POLITECNICO DI TORINO

Master's Degree in Computer Engineering

PoliTO Students App

User centered development of an open source mobile application

Supervisors

Prof. Fulvio CORNO

Prof. Luigi DE RUSSIS

Candidates

Umberto PEPATO

Luca PEZZOLLA

December 2022

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Context

Politecnico di Torino is one of the few Italian universities having an internal Information Technology department. As of now, most of the student and administrative services are carried out through bespoke tools, built internally over many years using diverse technologies, mainly hosted on managed infrastructure.

This includes all teaching-related activities: courses organization and management (including study material, recorded video lectures and virtual classrooms), interaction with students, computer-based and remote exams as well as secretarial services: enrollments, tuition fee payments, university career organization and more.

It is evident how in such a context the student-facing mobile application and web portal are essential tools for carrying out everyday academic and organizational activities.

While having an in-house development team is surely beneficial for building customized solutions that are tailored to the needs of the university, it can also present challenges in terms of workload. As a result, the efforts are coordinated towards delivering new features in a timely manner and not always dedicated to thorough planning, design and testing.

This project stems from a technological need related to **PolitoAPP**, the previous mobile application of the university. The app was developed starting from 2016 using Apache Cordova: a framework that allows to develop cross-platform mobile applications with a unified interface based on web technologies.

Unfortunately, Cordova has been falling in popularity in recent years, with support and resources being decreasingly available and a growing complexity when developing functionalities that need to interact with the operating system of the device.

This has led to the need of re-implementing the mobile application using state-of-the-art technologies, and doing so while creating a collaborative environment in which the internal team, students and external consultants can effectively cooperate to make the project evolve over time.

1.2 Goal of the thesis

The goal of this thesis is to **design and develop the new mobile application for students** as an open source project, and to do so by applying a user centered development approach.

You may have some questions: let's solve them right away.

1.2.1 What is User Centered Development?

It is a software development approach in which end users are involved in all phases of the development process. By making users part of the process, we make sure the product we are building actually satisfies their needs.

While this may seem trivial, it is actually one of the biggest challenges of modern software development, as beautifully stated by Donald Norman in The Invisible Computer [1].

> The traditional sequence of product design is technology-driven. Marketing provides a list of essential features: the engineers state what neat new technical tricks and tools they are ready to deploy. (...) Then after all is finished and the product ready to ship, call in the technical writers to explain it to the customers.
>
> Guess what: this process doesn't work. (...) It is difficult for a company to make the transition to a consumer-driven marketplace. It requires an entirely new approach to products, it requires starting with an understanding of customers and their needs, their real needs, not the feature sets so loved by marketing. It means using social scientists to collect, analyze and work with the data, moreover social scientists on an even footing with engineers and technologists. It means structuring the whole product process differently than before. It may mean reorganizing the company.

The IT department of Politecnico di Torino made a willful choice to embark in this journey with us while working on this project: **we would like to thank them right away for the effort they put into making this leap**.

You'll learn more on the actual process and the challenges we faced during the following chapters.

1.2.2   Why should it be for students only?

The previous mobile app was used by students, teachers and other university staff. Those users have significantly different needs, and while some features may be used by all of them the tasks they are performing are not.

Designing an application for students only allows to focus on the tasks they perform, and drastically improve their user experience while interacting with the app.

1.2.3   What languages and frameworks will be used?

We chose to write the new app using **React Native**, a popular framework for cross-platform mobile applications. React Native has the advantage of giving access to most native features of the underlying operating system through abstractions.

While working on a single codebase, it is however possible to introduce customizations in the UI to obtain a graphical appearance that is close to the one of a native application.

Both React and React Native are taught during the Master's Degree in Computer Engineering: this means that the codebase will be easily understandable by students.

1.2.4   What are the benefits of developing it as an open source project?

Releasing the code of the app as open source allows students to have direct visibility on the inner workings of the application.

This can both be a learning opportunity for students interested in mobile development, and a chance to get their hands dirty with a real-world project. They can open issues or feature proposals directly on the GitHub repository, and even contribute directly to the project by opening pull requests.

In more general terms, the use of open source software in the public sector is highly beneficial due to its transparency and adaptability. By utilizing open source options, government agencies and other public institutions can ensure that their technology is accessible and can be customized to meet the needs of their constituents. A

crucial aspect is that often it has a lower cost than proprietary alternatives, making it more financially viable for public institutions in times of big cuts in public spending.

Moreover, the app may serve as starting point for other universities planning on building their own mobile application. This is also reflected in the Italian legal system, where articles 68 and 69 of the Digital Administration Code [2] mandate that all Public Administrations make commissioned or modified software available in open source for reuse.

1.3 Document structure

Each of the following chapters details a phase of the development process:

- **Needfinding** - getting a first understanding of user needs
- **Prototyping** - testing different user experience alternatives on users
- **Design** - formalizing graphical appearance of components and data layer structure
- **Implementation** - writing code for the app
- **User testing** - evaluating the end results

As you might have noticed, the actual development is just a small part of the process: performing the previous steps allowed us to start writing code with a clear idea of the product we are building and a lot of pre-defined choices that speed up the implementation stage.

Chapter 2

Needfinding

2.1 Preliminary analysis

The first step in the project was understanding the needs of students when they interact with the IT infrastructure of the university, and their experience in doing so.

The features offered by the previous mobile application are of course a good starting point, but we are also interested in understanding if something is missing, or if some existing feature is hard to discover or use.

We divided the students into several clusters so as to explore specific needs of each of them:

- **Bachelor students** have a limited knowledge of the university jargon

- **Master students** have extensive experience in using the app and the other student services, they have two student IDs

- **Master students with a previous career in other universities** can provide extra insights based on their previous experience

- **Students with disabilities** have needs specific to their condition, may interact with the app through assistive technologies

- **International students** have a limited knowledge of both the university jargon and the national practices

- **PhD students** have a different organization of lecture and transcript compared to other students, also have a teacher profile to act on the courses they teach

2.2 Methodology

We defined a script for a qualitative interview to be carried out with students, with some specific questions for each of the above clusters.

The script is based on open-ended, non leading questions, to allow students to express their needs. Based on their response, we might continue with follow-up questions to further clarify what they just said.

2.2.1 Script content

Interaction with the previous app

- Think to a normal day **during the semester**: how do you interact with the PoliTO app?

- What about during an **exam session**?

- Are there any features you'd like to reach more easily? Why?

- What actions you decide not to perform on the app? Why?

Master/PhD students only

- Have you ever **switched student ID** to a previous career while using the app? If yes, why?

Students with a previous career only

- Are there any features of your previous university's app that you think PoliTO app should have?

Students with disabilities only

- Which assistive technologies do you use while interacting with the current app?

- Which characteristics would make the app more usable for you?

International students only

- Can you think of any specific content for international students the app should contain?

- Which characteristics would make the app more usable for you other than the language?

Potential improvements for the future app

6

- Are there any actions you wanted to perform on the app but you've been forced to do on the web portal or in person?

- Is there any non-teaching information you'd like to find in the app?

- Are there any academic calendar events you'd like to receive a notification for?

- Is there something we didn't ask that you think might help us in working on this project?

2.3 Results

We carried out a total of 22 interviews with students from all clusters and studying in different degrees (as shown in figures 2.1, 2.2).

While this may seem a small number, performing those interviews allowed us to get a good perspective on the spectrum of users that interact with the app, the context in which they use it and the outcomes they expect from it.
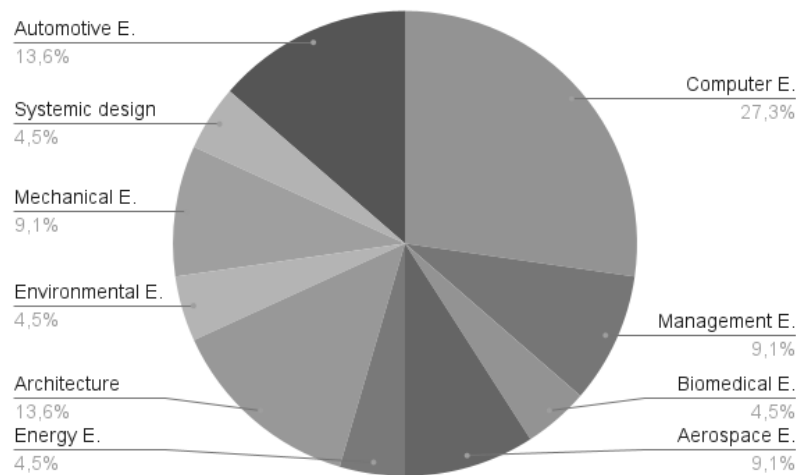
Figure 2.1: Interviewed students by degree type

We merged the feedback gathered during interviews with app store reviews and feature requests received over time by the IT department to create a comprehensive list of user experience improvements and new features that should be considered while designing the new application.

Each suggestion has been phrased as a user story to emphasize the advantages it produces and the targets that benefits from it (as a student if not specified).
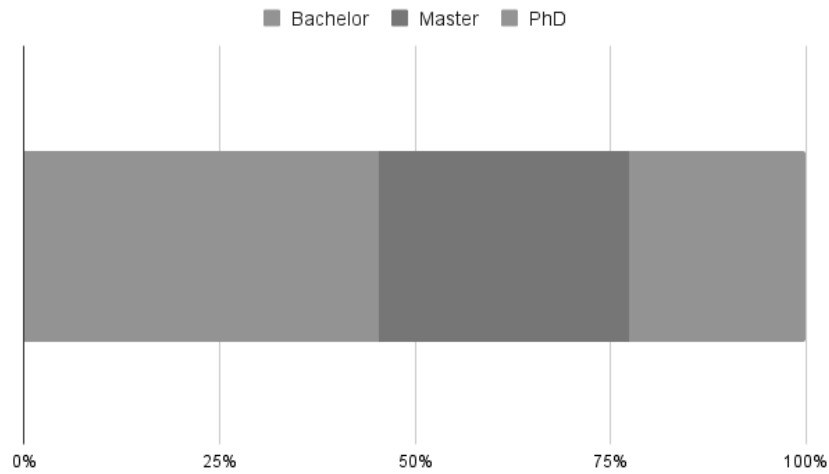
Figure 2.2: Interviewed students by study level

2.3.1 Courses

Feature proposals

As a student:

- I would like to create or update my Annual Personal Study Plan

- I would like to hide courses I already attended from the courses list

- I would like to upload course assignments even if they are not images

- I would like to access course files uploaded on Dropbox through the app

- I would like to see the exam statistics for previous editions of the course

As an architecture student, I would like to choose the teacher for the atelier.

UI improvements

As a student:

- I would like to play course videolectures inside the app, without being redirected to the browser, with playback speed control

- I would like to download course files without being redirected to the browser

2.3.2 Transcript

Feature proposals

As a PhD student, I would like to see the evaluation of grades and their weight in terms of hours.

2.3.3 Class calendar

UI improvements

As a student:

- I would like to hide courses I already attended from the calendar

- I would like to see exam appeals into the calendar

Feature proposals

As a student:

- I would like to add to my calendar a course that is not into my Study Plan

- I would like to add personal notes to calendar events

- I would like to add all lectures to the calendar app of my phone via iCal

- I would like to have a class calendar widget for the home screen of my phone

2.3.4 Bookings

Feature proposals

As a student, I would like to book an in person activity with an automatically assigned seat.

2.3.5 Email

Feature proposals

As a student, I would like to be guided in configuring the university email on my device.

2.3.6   Exams

Feature proposals

As a student, I would like to get get directions to reach the room in which the exam takes place from the exam screen.

2.3.7   Job offers

Feature proposals

As a student, I would like to filter the available offers.

2.3.8   People

Feature proposals

As a student, I would like to find the office location of a teacher together with its contact info.

2.3.9   Library

Feature proposals

As a student:

- I would like to look for books in a specific library

- I would like to know if a book is available

- I would like to know how to reach the library that has a given book

- I would like to book the pick-up of a book

- I would like to manage ongoing book loans (list, renew)

2.3.10   Maps

Feature proposals

As a student:

- I would like to search for a room by name

- I would like to know what services are available into a room (electric plugs, projectors...)

- I would like to know the suggested entrance to reach a room

- I would like to see the layout of a floor into the map

As a student with disabilities:

- I would like to search for equipped bathrooms
- I would like to know if a room is accessible

2.3.11   Free rooms

UI improvements

As a student:

- I would like to know until what time the room is free
- I would like to understand how to reach a free room
- I would like to search for them in an easier way, like in the web interface

2.3.12   New features

The following features proposals are not related to any feature that's already present into the previous apps.

As a student:

- I would like to check my taxes situation (contribution level, amounts, dates and payment status)
- I would like to have a faq/guides section unrelated from the support tickets
- I would like to search for internship offers
- I would like to request the start of an internship
- I would like to find the description of the activities of student associations and teams
- I would like to have a complete list of events organized by the university
- I would like to be notified new for events of specific categories I select
- I would like to have a digital representation of the smartcard for identification during exams or at other institutions
- I would like to be able to consult the academic calendar

Chapter 3

Prototyping

3.1 Feature analysis of the previous app

The first step in prototyping was fully analyzing the navigation patterns and features offered by the previous application.

As visible in Figure 3.1, the home screen presents a grid view containing direct links to all the features.

Through the app, a student can:

- interact with courses (notices, files, assignments, lectures)
- see his transcript
- look at his class calendar
- book activities (secretary appointments, lab usage)
- access email (opens in web browser)
- handle support tickets
- book exams
- read provisional exam results
- partake to in-classroom surveys
- navigate job offerings
- read university news
- explore the catalog of degree and courses
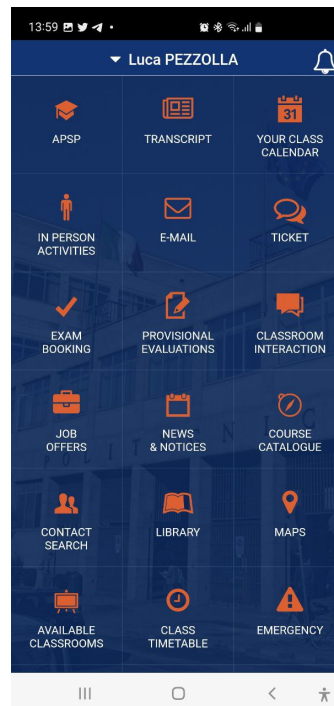
Figure 3.1: PolitoAPP - Main navigation

- search for contacts of staff (teacher, personnel)
- browse the library catalog
- find rooms in maps
- search for empty classrooms at a given time
- look at the timetable for any course (even not in its study plan)

That is a lot of features! The main issue in terms of usability is that they are represented flatly, without any grouping to help users identify them. Most of the feature are not cross linked: there is no reference to pass from courses to exams, or from lectures to people.

