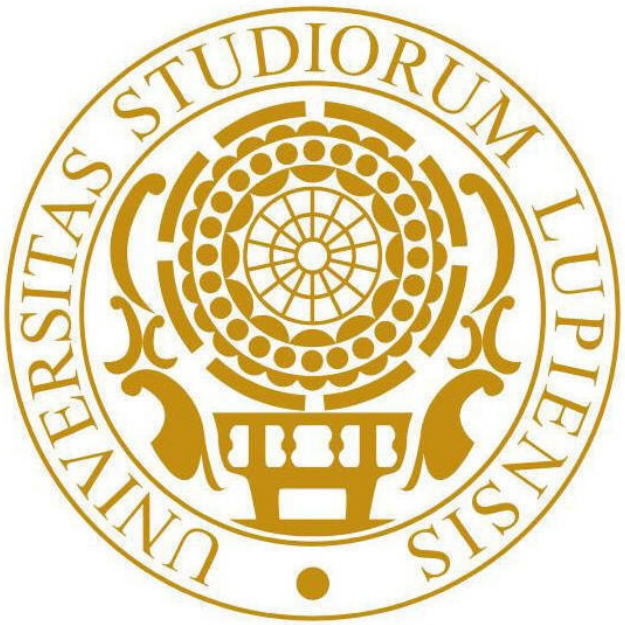


UNIVERSITA' DEL SALENTO

FACOLTA' DI INGEGNERIA

CORSO DI LAUREA MAGISTRALE IN COMPUTER ENGINEERING

PROGETTO DI
SOFTWARE ENGINEERING

Studenti:
Danilo Giovannico 20039574
Ilaria Malinconico 20034331

ANNO ACCADEMICO 2017/2018

Index

1. Description of the System Requirements

The Didactic Coordinator orders a software system that supports the didactic activity and it includes a web app and a mobile app.
The web app provides specific functionalities based on the user that is logged in the system as Professor or as Secretary. Its main purpose is handling the reports sent by the professors and taken care of by the Secretary.
The mobile app provides further functionalities for the user that is logged in as Student or as Professor. These functionalities include the possibility of exchanging messages, receiving notifications, downloading documents and sending feedback.

2. Requirements Analysis

The actors that interact with the software requested by the didactic Coordinator are:

- Secretary
- Professor
- Student

Using the web app, the Secretary is able to insert:

- New courses of study
- New subjects for the existing study courses
- New classrooms, with information regarding the geolocation
- New activities, choosing between Lesson or Exam
- New Users, choosing between Student or Professor

Furthermore, it can handle the reports sent by the professors. When the reporting is taken care of, the Secretary add a note or motivation if the issue has been resolved or refused. Anyway, in both cases, the state of the reporting is notified to the professors by the mobile app.

Using the web app, the Professor can:

- Upload the teaching material
- Send to the Secretary the reporting about the issues concerning one specific classroom
- View the state of his reports
- View the state of the reports regarding a certain classroom, even if they are sent by different professors

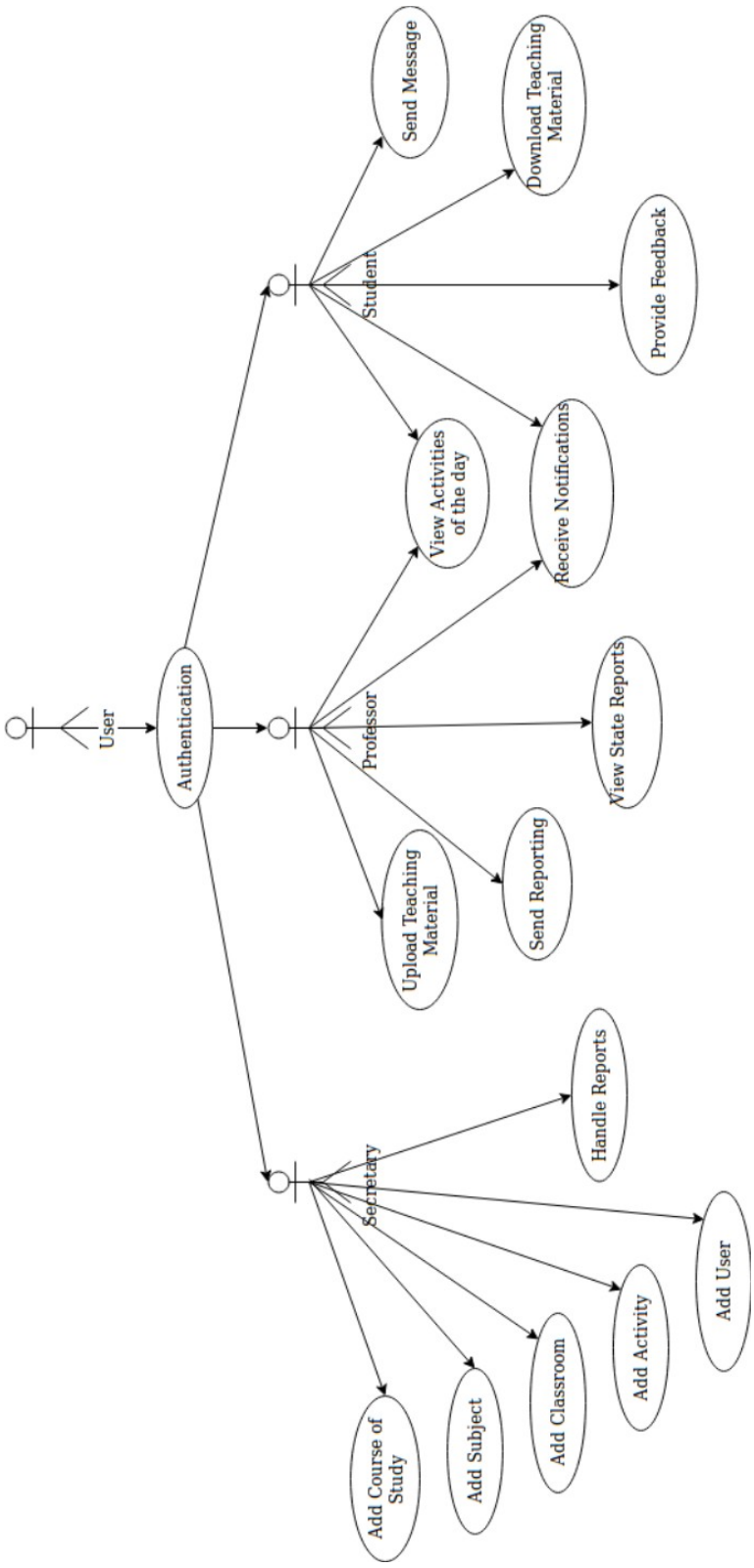
Instead, using the mobile app, the Professor can:

- View the activities of the day, including the map of the classrooms
- Receive the notification regarding his reports or classrooms
- Receive the notification when a student send a message to him
- Receive the notification when a student send a feedback regarding the lessons or the teaching material

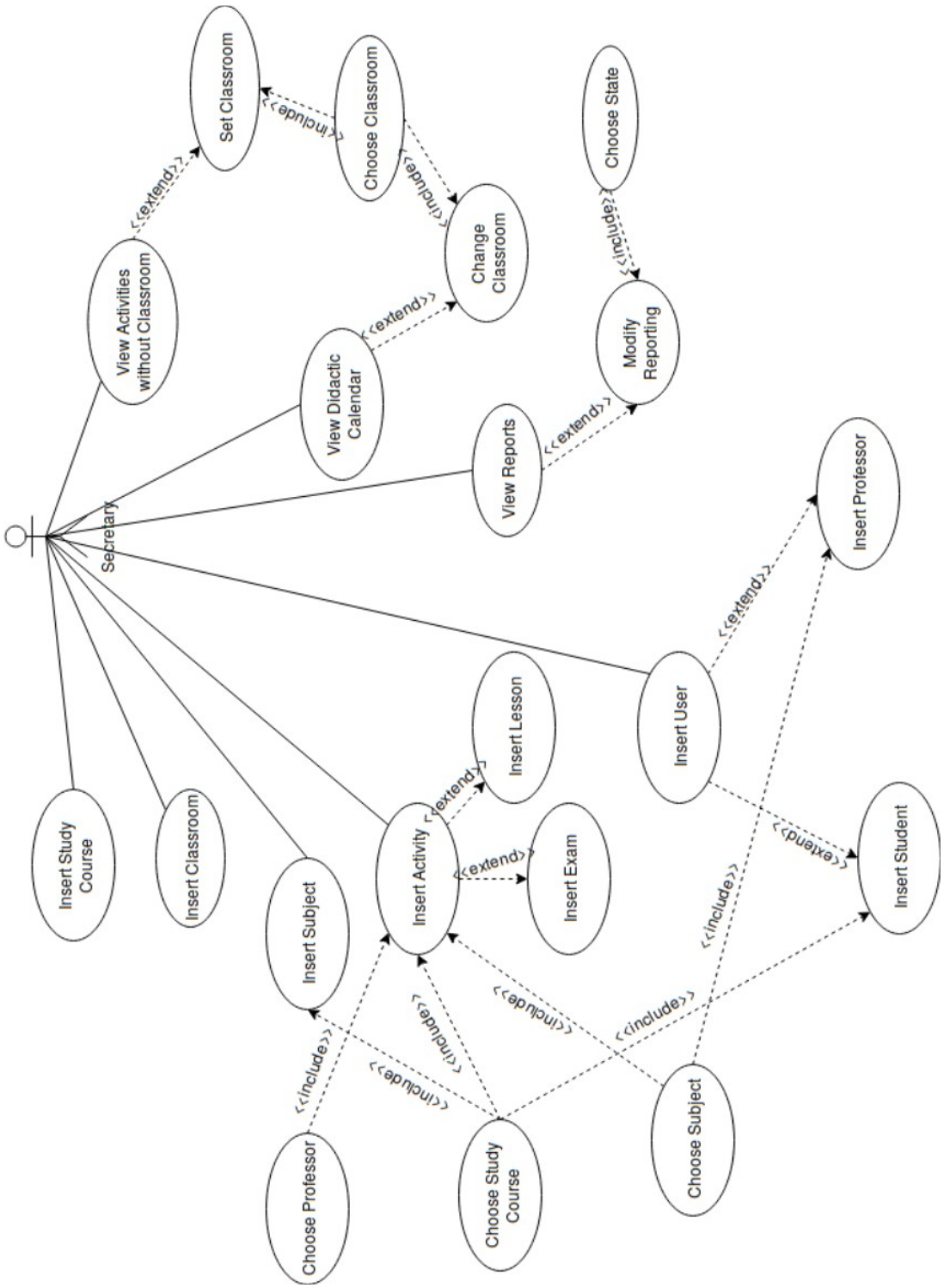
Using the mobile app, the Student can:

