SAPIENZA
UNIVERSITÀ DI ROMA

Progettazione e sviluppo della funzionalità di apertura di un verbale in InfoProf

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica

Corso di Laurea in Informatica

Candidato

Edoardo Gabrielli
Matricola 1693726

Relatore

Prof. Emanuele Panizzi

Anno Accademico 2019/2020

Progettazione e sviluppo della funzionalità di apertura di un verbale in InfoProf
Tesi di Laurea. Sapienza – Università di Roma

Questa tesi è stata composta con LaTeX e la classe Sapthesis.

Email dell'autore: gabrielli.1693726@studenti.uniroma1.it

Indice

Capitolo 1

Introduzione

1.1 InfoStud e InfoProf

L'app InfoStud nasce nel 2012 dalla necessità degli studenti di avere il sistema InfoStud sviluppato da InfoSapienza su mobile. Data la natura poco *mobile-friendly* del sistema web, nacque l'app ufficiale sotto il brand di SapienzaApps.

SapienzaApps, grazie al coordinamento del Prof. Emanuele Panizzi e all'aiuto degli studenti, pubblica e mantiene progetti come SeismoCloud, GeneroCity, InfoStud e InfoProf all'interno del Gamification Lab.

InfoStud è l'app che si rivolge unicamente agli studenti iscritti alla Sapienza e mette a disposizione un parco di funzionalità ampio che copre le funzioni standard della versione web (come visualizzazione e gestione esami) e andando oltre in alcuni casi: il sistema di gestione dell'orario didattico, la compilazione dei bollettini automatica, la prenotazione dei posti in biblioteca e molto altro.

Infoprof, invece, è l'app che si rivolge ai professori della Sapienza. Nata nel 2019, vuole essere uno strumento alternativo, e più intuitivo, al sistema web. La ricerca della semplicità di utilizzo è la ricetta che accomuna tutti i progetti nel laboratorio ed ha bisogno di un'attenta progettazione per essere raggiunta. Spesso infatti ci si affeziona ad una interfaccia senza coglierne i problemi di usabilità. Ciò significa che bisogna sempre mettersi in discussione attraverso lo studio dei bisogni dell'utente, chiamato *need finding*, la realizzazione di prototipi e il confronto con gli utenti finali tramite test di usabilità, con eventualmente molteplici iterazioni.

Lo stato dell'arte è un'app che permette di verbalizzare i voti degli studenti, attivare gli OPIS e cercare le aule. Queste funzionalità hanno un denominatore

in comune: sono casi d'uso in cui l'utente potrebbe non avere il computer quando ha bisogno di utilizzarle. La verbalizzazione infatti può essere fatta subito dopo un orale, senza dover aspettare di tornare in ufficio e registrare i voti di tutti gli studenti. L'OPIS, date le ultime disposizioni, deve essere attivato durante la lezione, ma un utente potrebbe non voler portare un computer in aula solo per attivare il codice OPIS.

Il tema principale del mio tirocinio è l'aggiunta della funzionalità di apertura di un verbale in Infoprof. Soprattutto all'inizio, però, mi sono occupato di risolvere dei bug e aggiungere alcune funzionalità minori ad entrambe le applicazioni. Nello specifico, in InfoStud ho lavorato a (immagine 1.1):

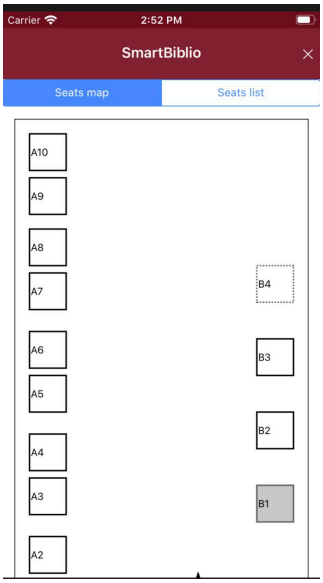
- L'aggiunta dei posti non gestiti dal sistema SmartBiblio. Prima venivano indicati solo come posti occupati;
- Risolto un bug in cui, nella pagina riguardante le aule, la barra di ricerca non riceveva alcun input;
- Nella pagina di login, il reset della password quando il login fallisce e l'aggiunta della funzionalità di "Ricorda password". Mentre la matricola ora viene sempre sempre salvata;
- L'aggiunta dell'opzione, nell'orario delle lezioni, per seguire dei corsi che non sono nella lista degli esami prenotabili;
- La darkmode, che prima utilizzava un nero tendente al grigio ed ora invece usa un nero puro. Questo per risparmiare batteria sui dispositivi dotati di schermo OLED;
- L'aggiunta della foto del profilo dello studente.

