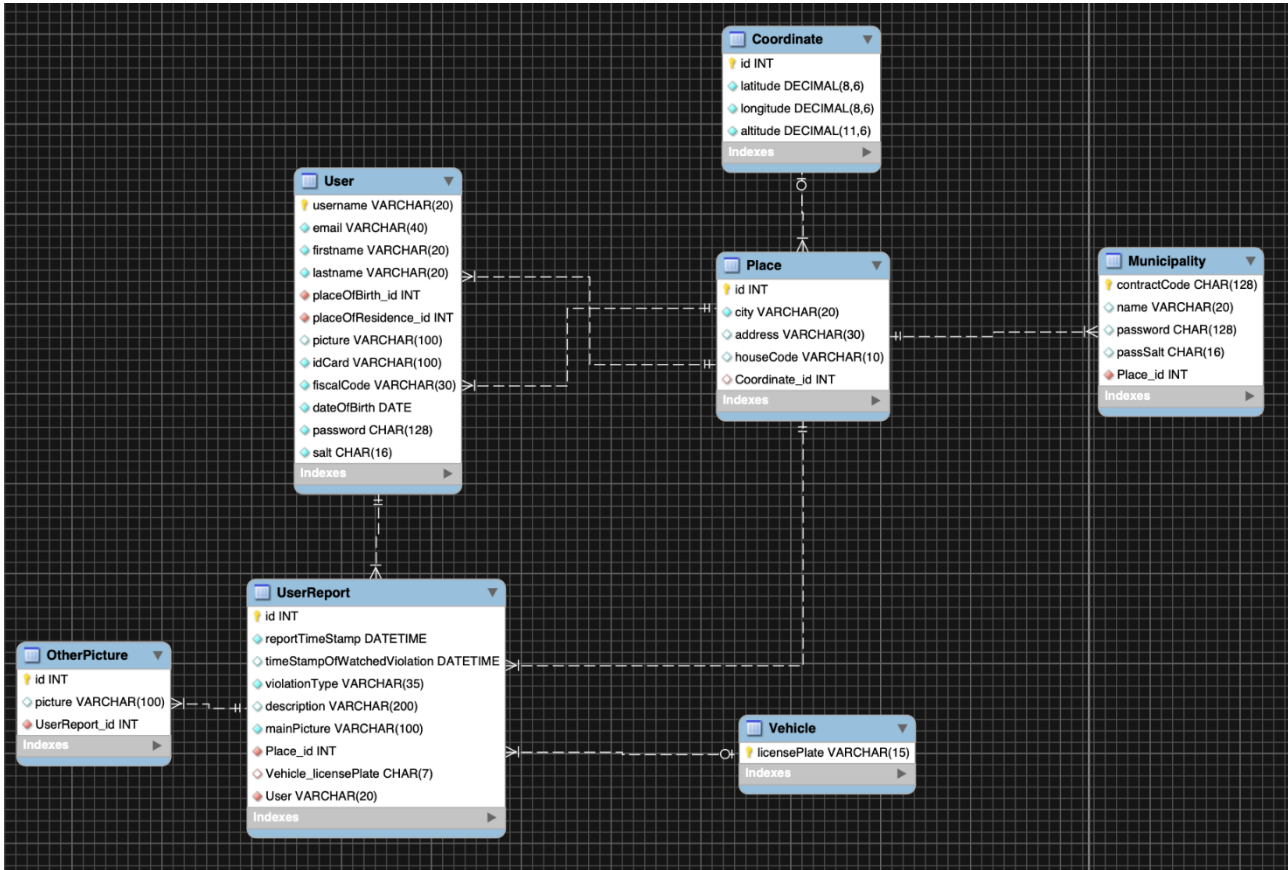
The “Dispatcher” component is in the class “Dispatcher” and its interface is “DispatcherInterface”.

The package “model” contains the classes of the class diagram in the Design Document. They are the data structures used in the Application Server.

The package “modelEntities” contains the classes that are mapped by Hibernate to the database. Each class correspond to a table in the database.

Each other package refers to one component specified in the Design Document, the package contains the public interfaces of the component and the classes that implement them.