- Write messages in the channel chats of the subjects that he studies and the message can be:
  - Public, visible to the professor and to the other students who study the same subject
  - Private, only visible to the professor or another student
- Download the teaching material
- Provide anonymously feedback about the lessons or the teaching material
- View the activities of the day, including the map of the classrooms
- Receive the notification when there has been a change of classroom for a lesson he has to attend
- Receive the notification when a professor send messages to him or when the professor uploads the teaching material

3. Use Cases Description



3.1 User Secretary



3.1.1 Use case: Insertion of the Course of Study
Pre-condition: Login and token verified
Post-condition: Creation of the new Course of Study

1. Write the name of the new study course
2. Click the button for the insertion

3.1.2 Use case: Insertion of the Subject
Pre-condition: Login and token verified
Post-condition: Creation of the new Subject

1. Write the name of the new subject
2. Select the Course of Study for the subject
    1. <include> Choose from a list of items
3. Write the year of the course in which the subject is taught
4. Click the button for the insertion

3.1.3 Use case: Insertion of the Classroom
Pre-condition: Login and token verified
Post-condition: Creation of the new Classroom

1. Write the name of the new classroom
2. Write the latitude
3. Write the longitude
4. Click the button for the insertion

3.1.4 Use case: Insertion of the Activity
Pre-condition: Login and token verified
Post-condition: Insertion of the new Activity

1. Choose the activity type between Exam and Lesson
2. Select the course of study
    1. <include> Choose from a list of items
3. Select the subject
    1. <include> Choose from a list of items
4. Select the professor
    1. <include> Choose from a list of items
5. Choose the starting time and date
6. Choose the ending time and date
7. Click the button for the insertion
    1. <extend> Insert Exam (if activity type is exam)
    2. <extend> Insert Lesson (if activity type is lesson)

3.1.5 Use case: Insertion of the User
Pre-condition: Login and token verified
Post-condition: Insertion of the new User

1. Write the username
2. Write the password
3. Choose the user type between Professor and Student
    1. <extend> Insert Professor
        1. Write the first name
        2. Write the last name
        3. Write the biography
        4. Write the reception time
        5. Select the subject
            1. <include> Choose from a list of items
        6. Click the button for the insertion
    2. <extend> Insert Student
        1. Write the first name

2. Write the last name
3. Choose the date of birth
4. Select the study course
    1. <include> Choose from a list of items
5. Choose the enrollment year
6. Click the button for the insertion

3.1.6 Use case: View Activities Without Classroom
Pre-condition: Login and token verified
Post-condition: Set a Classroom for the selected activity

1. Open the didactic calendar page
2. Display the activities that need a classroom
3. Select the activity to modify
4. Select the classroom
    1. <include> Choose from a list of items
5. Click the button for the setting
    1. <extend> Set the classroom for the selected activity

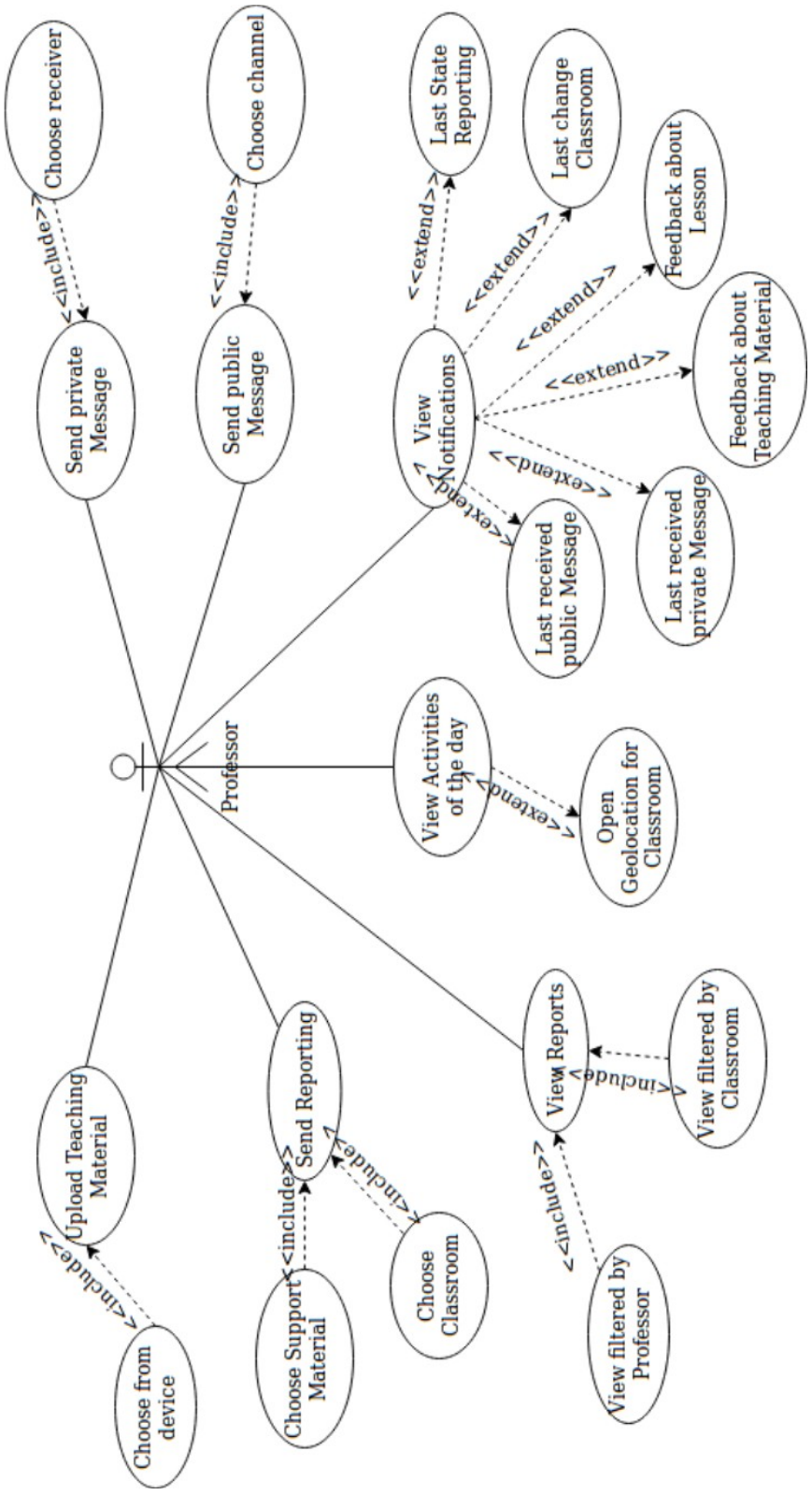
3.1.7 Use case: View Didactic Calendar
Pre-condition: Login and token verified
Post-condition: Change the Classroom for the selected activity

1. Open the didactic calendar page
2. Display the activities of the didactic calendar
3. Click the button to change the classroom for the selected activity
4. Select the classroom
    1. <include> Choose from a list of items
5. Click the button for the setting
    1. <extend> Set the new classroom for the selected activity

3.1.8 Use case: View Reports
Pre-condition: Login and token verified
Post-condition: Modify the Reporting

1. Open the reports page
2. Display the reports
3. Click the state of the reporting to modify the selected item
4. Select the new state
    1. <include> Choose from a list of items
5. Write a note to explain the change of state
6. Click the button for the modification
    1. <extend> Change the state and insert the note for the selected reporting

3.2 User Professor



3.2.1 Use case: View Notification
Pre-condition: Login in Firebase
Post-condition: Display of the last notifications

1. Redirect to index page
2. Display of the last notifications
    1. <extend> View last change of the state of the reports
    2. <extend> View last change of classroom of the activities
    3. <extend> View last group message
    4. <extend> View last private message
    5. <extend> View last feedback about the lessons
    6. <extend> View last feedback about the teaching materials

3.2.2 Use case: Send private Message
Pre-condition: Login in Firebase
Post-condition: Send private Message

1. Access to search page
2. Select the receiver
    1. <include> Choose from a list of users
3. Write the message
4. Send the message

3.2.3 Use case: Send group Message
Pre-condition: Login in Firebase
Post-condition: Send group Message

1. Access to channel page
2. Select the channel
    1. <include> Choose from a list of channels, each one for a different subject
3. Write the message
4. Send the message

3.2.4 Use case: View Activities of the day
Pre-condition: Login in Firebase
Post-condition: Display of the Activities of the day

1. Access to calendar page
2. Display of all the activities scheduled for the current day
    1. <extend> Click the name of the classroom to open the geolocation page

3.2.5 Use case: Upload Teaching Material
Pre-condition: Login and token verified
Post-condition: Upload Teaching Material

1. Open the upload material page
2. Select the file
    1. <include> Choose from device the file to upload
3. Click the button to upload the file
4. Creation of the professor directory
5. Save the file into the professor directory