3.2 Heuristic evaluation of the previous app

Before creating prototypes of the new app we focused on identifying challenging aspects in the user interface of the previous one by applying the heuristic evaluation approach proposed by Nielsen.[3]

This means performing common tasks while checking that the **10 Usability**

Heuristics for User Interface Design are met.[4] As you might expect, some of the usability issues we identified were also described by students during the interview phase.

The full evaluation is quite lengthy, we'll just describe one of the tasks analyzed by way of example.

3.2.1 Going to an exam

As a student, **I want to check my schedule and reach the classroom** in which my exam takes place.

The first step to do so is getting into Exam booking: once there, I see the list of the available exams, and by clicking on the one I am interested in I discover in which room it will take place (Figure 3.2).
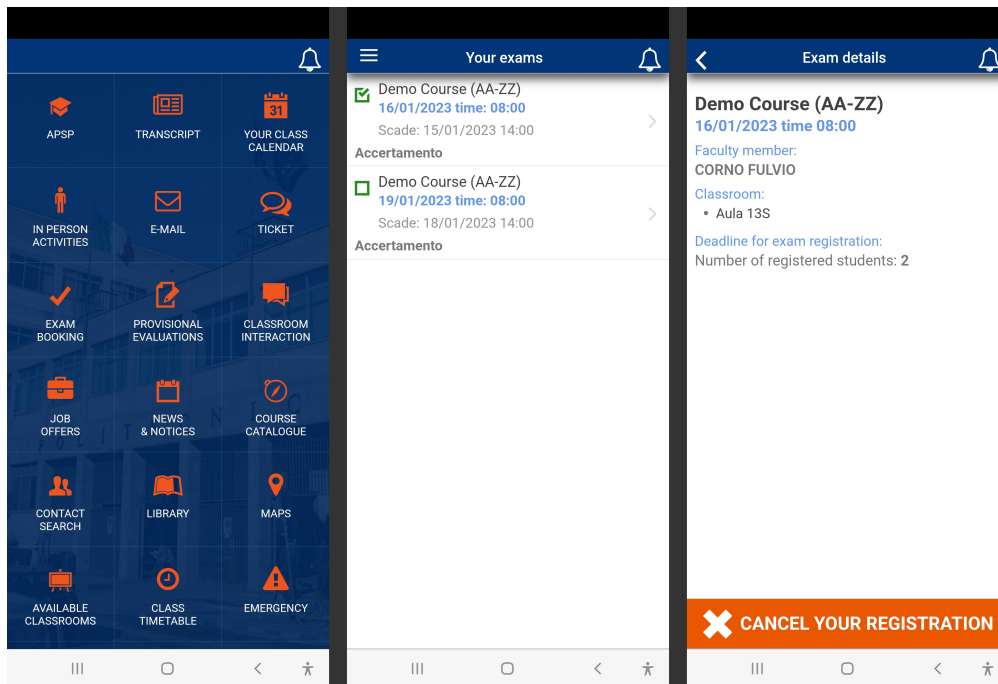
Figure 3.2: PolitoAPP - Finding an exam

Unfortunately, there is no way to understand where the room is located: to do so, I have to remember the room name, go back to the homepage and into Map (Figure 3.3).

This is obviously a painful user experience, and in particular it breaks the following heuristics:

14

- **#6 - Recognition rather than recall**, you need to remember the room name across multiple screens to look for it in a (long) list

- **#7 - Flexibility and efficiency of use**, that interaction could really use a shortcut

Figure 3.3: PolitoAPP - Finding a room

3.3 Competitor analysis

Once we concluded the evaluation of the previous app, we moved to analyze the set of **features and user experience patterns used by student apps of other universities** in Italy and abroad.

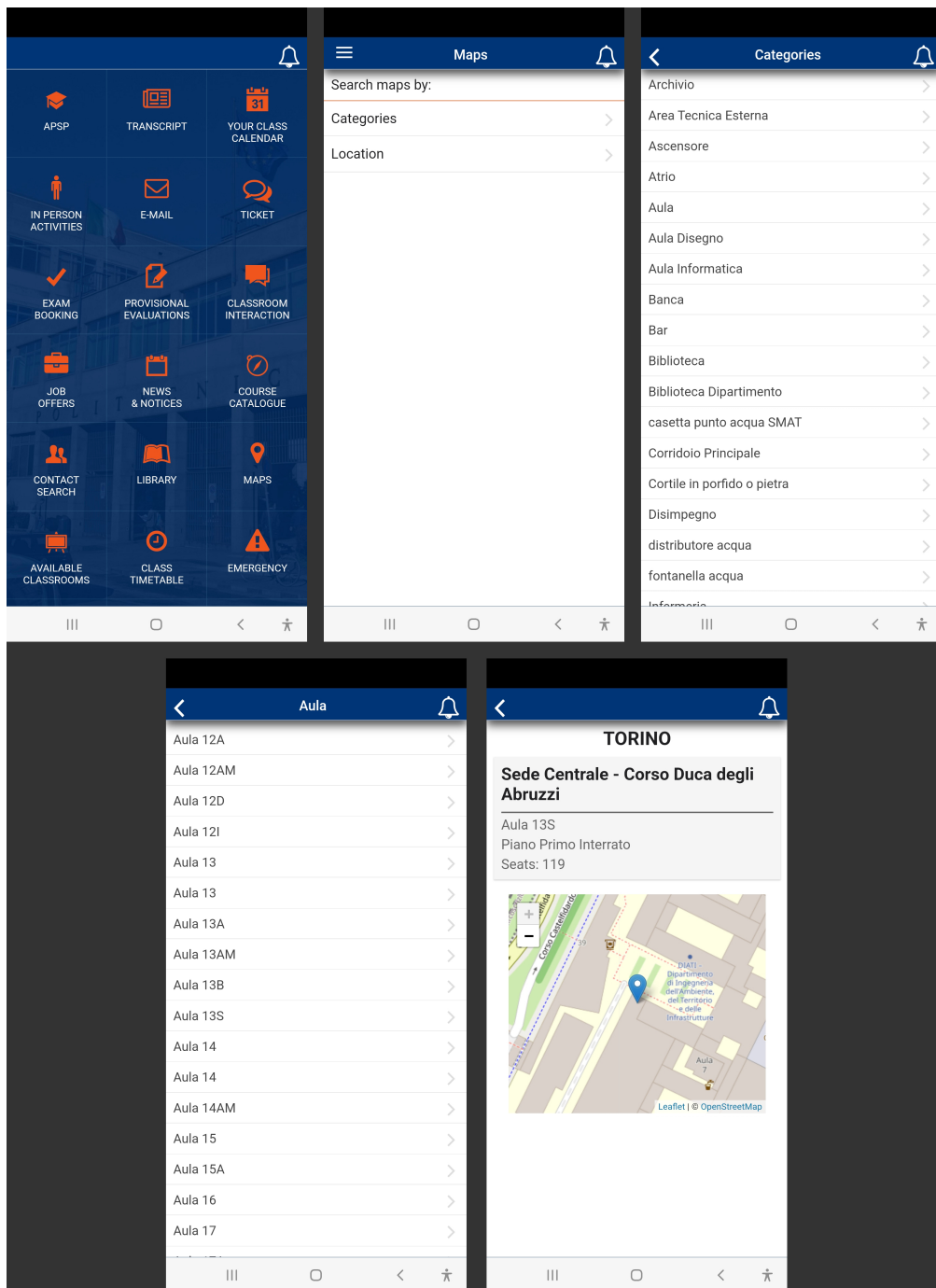Where possible, we tested the guest features first hand, while the student features were documented by interviewing a student from each university. In some cases, the student's career status did not allow us to navigate all the features of the app.

We will briefly describe the highlights of each app in terms of main navigation, plus any interesting features/UX patterns.

3.3.1 Università degli Studi di Siena

The app was developed in 2017 by Cineca, an inter-university consortium acting as a technology provider for many universities in Italy.

The home page consists of a customizable bookmark list while the complete navigation is contained in the side drawer (Figure 3.4).
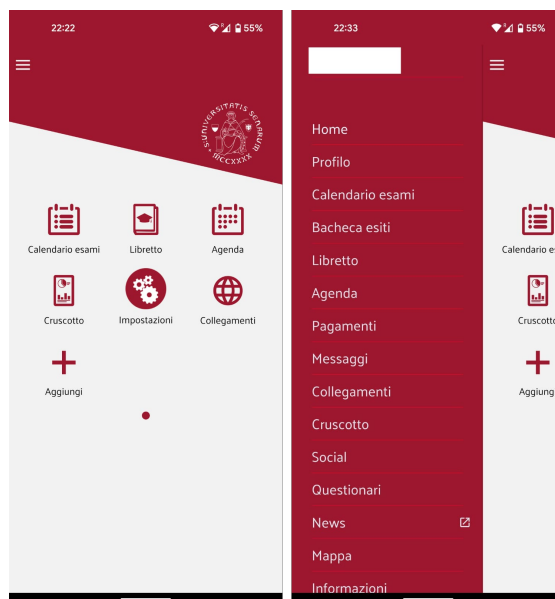

Figure 3.4: Unisi - Main navigation

The most interesting feature was the agenda, a filterable view over the activities the student should perform (Figure 3.5).
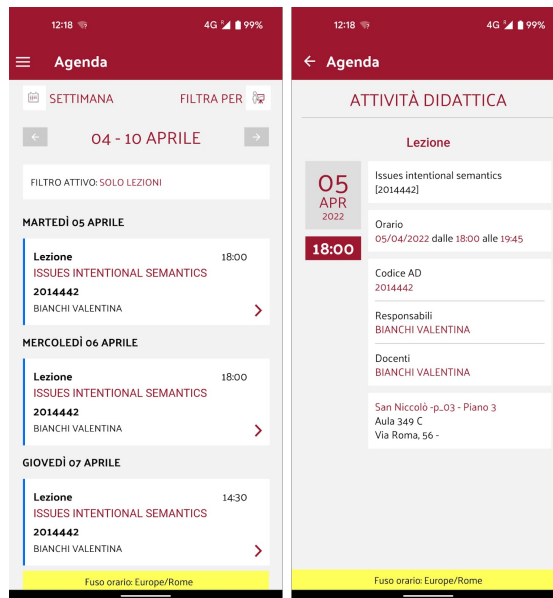
Figure 3.5: Unisi - Agenda

3.3.2   Università degli Studi di Napoli "L'Orientale"

This app was developed in 2021 by Cineca: being an external provider, new versions are not timely supplied to all universities. Therefore we can also evaluate the evolution over time of their UI compared to the previous one.

The home page has a block structure, with the addition of a bottom bar to reach the most used features. The elements present in the bottom bar and on the homepage can be configured by logged in users using the gear in the upper left corner(Figure 3.6). Comparatively speaking, the developers felt the need to add the bottom navigation for quick access to features when not in the homepage. The accessibility of features placed from the 10th position onwards is reduced, as getting that the blocks in the homepage are wrapped into a slider is not immediate.

In this version, the agenda is available with a monthly view only: the dots are color-coded to specify event type (orange is an exam) (Figure 3.7). Unfortunately, the student we interviewed had no lectures in her career at the time, so we couldn't get more insights on the UI of the single event.
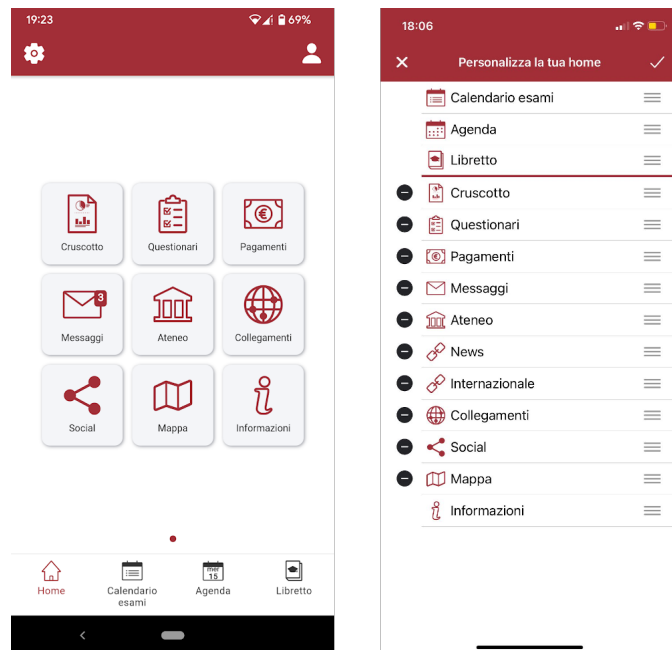
Figure 3.6: Unior - Navigation
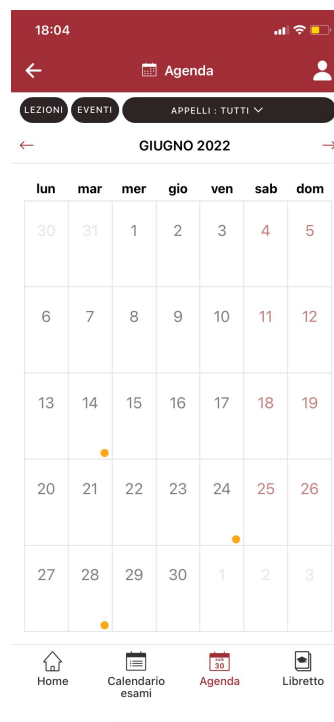
Figure 3.7: Unior - Agenda

3.3.3   Università degli Studi di Roma "La Sapienza"

This app was developed by a group of Computer Science students starting from 2018.

