



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](#_Toc29658268)

[1 Introduction 4](#_Toc29658269)

[1.A Scope 4](#_Toc29658270)

[2 Links to the source code and instructions 5](#_Toc29658271)

[3 Implemented Requirements 6](#_Toc29658272)

[4 Adopted Development Frameworks 7](#_Toc29658273)

[4.A Programming Languages 7](#_Toc29658274)

[4.B Middleware 7](#_Toc29658275)

[4.C Framework 7](#_Toc29658276)

[4.D API 7](#_Toc29658277)

[4.E Software 7](#_Toc29658278)

[5 Structure of the Source Code 9](#_Toc29658279)

[5.A Client 9](#_Toc29658280)

[5.B Application Server 9](#_Toc29658281)

[5.C Web Server 10](#_Toc29658282)

[6 Performed Testing 11](#_Toc29658283)

[6.A For the Client 11](#_Toc29658284)

[6.B For the Application Server 11](#_Toc29658285)

[6.B.1 authorizationPack 11](#_Toc29658286)

[6.B.2 data\_analysis\_manager 11](#_Toc29658287)

[6.B.3 dataManagerAdapterPack 11](#_Toc29658288)

[6.B.4 elaborationManagerTest 12](#_Toc29658289)

[6.B.5 mapsserviceadapter 12](#_Toc29658290)

[6.B.6 model 12](#_Toc29658291)

[6.B.7 modelEntities 12](#_Toc29658292)

[6.B.8 registrationManager 12](#_Toc29658293)

[6.B.9 Dispatcher Test 12](#_Toc29658294)

[6.C For the Web Server 12](#_Toc29658295)

[7 Installation Instructions 13](#_Toc29658296)

[7.A Set the database 13](#_Toc29658297)

[7.B Run the Application Server 13](#_Toc29658298)

[7.C Run the Web Server 13](#_Toc29658299)

[7.D Run the Client on a Smartphone 13](#_Toc29658300)

[7.E Run the Client on a Browser 13](#_Toc29658301)

[7.F Run the Client on iOS 14](#_Toc29658302)

[8 Effort spent 16](#_Toc29658303)

[9 References 17](#_Toc29658304)

# 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.

## 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.

# Links to the source code and instructions

Instructions for the software to install:  
<https://github.com/gianfi12/AbboAccordiBonetti/blob/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>

# 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[[1]](#footnote-1)
  + 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.