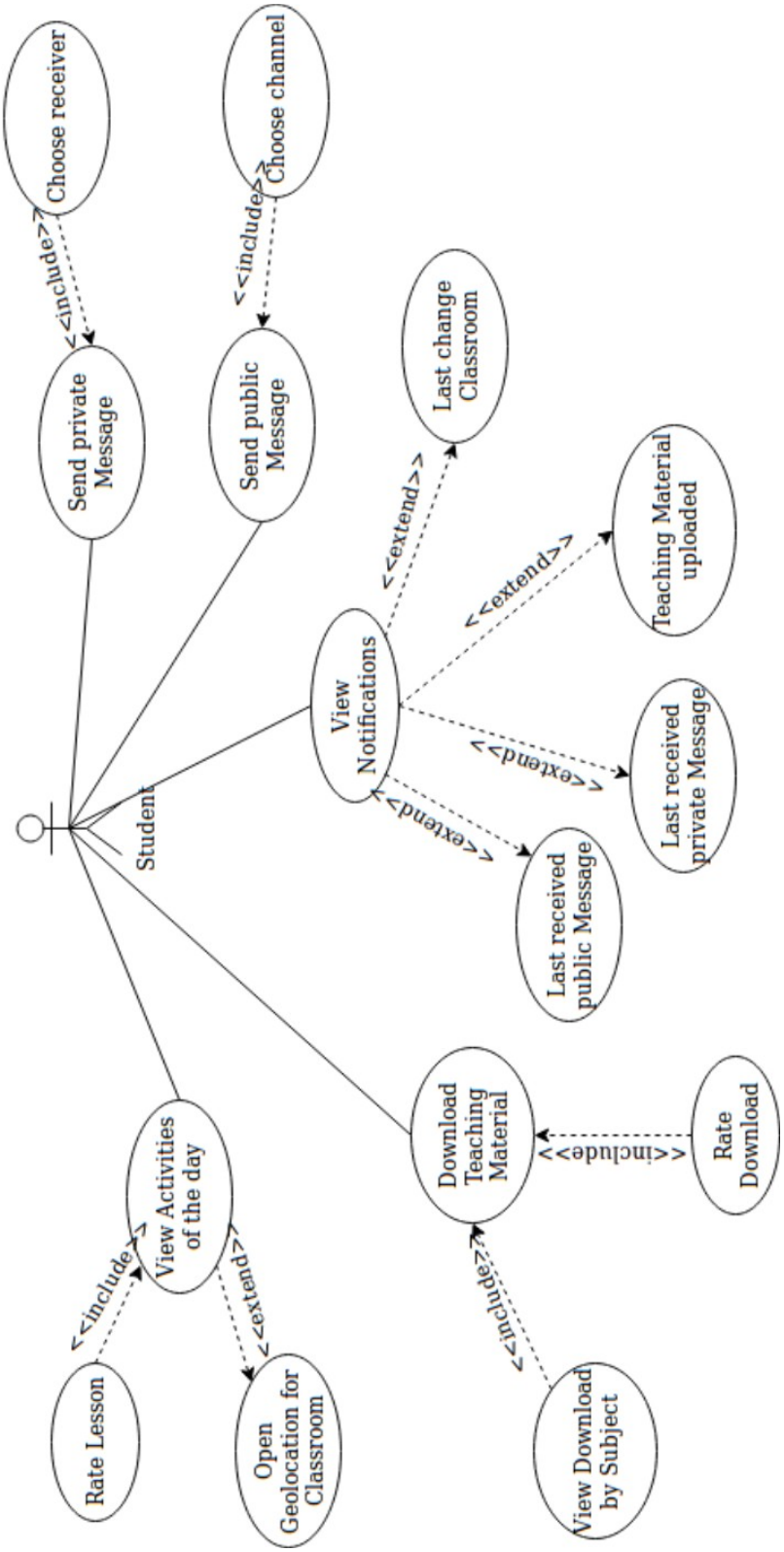
3.2.6 Use case: View Reports
Pre-condition: Login and token verified
Post-condition: Display of all the Reports

1. Open the reports page
2. Display of all the reports
    1. <include> Filter reports by classroom
    2. <include> Filter reports by professor's first name or last name

3.2.7 Use case: Send Reporting
Pre-condition: Login and token verified
Post-condition: Send Reporting

1. Open the reporting page
2. Select the classroom
    1. <include> Choose from a list of items
3. Select the support material
    1. <include> Choose from a list of items
4. Write a note for the problem to reporting
5. Click the button to send the reporting

3.3 User Student



3.3.1 Use case: Download Teaching Material
Pre-condition: Login in Firebase
Post-condition: Download Teaching Material

1. Access to download page
2. Display of all teaching materials by subject
    1. <include> Filter teaching materials by subject
3. Download selected teaching material
    1. <include> Rate teaching material downloaded
        1. Select a vote (1 – 5)
        2. Write a note
        3. Click the button to save the feedback

3.3.2 Use case: View Notification
Pre-condition: Login in Firebase
Post-condition: Display of the last notifications

1. Redirect to index page
2. Display of the last notifications
    1. <extend> View last change of classroom of the activities
    2. <extend> View last group message
    3. <extend> View last private message
    4. <extend> View last uploaded teaching material

3.3.3 Use case: View Activities of the day
Pre-condition: Login in Firebase
Post-condition: Display of the Activities of the day

1. Access to calendar page
2. Display of all the activities scheduled for the current day
    1. <extend> Click the name of the classroom to open the geolocation page
    2. <include> Rate attended lesson
        1. Select a vote (1 – 5)
        2. Click the button to save feedback

3.3.4 Use case: Send private Message
Pre-condition: Login in Firebase
Post-condition: Send private Message

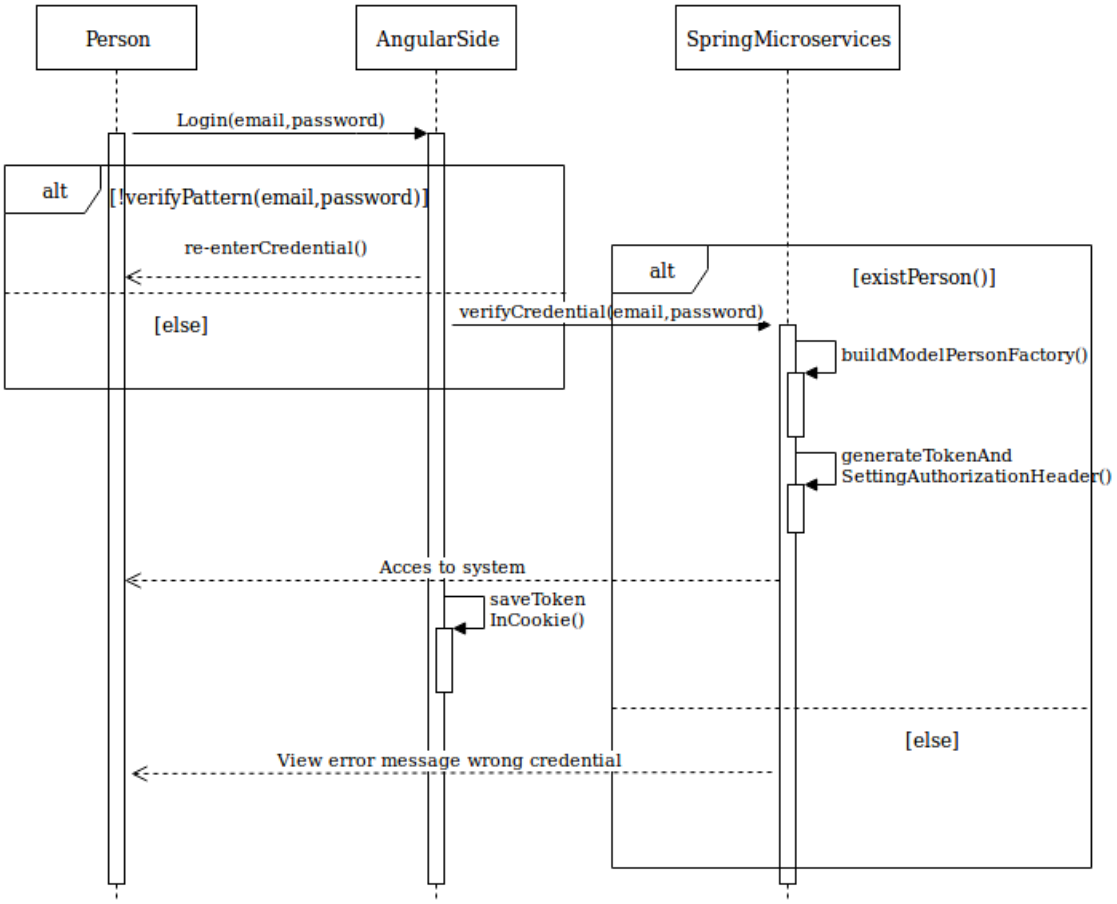
1. Access to search page
2. Select the receiver
    1. <include> Choose from a list of users
3. Write message
4. Send the message

3.3.5 Use case: Send group Message
Pre-condition: Login in Firebase
Post-condition: Send group Message

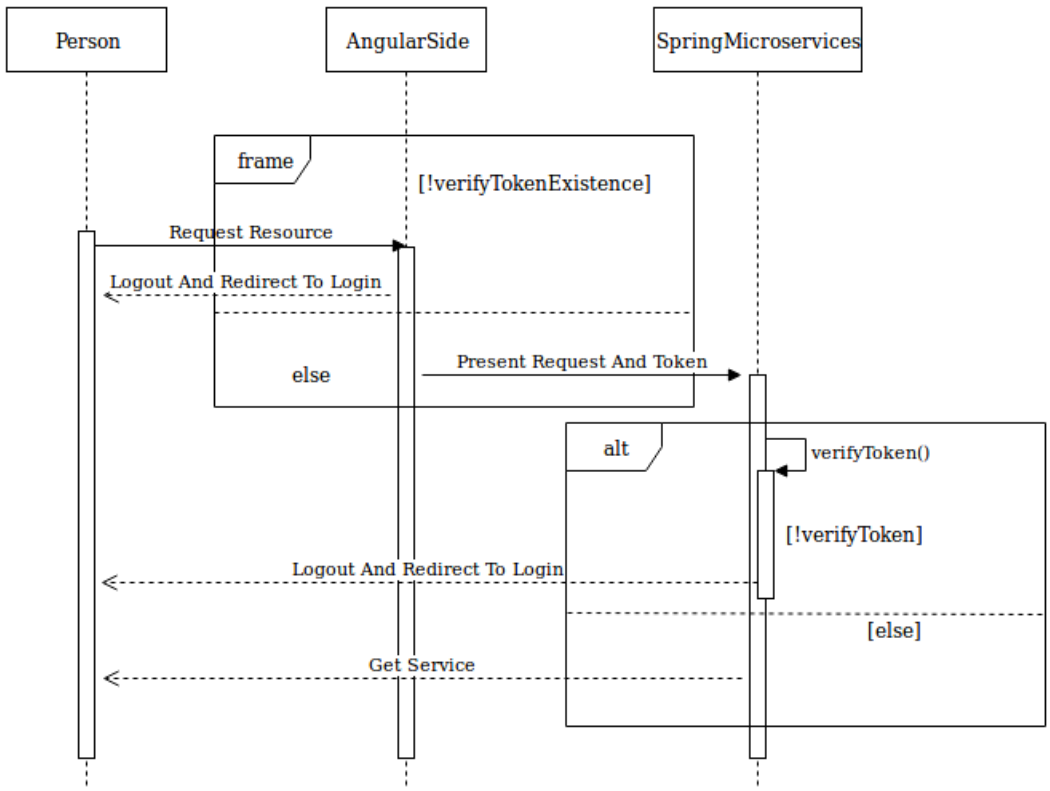
1. Access to channel page
2. Select the channel
    1. <include> Choose from a list of channels, each one for a different subject
3. Write the message
4. Send the message

4. Sequence Diagram for JWT

Login System:



Request
Service:



4.1 Login System

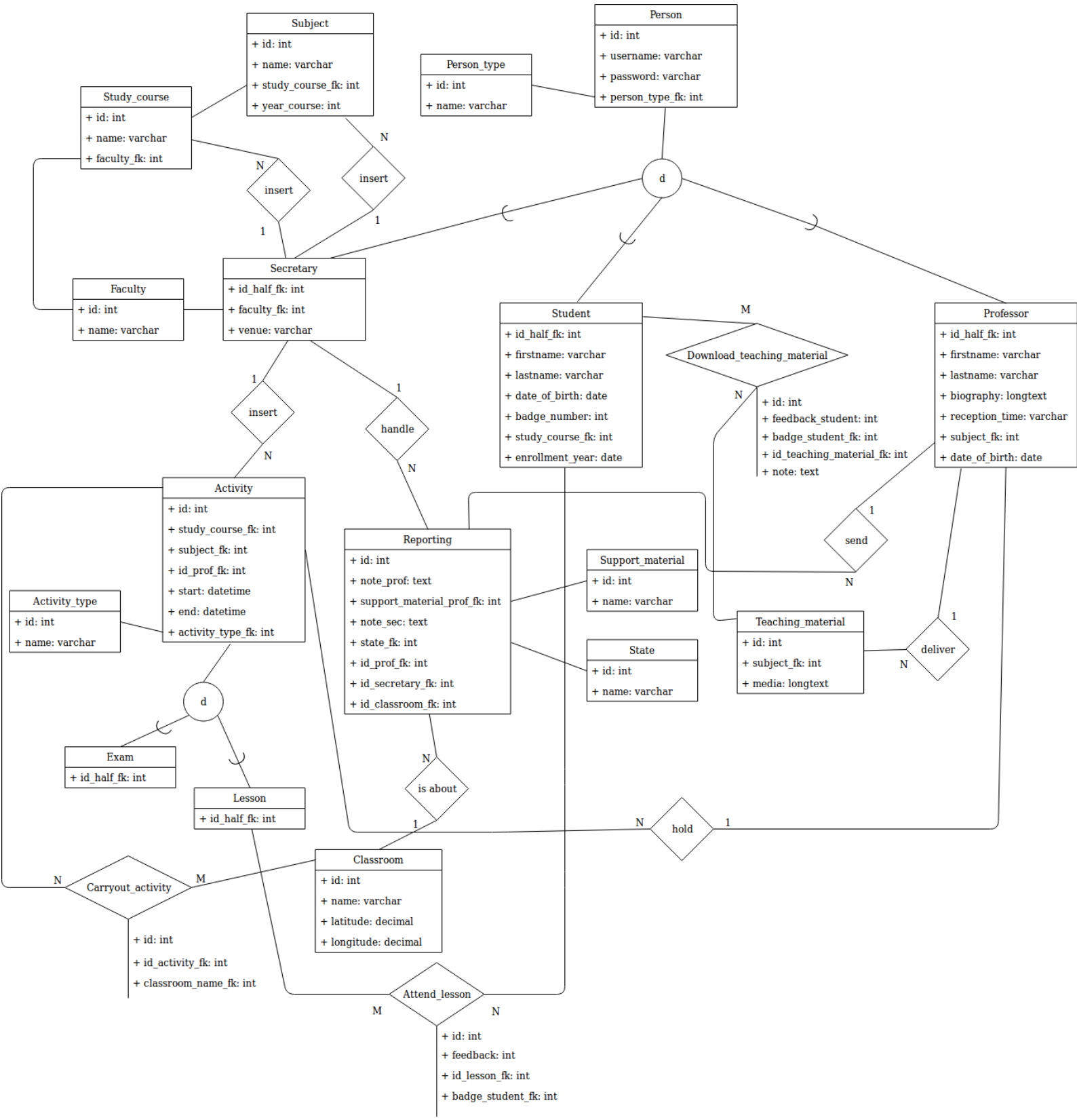
This sequence diagram explains the operation of the login. The first step is to enter the credentials of the user: the credentials must comply with a specific pattern for the login key to be activated. As the login button is clicked, the mail and password are sent to the back end and then it is performed a search for matches with the data in the database. If the user does not exist or the credentials are wrong, an error message is returned; otherwise, a token is generated, which will be set in the header in an AUTHORIZATION field and returned along with the person's template logged on to the client. The token and data for the logged-in person are saved in cookies (tokens) and localstorage (user data), respectively.

4.2 Request Service

This sequence diagram shows how the services of the Web Restful APIs are requested. When a user needs to view the data and then request it by using the different components of the page, routing is enabled for displaying the resources to the front end. Each route must be checked for the existence of the token. When the token does not exist, a redirect to the login page will be made a redirect and also a cleanup of cookies and localstorage; otherwise the request is sent to the back end service, which initially checks the token and then invokes the data from the service. If the token is not verified, or has expired, an error will be returned to the front end and a logout with redirect to the login page is performed and also a cleanup of localstorage and cookies; otherwise the requested data (JSON) will be returned correctly.

5. Database Description

5.1 Relational Database



The table Person represents the generic user who is logged in the system. The person_type foreign key is used to specify the type of the user, in fact:

- 1 is the Secretary
- 2 is the Professor
- 3 is the Student

are taken from the static table Person_type.
Then, the tables Secretary, Professor and Student are disjoint specialization of the table Person and their id is half primary key and half foreign key, since it is derived from the parent table.
The table Secretary has a faculty foreign key which is used to define the name of the faculty the secretary is associated to:

- 1 stands for Engineering
- 2 stands for Medicine
- 3 stands for Letters

that are taken from the table Faculty.
The table Professor has a subject foreign key which is used to define the details of the subject that the professor teaches.
The table Student has a study_course foreign key which is used to define the details of the study course that the student is enrolled to.
The table Activity represents the generic activity that the secretary can insert into the system. The activity_type foreign key is used to specify the type of the activity, in fact:

- 1 is Lesson
- 2 is Exam

are taken from the static table Activity_type.
The table Activity has three different foreign keys:

- study course, used to define the details of the study course that the activity refers to
- subject, used to define the details of the subject that the activity is about
- professor id, used to define the details of the professor who keeps the activity

Both tables Lesson and Exam are disjoint specialization of the table Activity and their id is half primary key and half foreign key, since it is derived from the parent table.
The table Attend_lesson is derived from a N:M relation of the ER diagram and represents the connection between the student and the lesson he attends. This table has two foreign keys:

- id_lesson, used to define the details of the lesson the student attend
- badge_student, used to define the details of the student who attends the lesson

This table also has a field that the students have to fill, which is the feedback given to the lesson.
The table Study_course represents the course of study that the students are enrolled to. It has a faculty foreign key used to define the name of the faculty the course is associated to, taken from the Faculty table.
The table Subject represents the subjects that the professors teach and it has a foreign key which is used to define the course of study they are associated to.
The table Classroom represents the classrooms in which the activities are carried out. It contains the information about the latitude and the longitude of the classroom, later used to implement the geolocation.
The table Carryout_activity is derived from a N:M relation of the ER diagram and represents the connection between the activity and the classroom in which it is carried out. This table has two foreign keys:

- id_activity, used to define the details of the activity which is performed in that specific classroom
- classroom_name, used to define the details of the classroom assigned to that specific activity

The table Teaching_material represents the didactic documents that the professor upload to the system, so that they are available to the students to download. It has a subject foreign key, used to define the subject the material refers to.
The table Download_teaching_material is derived from a N:M relation of the ER diagram and represents the connection between the student and the teaching material that he downloads. This table has two foreign keys:

- badge_student, used to define the details of the student who downloads the material
- id_teaching_material, used to define the materials that is downloaded.

This table also has two fields that the students have to fill, which are the feedback given to the material and a text note about the documents.
The table Reporting represents the reports that the professors send to the secretary about the problems that a classroom has. It has a few foreign keys:

- support_material_prof, used to define the support material of the classroom that does not work well
- state, used to define the state of the reporting and it is modified by the secretary when the request is analyzed
- id_prof, used to define the professor who sends the reporting
- id_secretary, used to define the secretary that modifies the state of the reporting and write a note about that
- id_classroom, used to define the classroom that the reporting is about

The state foreign key is chosen from a static table State, filled as shown below:

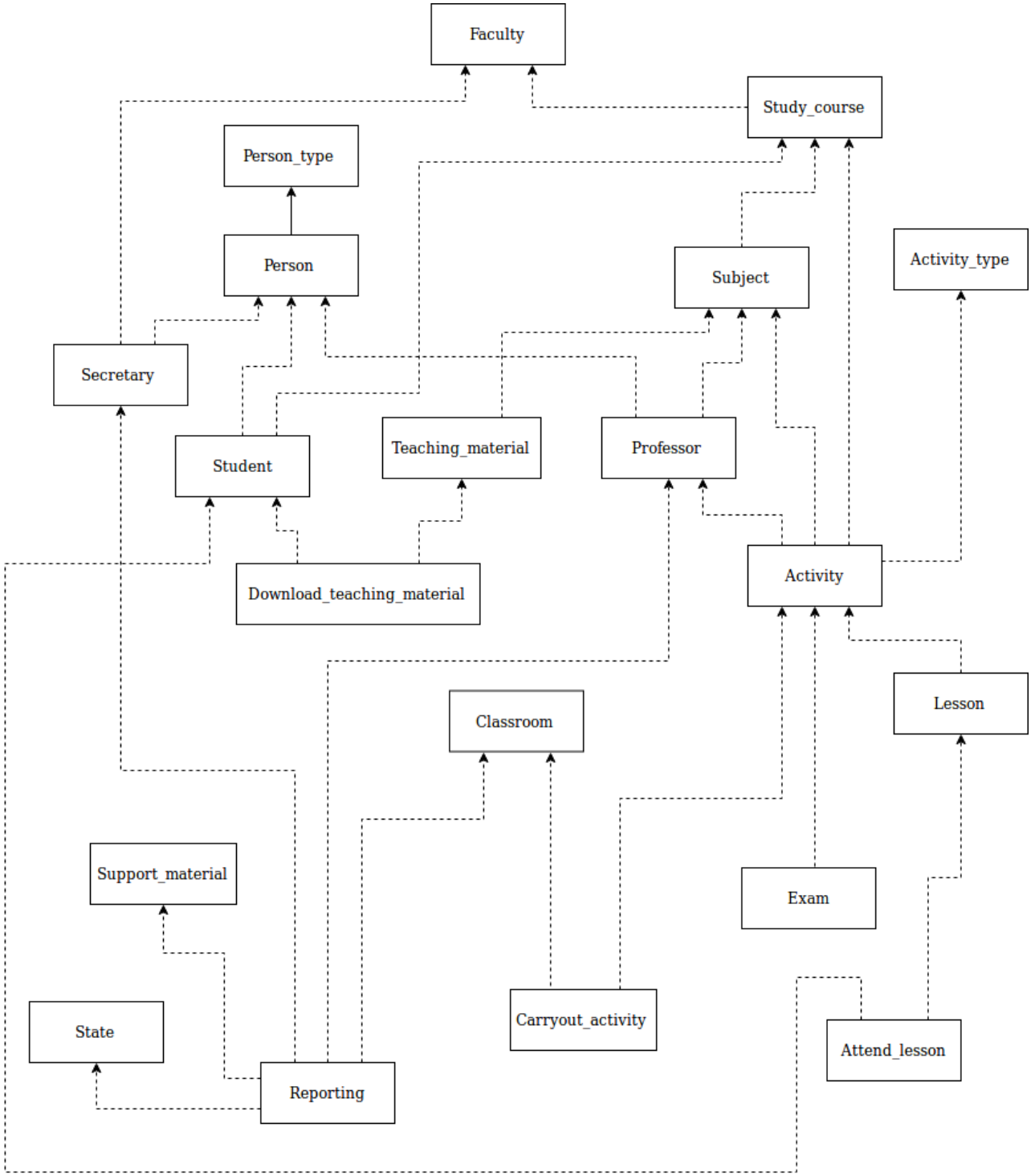
- 1 is Taken In
- 2 is In Progress
- 3 is Resolved
- 4 is Refused