They chose to include an "opinionated" bottom navigation: the features that are rapidly accessible are Transcript, Agenda, Rooms and Bookings (The fourth one is missing in the screenshots, as the testing student career was not active at the time). The homepage is the Transcript, while all other features are hosted into the side drawer. (Figure 3.8)
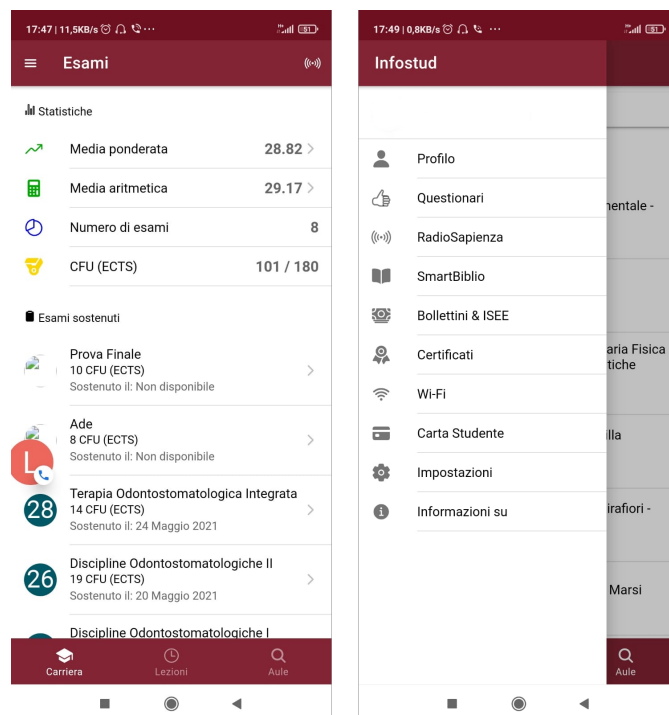
Figure 3.8: Uniroma1 - Navigation

The agenda is available with a weekly view only (Figure 3.9). The action bar contains two actions, which allow to:
- create a custom event on the calendar based on the user's needs
- add a course not present in the teaching schedule to the calendar
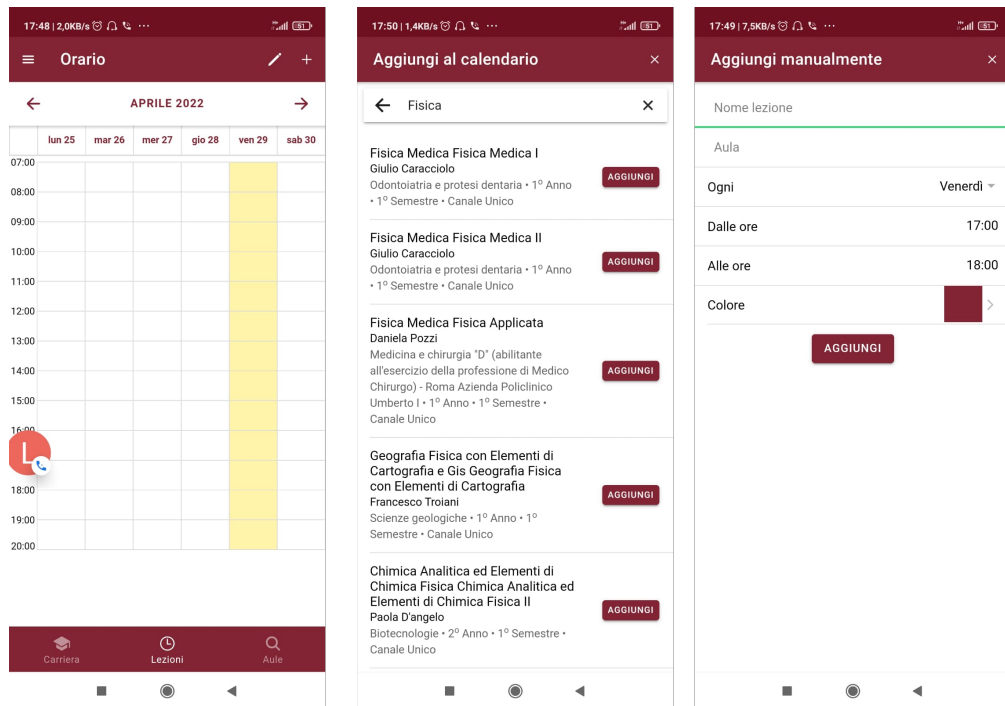
Figure 3.9: Uniroma1 - Agenda

3.3.4   EPFL - École Polytechnique Fédérale de Lausanne

The app is developed by PocketCampus, which markets this solution for universities.

The homepage hosts a list of features accessible through the app. It is possible to reorder the list items using a long press, or hide them using the pencil in the action bar (Figure 3.10).

We could not evaluate the agenda, as we interviewed a student with no active courses.

An interesting and well integrated feature is the Directory, allowing access to the contacts of teachers and other staff (Figure 3.11). It provides fast access to recently visited persons, and also allows to search all members belonging to a specific department/research group by just clicking on the corresponding item. The profiles are also linked into courses (for teachers) and other person-related features.
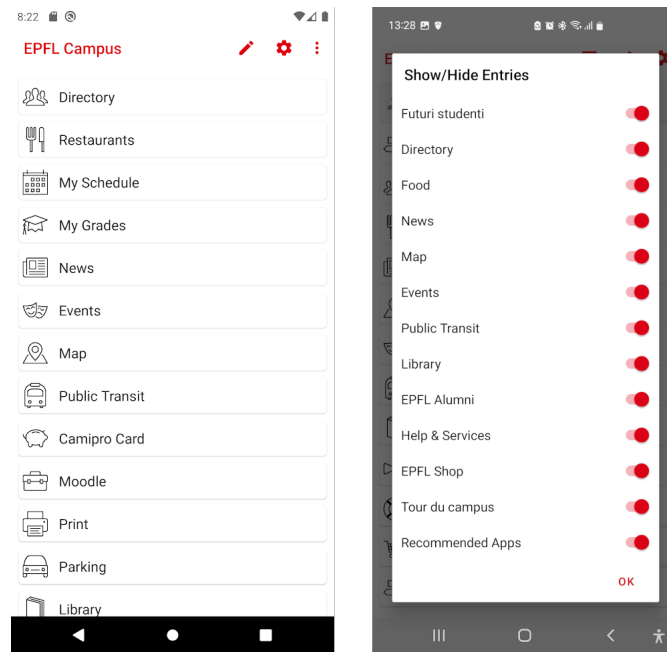
Figure 3.10: EPFL - Navigation
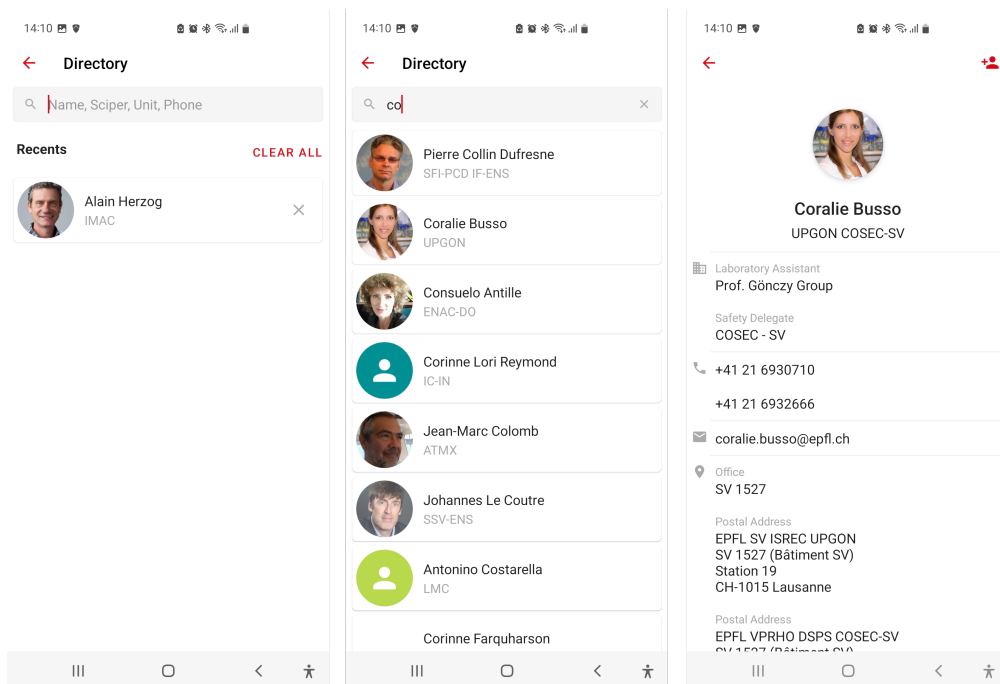
Figure 3.11: EPFL - Directory

3.3.5   Imperial College

The app was developed internally, but it has a low level of integration with the features offered by the university. The homepage as a customizable blocks structure, though the blocks are just website links opening into the integrated webview (Figure 3.12).
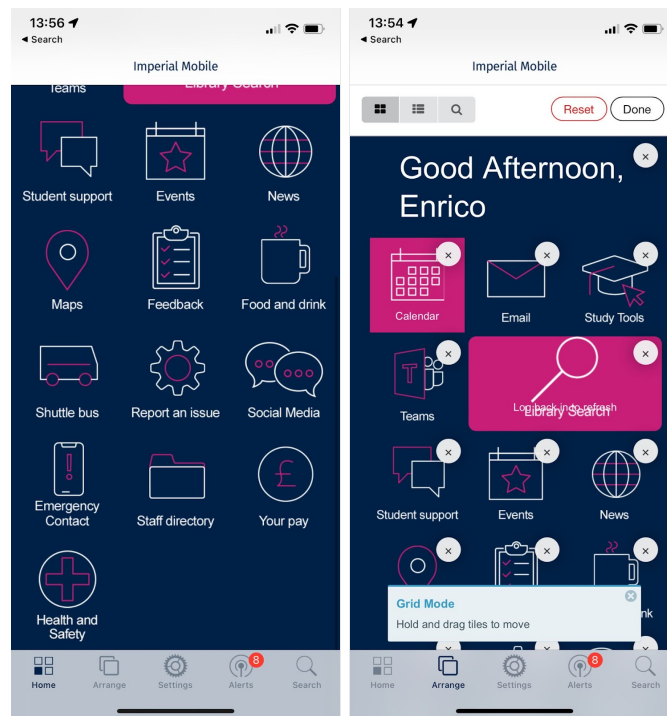
Figure 3.12: Imperial College - Navigation

The bottom navigation contains the few features offered by the application: homepage customization, app settings, notifications archive and campus buildings search.

3.4   Low-fidelity prototyping

We now have a good understanding of the different patterns and navigation strategies applied by universities: we can create low fidelity prototypes of the new app. The reason we are creating them is allow real users to perform tasks on it to evaluate: that is an usability test, more on that in the next section.

Low fidelity means we are not interested in the graphical appearance: our prototype will be a browsable interface in grayscale, just representing the hierarchy of content. This minimizes the effort in creating them, and reduces the chance of testers getting

distracted.

Based on the feedback provided by users and the analysis of competitors, we already made some choices with respect to information hierarchy that are reflected in both prototypes.

The most relevant one is the **introduction of an "Agenda" feature**, that is an extension of the pre-existing "Class calendar". While the calendar in the previous app only contained programmed lectures, the new Agenda improves usability by directly linking programmed lectures with their recording, when available. The Agenda will also include the other time-based events the student might have (exams, bookings, deadlines) in order to provide a unified view over the commitments of the user.

The other common choice was **merging "Transcript" and "Provisional evaluations"** into a single feature, since provisional evaluations are just unconfirmed transcript items.

We chose to create two different prototypes applying different main navigation approaches: both prototypes were created using Figma, a popular web-based tool for interface design.

We'll briefly present both of them before discussing how they were tested on users.

3.4.1   Prototype 1 - Per-feature screen

This version has a navigation approach that is closer to the one of the previous app: the bottom navigation hosts Agenda, Courses, Exams and Transcript, the most used features as emerged during the interviews. The rest of the features are accessible through the navigation drawer. (Figure 3.13)

The Agenda allows to easily access all the commitments of a student: each event includes hyperlinks to easily access the related features (e.g. for a lecture, video recording, room location, course files, teacher contact info, Figure 3.14).
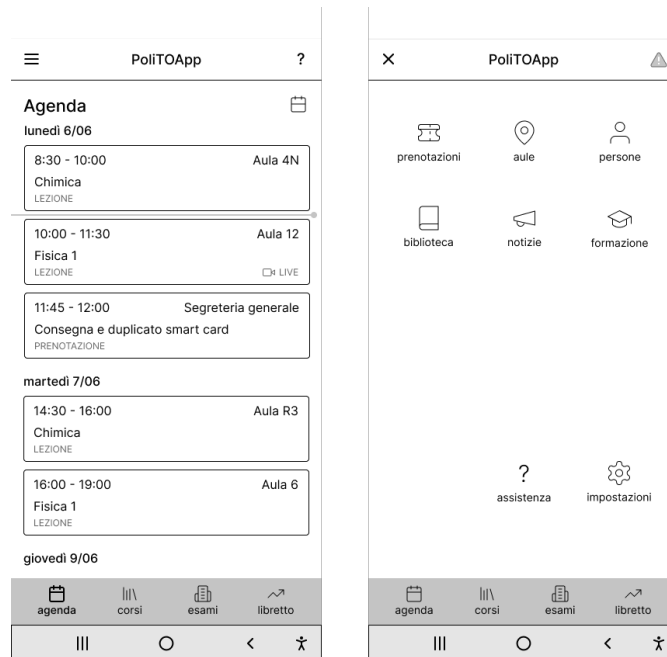
Figure 3.13: Low fidelity prototype 1 - Navigation

Figure 3.14: Low fidelity prototype 1 - Agenda/Lecture

You can navigate through the whole prototype by looking at it on Figma (`https://www.figma.com/proto/07da7DFUWpt6u3zYs7Ju2l/PolitoApp---Per-feature-screen?node-id=4%3A38`).

3.4.2 Prototype 2 - Aggregated features

The second low fidelity prototype is characterized by a hierarchical re-organization of features based on semantic grouping. On one hand, many related but - from a UX perspective - distant features were grouped together into the same navigation cluster. See for example "Transcript" and "Provisional grades", merged into the "Transcript" page. On the other, some features are represented in multiple sections along different dimensions: "Exam bookings" is displayed in the list of exam calls as well as in the "Agenda" section.

The 4 main macro-sections that emerged from this reorganization were used as the main bottom navigation elements, with user-related items in a dedicated modal. In an effort to reduce complexity and improve feature discoverability, this prototype does not include any additional top-level navigation elements such as drawer menus.
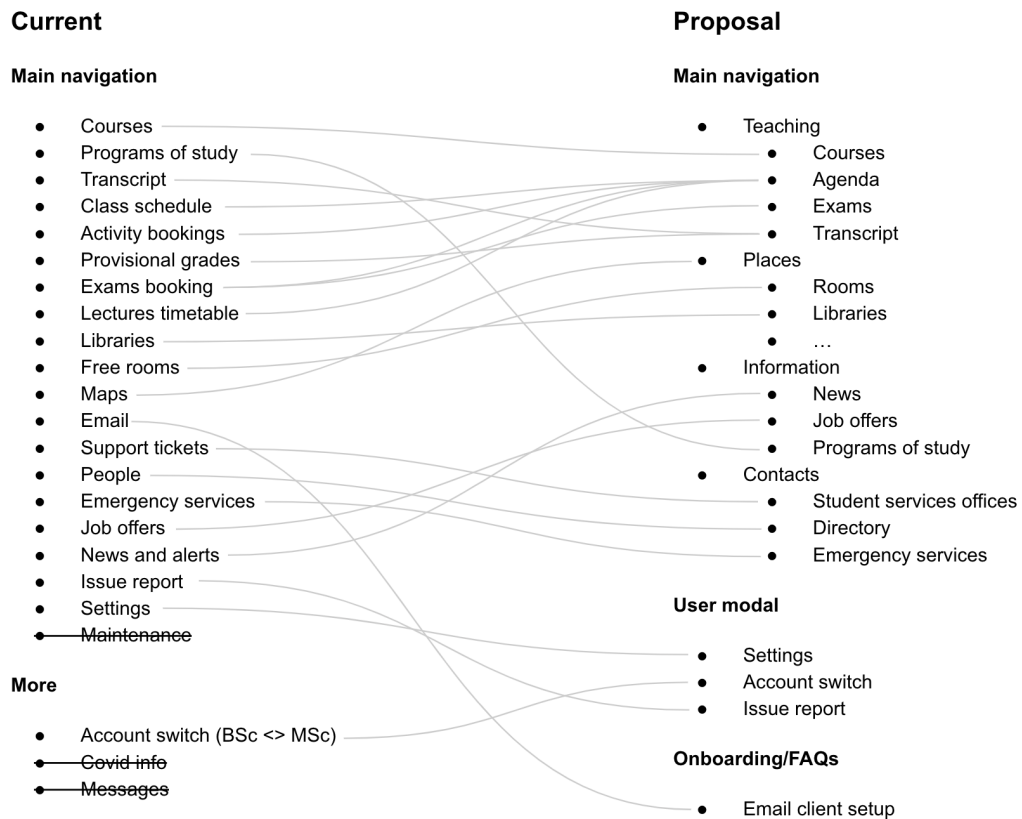
Figure 3.15: Hierarchical re-organization of features (first version)

At startup, the user lands on the teaching screen, where relevant extracts of the

subsections are shown: the main courses of the current period, a glance to the agenda, the upcoming exam calls, each with a dedicated call-to-action to navigate to the corresponding screen.

