School Application Client: Documentation

Desenvolvimento de Software e Sistemas Móveis (DSSMV)

Licenciatura em Engenharia de Telecomunicações e Informática

LETI/ISEP

2024/25

Paulo Baltarejo Sousa

pbs@isep.ipp.pt

Disclaimer

Material and Slides

Some of the material/slides are adapted from various:

- Presentations found on the internet;
- Books;
- Web sites;
- ...

Outline

Context/Problem

Context/Problem

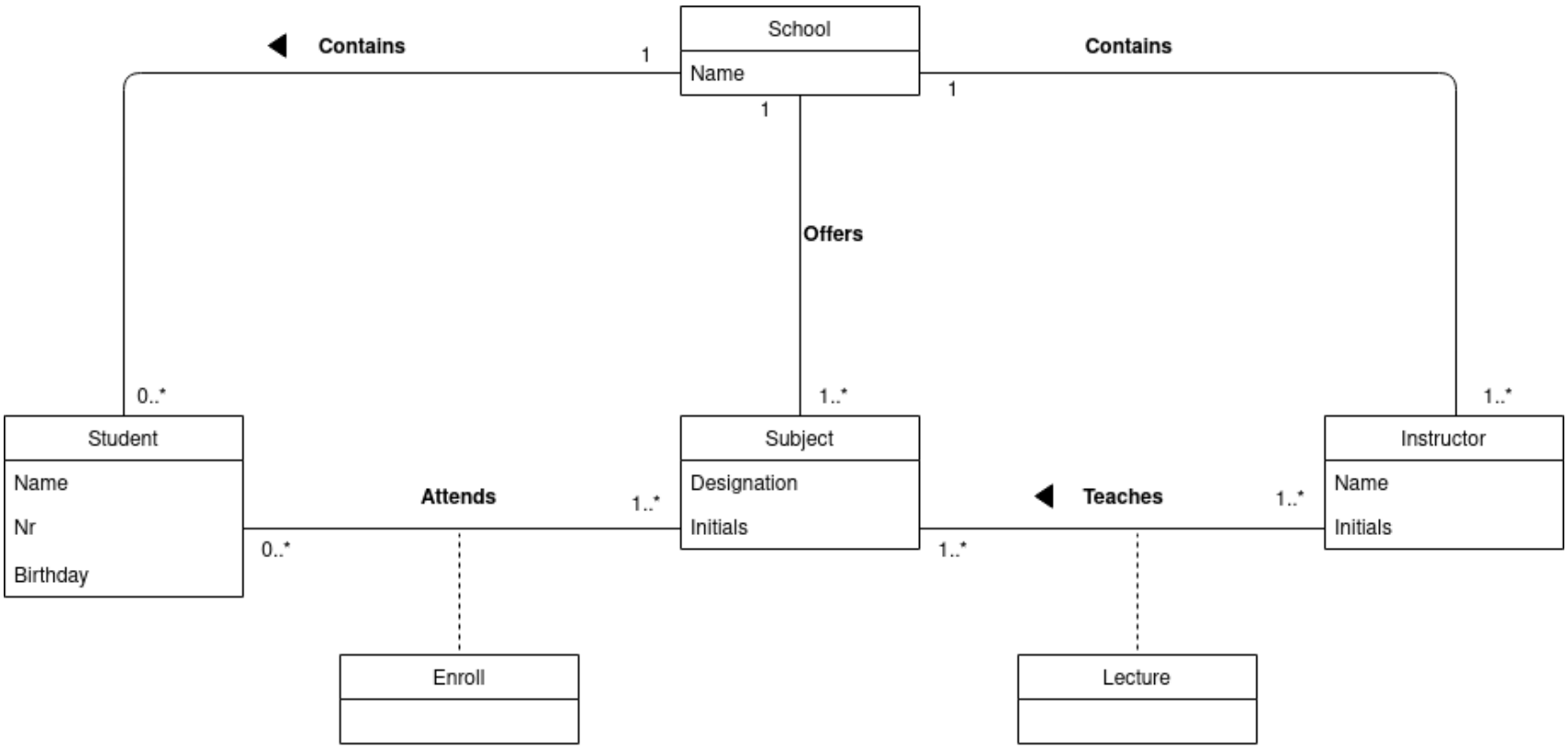
- Develop a mobile client app to interact with the `School API` server application.
- `School API` is a REST API web service.
- It provides a set of functionalities to manage a school [1].
  - School features:
    - School has students, instructors and offers subjects.
    - Instructors can teach many subjects.
    - The same subject can be taught by different instructors.
    - Student can attend to many subjects.
  - School functionalities:
    - CRUD Students
    - CRUD Subjects
    - ...

[1] Using `http://localhost:8585/swagger-ui.html`

Analysis

Domain Model

- *Explain the purpose of each entity, relationships and cardinality*.