The support_material foreign key is chosen from a static table Support_material, filled as shown below:

- 1 is WiFi
- 2 is Video projector
- 3 is Air conditioning
- 4 is Electronic whiteboard

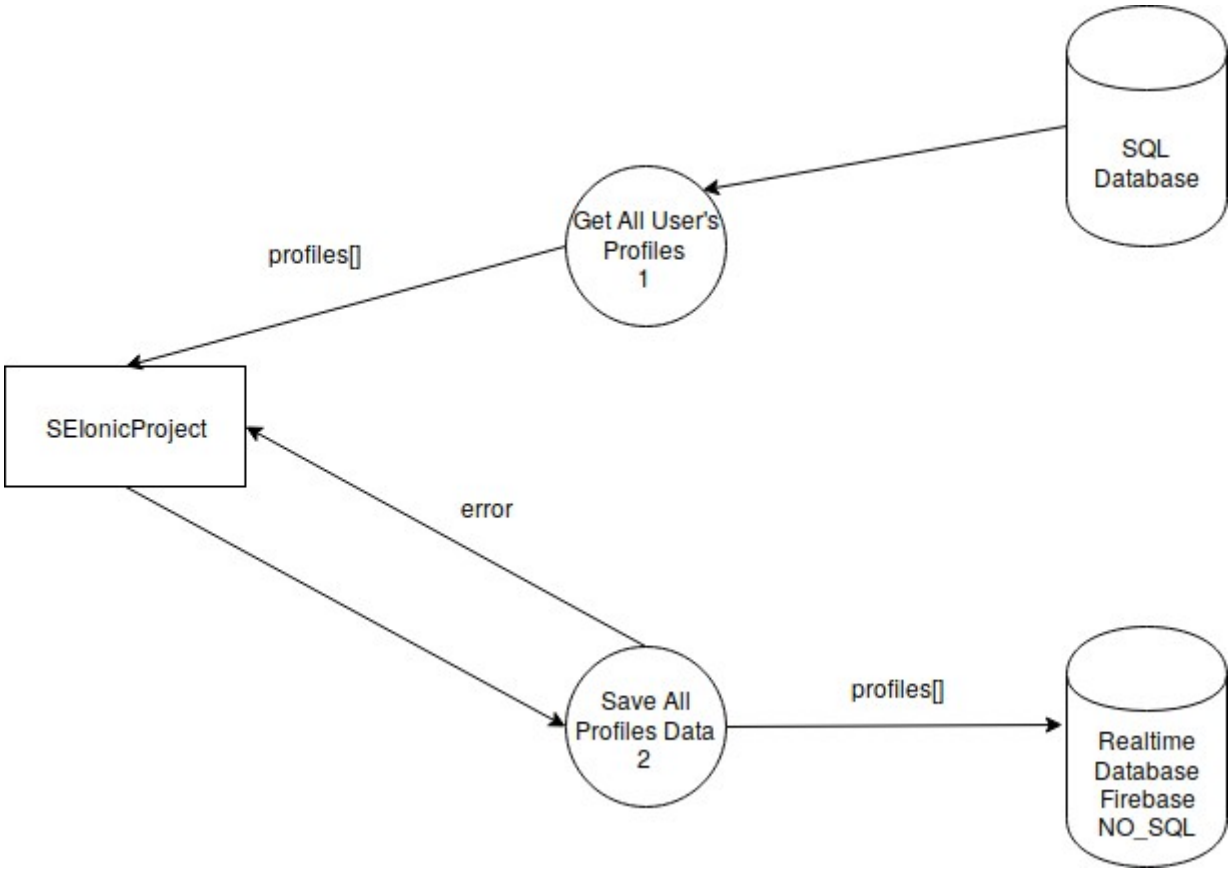
5.2 Dependency Diagram

Based on what described in the previous chapter, we can extract the Dependency Diagram as follows.

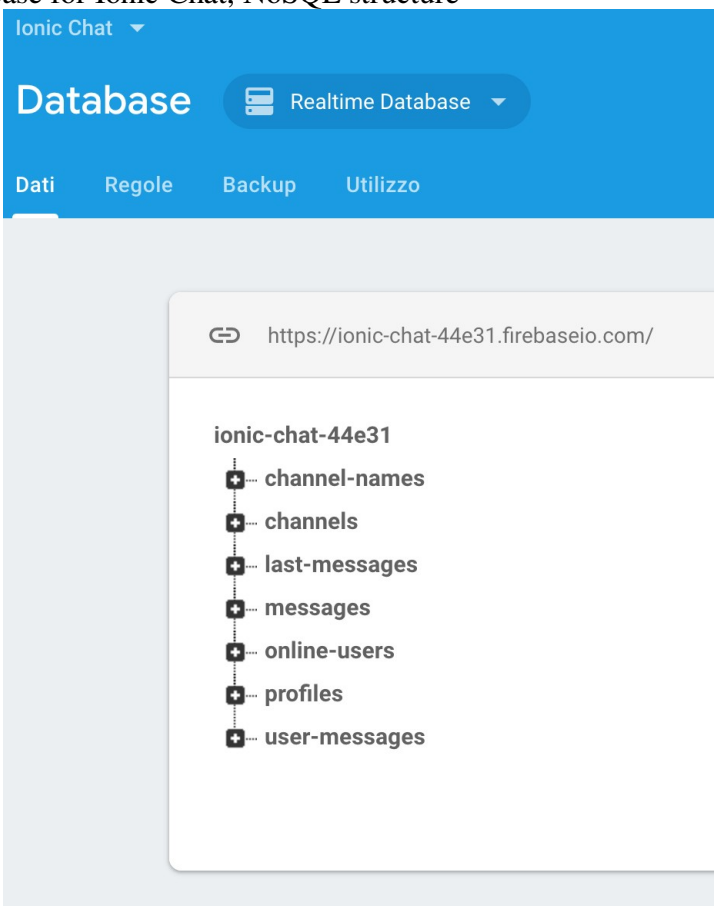


5.3 Non-Relational Database

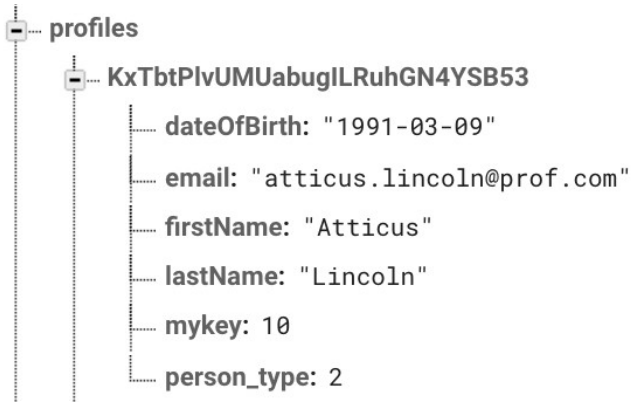
5.3.1 Starting the Ionic Chat



5.3.2 Real Time Database for Ionic Chat, NoSQL structure



Profiles

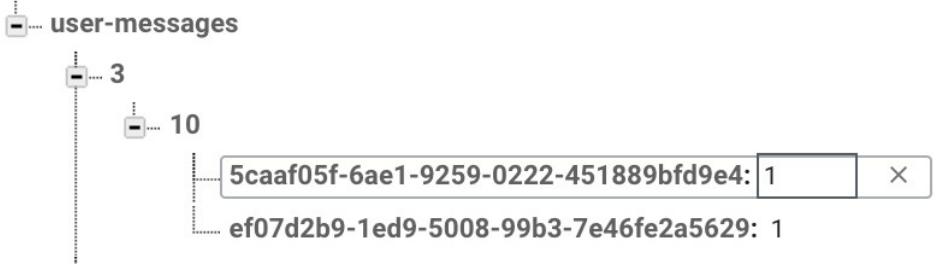


Profiles are acquired (email and password) at the start of the app-mobile and activated at the first login, setting name, surname, date of birth and type of person.

Messages

When a message is sent, the following structure is created. The message key is instantiated in addition to data about the sender and recipient.

User Messages

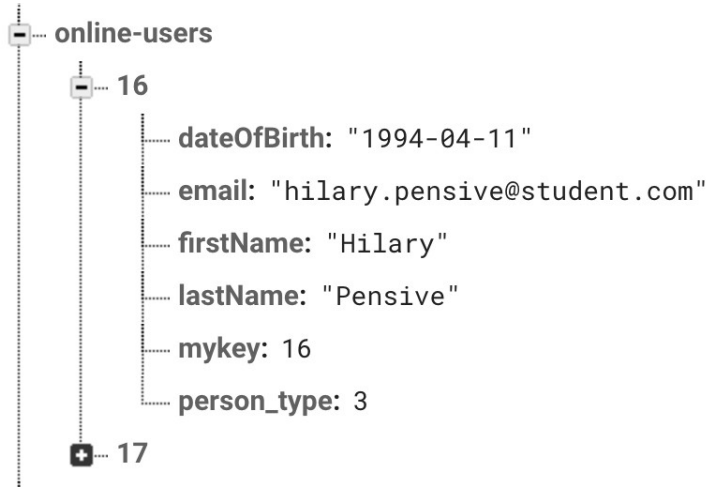


Messages are tracked where the SnapShot keys indicate the ids of the sender and of the recipient, in order of succession, and the ValueChanges indicate the message key.