Figure 3.16: Low fidelity prototype 2 - Teaching screen

The course screen uses a tabbed interface to represent all the course contents in an organized way. All entities are deeply linked across the UI: extracts and single item representations always lead to the complete detail screen through a navigation action.

Figure 3.17: Low fidelity prototype 2 - Course screen

The maps screen is structured as a fullscreen interactive map with the relevant university campus pre-selected. Here places are navigable in space using the map and the provided filters, but also laid out in a list in the collapsable bottom sheet.
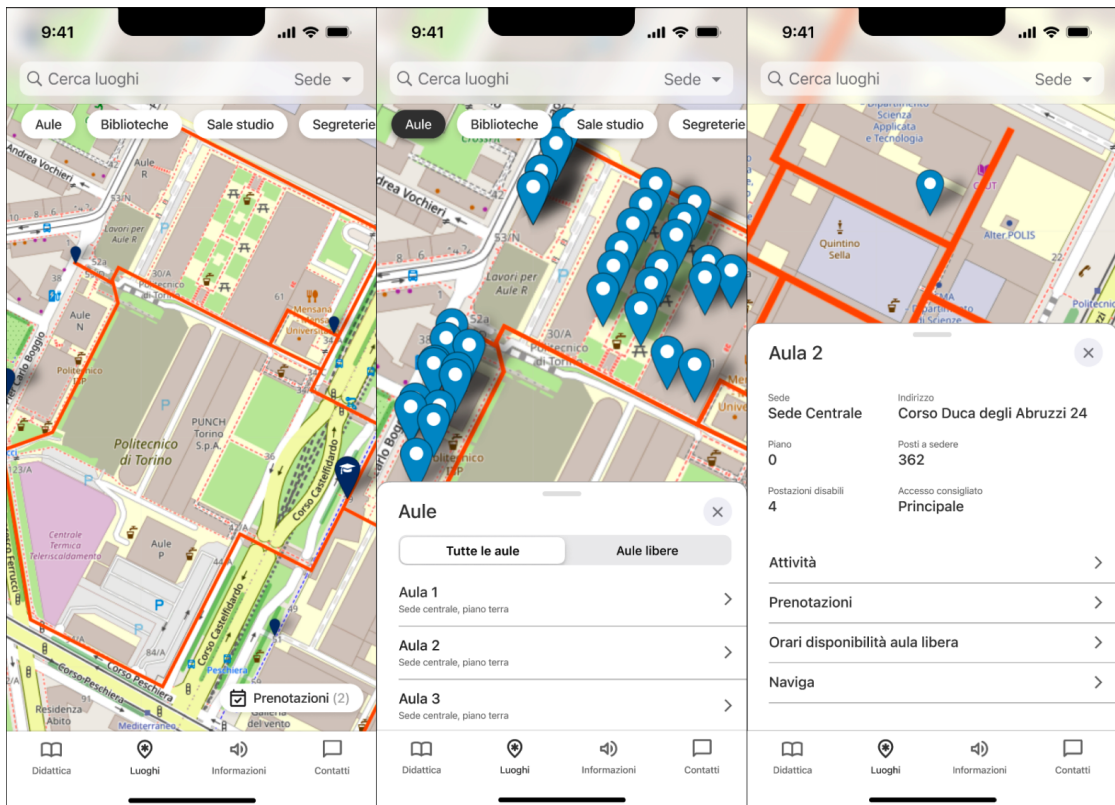
Figure 3.18: Low fidelity prototype 2 - Places screen

The complete interactive mockup is available at `https://www.figma.com/file/DvLvM0xz2Up0vgxhNt5E0l/PolitoApp-(aggregated-features)?node-id=0%3A1&t=rkyCBOGUIC4cg0qB-1`

3.5 Usability testing

We defined a common list of relevant activities to be performed by users testing the prototypes:

- Download a file of a course

- Open a videolecture

- Check the time and location of the next classroom

- Find a free room at a given time

- Finding the location of a room

30

- Book an exam

- Create a support ticket

Note that for most of this tasks there are now multiple ways to perform them: for example opening a videolecture can be done either through the agenda or from the course it belongs to. Providing shortcut for users increases their ability to navigate through the UI.

For each of those tasks, we asked testers to **evaluate the perceived complexity** on a scale from 1 - very hard to 7 - very simple. This scale is also known as SEQ, Single Ease Question, and is commonly used in usability testing activities.

Moreover, we observed testers while performing tasks and took notes of the **number and type of errors** they performed while trying to accomplish those tasks.

Each prototype was tested with a group of 20 students: each tester only interacted with one of the two prototypes, in order to avoid biasing.

We then summarized the results in terms of average SEQ and error-free rate ,that is, the percentage of tasks performed without committing any error (Table 3.1).

As you can see, both prototypes performed quite well, and we gathered very useful feedback by watching users interact with them and collecting their final remarks after testing.

The agenda was commonly appreciated as a feature, but many users expected some graphical way to rapidly distinguish different courses. The room search was coupled too deeply with the free rooms, causing confusion when interacting with it.

Besides those potential improvements, users were able to successfully interact with both prototypes: **we chose to move on with the grouped hierarchy of prototype 2**, expanding it with some features that were most explored in prototype 1.

	Prototype 1		Prototype 2	
	SEQ	**error-free**	**SEQ**	**error-free**
Task 1 - Files	6,45	75%	6,50	90%
Task 2 - Videolectures	6,65	90%	6,85	95%
Task 3 - Agenda	6,95	95%	6,25	95%
Task 4 - Free room	6,40	100%	6,65	95%
Task 5 - Room location	5,30	70%	5,80	65%
Task 6 - Exam booking	6,75	85%	6,60	80%
Task 7 - Ticket	N/A	N/A	6,40	70%
Average	**6,42 / 7**	**86%**	**6,44 / 7**	**84%**

Table 3.1: Low fidelity usability testing: results

Chapter 4

Application design

4.1 Introduction

Once the main navigation and UX patterns were clarified we could move on to designing the actual application. This implies two separate activities:

- **understand how the app will interact with the servers** of PoliTO to fetch data and perform operations

- **draft the graphical appearance of the UI** as a set of reusable components to represent information coherently

4.2 API formal definition

The features provided by the previous app rely on multiple data sources belonging to the IT infrastructure of the Politecnico, wrapped by a set of PHP files providing a unified authentication layer.

Being a pass-through for multiple services, the responses lacked coherence in naming conventions and object representation: this was a big obstacle for readability, and would have made harder allowing external contributions to the project.

Therefore we chose to **design a new API specification** based on the REST paradigm, including all the features that were previously provided, plus some extras based on the content of the low-fidelity prototypes.

While performing this activity, we chose to follow these main patterns:

- provide authentication via a **Bearer token in the Authorization header**

- use HTTP methods to describe mutations

- convert all response fields to readable english wording, in camelCase format

The new API specification was formally described using **OpenAPI Specification 3.0** [5], an industry-wide standard for REST API representation through JSON objects in YAML format.

The complete specification will be publicly available on the GitHub organization of PoliTO, at `https://github.com/polito/api-spec`.

Having a formal description before writing any code produces multiple positive side effects, we'll briefly discuss all of them.

4.2.1   Planning

Having a shared representation allowed us to discuss API functionality with a clear idea of the expected output: in this way we were able to effectively engage members of the IT team that were not previously involved in the project.

This also made easier prioritizing the development of the new API based on the expected roadmap for the development of the app.

4.2.2   Documentation

Using the OpenAPI specification allows to use Swagger UI to obtain a graphical interface for it: in this way the API documentation is always updated with respect to its formal definition (as in Figure 4.1).
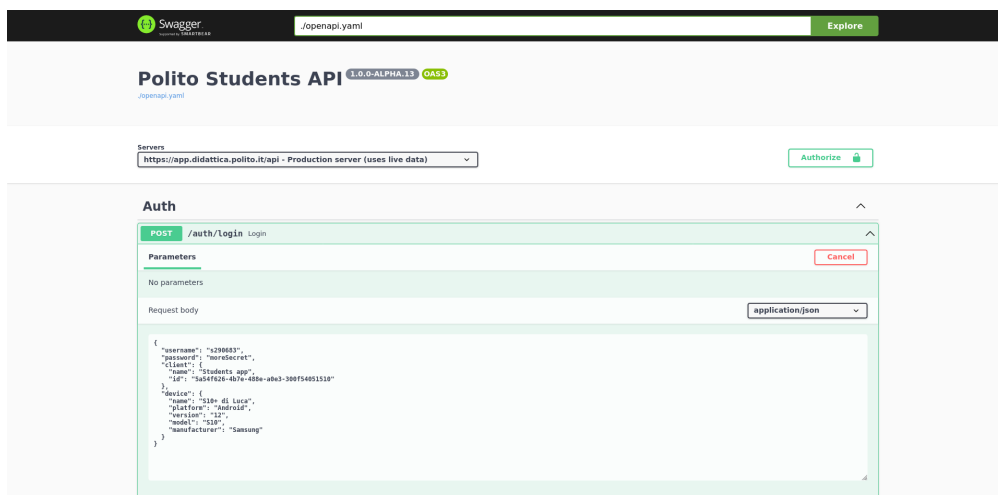

Figure 4.1: Swagger UI API specification

34

4.2.3 Mocking

While the new API was still under development, we were able to **autogenerate a mock API** compatible with the specification by adding relevant examples to the formal definition and using Prism [6] to dynamically generate a mock server.

Of course, this does not replace the need for a production API, but allowed us to test the interaction of the app with a remote server from day one, minimizing the need for refactoring when switching to the real API.

4.2.4 Tooling

Another interesting side effect of using a formal specification is the ability to **auto-generate strictly typed client SDKs and server stubs** based on the model definitions included into the specification.

There are multiple tools with the ability to generate such clients, we chose to use OpenAPI generator (`https://openapi-generator.tech/`) to obtain a TypeScript fetch client to be referenced as a dependency into the app.

If you are interested to know more on how we handled the generation and versioning of this client keep on reading, or jump to **subsection 5.4.1**.

4.2.5 Testing

Last but not least, having a formal definition allows to **easily identify incoherence in response structure and types** between the definition and the output of the real API.

This was not implemented for PoliTO at this time, but could be applied either into the server (testing all responses before sending them) or as a step into the continuous deployment pipeline to ensure coherence before deploying a new version of the app to students.

4.3 High fidelity prototyping

The data that emerged from end-user testing, informal feedback and reviews with the IT department team members was then used to produce a high fidelity mockup that synthesized the best aspects of both of the previous prototypes.

4.3.1 Providing a near-native experience

A challenging aspect of this phase was conciliating a modern and intuitive UX with React Native's technical limitations and external requirements, brand identity expression above all. In particular, the framework has been criticized for creating apps that feel too homogenized (writing a single codebase can result in apps that have a similar look and feel across different operating systems) yet it was crucial for us to provide a near-native experience for both Android an iOS users, therefore we chose not to use a general-purpose Design System. Despite this, we found NativeBase's Figma Kit a solid starting point to create our custom theming and component system.

Four different variants of the final prototype were created in order to showcase the light and dark themes as well as platform-specific features and design elements.

In order to convey the aforementioned native feel we incorporated small variations on many UI components driven by the thoughtful consideration of both design guidelines.

As an example, iOS lists use dividers and are enclosed in flat surfaces with rounded corners (as a reference to inset grouped lists) while on Android they use full-width elevated surfaces without dividers (in conformance with Material Design guidelines).

Other variations include headers: translucent on iOS while opaque and elevated Android, list navigation indicators, call-to-action buttons.

4.3.2 Structure and navigation

The hierarchical reorganization of the main features was refined across multiple iterations, the last one of which is shown here.