Non-Functional Requirements (I)

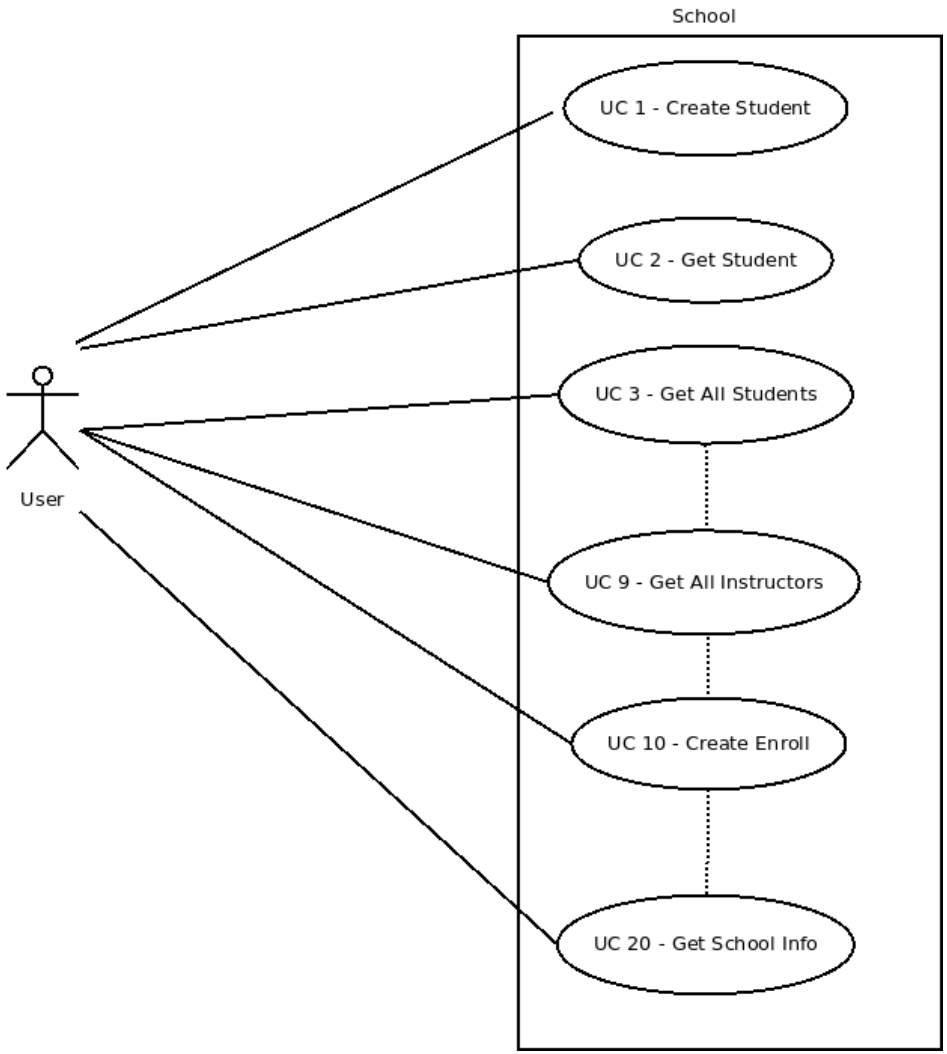
- **Usability**
  - Graphical Interface
- **Reliability**
  - No
- **Performance**
  - No
- **Supportability**
  - Huge amount of tests (unit tests)
  - Android platform
  - Minimal Android API 16

Non-Functional Requirements (II)

- +
  - **Design constraints**
    - No
  - **Implementation constraints**
    - Java language
  - **Interface constraints**
    - Android Graphical Interface

Functional Requirements (I)

- Funcionalities



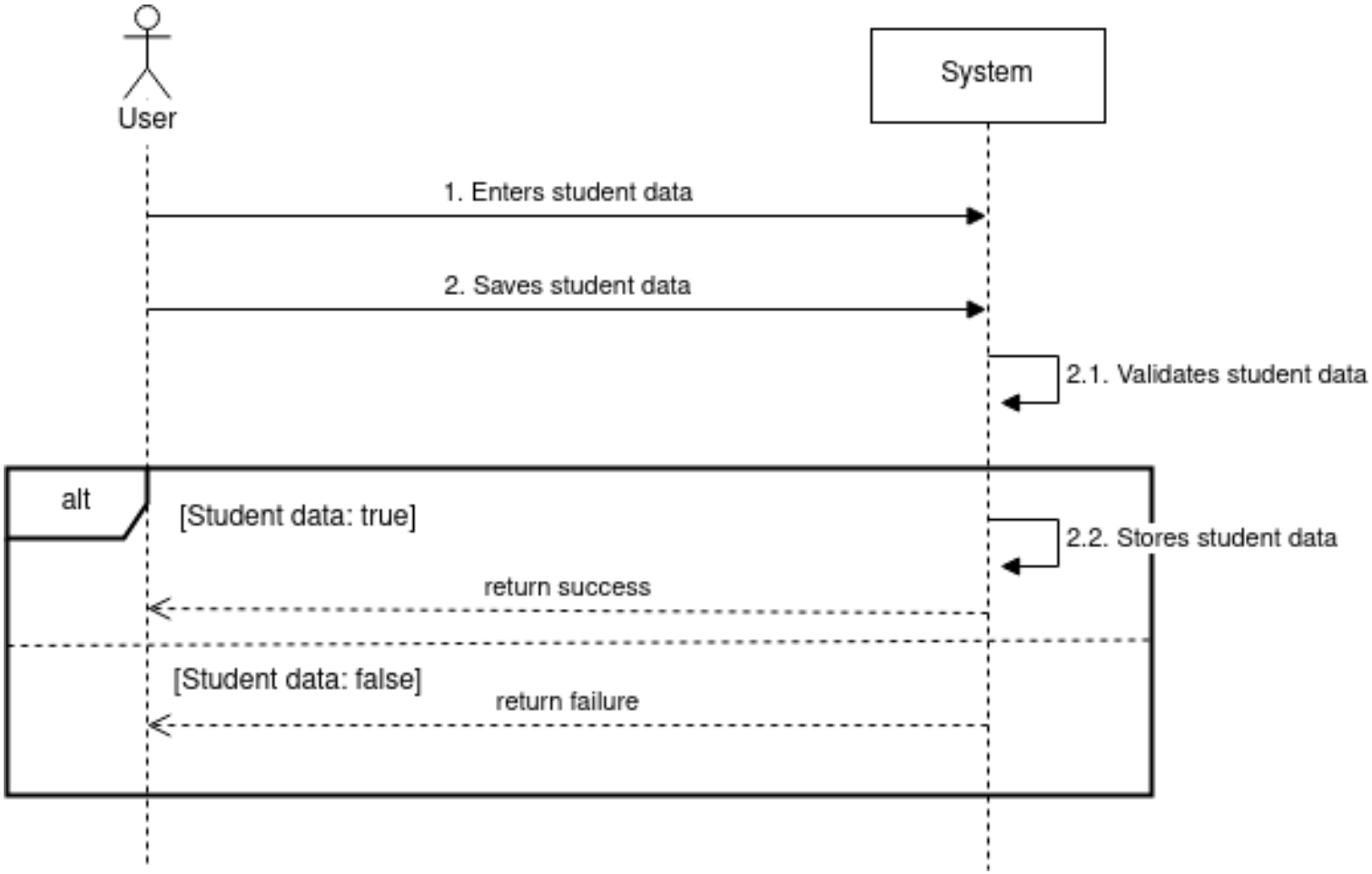
Functional Requirements (II)