Last Messages

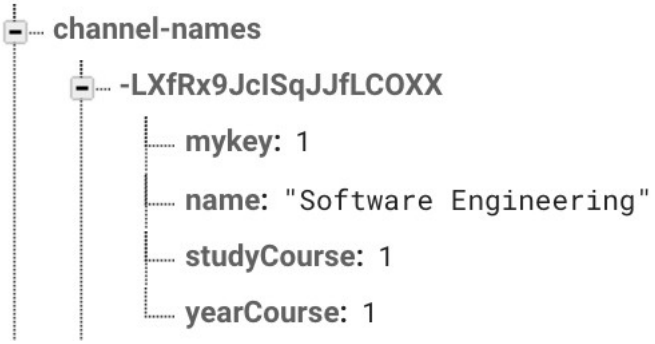
It keeps track of the last messages sent by a pair of users. The structure is similar to that of user messages.

Online Users



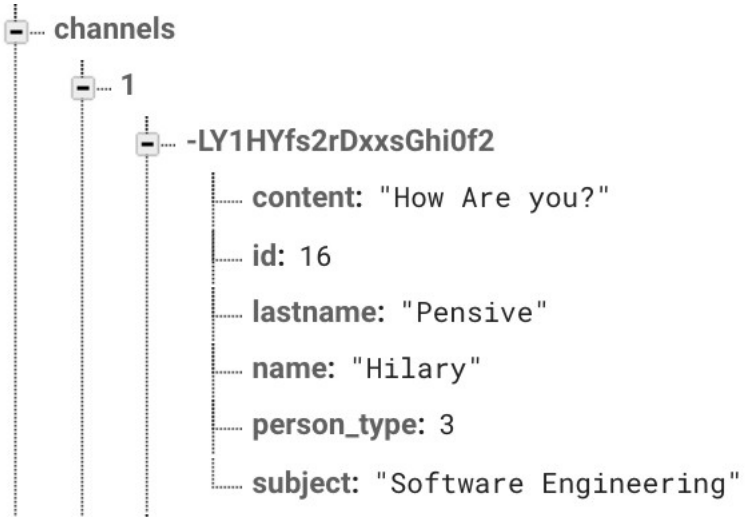
List of online users with a SnapShot key that represents the ID and contains the online user profile.

Channel Names



The channel-names are created when the access to the channel page is performed by listening to the SQL database and then there is a check for any missed channels: the channel identifies the subject to the course on which it is registered. The structure is shown above, where each course has its own key, corresponding to the ID of the Relational database.

Channels



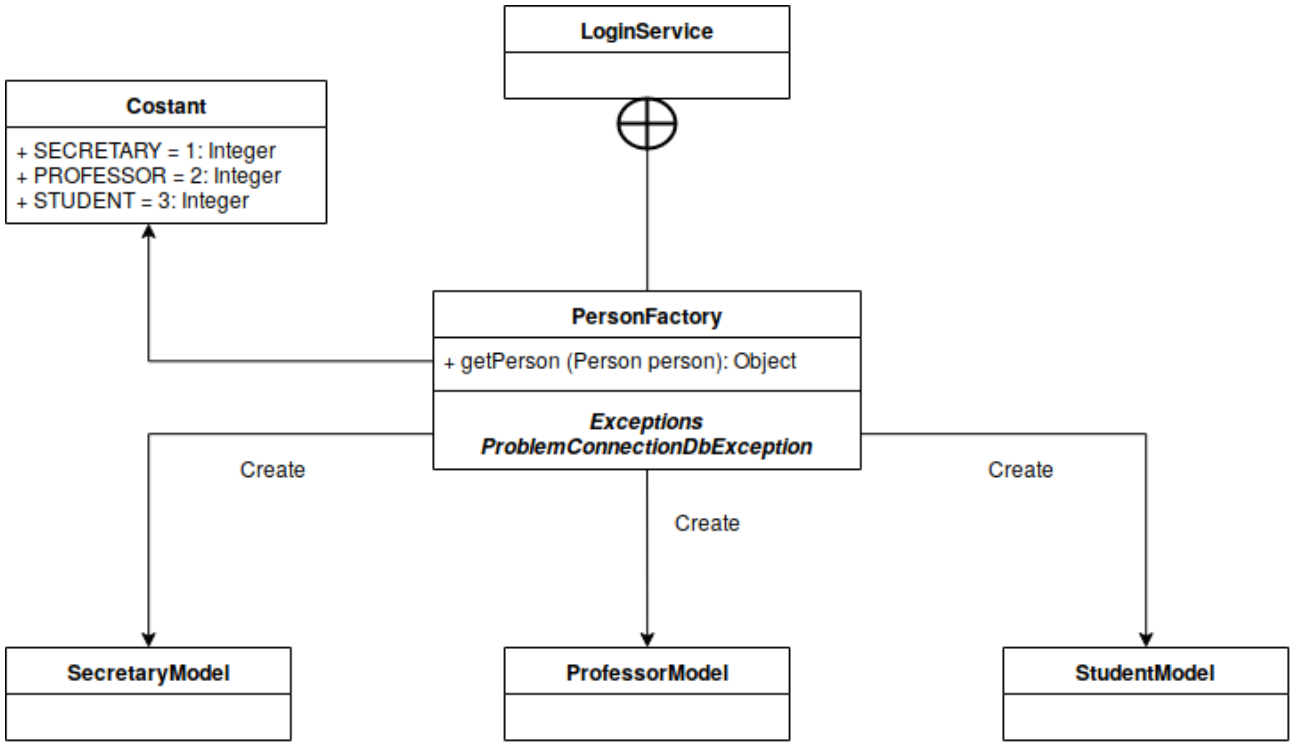
All the messages sent to the channel are tracked, where the SnapShot value indicates the subject id and the message sender is included in it.

Read and write permissions on the Real Time Database for Chat Messaging NOSQL

```
{
  /* Visit https://firebase.google.com
  "rules": {
    "messages": {
      ".read": true,
      ".write": true
    },
    "online-users": {
      ".read": true,
      ".write": true
    },
    "channel-names": {
      ".read": true,
      ".write": true
    },
    "channels": {
      ".read": true,
      ".write": true
    },
    "last-messages": {
      ".read": true,
      ".write": true
    },
    "user-messages": {
      ".read": true,
      ".write": true
    },
      "profiles": {
        ".indexOn": ["firstName"],
          ".read": true,
            "$uid": {
              ".write": true,
                ".read": true,
            }
      }
  }
}
```