Current

Main navigation

- Courses
- Programs of study
- Transcript
- Class schedule
- Activity bookings
- Provisional grades
- Exams booking
- Lectures timetable
- Libraries
- Free rooms
- Maps
- Email
- Support tickets
- People
- Emergency services
- Job offers
- News and alerts
- Issue report
- Settings
- ~~Maintenance~~

More

- Account switch (BSc <> MSc)
- ~~Covid info~~
- ~~Messages~~

Proposal

Main navigation

- Teaching
  - Courses
  - Transcript
  - Exams
- Agenda
- Places
- Services
  - Programs of study
  - Job offers
  - News
  - Activity bookings
  - Support
  - Directory
  - Emergency services
  - Libraries
- User
  - Settings
  - Account switch
  - Issue report

Onboarding/FAQs
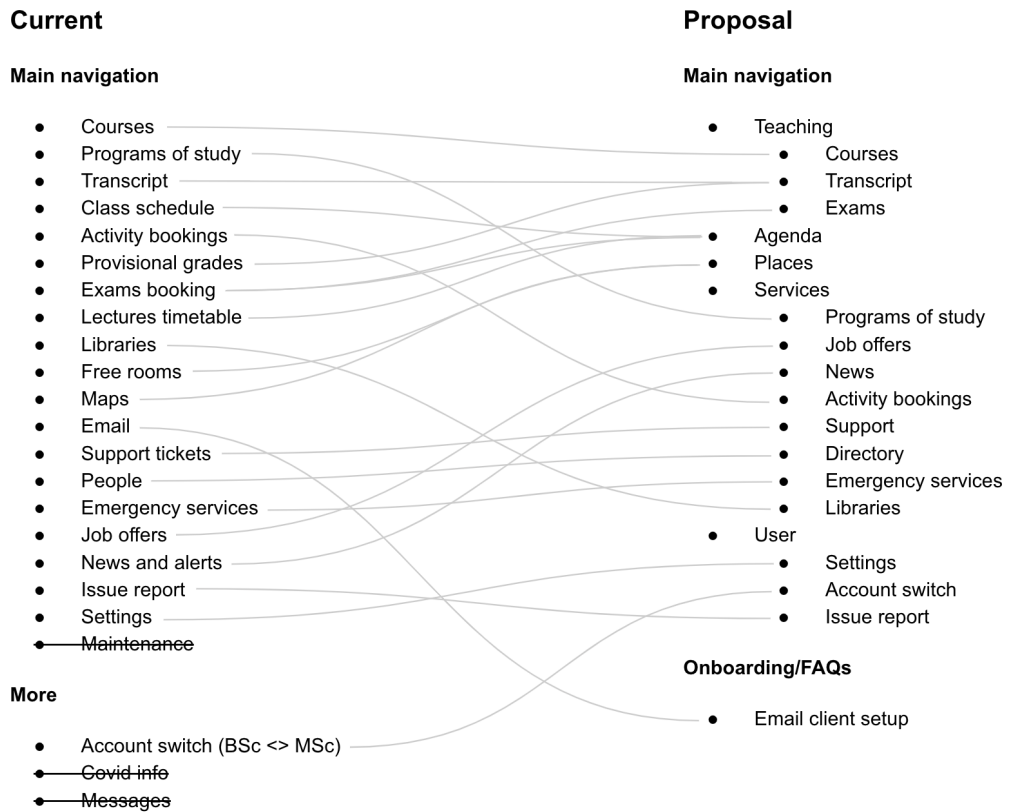
- Email client setup

Figure 4.2: Hierarchical re-organization of features (latest version)

The bottom tab bar was chosen as the main navigation method, with one section for each top-level semantic area. Customizable visual clues were added to course representations to better recognize them in the various areas of the application where they are shown.
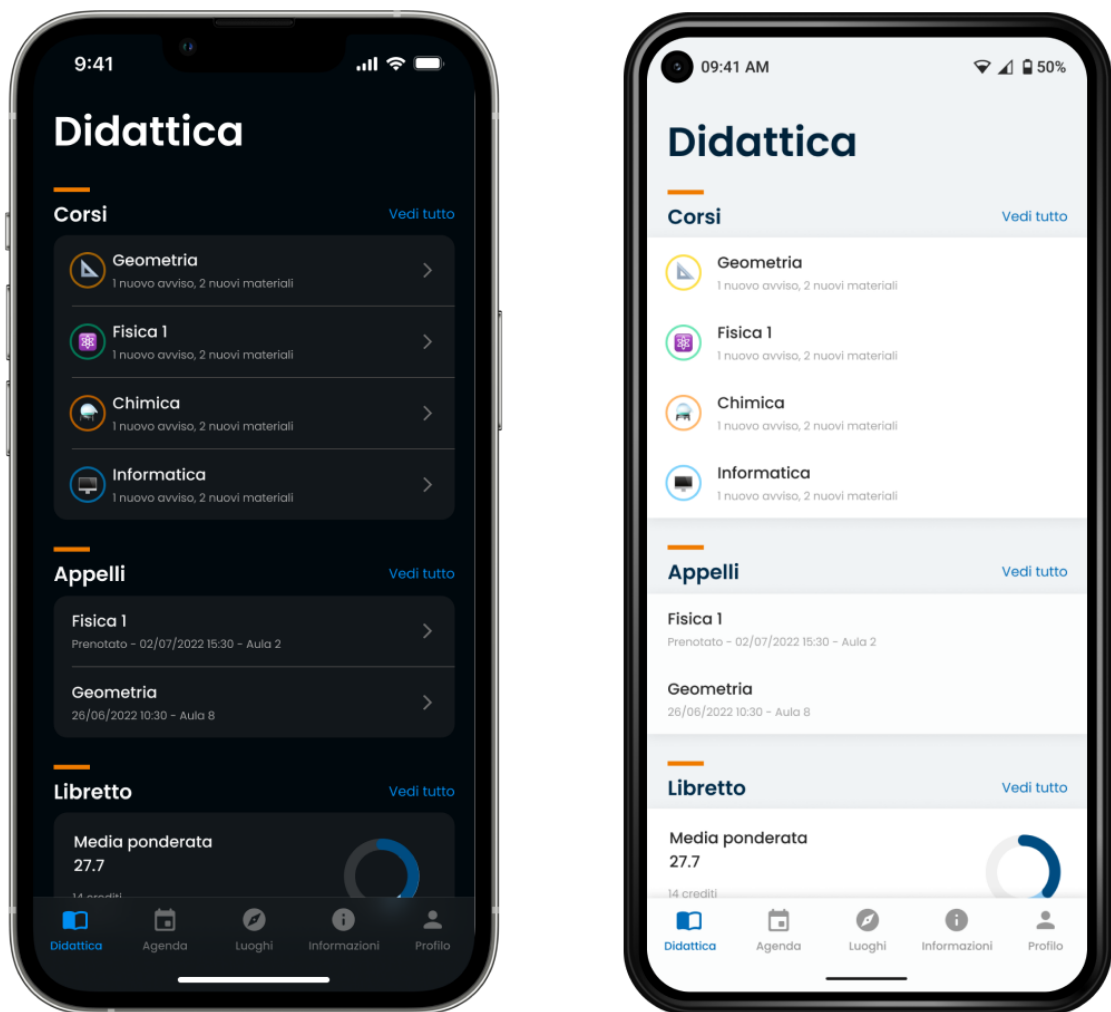
Figure 4.3: High fidelity prototype - Teaching screen

The agenda screen shows a linear list of all the commitments, grouped by date. The same visual clues cited before are applied in order to recognize courses at a glance. A live indicator shows that a lesson is being taught in the present time and filters are used to limit the amount of events shown to a specific category.
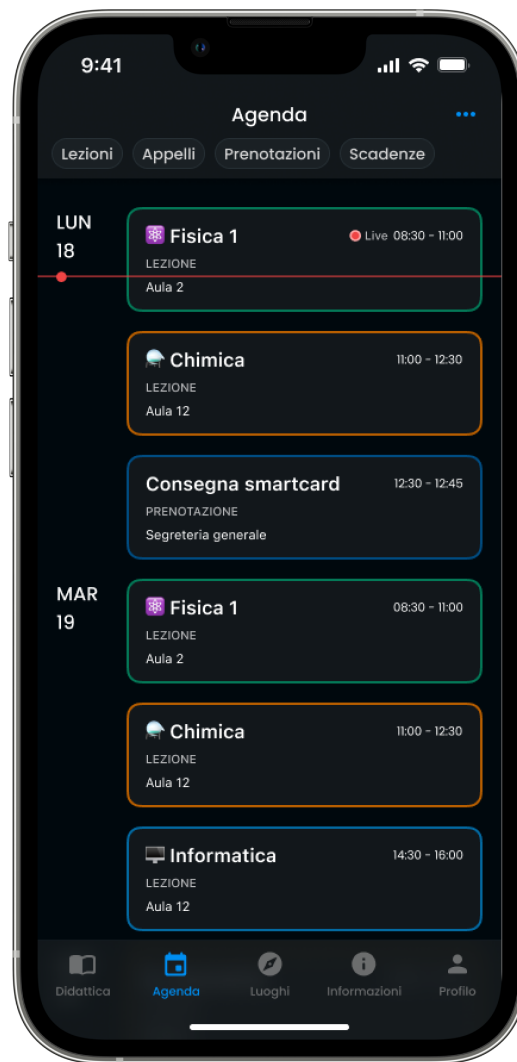
Figure 4.4: High fidelity prototype - Agenda screen

The places tab shows an interactive filterable map with a campus selector and a global search field. Lists and details of places are laid out in a translucent collapsable bottom sheet.

Figure 4.5: High fidelity prototype - Places screen

The user tab shows the main information about the student including a digital copy of her smartcard. From this page, the settings and notifications archive can be reached, as well as the logout call-to-action.

Figure 4.6: High fidelity prototype - User screen

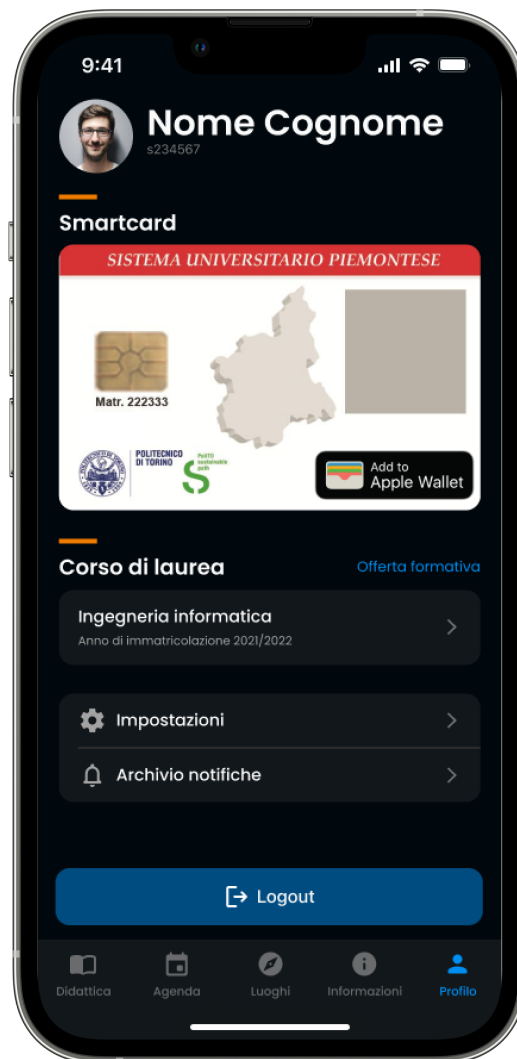The complete interactive mockup is available at `https://www.figma.com/file/2AhHO6LfDD25WTqd074f0X/PolitoApp-(high-fidelity---public)?node-id=984%3A11390&t=4zhkhuABfK5uWrCl-1`

Chapter 5

Implementation

5.1 Project management

Given the goal of releasing the whole project as open source, we chose to use GitHub to host the codebase and track the progress in development.

The first step in doing so was creating a GitHub Organization for PoliTO, which will be publicly available at `https://github.com/polito/`.

We then created a GitHub Project to handle development tasks, using a Scrum-like task management approach (Figure 5.1).

Figure 5.1: GitHub Project - Tasks backlog

GitHub Project tasks can be easily converted to repository issues: this allowed

us to track progress in the implementation of them by just referencing the issue numbers into the commit messages.

Having to start from scratch, our first milestone was the development of a fully working beta version containing all the features included into the Teaching tab.

We worked on two separate repositories:

- **@polito/students-app** (at `https://github.com/polito/students-app`), containing the code of the mobile application

- **@polito/api-spec** (at `https://github.com/polito/api-spec`), containing the formal specification of the API as described in section 4.2

5.2   Project structure

Since the project will receive contributions from multiple stakeholders, we focused on defining from day one **a series of rules and patterns to maximize coherency** across them.

We chose to split the code for different features in modules to keep the main areas semantically organized (Figure 5.2). Each module is divided by entity type (components, hooks, styles, screens). The core module contains general-purpose items which are used across the app.

```
├── assets
├── src
│   ├── App.tsx                # App entry point
│   ├── core                   # Core module
│   │   ├── components
│   │   │   └── RootNavigator.tsx
│   │   ├── hooks
│   │   └── screens
│   ├── features               # Feature modules
│   │   ├── teaching
│   │   │   ├── components
│   │   :   ├── hooks
│   │   │   └── screens
│   └── utils                  # Utilities
└── lib                        # Library modules
```
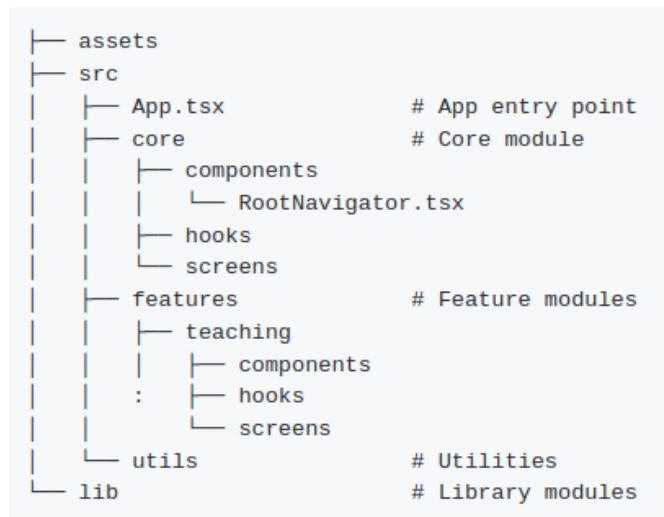
Figure 5.2: GitHub repository structure

The lib folder is used to isolate design system components that may be extracted into a dedicated package in the future to be reused in the Teachers app.

43

For what concerns code styling, we suggest contributors to follow the **Google TypeScript Style Guide** [7].

We chose to enforce some formatting and linting rules in git hooks [8] (that is, scripts that are automatically executed when certain important actions occur in the version control flow). To do so, we used several tools:

- **husky** (`https://typicode.github.io/husky`) allows to distribute hooks as part of the npm project in version control so that all collaborators are synced to the same workflow

- **lint-staged** (`https://github.com/okonet/lint-staged`) allows to execute hooks on staged files only

- **eslint** (`https://github.com/eslint/eslint`) automatically identifies problematic patterns and prevents them from getting into the codebase

- **prettier** (`https://github.com/prettier/prettier`) enforces code formatting rules (quote style, import order, indentation...)

Finally, we established some ground rules for our Git workflow, inspired by Git Flow [9]:

- using **Conventional Commits** (`https://www.conventionalcommits.org/en/v1.0.0/`) to give consistency to commit messages

- applying *feature/<scope-with-hyphens>* and *hotfix/<scope-with-hyphens>* branch names, to be detached from *develop* or the parent feature branch and merged back via PR.

These conventions will be beneficial for generating clean and consistent changelogs and release notes.

The full list of contribution guidelines are embedded into the repository as a Markdown document: `https://github.com/polito/students-app/blob/main/CONTRIBUTING.md` .

5.3   Technological choices

While React Native by itself provides many building blocks for mobile applications, we had to pick several libraries to expand its functionalities: we'll briefly discuss the most relevant choices.

5.3.1   React Navigation

React Navigation is a routing and navigation library for React Native: it provides a set of components to represent screens and their context.

We chose to **implement navigation with a two-layered approach**:

- A BottomTabsNavigator, providing the ability to move across different tabs

- One NativeStackNavigator inside each tab, handling the reprentation of a single screen and the history. Being "native", it provides the **graphical appearance and features of using the equivalent native navigator** on both OSs

By using React Navigation we were able to strictly type route parameters and easily handle navigation across screens.