# Adopted Development Frameworks

## 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.

## 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.

## 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.

## 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.

## 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.

# 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.

## 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*

## Application Server

In

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

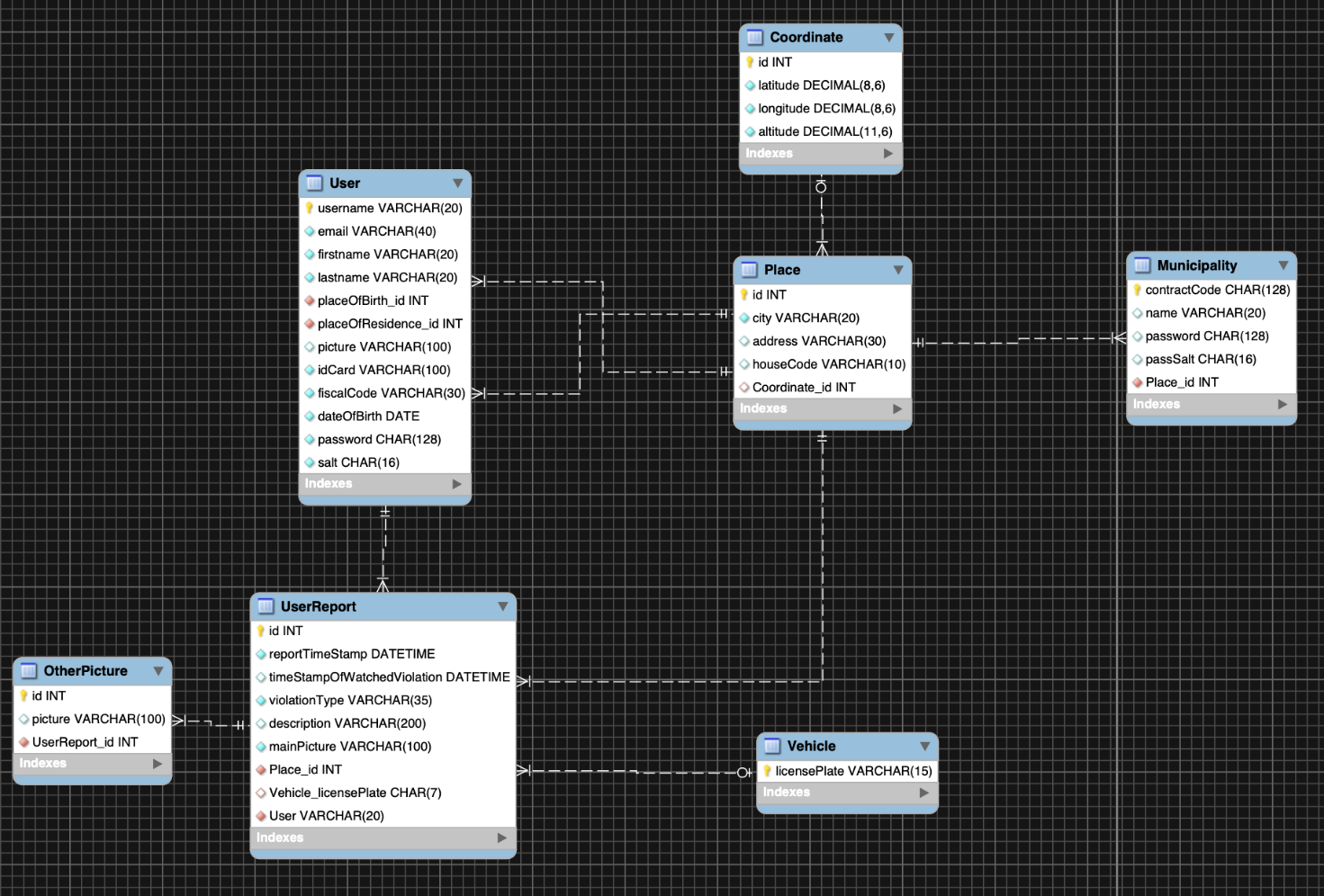
The “Dispatcher” component is in the class “Dispatcher” and its interface is “DispatcherInterface”.

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:



## Web Server

# Performed Testing

## 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.

## 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.

### authorizationPack

### 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.

### 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.

### elaborationManagerTest

### 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.

### model

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

### modelEntities

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

### registrationManager

### Dispatcher Test

In the DispatcherTest we have also tested the methods “accessReports” and “requestDataAnalysis” which require the implementation of the DataAnalysisManager, but it was not implemented yet so we have used Mockito to mock the DataAnalysisInterface.

## For the Web Server

# Installation Instructions

First install the software as explained in:  
<https://github.com/gianfi12/AbboAccordiBonetti/blob/master/SoftwareToInstall/SoftwareToInstall.docx>

The following phases indicates how to run the software:

## 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/AbboAccordiBonetti/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/AbboAccordiBonetti/tree/master/JARsAndOtherFiles>*)* inside the directory glassfish4/glassfish/domains/domain1 of glassfish.