The following pictures illustrates the entity-relationship diagram of the database:



## 5.C WEB SERVER

The implementation of the web server can be found at:

<https://github.com/gianfi12/AbboAccordiBonetti/tree/master/Implementation/SafeStreets>

it already contains all the dependency necessary to run the web server inside the *pom.xml* file as we have used Maven for the development.

## 6 PERFORMED TESTING

---

### 6.A FOR THE CLIENT

The tests can be found in */Implementation/safe\_streets\_client/test*.

Flutter framework offers three main types of tests: unit, widget and integration. Widget testing focuses on the appearance and behaviour of the app components, but the testing framework is not complete yet and caused many problems, for this reason it was not used. Unit testing was performed where possible, to ensure the correct behaviour of the methods. The app was also tested manually.

In a real environment, with more time and expertise at hand, it would have been better to test the app in different environments, and then to pre-release it to a small group of testers.

### 6.B FOR THE APPLICATION SERVER

The tests are in the package:

<https://github.com/gianfi12/AbboAccordiBonetti/tree/master/Implementation/SafeStreetsRPC/SafeStreetsSOAP/src/test/java/com/SafeStreets>

Inside it there is the test for the Dispatcher (DispatcherTest) and a package for each component.

Each class contains some methods and each method tests one method of the source code with one input of data or with more input of data if the method is simple.

Each method of the test is well commented with the Javadoc, please read the Javadoc for more details:

<https://github.com/gianfi12/AbboAccordiBonetti/javadoc>

*Inside the directory*

<https://github.com/gianfi12/AbboAccordiBonetti/tree/master/Implementation/SafeStreetsRPC/SafeStreetsSOAP/src/test/resources/image>

there are the images used for testing.

The database has some tuples in order to test the functionalities of the system.

We have also tested the entire system: we have activated the database, the application server and the web server and we have tried the following features on two smartphone running Android, on one smartphone running iOS and on the browser Google Chrome: login of the user and of the municipality, registration of the user and of the municipality, new report from the user, view statistics for the user and for the municipality and access reports for the municipality.

#### 6.B.1 authorizationPack

This component has been tested by looking at how it grants the access to the client in the system, basically the tests look on the fact that the module has to return an access to the municipality and the user, while in the other test the component correctly responded that no user has been recognized.

#### 6.B.2 data\_analysis\_manager

It has been verified that the DataAnalysisManager can calculate all the type of statistics. For each type of statistic the returned statistics have been checked whether they have been correctly calculated.

It has been verified that the DataAnalysisManager can return the reports done by the users.

### 6.B.3 dataManagerAdapterPack

All the public methods and the package methods of the DataManagerAdapter have been tested, for example whether the DataManagerAdapter can return correctly one specific municipality or user, the reports done by the users, whether it can store correctly a municipality or a user and whether it can execute specific SQL queries.

### 6.B.4 elaborationManagerTest

The Elaboration Manager has been tested by looking on how it memorized the new report request, the tests were not expensive, because this component it has simply to save the received report. The tests instead were focused on how the objects were deserialized, in order to avoid error with the convention, for example with the date format.

### 6.B.5 mapsserviceadapter

The values returned from the maps service were not tested, neither were the library methods for accessing the service. The translation between different coordinates representation was tested to avoid accidental swapping of latitude and longitude values. Similar tests were performed on the other methods of the class. Simple test (on throw, constructor, etc.) were omitted due to time limits.

### 6.B.6 model

In the model we have tested the methods that convert a class of the model to the respective class of the modelEntities and the method isEqual which verifies whether two objects have the same attributes.

### 6.B.7 modelEntities

In the model we have tested the methods that convert a class of the modelEntities to the respective class of the model.