Moreover, React Navigation also provides a way to **abstract the acquisition of screen navigation analytics**: we tested this behaviour by integrating Web Analytics Italia, a customized Matomo instance built for the Public Administration.

Another benefit for future implementations is that React Navigation also allows to **define deep links by associating our app with a custom scheme**: this means that we will be able to directly open PoliTO Students on the screen of a lecture from the calendar event on the phone of a user.

A full overview of the features offered by React Navigation is available at `https://reactnavigation.org/docs/getting-started`

5.3.2   React Query

React Query provides a **declarative approach to completely abstract state management related to data fetching**: it handles caching, background updates and stale data.

The whole library is based on two main concepts: queries and mutations.

This has several implications on the new features we were able to introduce, so let's briefly discuss them.

Queries

A query is a declarative dependency on an asynchronous source of data that is tied to a unique key.

```
const { isLoading, error, data } = useQuery(
  ['exams'],
  () => fetch('https://app.didattica.polito.it/api/exams')
        .then(res => res.json()),
  {
    staleTime: 600 // seconds
  }
)
```

The *useQuery* hook binds the result of a promise to a unique key, possibly containing variables. The third, optional parameter of the hook allows to customize the query behaviour via options: **staleTime defines after how much time the data provided by the promise should be considered stale and background-refetched**.

This means that using the same query across multiple screens will not trigger more API calls unless the staleTime has passed, and when it does it will be transparent to the user.

Mutations

Unlike queries, mutations are typically used to create/update/delete data or perform server side-effects.

Their response is not state-persisted, but **the success of a mutation is typically handled by invalidating one or more queries**.

```
const bookExamMutation = useMutation(
 () => fetch(`https://app.didattica.polito.it/api/exams/${examId}`,
 ↪  requestOptions),
 {
   onSuccess: () => queryClient.invalidateQueries(['exams'])
 })
```

As you can see, React Query is a really powerful tool for abstracting the complexity of synchronization with a remote data source.

For a full overview of the features it offers, please refer to `https://tanstack.com /query/v4/docs/overview`.

5.4 DevOps

The release process for the previous version of the application was entirely manual. In the spirit of applying state-of-the-art technologies and patterns, we chose to introduce some automations both for the API client and the app via GitHub Actions workflows.

5.4.1 API Spec

As briefly anticipated in section 4.2, one of the many advantages of having a formal API specification is the ability to generate strictly typed clients to interact with the API.

We chose to **automate the generation of a TypeScript fetch client**: the client is then published as a package into the private registry of the GitHub Organization (in the future, into the public npm registry).

The client is auto-generated through a GitHub action, running at every new tagged version using the Semantic Versioning format [10]. In this way adding new API features results in the following process:

1. Describe the expected output of the new feature by creating a new branch in the specification repository

2. Develop the new features

3. Verify coherency with the specification

4. Merge the updated specification into the main branch, tagging the new version

5. Update the version of ***@polito/api-client*** in the package.json file of the app to get the new features

To generate the client we used OpenAPI Generator (`https://openapi-generat or.tech/`), an open source project providing client generation capabilities from a YAML specification.

5.4.2 Students app

The workflow for the mobile application is more complex, since it has to generate artifacts for both Android and iOS devices (Figure 5.3).

The first step in the workflow is the installation of dependencies and the execution of tests.
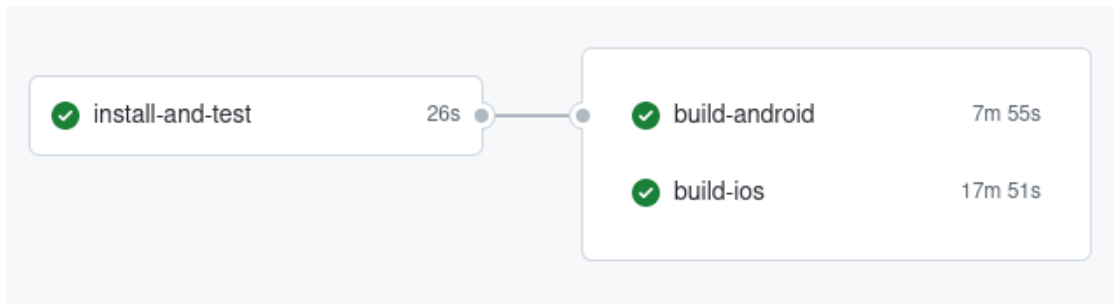
Figure 5.3: GitHub workflow - Students App

After the first step is completed, the two build process can be started in parallel: the android build runs on a virtual Ubuntu runner provided by GitHub, while the iOS build (that requires to be executed on macOS) runs on a self-hosted device inside the IT infrastructure of Politecnico.

The workflow currently runs for pull requests, and pushes on branches main and develop: its output is just the two build artifacts, available in the summary of the Action instance.

We also considered using Fastlane (`https://fastlane.tools/`) to **automate app stores deployment**, but since at the time we are writing the app is not ready yet for release this step could not be tested.

5.5 The first alpha release

Given the limited time frame of the thesis project, the focus of the implementation activity was primarily directed towards the teaching-related features (where the most important interactions occur), approximately corresponding to the first top-level navigation tab.

As noted in **subsection 4.3.2**, the hierarchical reorganization was used as a guiding reference for the main navigation. The main evolution from the high-fidelity prototype is that the **Information** section (now called **Services**) is being restructured in order to incorporate a number of different services such as **Libraries**, **Directory** and **Courses catalog**.

Other changes were made as technical factors demanded them; in particular the iOS/Material Design 3-style collapsing headers proved to be too difficult to replicate on Android in this first phase: normal single-row headers were used instead. Also, the dark theme color palette was revised in order to better express the brand's primary color.

5.5.1 Teaching home screen

The teaching home screen retained the subsection extracts-based structure, with minor changes to course visual clues targeted to improve accessibility.
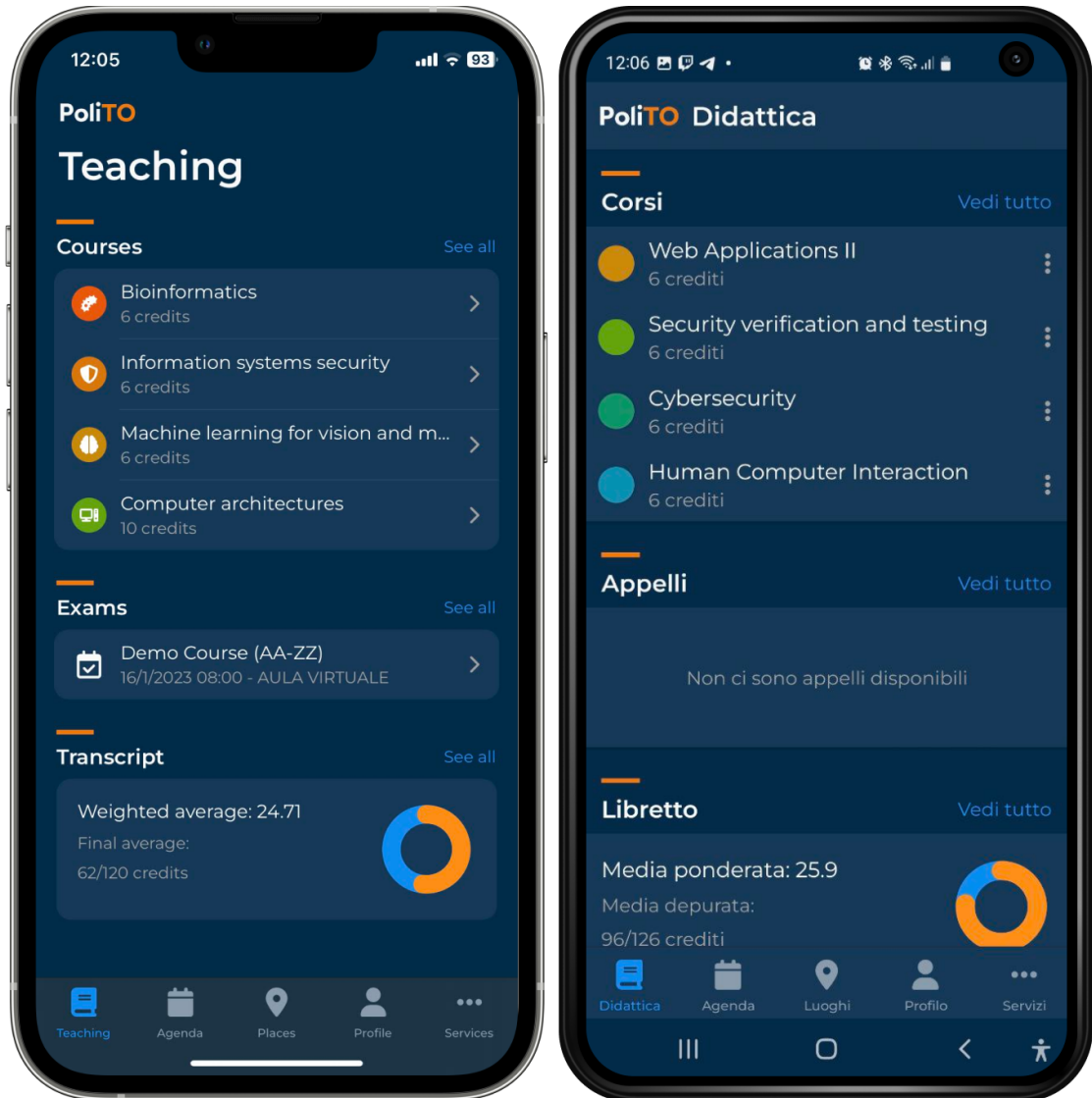
Figure 5.4: The Teaching home screen in the first alpha release (dark theme)
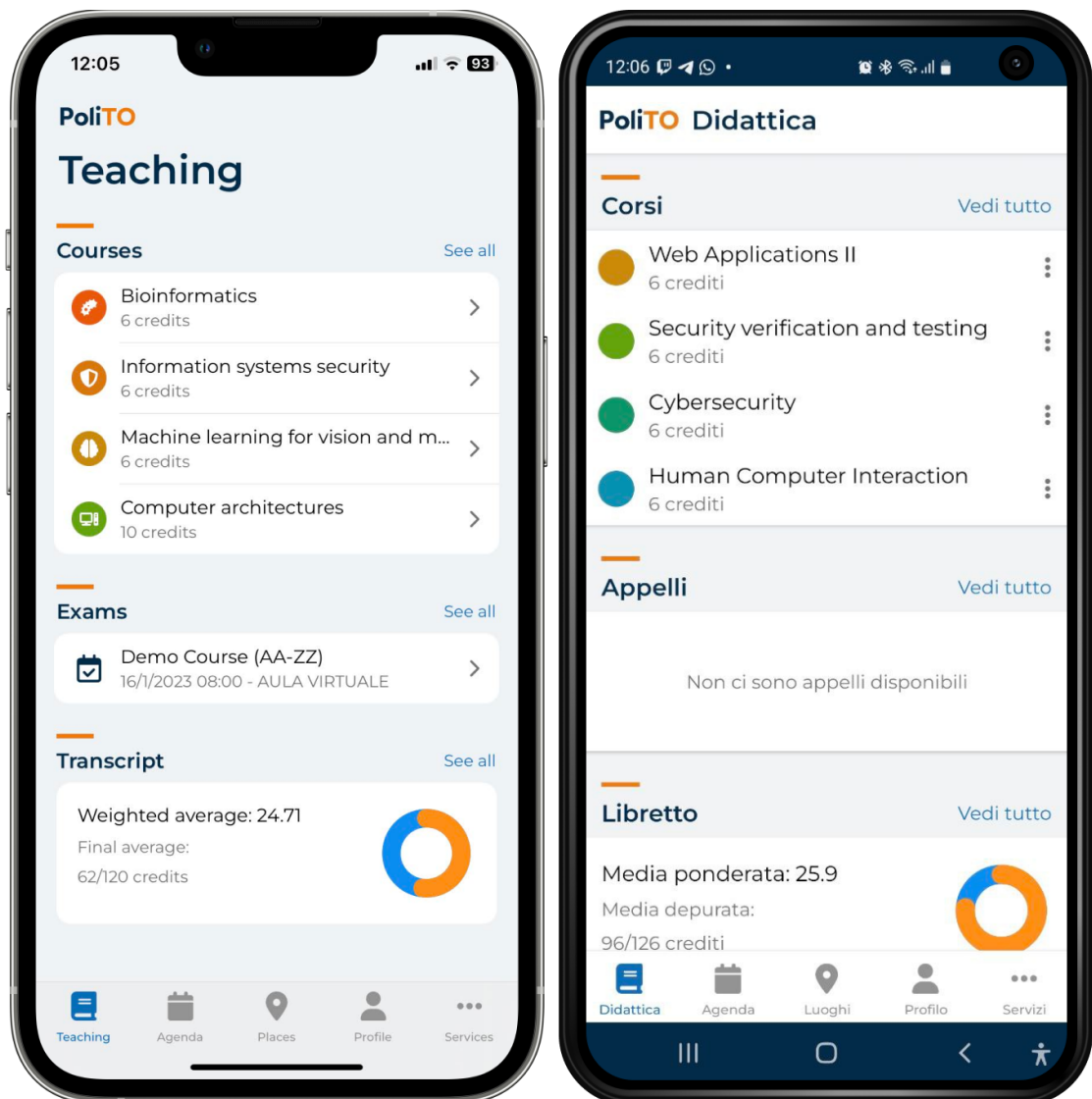
Figure 5.5: The Teaching home screen in the first alpha release (light theme)

5.5.2 Course screen

As for the course screen, aesthetic changes were made to some UI elements such as tab indicators, the info tab was streamlined, showing less metrics at the top and more important information in extended form (full course title and code).