- Specification: UC 1 - Create Student

Description	Add one student to the system
Pre-condition	There are no preconditions associated with this use case.
Post-condition	The number of students has to increase one.
Basic path	1. The User enters the student data. 2. The User saves the student data. 3. The System validates the student data. 4. The System stores the student data. 5. The System returns success.
Alternative path	• Invalid data • The System returns failure.

Functional Requirements (III)

- SSD: UC 1 - Create Student



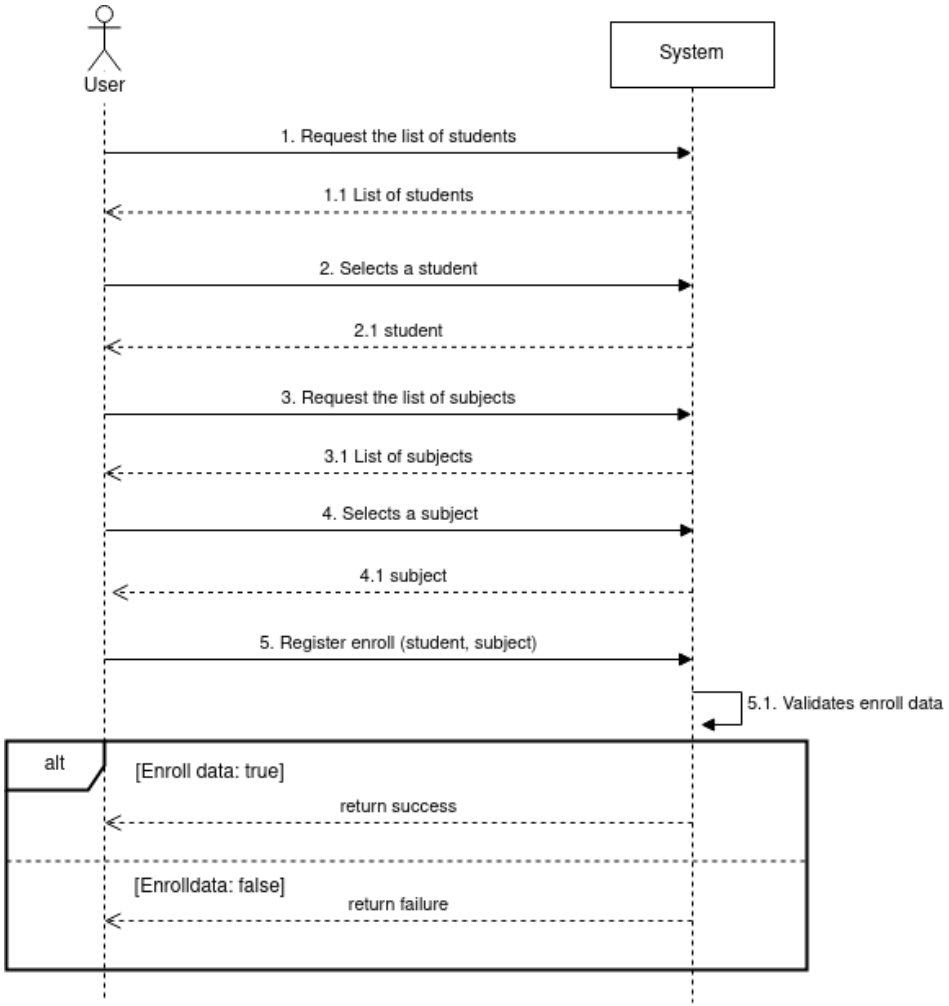
Functional Requirements (IV)

- Specification: UC 10 - Create Enroll

Description	Associate one student to one subject
Pre-condition	The student and the subject must exist in the system
Post-condition	One more Enroll.
Basic path	1. The User requests the list of students. 2. The User selects one student. 3. The User requests the list of subjects. 4. The User selects one subject. 5. The User register the enroll. 6. The System validates the enroll data. 7. The System stores the enroll. 8. The System returns success.
Alternative path	• Invalid data • The System returns failure.