Mentre su Infoprof mi sono occupato:

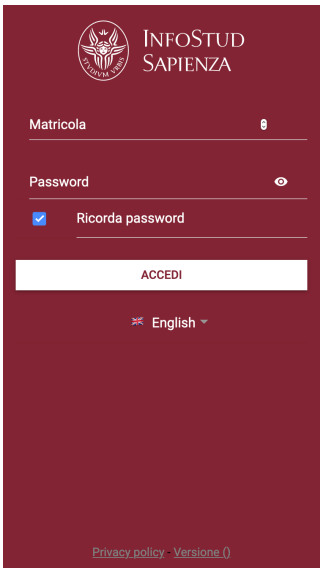
- Di migliorare la registrazione dei voti degli studenti, in particolare aggiungendo dei vincoli alla data di registrazione e dei default;
- Al reset della password nel caso in cui il login fallisce;
- Di migliorare il sistema di traduzione, che prima non coinvolgeva adeguatamente tutti i componenti dell'applicazione;

- E infine di vari ed eventuali piccoli errori.

Questa attività iniziale è stata utile per diversi fattori: innanzi tutto per conoscere entrambi i progetti che sono di medio-grandi dimensioni e richiedono quindi un po' di tempo, inoltre per approfondire le tecnologie alla base di Ionic, per me nuove, e infine per collaborare insieme agli altri colleghi. Solo nell'ultimo periodo invece mi sono dedicato full-time all'apertura del verbale, nel quale però ho svolto un'attività di progettazione molto più approfondita. Come dicevo prima, infatti, il processo non è immediato. Durante il tirocinio ho svolto delle interviste/sondaggi agli utenti finali (cioè i docenti) per catturare quanti più dettagli possibili sull'utilizzo che non conoscevo, individuando così i *needs* degli utenti. In seguito ho creato iterativamente i prototipi analizzandoli con dei test di usabilità con gli utenti oppure grazie a delle euristiche che descriverò più avanti. Lo step finale è stato quello dell'implementazione, grazie al quale oggi la funzionalità è effettivamente funzionante.

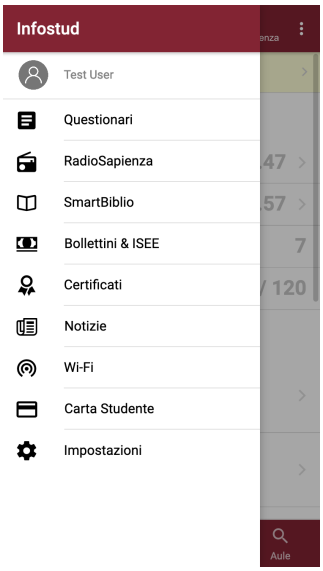


(a) Il posto tratteggiato indica un posto non gestito.

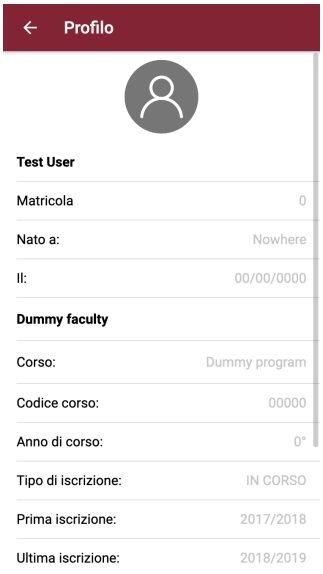


(b) L'opzione per ricordare la password.

(c) La darkmode, ora più scura.



(d) Immagine del profilo con nome nel menù laterale.