## Run the Application Server

## Run the Web Server

## 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.

## 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.

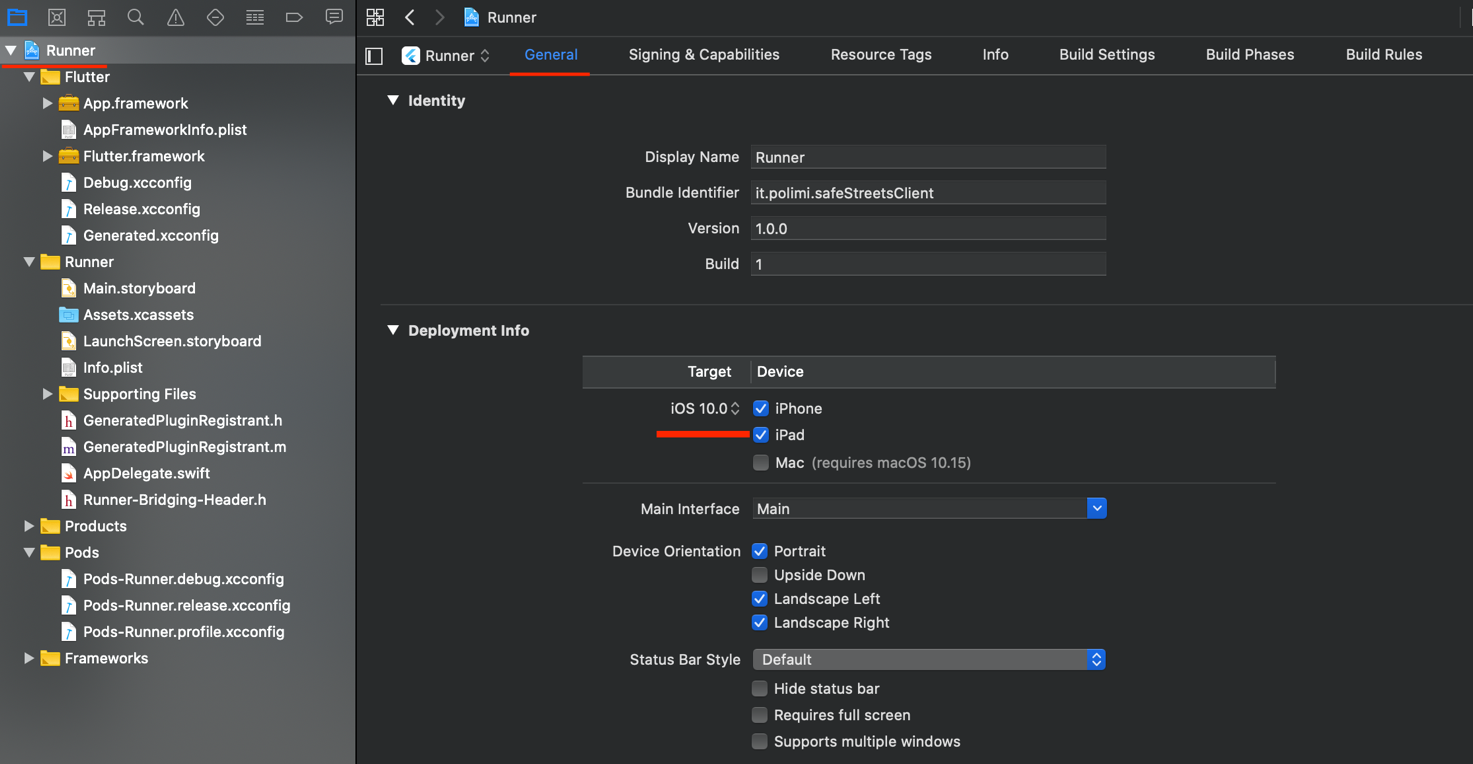
Launch the app on chrome or as a local page.

## Run the Client on iOS

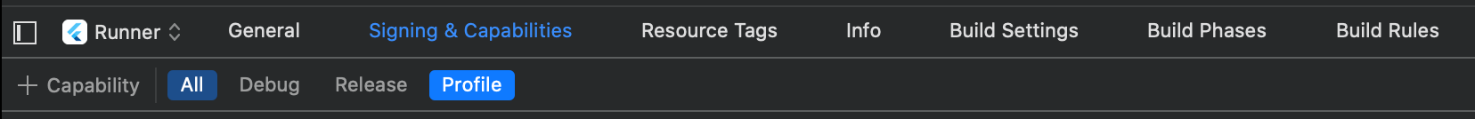
First you need to compile the client application so read the section 7.D Run the Client and if the application works on iOS then this section is useless, otherwise continue with this section.