Functional Requirements (V)

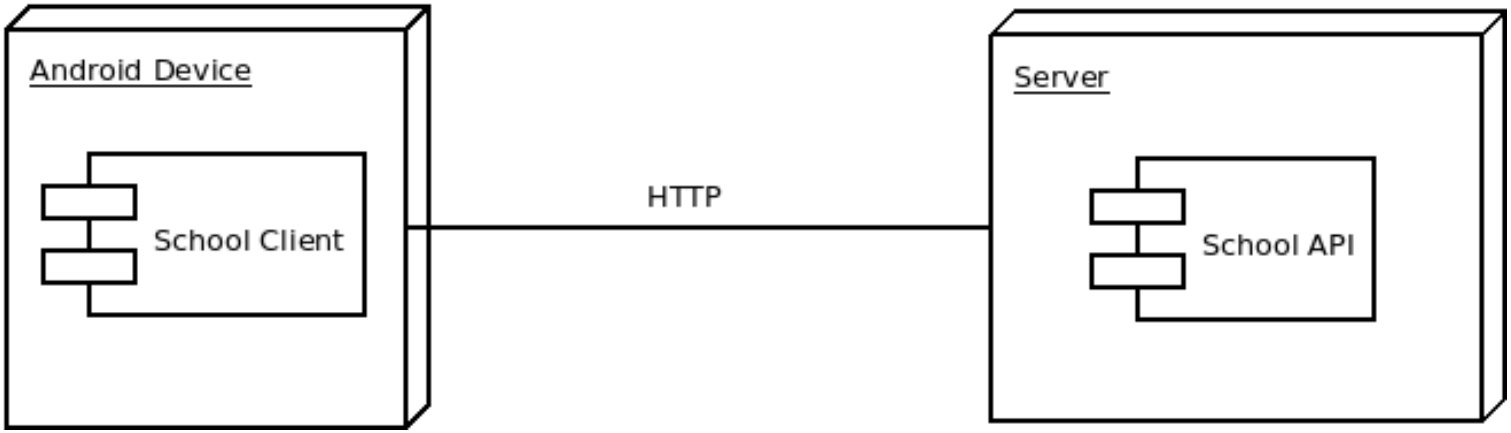
- SSD: UC 10 - Create Enroll



Design

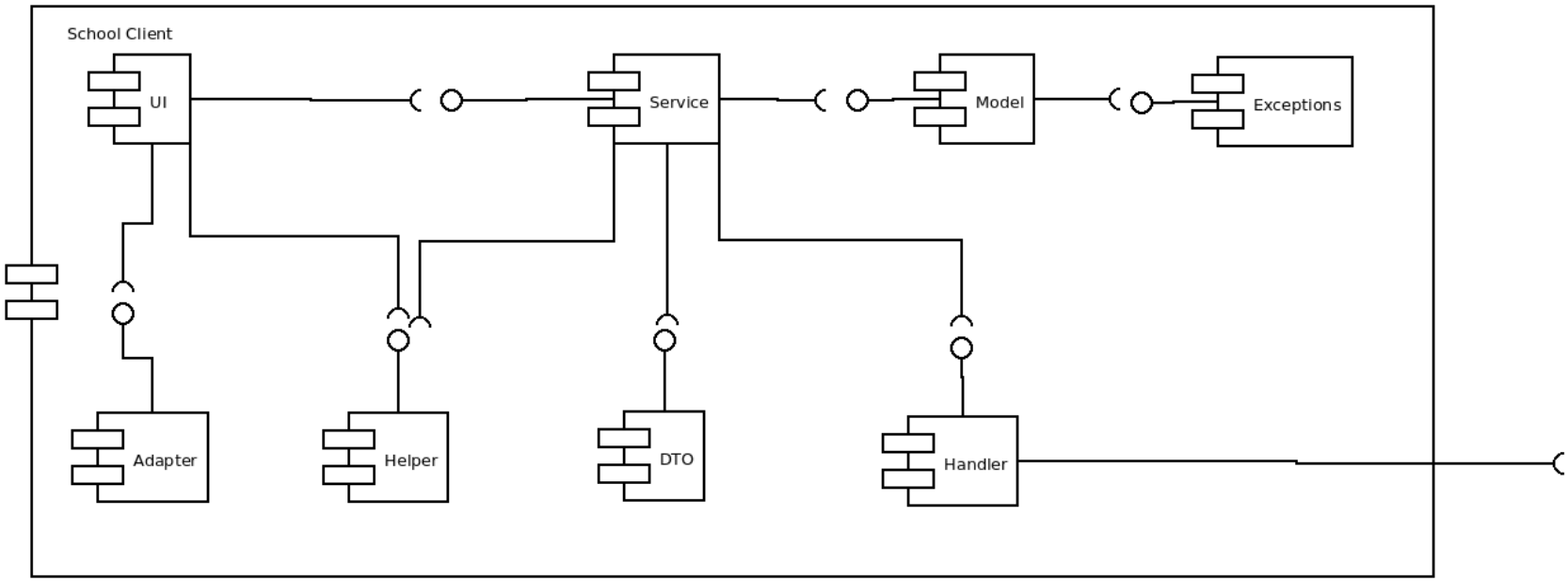
Physical Architecture

- Usage of deployment diagram



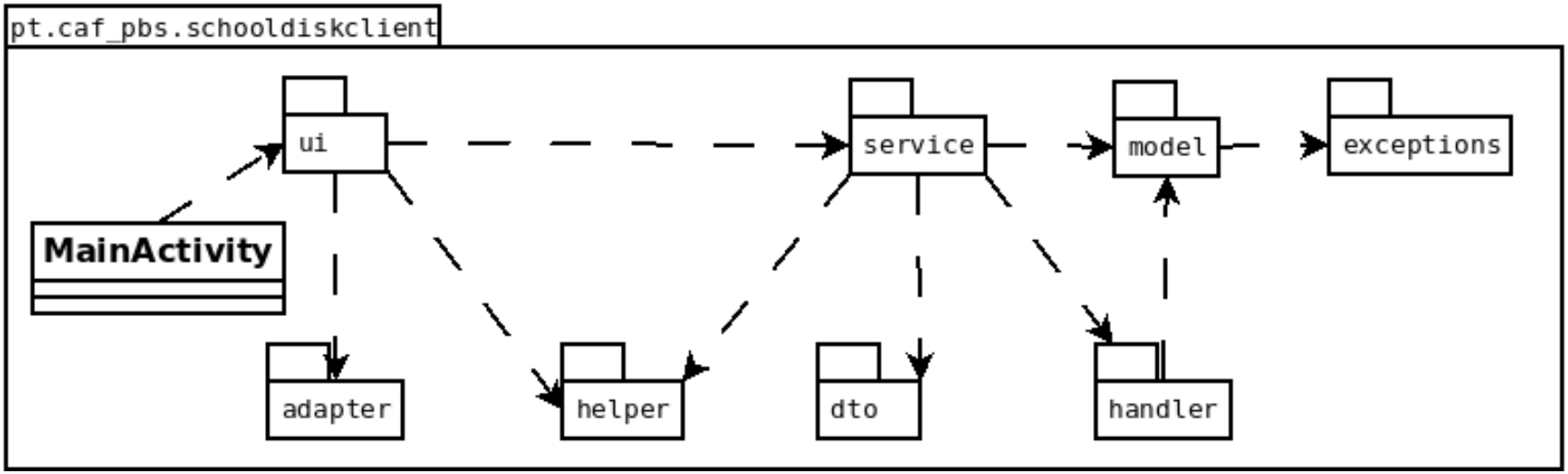
Logical Architecture

- Usage of component diagram



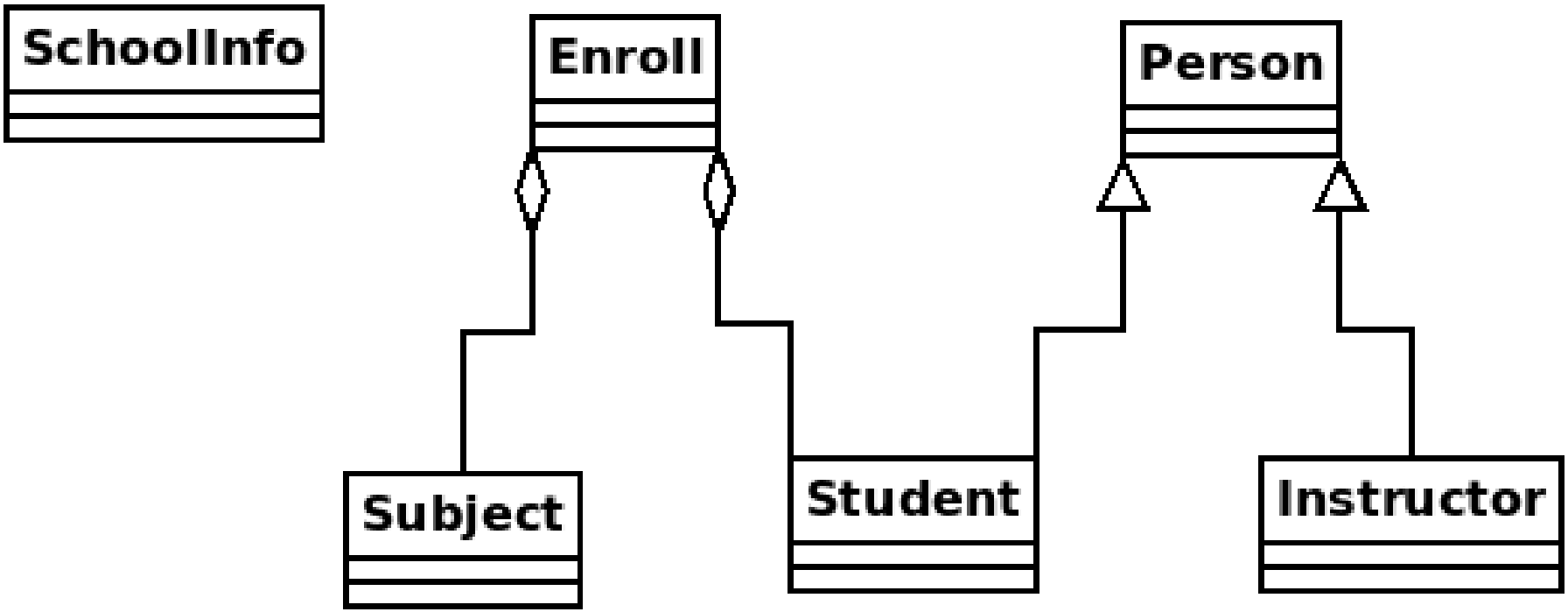
Code Organization

- Usage of package diagram



`model` Class Diagram

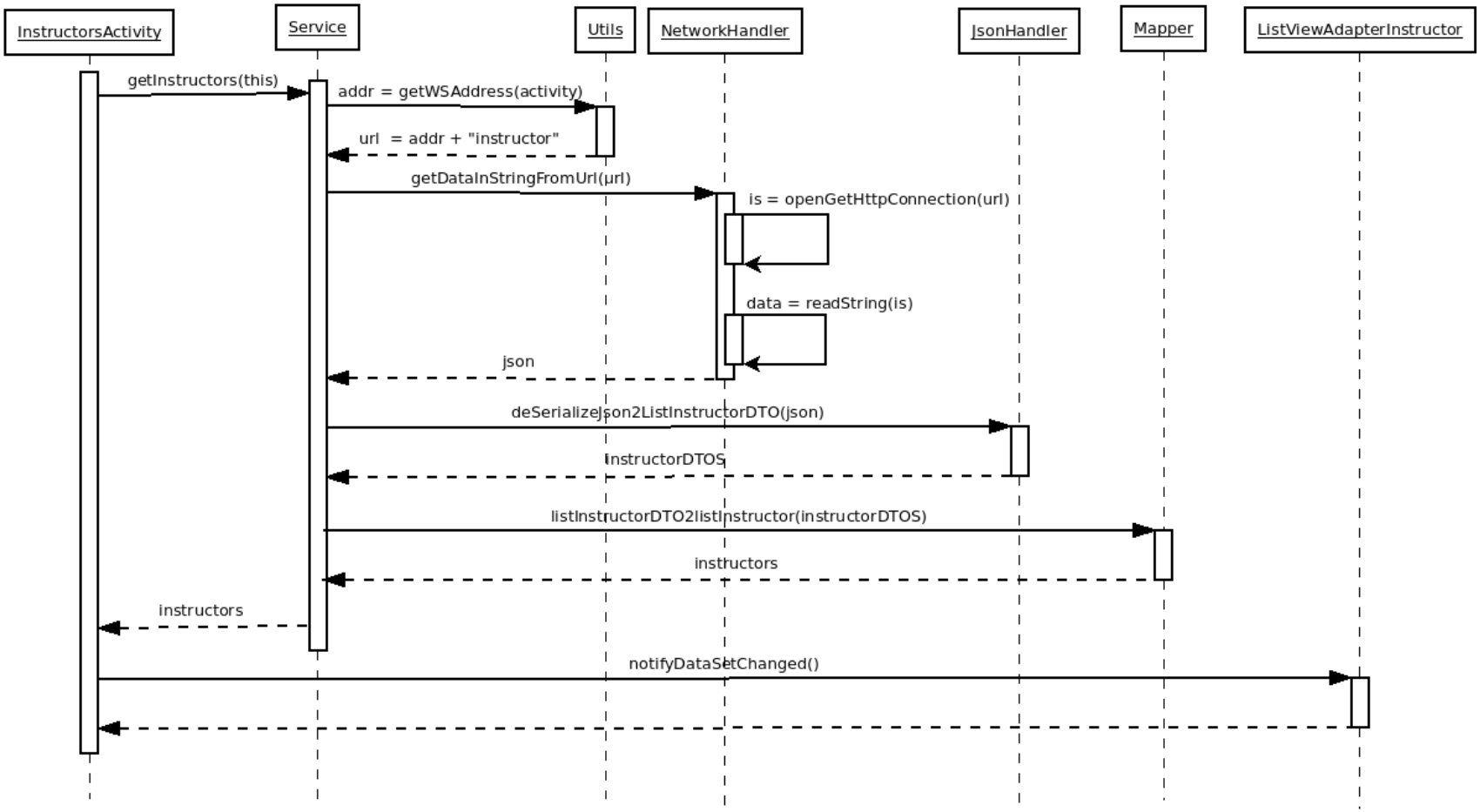
- Usage of class diagram