### 6.B.8 registrationManager

The test of the registration manager has been done by trying to provide wrong credentials and see if for example someone can register with the same nickname, obviously has been tested early with some unit testing, and then its functionalities has been tested again with the integration testing with the Dispatcher

### 6.B.9 Dispatcher Test

In the DispatcherTest we have also tested the methods “accessReports” and “requestDataAnalysis” which require the implementation of the DataManager, but it was not implemented yet so we have used Mockito to mock the DataManagerInterface. The major part of the Dispatcher test is present on the Client part, from the Android Studio project, under the path *safe\_streets\_client/test/handler/backend\_test.dart* in this way we have been able to test directly the functionality of the Dispatcher when invoked from the client, and also to look at how the objects are deserialized by the provided method

## 7 INSTALLATION INSTRUCTIONS

---

First install the software as explained in:

<https://github.com/gianfi12/Abbo.AccordiBonetti/tree/master/SoftwareToInstall/SoftwareToInstall.docx>

The following phases indicates how to run the software:

### 7.A SET THE DATABASE

Run the MySQL Community Server and the MySQL Workbench.

Create the new schema with the query:

```
CREATE DATABASE safe_streets_db
```

Create the user “admin” with password “admin” with:

```
CREATE USER 'admin'@'localhost' IDENTIFIED BY 'admin'
```

Grant the privileges on the new schema to the admin user:

```
GRANT ALL PRIVILEGES ON safe_streets_db.* TO 'admin'@'localhost';
```

Import the file “database.sql” (which is present inside the directory

<https://github.com/gianfi12/Abbo.AccordiBonetti/tree/master/JARsAndOtherFiles>) with the MySQL Workbench database:

select from the top menu “Server/Data Import”, select the button “Import from self-contained file”, choose the file “database.sql”, choose as Default Target Schema the schema “safe\_streets\_db” and press the button “Start Import”.

Finally copy the directory picturesData (which is present inside the directory

<https://github.com/gianfi12/Abbo.AccordiBonetti/tree/master/JARsAndOtherFiles>) inside the directory glassfish4/glassfish/domains/domain1 of glassfish.

### 7.B RUN THE APPLICATION SERVER

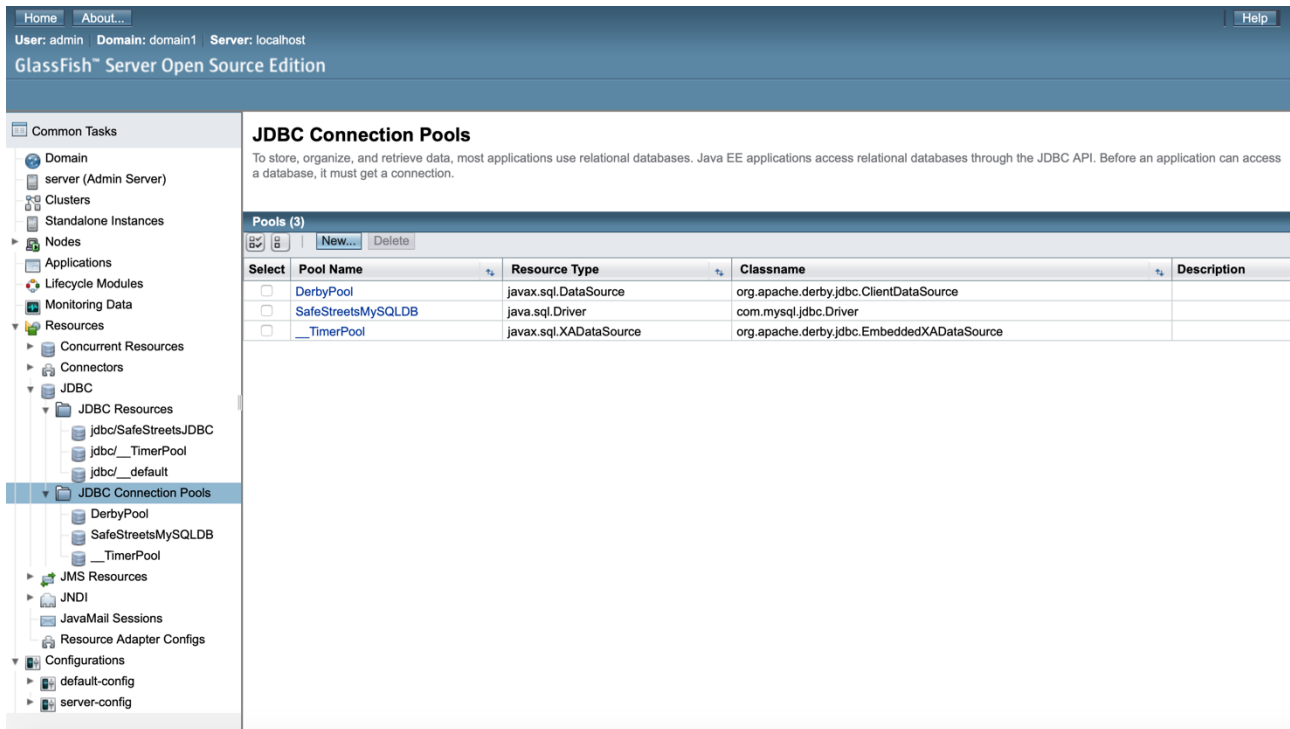
After the installation of Glassfish, you can launch it by looking inside the /bin/ folder and here you find the file *asadmin*, launch it with (on the Linux shell):

```
$ ./asadmin start-domain
```