(e) Immagine del profilo nella pagina del profilo utente.

Figura 1.1. Alcune delle issues a cui ho lavorato.

1.2 Perché?

Lo smartphone è ormai uno strumento radicato nella nostra quotidianità. Sono innumerevoli i modi in cui ha cambiato la nostra vita e la produttività è uno degli aspetti centrali. Nel settore pubblico l'evoluzione dei sistemi informatici però pecca di una rigorosa analisi progettazione, il che comporta perdite di tempo e difficoltà nell'utilizzo. Dall'indagine preliminare che ho svolto infatti emerge che alcuni intervistati trovano l'interfaccia web *error-prone*, mentre un professore assunto recentemente, ha dichiarato che la piattaforma non è affatto user-friendly e che la logica vorrebbe che ci fosse un tasto "crea appello", non che debba andarlo a cercare in "verbalizzazione". Anche all'interno del team di sviluppo, quando ci è stata presentata per la prima volta l'interfaccia, abbiamo faticato a capire come svolgere i task che un professore è tenuto normalmente a fare. Tolti errori grossolani come l'assegnazione di termini differenti per rappresentare lo stesso concetto, esistono delle criticità che vanno contro la logica comune: se andiamo ad osservare la lista degli insegnamenti, alla destra di ogni elemento possiamo osservare un numero che rappresenta la quantità di *corsi* selezionati accoppiati con quell'insegnamento. Se si clicca il checkbox dell'elemento e si va all'interno del sotto-menù, deselezionando tutti i corsi, l'insegnamento rimarrà selezionato. Una buona interfaccia, invece, dovrebbe essere coerente, efficiente e aiutare l'utente ad evitare gli errori.

Capitolo 2

Metodologie di sviluppo

2.1 Organizzazione

Il team è composto da studenti impiegati nello sviluppo per circa tre mesi. La prassi è iniziare ad ambientarsi nei progetti attraverso l'attività di bug-fixing: familiarizzare con il codice e con i colleghi è centrale dato che per molti di noi è la prima esperienza all'interno di un progetto di tali dimensioni.

La metodologia di sviluppo che si adatta meglio alle nostre esigenze è l'Agile, il quale si basa su un insieme di regole il cui senso generale è un approccio adatto ai cambiamenti, non fisso e rigoroso, che misura i progressi attraverso codice funzionante e incoraggia il rilascio continuo del software per soddisfare l'utente finale.

Un aspetto fondamentale che si ricollega a questo è l'integrazione continua del codice prodotto dagli sviluppatori: nel laboratorio utilizziamo GitLab, un sistema di controllo versione che tiene traccia delle modifiche, segnala i conflitti che possono generarsi dalla modifica simultanea dello stesso file e aiuta a gestire i compiti di ogni singolo sviluppatore. Ad ogni *merge request*, le modifiche apportate al codice sono revisionate da un controllo umano che garantisce la qualità del codice. Se infatti lo stile non è coerente con il resto del progetto, se ci sono bug introdotti dalle modifiche o altri problemi, la merge request viene respinta.