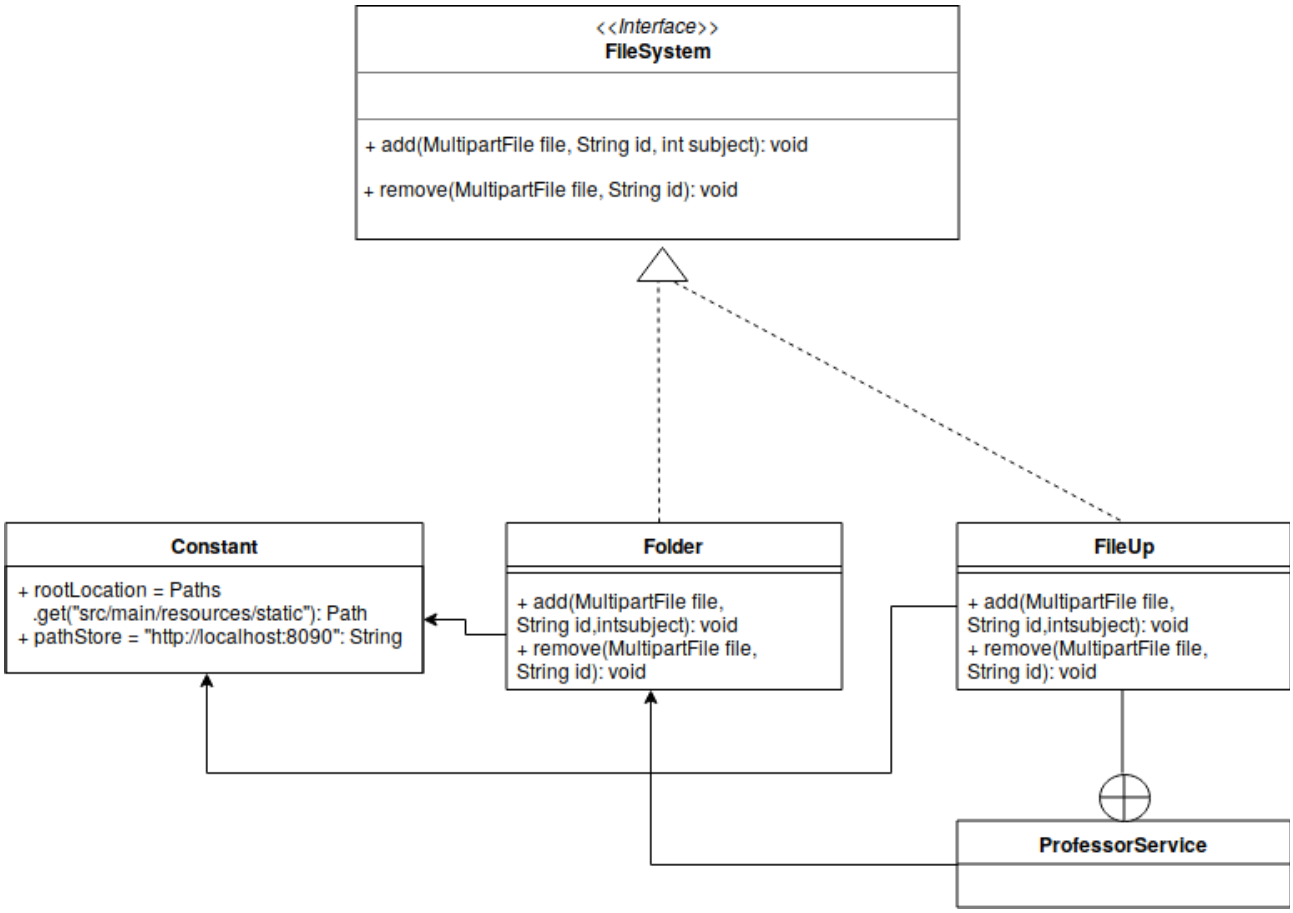
6. Design Pattern

6.1 Abstract Factory

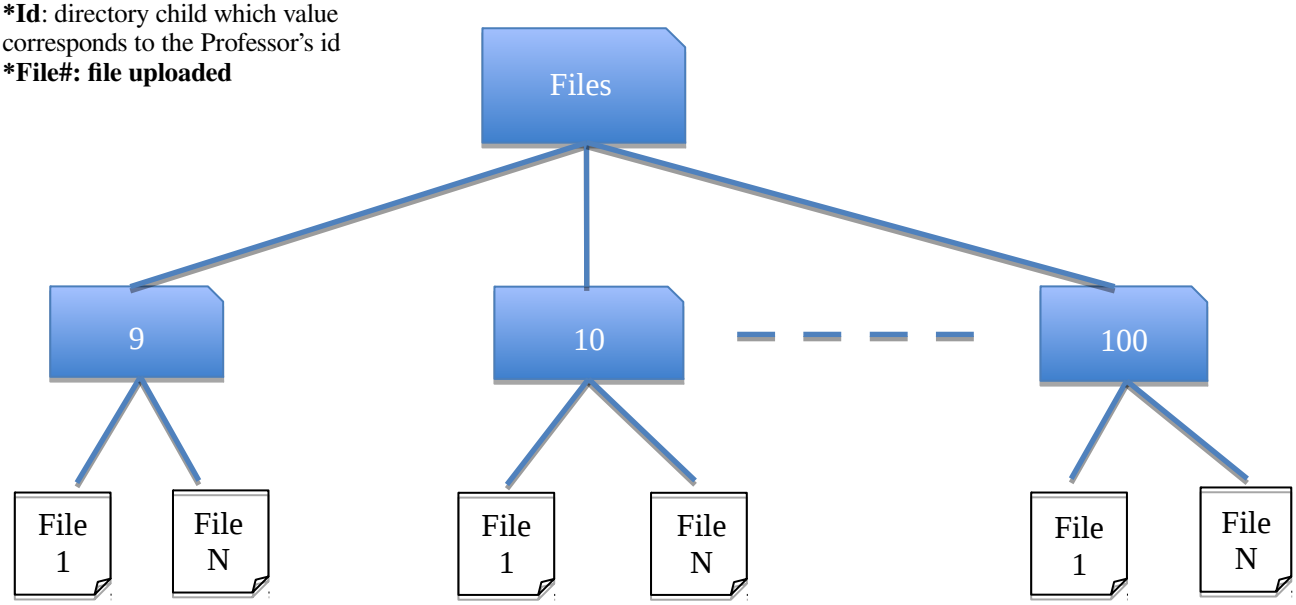


In class-based programming, the factory method pattern is a creational pattern that uses factory methods to deal with the problem of creating objects without having to specify the exact class of the object that will be created.
We used this design pattern to generate the user model logged in to the client architecture; the PersonFactory was included as a class nested within the LoginService. It is responsible for generating, based on the logged user, the correct person model (SecretaryModel, ProfessorModel, StudentModel) which will then be returned as a generic Object to the PersonController and consequently to the front end; where then a check will be made, with redirect to the page accordingly (Routing Angular), on the attribute type person (thanks to the fixed enum).

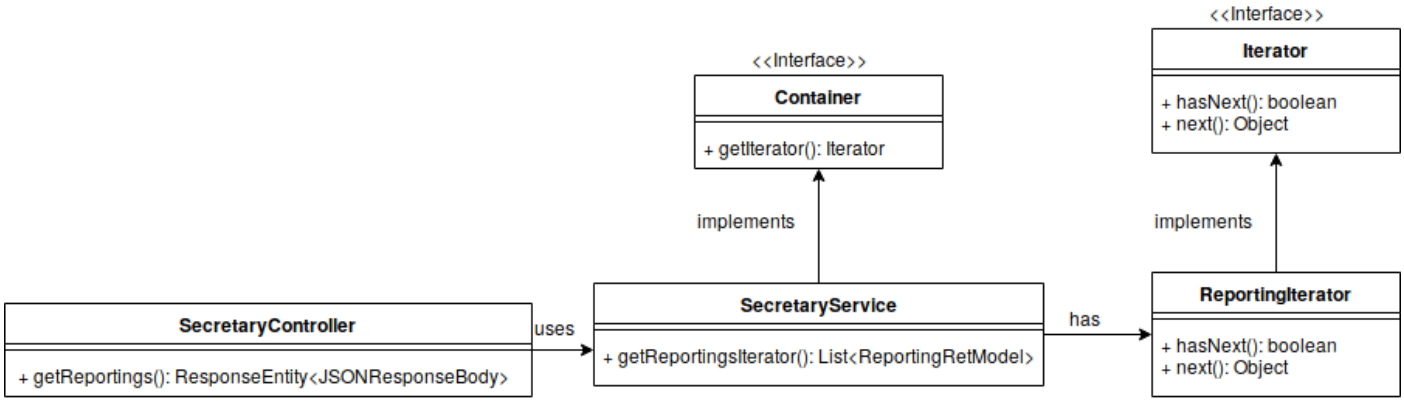
6.2 Composite



In software engineering, the composite pattern is a partitioning design pattern and falls into the category of structural patterns. The composite pattern describes a group of objects that is treated the same way as a single instance of the same type of object. The intent of a composite is to "compose" objects into tree structures to represent part-whole hierarchies. Implementing the composite pattern lets clients treat individual objects and compositions uniformly.
We used the composite pattern to create the hierarchical structure of directories and files on upload. The common interface is FileSystem and the two classes that implement it are Folder and FileUp that override the methods, the composite class in our case is the ProfessorService that uses the previous classes to generate the tree related to the upload. An example result can be the following:
***Files**: directory parent
***Id**: directory child which value corresponds to the Professor's id
***File#: file uploaded**



6.3 Iterator



Iterator pattern is a behavioral pattern and it is commonly used in Java and .Net programming environment. It is used to access the elements of a collection object in a sequential manner, without knowing the underlying representation.
We implemented this pattern in order to access the list of Report Objects contained in the database. We created two interfaces, Container and Iterator, where the first one returns the iterator and the second one develop the navigation method. The SecretaryController uses the SecretaryService, which is the class that implements the Container and has an inner class ReportingsIterator that gets the Objects from the database. This class is created as new in the override of the getIterator() method of the Container interface. Then, the method is used by the method getReportingsIterator().

7. Test Description

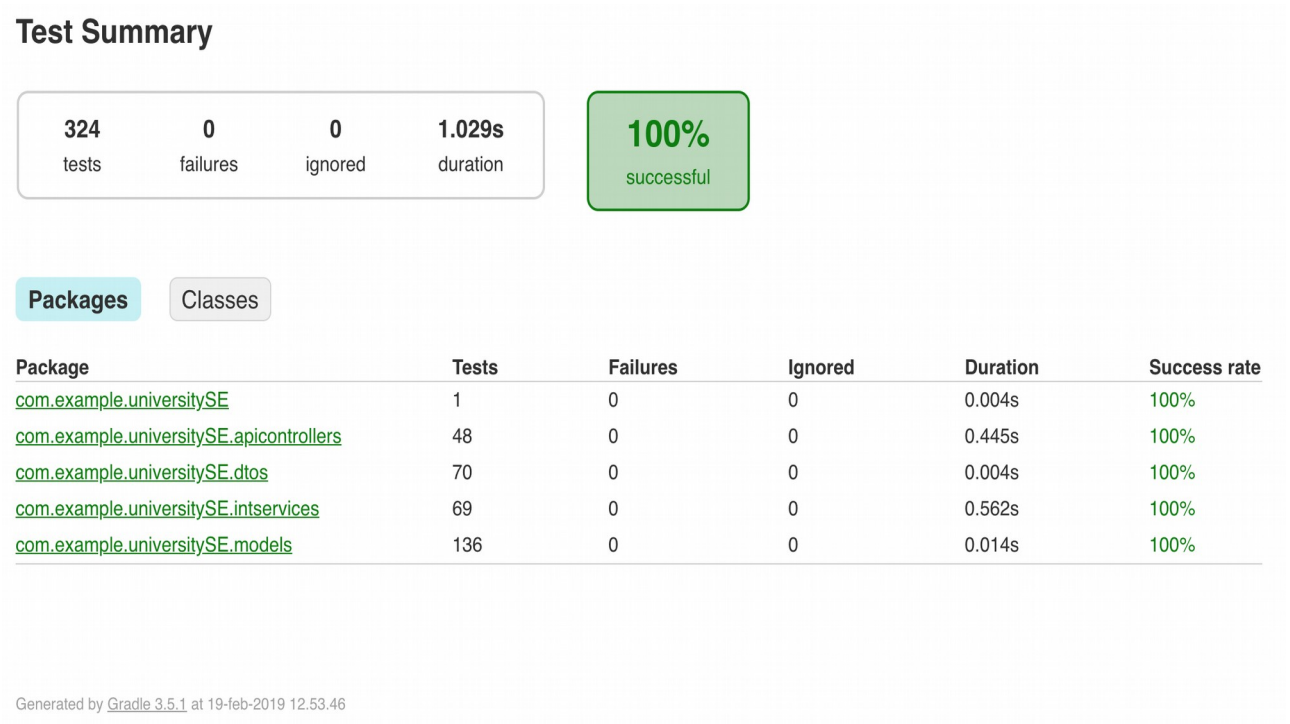
7.1 JACOCO

JaCoCo is an open-source toolkit for measuring and reporting Java code coverage. JaCoCo is distributed under the terms of the Eclipse Public License. It was developed as a replacement for EMMA, under the umbrella of the EclEmma plug-in for Eclipse.

Features

JaCoCo offers instructions, line and branch coverage.
In contrast to Atlassian Clover and OpenClover, which require instrumenting the source code, JaCoCo can instrument Java bytecode using two different approaches:
- like JCov(1) on the fly while running the code with a Java agent (computational process with communicative and autonomy functions)
- like Cobertura and JCov prior to execution (offline)
And can be configured to store the collected data in a file, or send it via TCP. Files from multiple runs or code parts can be merged easily. Unlike Cobertura and EMMA it fully supports Java 7, Java 8, Java 9, Java 10 and Java 11.

- JCov is the tool which has been developed and used with Sun JDK (and later Oracle JDK) from the very beginning of Java: from the version 1.1. JCov is capable of measuring and reporting Java code coverage. JCov is distributed under the terms of the GNU Public License (version 2, with the Classpath Exception). JCov has become open-source as a part of OpenJDK code tools project in 2014.



We used Jacoco to get a test summary of the overall code of the project, regarding the executed tests.