Connect to Glassfish with a browser by inserting in the browser:

```
localhost:4848
```

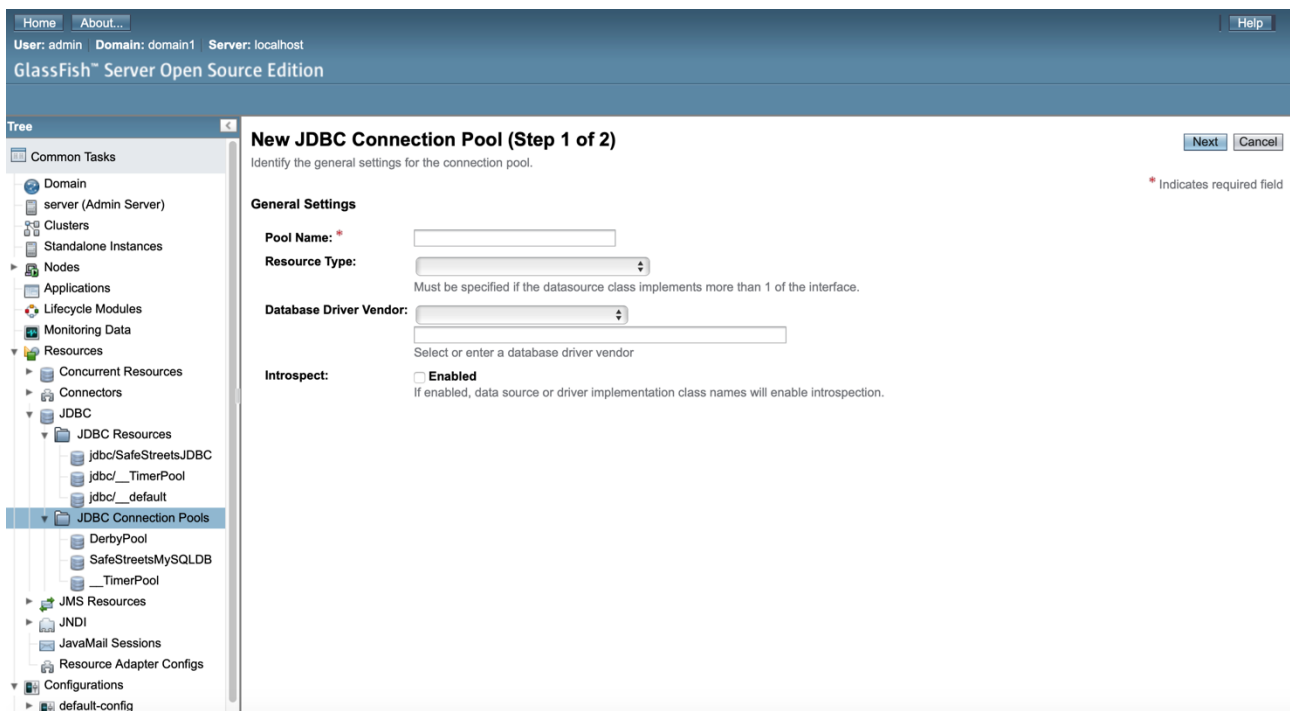
Go to JDBC/JDBC Connection Pools and press the button “New”



The screenshot shows the GlassFish Server Open Source Edition administration console. The left sidebar contains a tree view of the server's configuration, with 'JDBC Connection Pools' selected under the 'JDBC' section. The main content area is titled 'JDBC Connection Pools' and includes a description: 'To store, organize, and retrieve data, most applications use relational databases. Java EE applications access relational databases through the JDBC API. Before an application can access a database, it must get a connection.' Below this is a table titled 'Pools (3)' with columns: Select, Pool Name, Resource Type, Classname, and Description. The table lists three pools: DerbyPool, SafeStreetsMySQLDB, and \_\_TimerPool.