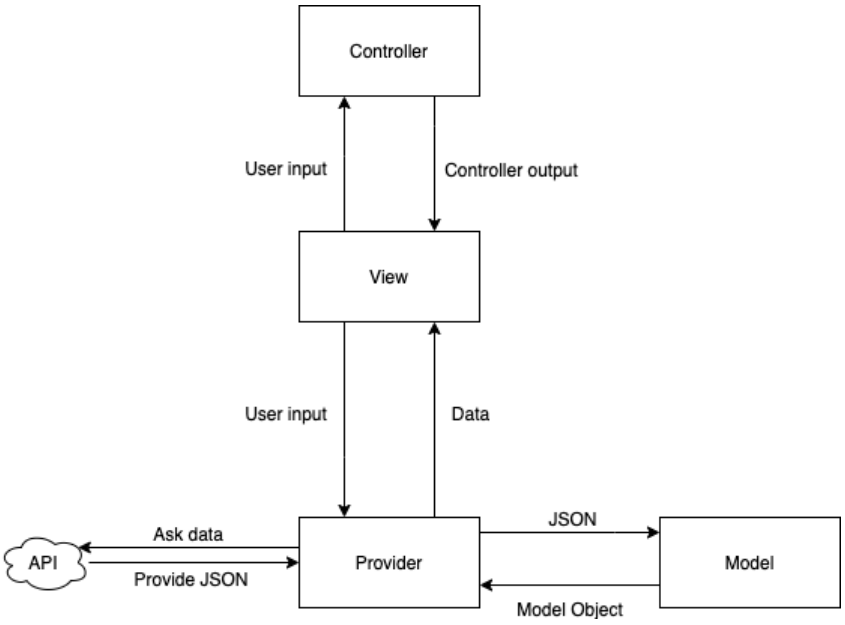
2.2 Tecnologie utilizzate

Per sviluppare il front-end delle applicazioni utilizziamo Ionic Framework v3, che è basato su AngularJS. Ionic permette di utilizzare tecnologie web per creare interfacce secondo gli standard di Android e iOS. Il vantaggio dell'uso di Angular è che il codice non va adattato ad alcuna piattaforma, inoltre per chiunque abbia già una discreta conoscenza della programmazione web, risulta immediato iniziare a programmare. Tutto ciò velocizza molto la scrittura di una UI, ma bisogna fare dei compromessi: l'interfaccia non segue diligentemente gli standard imposti dal Material Design o dall'iOS UI Design perché deve adattarsi a entrambe le piattaforme. Un'altra criticità è che di fatto l'app è un'interfaccia web con caratteristiche *app-like*, ovvero una Progressive Web App come Google le definisce [1]. Questo spesso va a peggiorare significativamente le prestazioni.

InfoStud ed InfoProf sono progettati seguendo un pattern architetturale la cui idea generale è quella di tenere separate le responsabilità di ogni componente. Ciò si traduce sinteticamente in:

- Model: rappresentano le classi, contengono i dati dell'applicazione a cui si accede attraverso i metodi che forniscono;
- View: la view accede ai dati forniti dai modelli e li presenta all'utente. Può comunicare con i controller, i provider e i model;
- Controller: il controller risponde alle azioni dell'utente intercettate dalla view e si occupa soltanto della logica di business, a volte comunica con i provider;
- Provider: di fatto "fornisce" i dati. Ha il compito di accedere alle API ed aggiornare i model.

Figura 2.1. Schema del funzionamento del pattern utilizzato da InfoStud e InfoProf.



I vantaggi dell'utilizzo di questo modello è quello di separare la logica di presentazione, quella di business e quella di accesso ai dati in entità distinte tra loro in modo da favorire il lavoro di gruppo. Inoltre, essendo Angular un framework che impone una separazione in moduli del codice [2], una divisione così fatta ne incoraggia il riuso.

Capitolo 3

Progettazione

L'attività di progettazione consiste nel trovare i bisogni dell'utente attraverso delle domande rivolte agli stessi, costruire un prototipo che rispecchi ciò che è emerso dalle risposte, eseguire dei test di usabilità sul prototipo e, a seconda delle criticità emerse dai test, modificare il prototipo e ripetere i test finché si è trovata un'interfaccia che funzioni per gran parte degli utenti.

3.1 Need-finding

La prima cosa che ho fatto è capire quali sono i bisogni dell'utente. Non avendo mai utilizzato InfoStud docenti, il primo approccio al sistema è stato insieme al Prof. Panizzi che mi ha mostrato il processo di apertura di un verbale d'esame, le varie funzionalità già esistenti e i dati richiesti nel form che il docente è tenuto a compilare. Da questa dimostrazione sono emersi dei problemi: per un utente inesperto come me è difficile prendere confidenza con l'interfaccia in quanto, ad esempio, vengono utilizzati diversi termini per rappresentare lo stesso concetto, c'è una lista molto grande di insegnamenti da poter selezionare e bisogna prestare attenzione a quali sono quelli che si selezionano perché in alcuni casi il sistema potrebbe restituire un errore. Inoltre l'interfaccia non spiega come inserire il giorno dell'esame (quello che effettivamente gli studenti vedono in fase di prenotazione), bisogna chiederlo a chi già ne è a conoscenza.