7.2 JUnit and Mockito

On the other hand, we could have computed the results using mathematical calculations. For example, the coverage is computed as follows:

Package	Classes	Methods	Lines
com.example.universitySE.apicontrollers	100% (5/5)	94% (48/51)	77% (67/87)

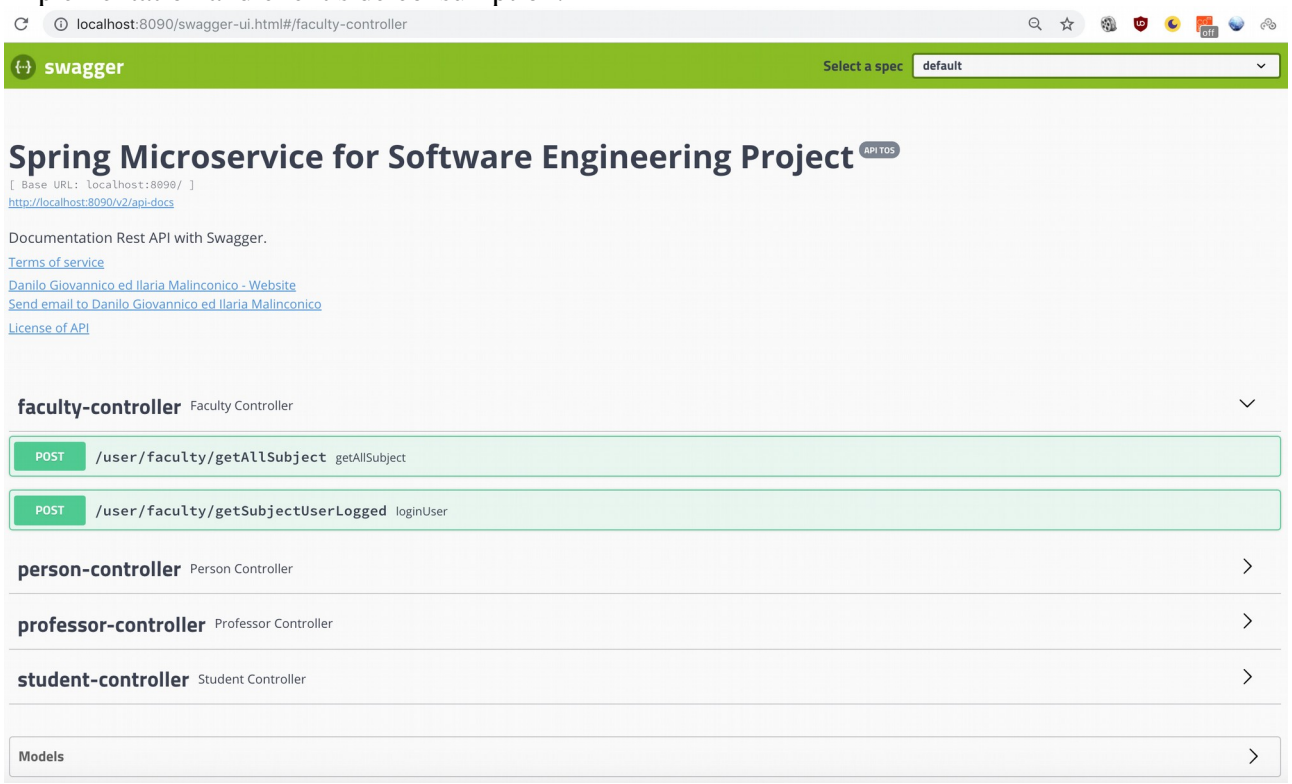
We used JUnit, which is an open source framework designed for the purpose of writing and running tests in the Java programming language. JUnit, originally written by Erich Gamma and Kent Beck, has been important in the evolution of test-driven development, which is part of a larger software design paradigm known as Extreme Programming (XP).
And we used also Mockito, which is a mocking framework, JAVA-based library that is used for effective unit testing of JAVA applications. Mockito is used to mock interfaces so that a dummy functionality can be added to a mock interface that can be used in unit testing.

8. Documentation Tools

We used the most used tools in order to automatically produce the documentation for Spring, Angular and Ionic.

8.1 SWAGGER UI

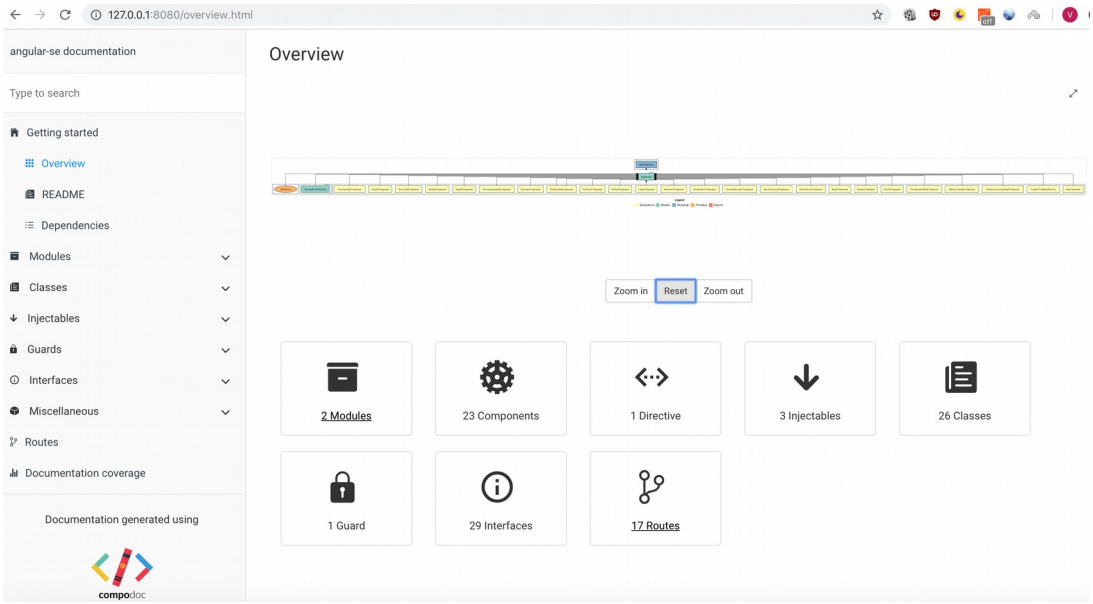
The Swagger UI is an open source project to visually render documentation for an API defined with the OpenAPI (Swagger) Specification. Swagger UI lets you visualize and interact with the API's resources without having any of the implementation logic in place, making it easy for back end implementation and client side consumption.



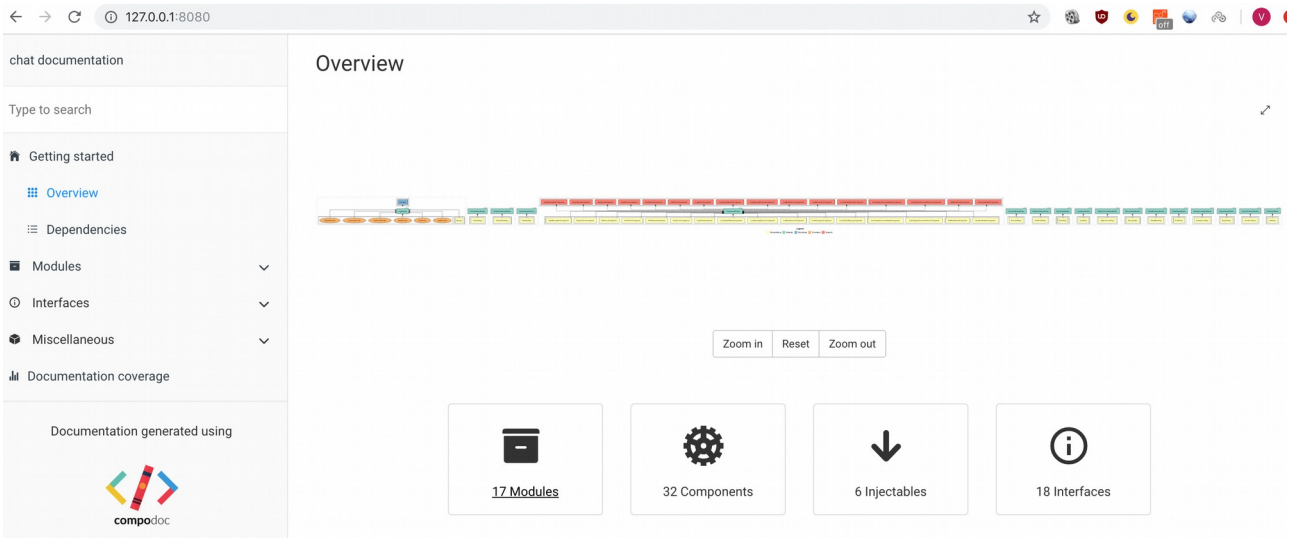
8.2 COMPODOC

Compodoc is a documentation tool for your Angular application.
The goal of the tool is to generate a documentation for all the common APIs of Angular : modules, components, injectables, routes, directives, pipes and classical classes.

- AngularSide



- SEIonicProject



9. Sprint Backlog

Week 1					
	07/01/2019	08/01/2019	09/01/2019	10/01/2019	11/01/2019
Database SQL	4				
Firebase Realtime Database NoSQL		2			
Chat Ionic: Firebase Settings, Firebase services			2		
Set workspace, Generation domain				2	
BE: Authentication JWT, Verification JWT, utils, shared, configuration CORS					5

Week 2					
	14/01/2019	15/01/2019	16/01/2019	17/01/2019	18/01/2019
Person Controller, Login service, setting authorization header jwt, factory pattern	4	2			
Angular setting, routes, activation routes, login (cookie/localstorage)		3	1		
FE: common utils			1		
Generic BE/FE Professor				2	
Generic BE/FE Secretary					2

Week 3					
	21/01/2019	22/01/2019	23/01/2019	24/01/2019	25/01/2019
BE Professor: reporting, file upload, composite pattern	3	3	2		
FE Professor				2	2
BE Secretary: insert, change classroom, didactic calendar	4	3	3		
BE Secretary: update reporting, iterator pattern		3	2		
FE Secretary				2	2

Week 4					
	28/01/2019	29/01/2019	30/01/2019	31/01/2019	01/02/2019
Generic BE/Mobile App Student	3				
BE Student: download, rating download and lesson		3	3		
Student Mobile App			1	2	
BE Secretary: show today activities. Mobile App Secretary: geolocation				4	3
Professor Mobile App					4

Week 5					
	04/02/2019	05/02/2019	06/02/2019	07/02/2019	08/02/2019
Mobile App: chat, channels	3	3	2	2	
Mobile App: Synchronize NoSQL with SQL Database for registration user and channels' names		2			
Mobile App: Professor dashboard for notification			3	3	
Mobile App: Student dashboard for notification			3	3	
FE/Mobile App: CSS/HTML styling					3

Week 6					
	11/02/2019	12/02/2019	13/02/2019	14/02/2019	15/02/2019
Bug fixing	2	2			
Testing: Mockito, Junit, Jacoco		2	3	3	
Documentation BE + Swagger				2	
Documentation FE + Compodoc					2
Documentation Mobile App + Compodoc					2

10. Burndown Chart