Select	Pool Name	Resource Type	Classname	Description
<input type="checkbox"/>	DerbyPool	javax.sql.DataSource	org.apache.derby.jdbc.ClientDataSource	
<input type="checkbox"/>	SafeStreetsMySQLDB	java.sql.Driver	com.mysql.jdbc.Driver	
<input type="checkbox"/>	__TimerPool	javax.sql.XADataSource	org.apache.derby.jdbc.EmbeddedXADataSource	

Now you should have



The screenshot shows the 'New JDBC Connection Pool (Step 1 of 2)' configuration page. The left sidebar is the same as the previous screenshot. The main content area is titled 'New JDBC Connection Pool (Step 1 of 2)' and includes a description: 'Identify the general settings for the connection pool.' Below this is a section titled 'General Settings' with the following fields: Pool Name (required), Resource Type (dropdown), Database Driver Vendor (dropdown), and Introspect (checkbox). The 'Pool Name' field is highlighted with a red asterisk, indicating it is a required field.

**General Settings**

Pool Name: \*

Resource Type:

Database Driver Vendor:

Introspect: ☐ Enabled

As Pool Name write “SafeStreetsMySQLDB”, select “java.sql.Driver” as ResourceType, select “MySql” as Database Driver Vendor and click Next.

Now at the end of the page you must set these properties:

Additional Properties (7)			
<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> Add Property Delete Properties			
Select	Name	Value	Description
<input type="checkbox"/>	dataBase	safe_streets_db	
<input type="checkbox"/>	portNumber	3306	
<input type="checkbox"/>	user	admin	
<input type="checkbox"/>	url	jdbc:mysql://localhost:3306/safe_streets_db	
<input type="checkbox"/>	useSSL	false	
<input type="checkbox"/>	allowPublicKeyRetrieval	true	
<input type="checkbox"/>	password	admin	

Then click Finish. You can verify the correct setup by clicking on “SafeStreetsMySQLDB” and by pressing the button “ping”.

Then go to JDBC/JDBC Resources and press New.

As JNDI Name write “SafeStreetsJDBC”, as Pool Name write “SafeStreetsMySQLDB” then click Ok.

GlassFish™ Server Open Source Edition

Tree

- Common Tasks
  - Domain
    - server (Admin Server)
      - Clusters
        - Standalone Instances
      - Nodes
        - Applications
        - Lifecycle Modules
        - Monitoring Data
      - Resources
        - Concurrent Resources
        - Connectors
        - JDBC
          - JDBC Resources
            - jdbc/SafeStreetsJDBC
            - jdbc/\_TimerPool
            - jdbc/\_default
          - JDBC Connection Pools
            - DerbyPool
            - SafeStreetsMySQLDB
            - \_TimerPool
        - JMS Resources
        - JNDI
        - JavaMail Sessions
        - Resource Adapter Configs

New JDBC Resource

Specify a unique JNDI name that identifies the JDBC resource you want to create. The name must contain only alphanumeric, underscore, dash, or dot characters.

JNDI Name: \* SafeStreetsJDBC

Pool Name: SafeStreetsMySQLDB

Use the JDBC Connection Pools page to create new pools

Description:

Status: ☒ Enabled

Additional Properties (0)

Add Property Delete Properties

Select	Name	Value	Description
No items found.			

OK Cancel

This finishes the configuration on the browser.

This command will start the glassfish server, and its command line, from here you can deploy the Application by running:

```
$ asadmin> deploy /path_to_war/SafeStreetsSOAP.war
```

You can also deploy the application more easily by using Glassfish through the GUI, by typing inside your browser the following address: *ip\_address:4848* and then you select the “Application” voice from the Menu, and from here you can select which war you want to deploy.