Connect your iPhone or iPad or iPod to the Mac, run Xcode and open with Xcode the file <https://github.com/gianfi12/AbboAccordiBonetti/tree/master/Implementation/safe_streets_client/ios/Runner.xcodeproj>

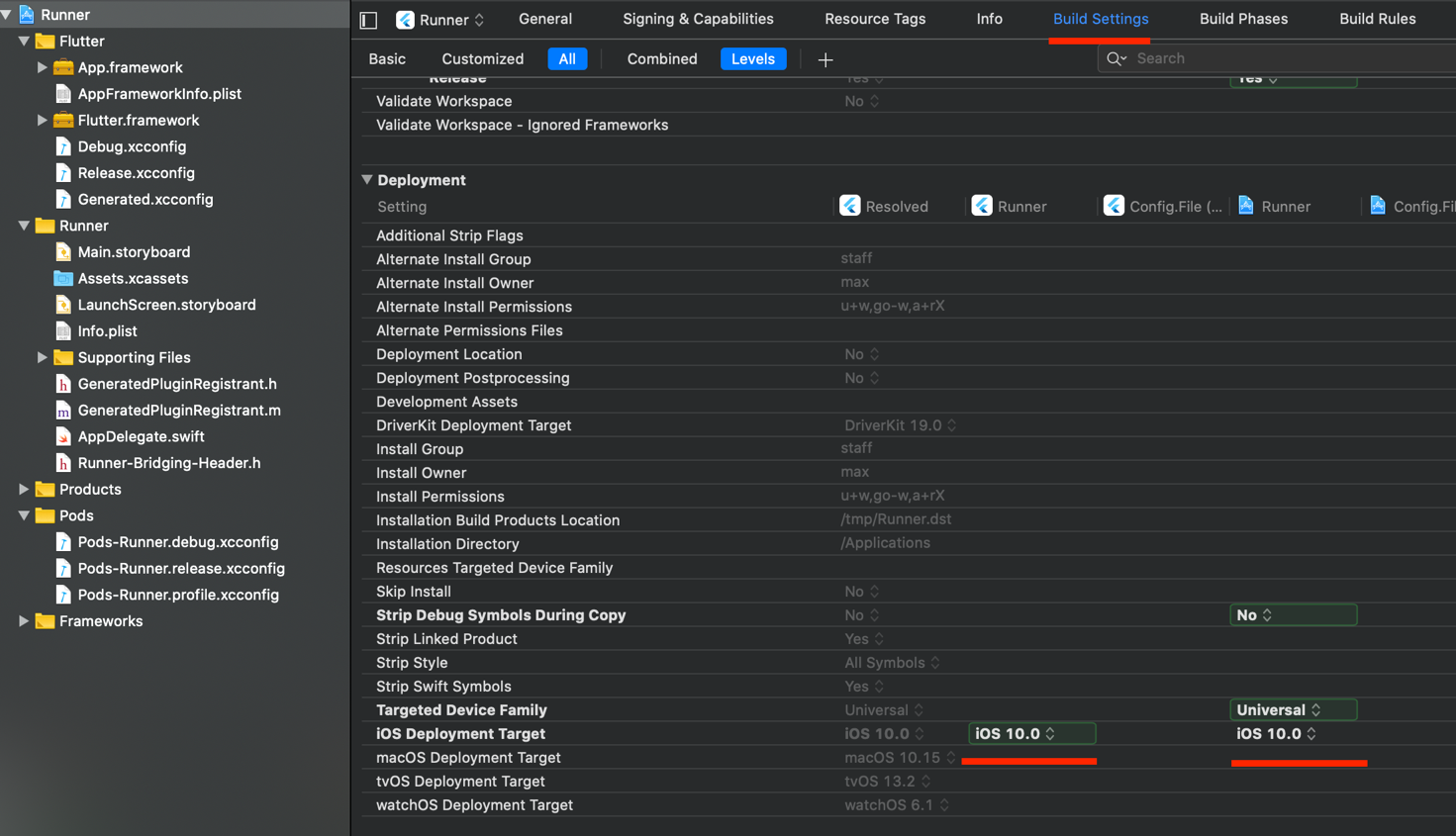
Select “Runner” from the top-left, then select “General” and verify that the Target is iOS 10.0