File and video lecture card grids were abandoned in favor of flat lists, primarily because graphic covers were unavailable in most contexts. Nevertheless, this structure facilitates browsing long lists and searching for specific items. In the

files tab only the last files in order of upload date and time are shown, with the complete browsing experience reachable through a **Browse files** call-to-action.
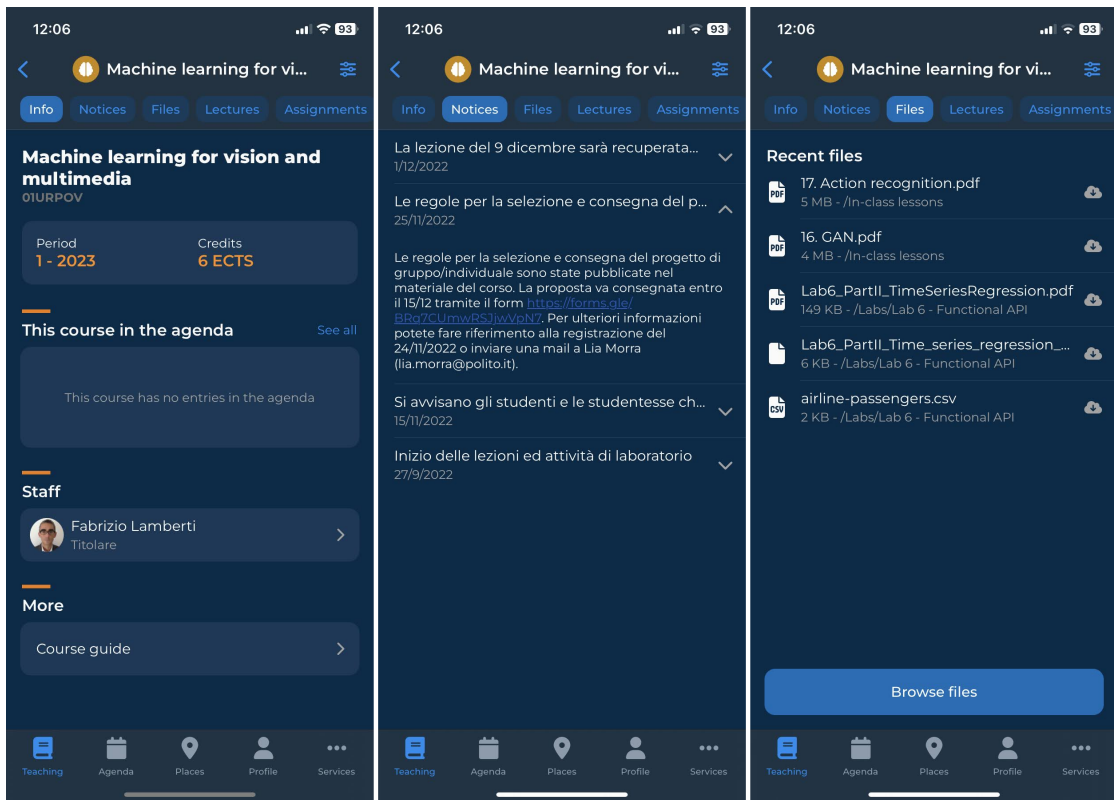
Figure 5.6: Course screen's info, notices and files tabs (dark theme)

The files browsing is carried out in a dedicated navigation flow, with one screen per folder.

Lectures are laid out in a sectioned list, where sticky translucent headers are used to divide as well as collapse/expand the various types of lectures (video lectures, virtual classrooms and other years' lectures).

A preferences screen was created in order to edit the main course-related preferences (colour and icon, course visibility in extracts, notification types as well as clean the local file cache).
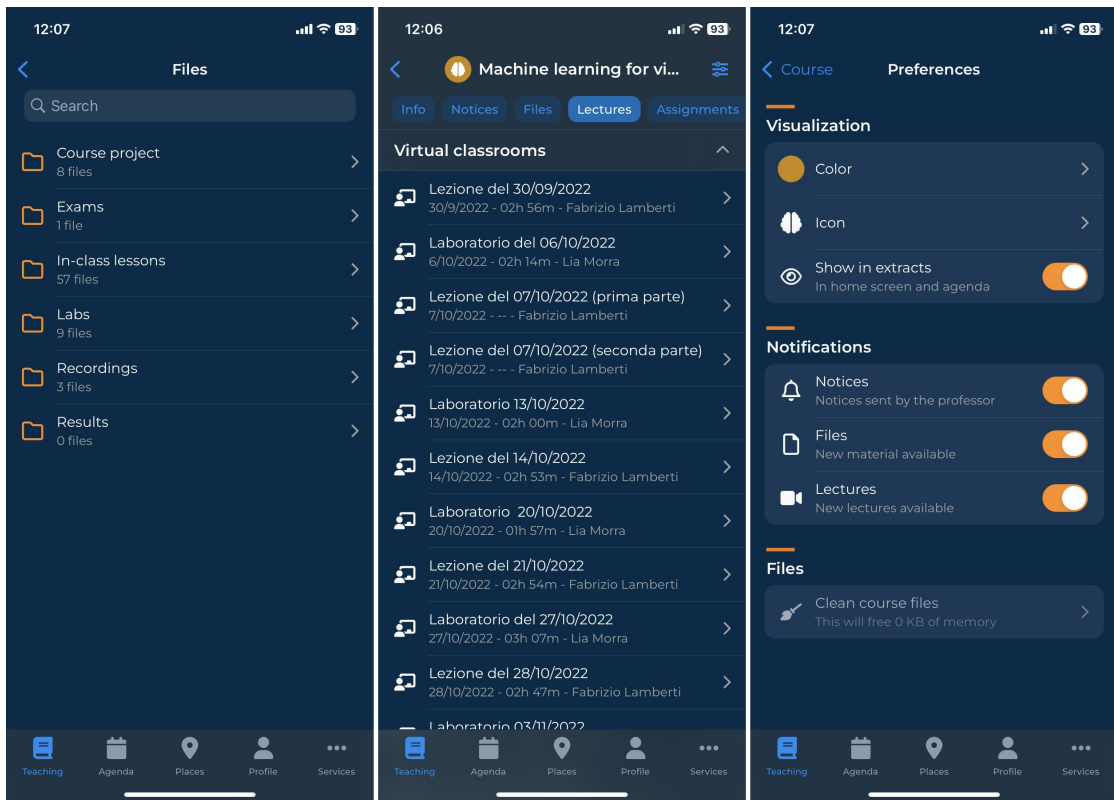
Figure 5.7: Course screen's lectures tab, course preferences and file browsing screens (dark theme)

5.5.3 Exam screen

The Exam screen shows relevant information on the location and staff involved in the call, with links to the relative screens. A call-to-action button allows to book the call (and cancel existing bookings). The loading and success states of such mutation are shown in-place.
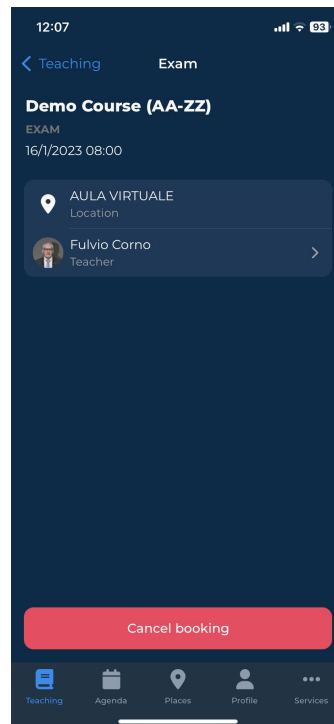
Figure 5.8: Exam screen (dark theme)

5.5.4   Transcript screen

The Transcript screen uses circular graphs in order to show the progress of the whole career and the current academic year, both in terms of attended and acquired ECTSs.
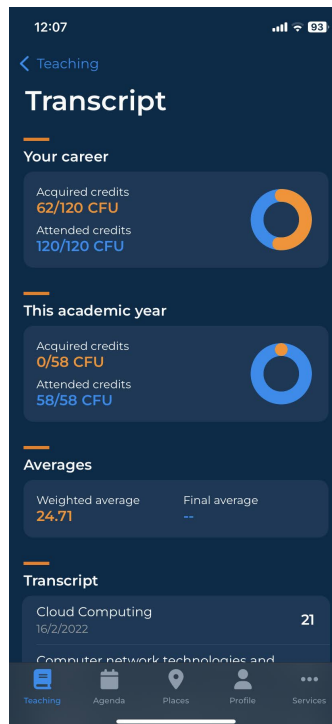
Figure 5.9: Transcript screen (dark theme)

5.5.5 Newly added features

Thanks to the above mentioned building blocks and the feedback provided by users throughout the design process we were able to introduce a good number of new features from the very beginning: we'll present the most relevant ones.

Course files persistence

In the previous app the download of course files was handled by redirecting the users to the web browser: this meant that it was hard for students to retrieve a previously downloaded file.

We chose to handle the file download internally: when clicking on a file, its download is started in the background and the progress is notified to the user with an indicator. Once it is completed, the file is marked as downloaded and immediately opened via a native intent, based on its type (Figure 5.10).

The mapping between API file and downloaded location is persisted: if some days later the student goes back to the course files, that file will be immediately available.
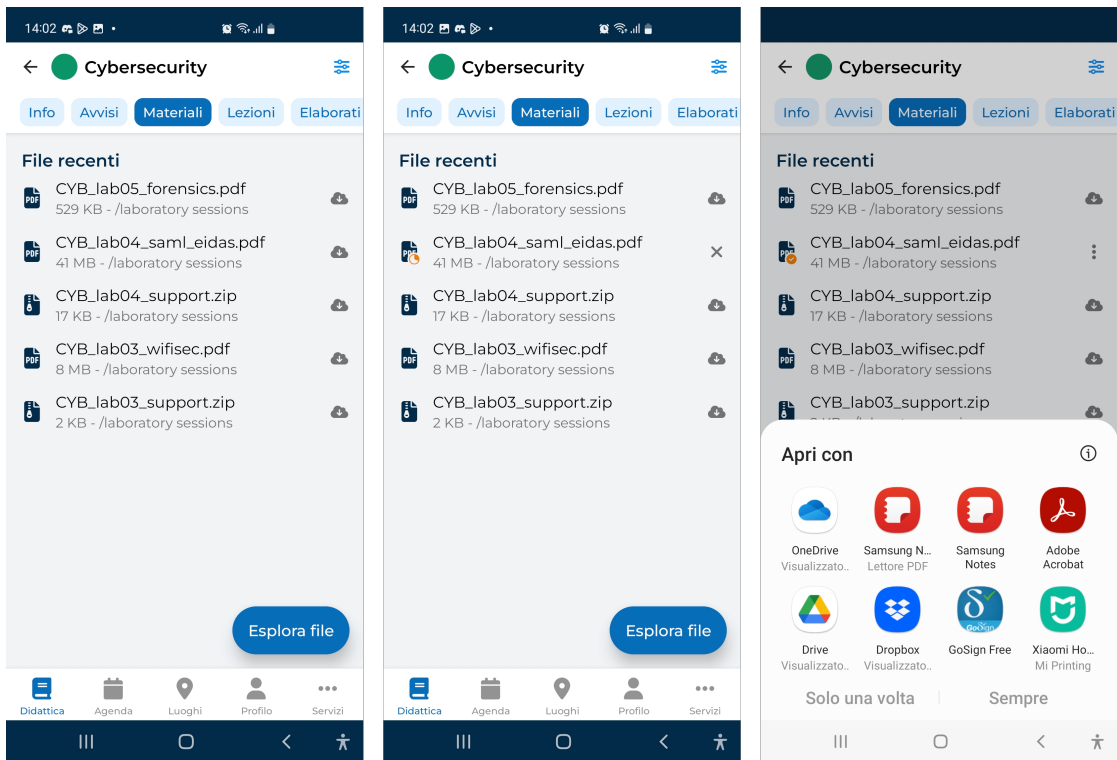
Figure 5.10: Polito Students - Course files

Course lectures

Also course lectures were previously opened in the browser: this meant that they were rendered via the standard web video player, that provides very little functionality and no contextual information aside from the video itself.

We introduced a **dedicated screen for the video lecture**: the video is rendered through the native player on iOS, and through a native player with a custom overlay on Android.

In both flavors, it provides fullscreen mode and playback speed controls.

Chapter 6

User Testing

6.1 Methodology

Once the development of the Teaching tab was completed, we moved to test the resulting UI by letting real students use the application on their phones.

We identified potential testers that were not part of the previous testing sessions: this guarantees a non-biased impression while performing the tasks.

We planned for half a day dedicated to testing, and confirmed their participation by asking testers to schedule their 20-minutes slot on Calendly: while doing so, they shared with us their PoliTO student ID and device type (Figure 6.1).

Figure 6.1: User testing booking on Calendly

With this information we could temporarily add to their student career a Demo Course, to be used during test to perform operations (such as booking an exam, or sending an assignment) that should not be abused on real-live courses.

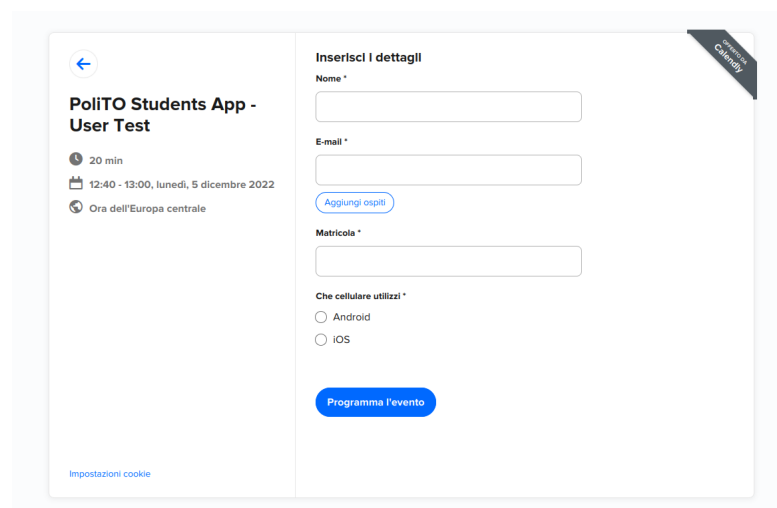During the test session we followed the protocol reported in section A.2: **we asked each tester to perform 4 tasks** among the 8 relevant tasks identified within the Teaching tab.

We assigned tasks so that each task was tested an equal number of times using a semi-Latin Rectangle: in this way the transfer of learning across subsequent tasks is minimized.

We chose to track the following metrics while performing each task:

- **SEQ - Single Ease Question**, the perceived complexity on a scale 1 to 7

- the number and type of errors, then elaborated as an error-free rate

- the time needed to complete the task

At the end of testing we asked each tester to compile the **SUS - System Usability Scale** [11], a 10 item questionnaire that allows to measure the perceived usability of a system.

Besides those metrics, we gathered feedback during the test session by asking for impressions after the first free exploration and at the end of all tasks.

We also screen-recorded the interaction with the app to be able to watch it again in case any usability issue arose.

6.2   Results

We performed tests involving 11 students from different degrees and at different times in their career (bachelors at their first semester, bachelors at their third year, masters) in order to create a varied sample.

The results were really encouraging: all students appreciated the new interface, recognizing the usage of graphical elements that are part of the brand identity of PoliTO.

We also validated the choice to respect the UX patterns of the Operating System of their phone: users immediately recognized the semantic meaning of ellipsis on Android, and expected a long-press or swipe gesture on iOS.

The average System Usability Scale, gathered after testing using the module reported in subsection A.3.2, **is 90.9, above the 95th percentile** for systems

evaluated using this method [12].

This confirms that the overall perception of the experience while using the app is highly positive.

The results of the single tasks are summarized in Table 6.1: as you can see the average SEQ is still high, but **some tasks were harder to accomplish, or resulted in a low error-free rate**.