Another available option is to use on IntelliJ with Glassfish: you have to import the project by specifying the location of the maven project (you select the pom.xml file), then you create configuration of Glassfish (local) and from it you specify the domain (usually is domain1), and then you can simply press run to start it, once you have selected the provided .war for the application server, in the *Deployment* part of the configuration.

You can check the correct work of the server by typing on a browser:

*<http://localhost:8080/SafeStreets.SOAP/DispatcherService>*

and from here you can access the definition of the method exposed by the server.

Important is that you must create a folder call */picturesData* inside the glassfish location, at the following path: */glassfish4/glassfish/Domains/domain1* because here after the deployment will be saved the provided photo by the user, also here in order to use the provided DBMS in which are already present some photo, you have to copy in this folder the pictures that are presents at this link:

*<https://github.com/gianfi12/Abbo.AccordiBonetti/tree/master/Implementation/SafeStreetsRPC/picturesData>*.

In order to run Glassfish you need to use Java 8.

*Note:* if you are using IntelliJ, you can configure the Database from it by following this guide:

*<https://www.jetbrains.com/help/idea/connecting-to-a-database.html>*.

## 7.C RUN THE WEB SERVER

The deployment of the web server is the same as for the deployment of the Application server, but in this case, you use the war of the web server, and you can now interact with the system by using any browser. Pay attention that you cannot simply run it from IntelliJ if you have already up the Application Server from IntelliJ in the same machine, but you have to manually deploy it from the glassfish console or from the Browser GUI of glassfish. After the deployment of the Web server, you can access it by typing inside the browser:

*[http://ip\\_address:8080/SafeStreets](http://ip_address:8080/SafeStreets)*

Pay attention to use the ip address of your machine and no the loopback address and the localhost address, because some browser (due to the CORS policy) blocks all the request made.

## 7.D RUN THE CLIENT ON A SMARTPHONE

Import the project in Android Studio, then create a virtual device or connect a compatible device

*<https://flutter.dev/docs/get-started/install>*

Run Flutter upgrade, Packages get and Packages upgrade if necessary, then launch the app.

## 7.E RUN THE CLIENT ON A BROWSER

Import the project in Android Studio, then follow the instructions at *<https://flutter.dev/docs/get-started/web>* to enable web support.

Launch the app on chrome or as a local page.

## 7.F RUN THE CLIENT ON IOS

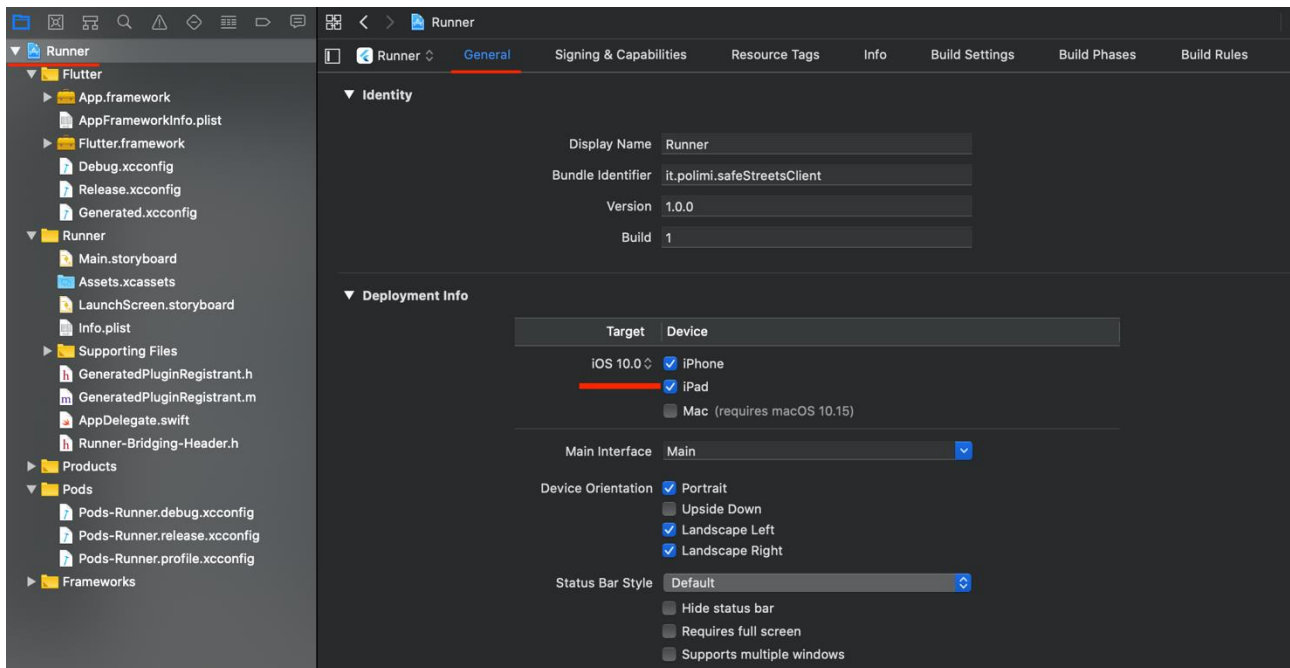
First you need to compile the client application so read the section 7.D Run the Client and if the application works on iOS then this section is useless, otherwise continue with this section.