A partire da questa premessa, ho creato un sondaggio inviando via email le seguenti domande:

1. Preferisce aprire un verbale unico o aprirne uno per ogni corso che tiene?

2. Trova confusionario il processo di apertura di un verbale? Perché?
3. Quanto tempo prima dalla data dell'esame apre il verbale?
4. Quanto tempo dà agli studenti per prenotarsi?
5. E quanti giorni prima, dal giorno dell'esame, chiude le prenotazioni?

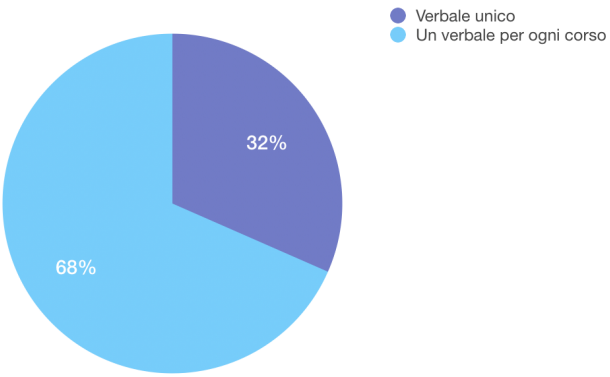
Il sondaggio ha solo una domanda aperta, tra l'altro opzionale, in quanto volevo renderlo molto corto in modo da aumentare le possibilità di risposta. Per coloro che, invece, avessero voluto dedicarmi più tempo ho chiesto di argomentare per avere risposte più complete.

Gli intervistati sono 60, presi casualmente in vari dipartimenti dell'ateneo in modo di avere un campione più eterogeneo possibile. I professori che hanno invece risposto sono stati 19.

3.1.1 Verbale unico o uno per ogni corso?

Come vediamo dall'immagine 3.1 la maggioranza degli intervistati preferisce aprire un verbale per ogni insegnamento. Ricordo che "un insegnamento", nella tabella, corrisponde a tutti gli insegnamenti vecchi che hanno cambiato nome e per ognuno esistono due righe: una per l'anno accademico corrente ed una per quello precedente.

Figura 3.1. Grafico delle risposte date alla prima domanda.



3.1.2 Trova confusionario il processo di apertura di un verbale?

Qui i risultati sono un po' sorprendenti perché la maggioranza sostiene di non avere problemi, sebbene comunque una percentuale non trascurabile abbia risposto il

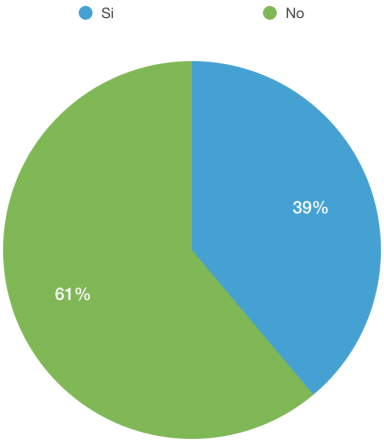
contrario. Ciò potrebbe essere imputabile al fatto che per molti anni gli utenti sono stati abituati a questo sistema ed hanno imparato ad utilizzarlo agevolmente, con degli escamotage. Uno fra tutti è quello di utilizzare la funzionalità del "Duplica appello", che di fatto copia un verbale creato in precedenza portandosi dietro tutte le informazioni già compilate.

Inoltre alcuni dei docenti che hanno risposto "no" alla domanda, si sono espressi anche sul perché. Cito:

- "Il processo di apertura di un verbale più che confusionario è piuttosto "error-prone" con campi che si riempiono automaticamente o campi che scompaiono quando si effettua una scelta da un menù a tendina."
- "Ho difficoltà a mettere gli appelli di esame perché non so mai per quali corsi debba essere valido l'esame. E sovente gli studenti non vedono l'appello o meglio alcuni lo vedono e altri no."
- "La scelta degli insegnamenti è caotica perché ci sono nomi ripetuti con diverse diciture e lo stesso codice (non parliamo poi dei corsi di studio associati, che non apro neppure); l'anno accademico non serve a niente e complica l'immissione anche perché crea il refresh dell'elenco materie; oltre ad "avvisi e note" bisognerebbe poter inserire il luogo dell'esame e l'orario o i vari appelli, se più d'uno."
- "Un problema che trovo riguarda la scelta dell'A.A di riferimento. Vorrei che non esistesse questa opzione."