Task	SEQ	error-free	time
A - Course files	5.6	33%	59s
B - Course appearance	6.9	100%	19s
C - Exam booking	6.7	100%	29s
D - Course assignment upload	5.9	38%	72s
E - Course lecture	7.0	86%	17s
F - Hiding a course	6.6	100%	11s
G - Transcript	7.0	100%	6s
H - Course notices	6.8	75%	16s
Average	**6.5**	**79%**	**29s**

Table 6.1: App user testing: results

Let's discuss what happened during those critical tasks, and what we learned from it.

A - Course files

We asked testers to locate a file that was not immediately visible while getting into the Files tab of the course: this was a deliberate choice, as we wanted to verify the ability to navigate through the folders.

The testers that were accustomed to use the previous app **expected all files to be included into the recent list, not just a selection of them**: they would stop in the Files tab, unsure of what to do next (Figure 6.2).

We discussed the issue with them: the most reasonable choice to help students transition from the previous app to the new one is showing all files ordered by date, even though they will typically access just the top ones.

We also understood that the CTA label was not clear enough: we will replace Esplora file/Browse files with a folder-related string.
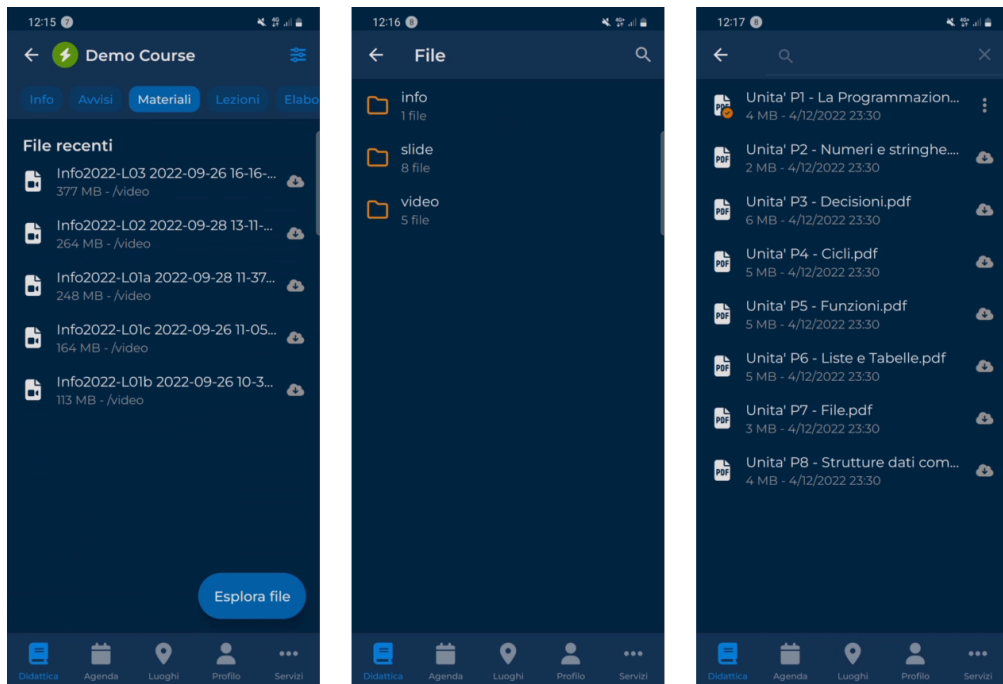
Figure 6.2: Task - Course files

C - Exam booking

We asked testers to move the exam booking from the first to the second call: **the ones with most experience at the Polytechnic automatically cancelled the existing booking** and then booked the new one, **while the new students typically booked the new call** without cancelling the previous one first.

This made us realize that the ban on booking multiple calls of the same course was not represented into the app nor enforced by the new API: we opened an issue to handle this behaviour.

D - Course assignment upload

We asked testers to upload a physical two-page document we provided to them as an assignment: the majority of them **got into the assignment upload screen, but then clicked on *Take a picture*** (Figure 6.3).

They did not even get to read *Scan a document*: they immediately linked taking a picture to the action they had in mind. They only understood something went wrong when they got to the confirmation screen, with no possibility of adding more pages to the assignment.
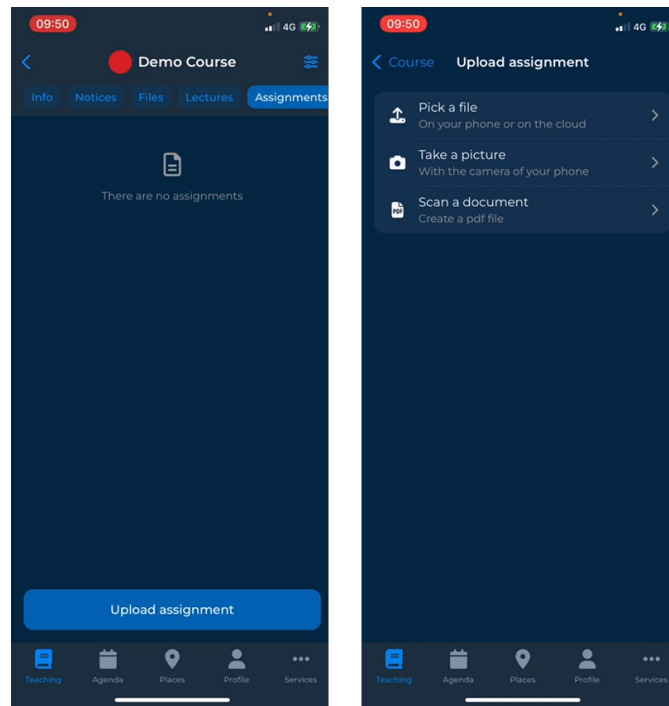
Figure 6.3: Task - Assignment upload

This made us realize that we might allow users taking a picture to "change their mind" and use the picture they took as the first page of the PDF.

Alternatively, we might remove the ability to upload a single picture as an assignment: it was provided by the previous app, but there is no real advantage to it, as the generated PDF just contains one picture per page.

F - Hiding a course

This task has an high SEQ and a really fast execution time, but was still able to teach us something: **one of the testers asked us why he could still see the exam calls of the course he just hid** into the homepage.

We did not think about it while planning for this feature, but it is totally reasonable to hide non-booked exam calls belonging to hidden courses: we took note of this improvement to the feature.

Other

While performing testing **we were also able to spot a bug not related to user experience** that would otherwise be hard to identify early: two of the testers

of Biomedical Engineering had in their Study Plan a course divided into multiple modules, all represented by the same course short code.

This caused these modules to appear duplicated into the Courses section in the homepage after any course was hidden from it, because of the non-uniqueness of the key associated to the items.

We are investigating how to properly obtain a unique identifier in this scenario.

Chapter 7

Conclusions

7.1 End results

In just a nine months time frame we designed and developed a new usable mobile application specifically designed for students.

Most importantly, while doing so **we were able to experiment some innovative processes** within the IT department of PoliTO **by applying for the first time a user centered approach** to the creation of a new product.

By keeping students involved in every step of our process we were able to consistently verify that what we were building was effectively responding to their needs.

As anticipated in section 5.5, the development of non-core features will continue in the following months. **The approach applied during this thesis will be iteratively applied when new features are added to the app**: we saw firsthand how half a day of user testing can help to easily identify usability issues and corner cases.

As engineers, **we are highly trained to solve complex problems, but we often forget the why behind the what we build**, the need behind the feature.

We thank the students who have collaborated in this process for helping us understand how the new app could improve their journey at PoliTO.

PoliTO Students will be published on app stores in the upcoming months, and the related source code will be made publicly available for review and contributions.

7.2 Future directions

While the work performed during this thesis has produced a direct result in the processes related to the mobile application for students, we believe **this experience might serve as a starting point for growth in two distinct aspects of the workflow of the IT department** of PoliTO.

We once again thank our interlocutors within the IT department for their commitment to this project and in tackling the challenges it entailed.

7.2.1 User centered approach

We have proven that applying a user centered approach in software design and development can be done, even inside a complex organization like PoliTO, and produces positive effects on the final product.

This positive experience can act as a guide for improving other bespoke tools to handle the needs of students, teachers and staff of the university.

7.2.2 Open source software

Public Administrations are shifting towards an open source approach, as mandated by the Digital Administration Code [2]: the new app is the first open source product developed internally by the IT department of PoliTO and freely made available for contributions and reuse by other universities.

PoliTO now has an university-wide GitHub organization that can host open source projects, and **while preparing this project for release has already tackled the challenges of making code publicly available** (readability and conventions, documentation, licensing, automations).

Hopefully, in the future more parts of the internal infrastructure of the university will be publicly auditable, contributing to the learning process of students and the technological growth of other universities in Italy and abroad.

Appendix A

Appendix

A.1 User testing consent form

Thank you for choosing to participate in the usability test for the new app for students of the Politecnico di Torino.

A.1.1 Testing method

We ask you to complete some tasks using the new application on your mobile phone. We will record the screen of your device in order to be able to analyze the choices you make as you navigate through the screens.

At the end of each activity we ask you to rate how difficult it was for you to complete the task.

At the end of the test we ask you to give us an overall evaluation of your experience using the application.

A.1.2 Purpose of the test

We are evaluating the usability of the Teaching tab of the new application, we will use the results collected to improve the user experience.

Remember that we are not here to evaluate you - we need your help to improve the application for all students.

A.1.3 Data Processing

The audio and video recordings will be kept only to evaluate the navigation choices made during the tasks, and will be deleted after analysis.

The information you share with us during the usability test will only be analyzed anonymously for the purposes described above.

You can decide to withdraw your consent at any time without consequences of any kind: to request it after taking part in the test, contact us indicating your tester code at me@lucapezzolla.com.

A.2 User testing protocol

A.2.1 Welcome

Briefly tell about the app project (tabbed structure, we are exploring the functionality of the first) and the purpose of this test (we are testing the app and not you).

We have temporarily added a zero CFU course to your profile, we will use that to perform the tasks.

Have the consent signed, write the tester code on the form.

Install the application on the student's mobile phone, have him log in without navigating in the app.

[Install screen recording app if needed]

A.2.2 Exploration

Give the student 30 seconds to navigate the app and get an idea: collect his first impressions of the interface.

A.2.3 Tasks

Introduce the student to the think-aloud methodology, to be used for all tasks.

Start recording the screen on the student's mobile phone (our audio backup, just in case).

Have the four tasks selected for the student performed one at a time (see table below).

Count the errors for each task (we should be able to verify them from the log).

At the end of each task, collect the response to the SEQ and the number of errors. You don't need to ask to go back to home when a task finishes, let him pick up where he left off.

A.2.4   Conclusion

Stop screen recording, forward recording to self.

Ask the tester to fill the SUS and collect the final comments.

Stop audio recording, give reward.

A.2.5   Task list

A. Open the presentation slides of the Demo Course

B. Customize color and icon for Demo Course

C. You have decided to take the Demo Course exam in the second call instead of the first, move the booking

D. Submit this document as an assignment for the Demo Course

E. Play the video lecture on double integrals

F. You will not follow the Demo Course this semester: hide it from the list on the home page

G. Check how many credits you have attended and acquired in your career

H. The lecture of Demo Course on Friday was been cancelled, check when it will be recovered

A.2.6   Mapping between testers and tasks

Tester	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6
1	E	B	H	C	A	D
5	G	C	E	H	F	A
7	B	D	A	F	H	G
10	A	G	F	D	B	E
11	F	A	B	G	D	H

Table A.1: User testing - Mapping between Android testers and tasks

Tester	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6
2	C	D	B	E	/	/
3	D	E	C	F	A	H
4	G	H	F	A	D	B
6	H	A	G	B	F	E
8	B	F	H	C	/	/
9	E	G	A	D	B	C

Table A.2: User testing - Mapping between iOS testers and tasks

A.3 User testing module

A.3.1 SEQ - Single ease question

Task #

How difficult was it to complete this task?

Difficult						Easy
1	2	3	4	5	6	7

A.3.2 SUS - System Usability Scale

For each of the statements below, please tick the box that best describes your experience using the app today.

	Strongly disagree				Strongly agree
1. I think that I would like to use this app frequently	1	2	3	4	5
2. I found the app unnecessarily complex	1	2	3	4	5
3. I thought the app was easy to use	1	2	3	4	5
4. I think that I would need the support of a technical person to be able to use this app	1	2	3	4	5
5. I found the various functions in this app were well integrated	1	2	3	4	5
6. I thought there was too much inconsistency in this app	1	2	3	4	5
7. I would imagine that most people would learn to use this app very quickly	1	2	3	4	5
8. I found the app very cumbersome to use	1	2	3	4	5
9. I felt very confident using the system	1	2	3	4	5
10. I needed to learn a lot of things before I could get going with this app	1	2	3	4	5

Bibliography

[1] Donald A. Norman. *The Invisible Computer*. MIT Press, 1998. Chap. 10 (cit. on p. 2).

[2] *Articles 68 and 69 of the Italian Digital Administration Code*. URL: `https://docs.italia.it/italia/developers-italia/gl-acquisition-and-reuse-software-for-pa-docs/en/stabile/software-reuse/reuse-model.html` (cit. on pp. 4, 63).

[3] Jakob Nielsen. *How to Conduct a Heuristic Evaluation*. Nielsen Norman Group. Nov. 1994. URL: `https://www.nngroup.com/articles/how-to-conduct-a-heuristic-evaluation/` (cit. on p. 13).

[4] Jakob Nielsen. *10 Usability Heuristics for User Interface Design*. Nielsen Norman Group. Nov. 2020. URL: `https://www.nngroup.com/articles/ten-usability-heuristics/` (cit. on p. 14).

[5] *OpenAPI Specification v3.0.0*. OpenAPI Initiative. July 2017. URL: `https://spec.openapis.org/oas/v3.0.0.html` (cit. on p. 34).

[6] *HTTP Mocking using Prism*. Stoplight. URL: `https://meta.stoplight.io/docs/prism/83dbbd75532cf-http-mocking` (cit. on p. 35).

[7] *TypeScript Style Guide*. Google. URL: `https://google.github.io/styleguide/tsguide.html` (cit. on p. 44).

[8] *Git Hooks documentation*. URL: `https://git-scm.com/book/en/v2/Customizing-Git-Git-Hooks` (cit. on p. 44).

[9] *Git Flow*. URL: `https://danielkummer.github.io/git-flow-cheatsheet/` (cit. on p. 44).

[10] Tom Preston-Werner. *Semantic Versioning 2.0.0*. URL: `https://semver.org/spec/v2.0.0.html` (cit. on p. 47).

[11] John Brooke. *SUS: A 'Quick and Dirty' Usability Scale*. A. L. McClelland, 1996 (cit. on p. 57).

[12]   Jeff Sauro. *A practical guide to the System Usability Scale: Background, Benchmarks & Best Practices.* CreateSpace Independent Publishing Platform, 2011 (cit. on p. 58).