Connect your iPhone or iPad or iPod to the Mac, run Xcode and open with Xcode the file

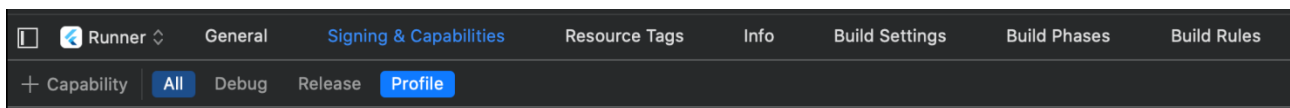
*[https://github.com/gianfi12/Abbo.AccordiBonetti/tree/master/Implementation/safe\\_streets\\_client/ios/Runner.xcodeproj](https://github.com/gianfi12/Abbo.AccordiBonetti/tree/master/Implementation/safe_streets_client/ios/Runner.xcodeproj)*



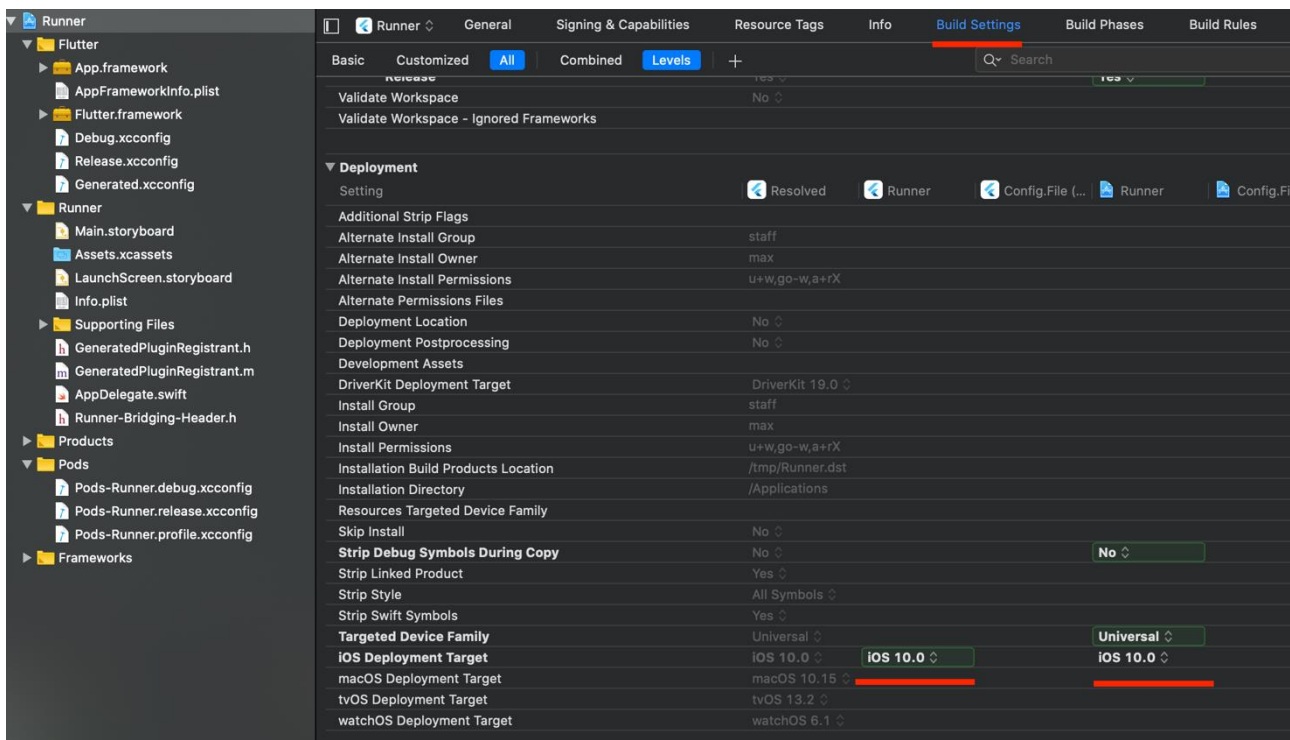
Select “Runner” from the top-left, then select “General” and verify that the Target is iOS 10.0