Ho avuto inoltre la fortuna di intervistare un docente assunto recentemente, che mi ha esplicitamente detto che ha avuto grosse difficoltà nell'aprire il suo primo verbale. Nessuno gli ha spiegato come fare, e che quindi ha impiegato molto tempo a cercare le funzionalità che gli servivano.

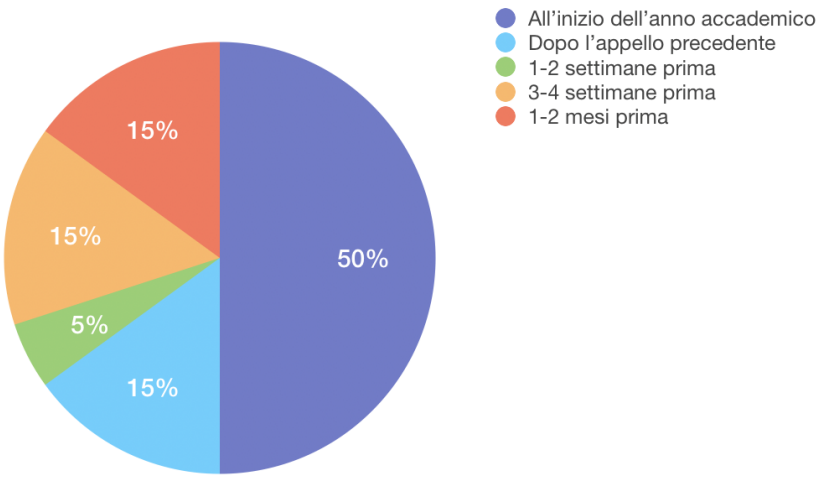
Figura 3.2. Grafico delle risposte date alla seconda domanda.



3.1.3 Quanto tempo prima dalla data dell'esame apre il verbale?

La metà degli intervistati ha dichiarato di aprire i verbali all'inizio dell'anno accademico. Infatti da quello che ho potuto constatare, molti dipartimenti impongono questa condizione ai docenti, quindi è necessario velocizzare il tempo di apertura di un singolo verbale perché aprirne sette per ogni insegnamento potrebbe essere un'attività che porta via diverso tempo.

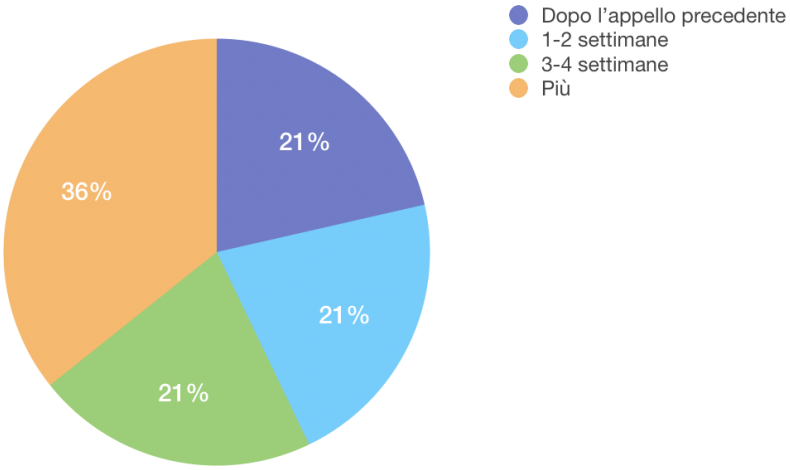
Figura 3.3. Grafico delle risposte date alla terza domanda.



3.1.4 Quanto tempo dà agli studenti per prenotarsi?

Qui le risposte sono piuttosto eterogenee.