- The same for the remaining packages

UC 9 - Get All Instructors

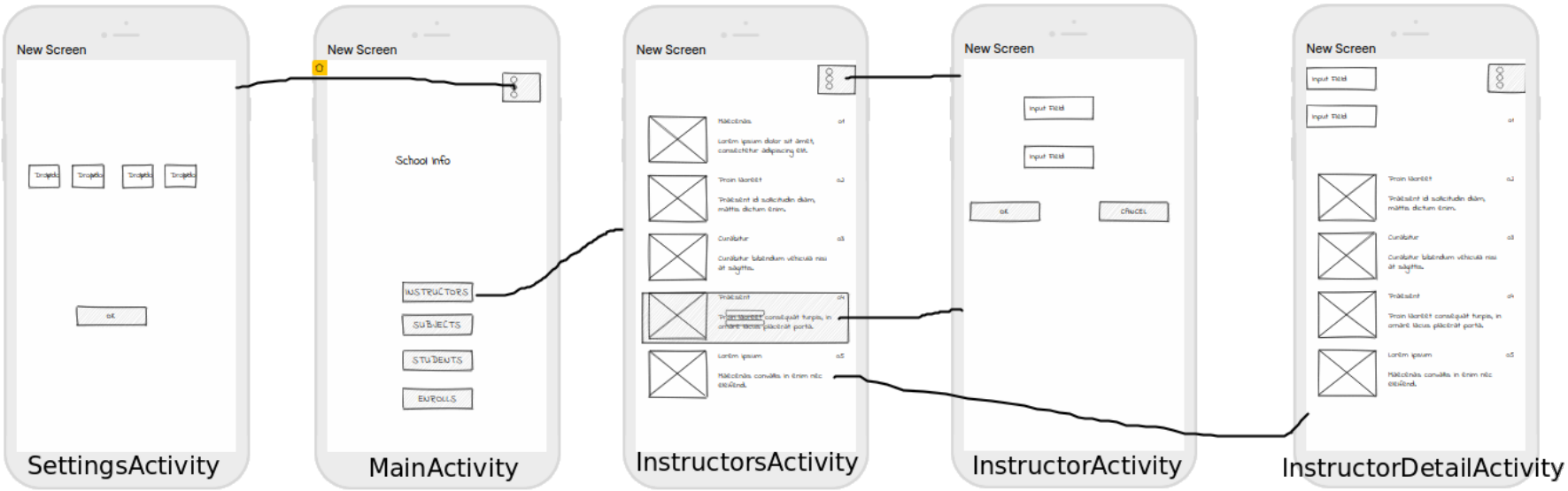
- Usage of sequence diagram



- The same for the remaining UCs

User Interface

- Mock ups



- The same for the remaining screens

Implementation

Overview

- Naming Convention: Camel Case.
- Code snippets cannot be presented as an image. It has to be a listing of text and each line has to be numbered.
  - Use the line number to help the reader
- Explain the implementation of the most important methods or functions (typically, conceptually different):