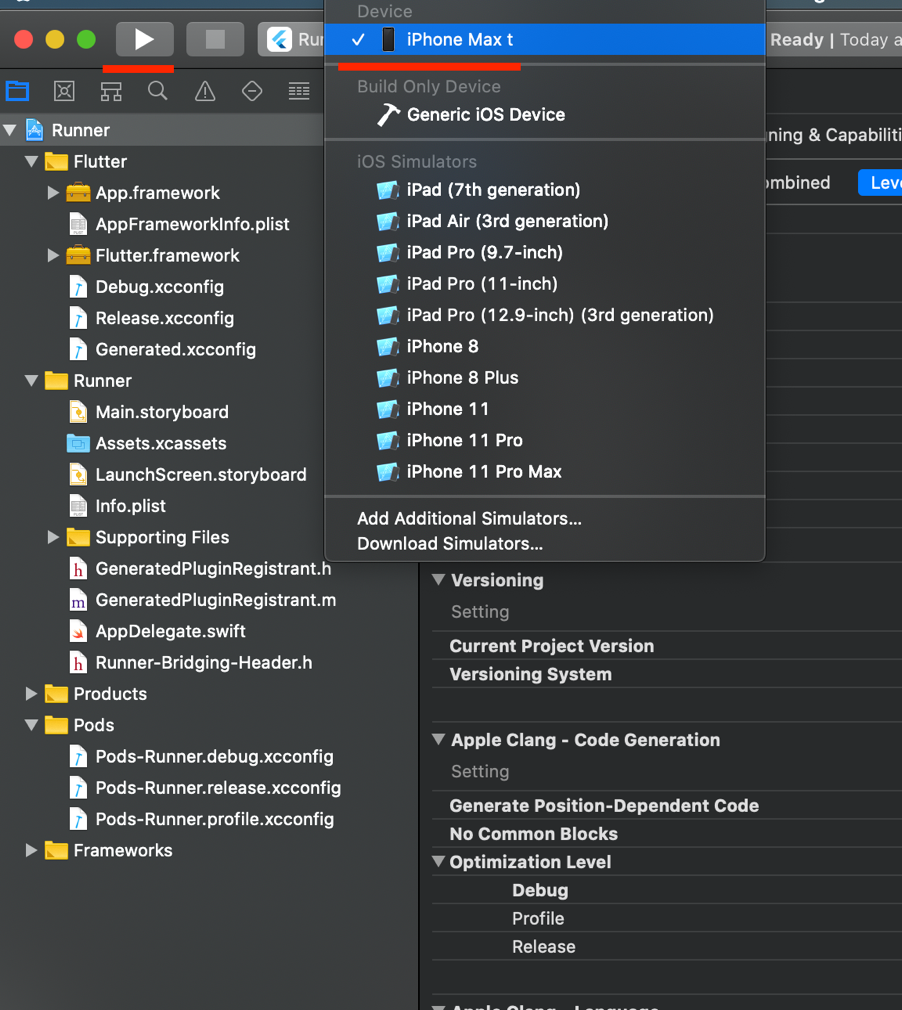
Select “Signing & Capabilities” and verify that no error is present



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



# Effort spent

* Abbo Giulio Antonio

|  |  |
| --- | --- |
| Description of the task | Hours |
| Implementation | 100 |

* Accordi Gianmarco

|  |  |
| --- | --- |
| Description of the task | Hours |
| Implementation | 0 |

* Bonetti Massimiliano

|  |  |
| --- | --- |
| Description of the task | Hours |
| Implementation | 100 |

# References

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

1. The user can not view the statistics specified. [↑](#footnote-ref-1)