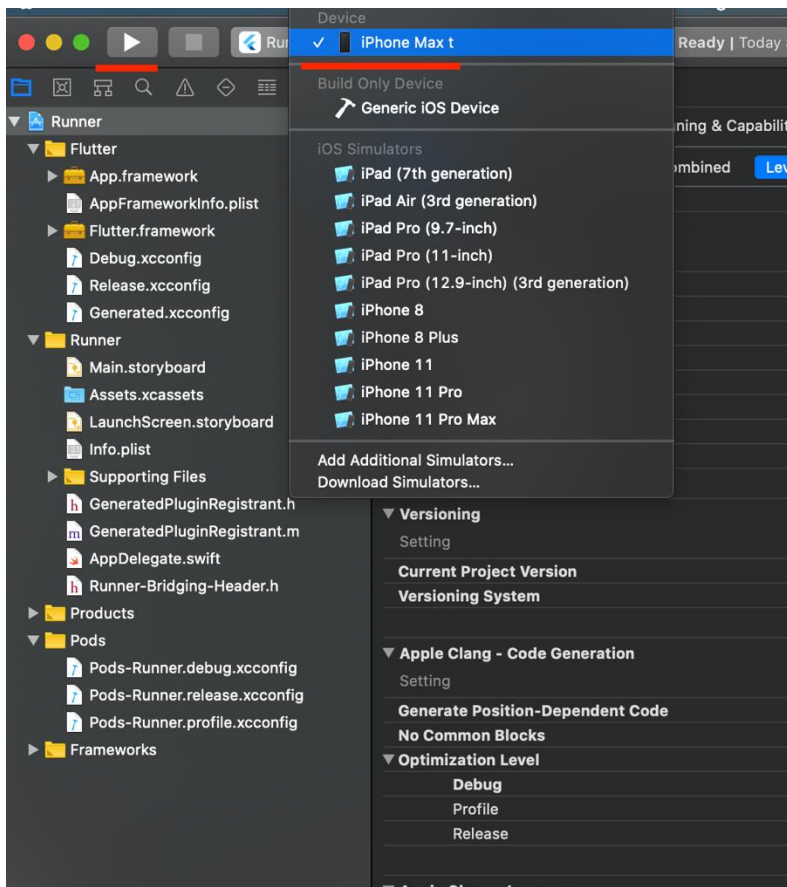
Select “Signing & Capabilities” and verify that no error is present



Select “Build Settings” and verify that “iOS Deployment Target” is set to 10.0



You can run the application by selecting your device or an iOS Simulator and by clicking build



## 8 EFFORT SPENT

- Abbo Giulio Antonio

DESCRIPTION OF THE TASK	HOURS
Implementation	104

- Accordi Gianmarco

DESCRIPTION OF THE TASK	HOURS
Implementation	102

- Bonetti Massimiliano

DESCRIPTION OF THE TASK	HOURS
Implementation	100

## 9 REFERENCES

---

- Slides of the Software Engineering 2 course by prof. Di Nitto at Politecnico di Milano