  - `openGetHttpConnection`
  - `serializeInstructorDTO2Json`
  - `deSerializeJson2ListInstructorDTO`
  - ...

List<InstructorDTO> deSerializeJson2ListInstructorDTO(String resp)

- `deSerializeJson2ListInstructorDTO` method receives a JSON formatted string and return a list of `InstructorDTO` objects (line 1)
- It is supposed that JSON formatted string holds an array.
- ...

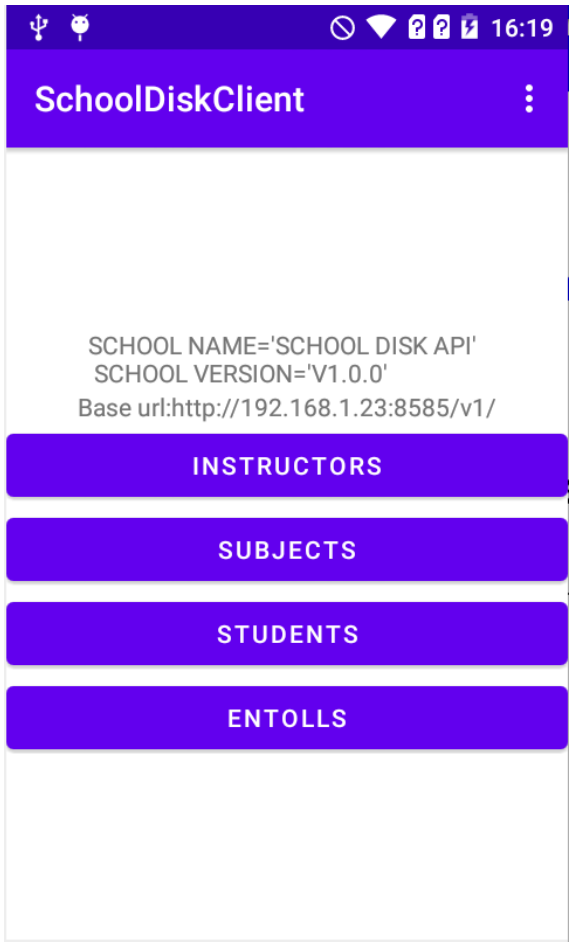
```
1  public static List<InstructorDTO> deSerializeJson2ListInstructorDTO(String resp)
       throws JSONException {
2   JSONArray jsonResponse = new JSONArray(resp);
3   List<InstructorDTO> list = new ArrayList<>();
4   for(int i = 0; i<jsonResponse.length();i++){
5     JSONObject jsonChildNode = jsonResponse.getJSONObject(i);
6     String name = jsonChildNode.optString("name");
7     String initials = jsonChildNode.optString("initials");
8     list.add(new InstructorDTO(initials, name));
9   }
10  return list;
11 }
```

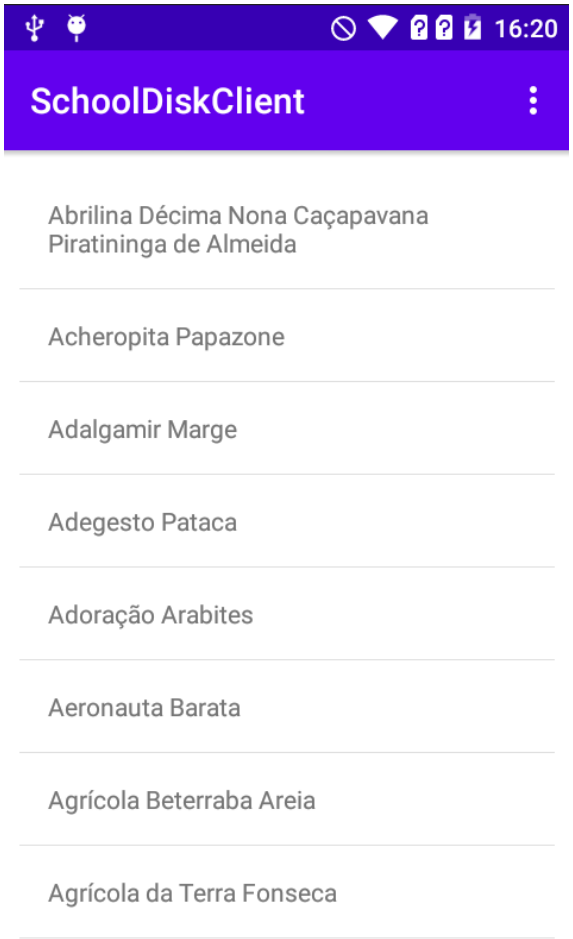
Tests

Type of tests

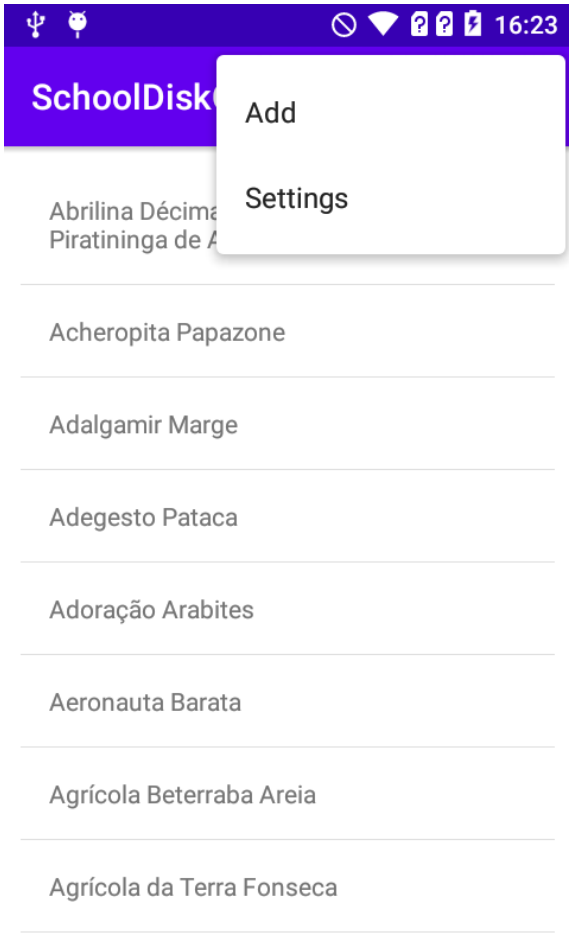
- Unit tests
- Functional tests (end to end tests)
- All tests should be described

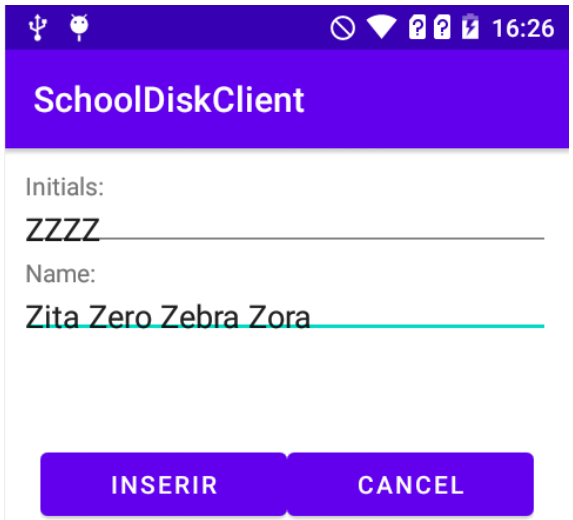
UC 9 - Get All Instructors

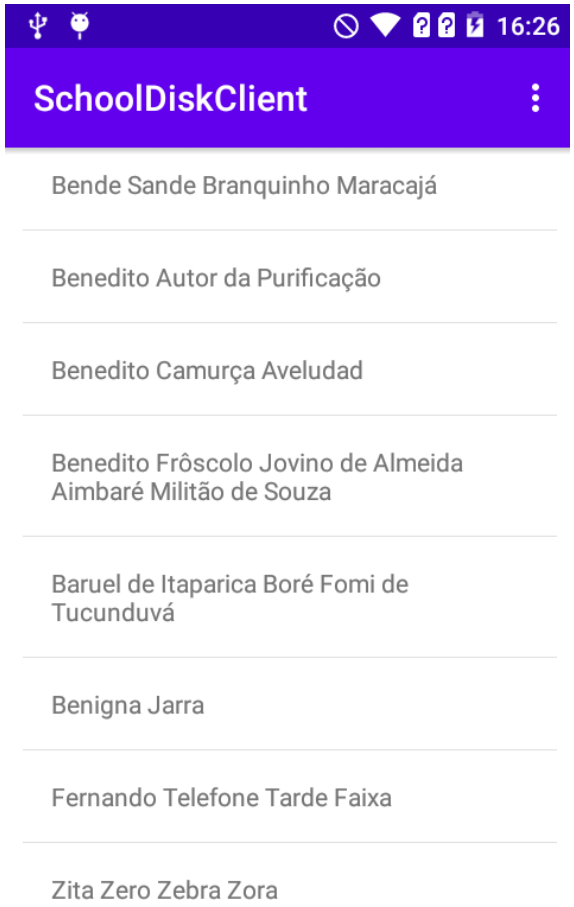




UC 8 - Create an Instructor







Conclusion

Conclusion

- Which is working properly and not
- Difficulties