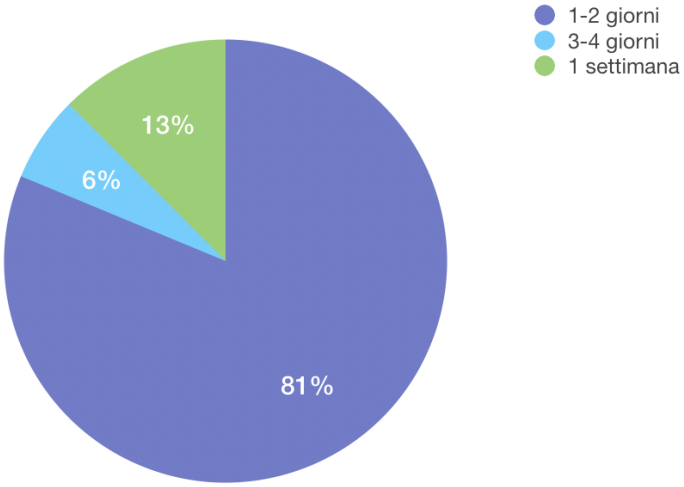
Figura 3.4. Grafico delle risposte date alla quarta domanda.



3.1.5 E quanti giorni prima, dal giorno dell'esame, chiude le prenotazioni?

Al contrario qui la maggior parte dei docenti preferisce chiedere le prenotazioni uno o due giorni prima dal giorno dell'esame.

Figura 3.5. Grafico delle risposte date alla quinta domanda.



3.2 Requisiti

Abbiamo visto che una buona percentuale dei partecipanti non è contenta dell'interfaccia desktop e mi ha dato degli spunti su cui lavorare, come l'aggiunta di orario e luogo d'esame. Inoltre sapere che diversi dipartimenti impongono l'apertura dei verbali ad inizio anno accademico è una delle motivazioni per cui è importante velocizzare il tempo di apertura di un verbale. Per raggiungere questo scopo ho pensato che una delle strade percorribili fosse quella di togliere un po' di informazioni su schermo e di non costringere l'utente a compilare tutti i campi del form. Dato che è impensabile eliminarli del tutto, si potrebbe creare un sistema di compilamento automatico e sintetizzare in qualche modo la tabella degli insegnamenti togliendo quegli elementi di ripetizione come l'anno accademico.

Dalle domande riguardanti le prenotazioni possiamo inoltre notare che, se da una parte le risposte sono state omogenee, per quanto riguarda invece il numero di giorni di prenotazione si osservano delle risposte molto frammentate. Quindi dare degli standard nella compilazione automatica (nel caso delle date) risulta essere poco coerente.

A partire dalle considerazioni fatte e dai problemi riscontrati sul sistema originale, ho stilato una lista di requisiti ad alto livello che guideranno tutte le iterazioni future:

1. L'ambiente in cui l'app funzionerà per la maggior parte del tempo è su uno smartphone quindi, essendo uno schermo piccolo, è necessario togliere le informazioni superflue.

2. L'apertura di un verbale deve avvenire nel minor tempo possibile, in modo da velocizzare l'apertura consecutiva di più verbali.

3. L'interfaccia dovrebbe essere ragionevolmente semplice da capire ed utilizzare, senza aver bisogno di una spiegazione.

3.3 Iterazioni

3.3.1 Valutazione dell'interfaccia

Prima di cominciare a parlare della fase di prototyping, introduco le tecniche di valutazione che ho utilizzato. Queste tecniche testano l'usabilità e le funzionalità di

un'interfaccia, in modo da valutare gli effetti che ha sull'utente ed eventualmente i problemi di usabilità. Per essere eseguite richiedono un artefatto che può essere sia un prototipo costruito con dei tool appositi, che un'implementazione del codice. In questo caso procederò ad iterare su un prototipo nella fase iniziale, per poi implementarne direttamente le funzionalità applicando le tecniche di valutazione a fine di ogni iterazione.

Inizialmente ho valutato il prototipo attraverso dei test di usabilità "sul campo", con la partecipazione di rappresentati scelti a caso dagli utenti stessi. Essi prevedono di avere un prototipo funzionante da far provare agli utenti, i quali durante il test descrivono ciò che stanno facendo (think aloud [3]). Nei casi in cui questo test produce poche informazioni ho introdotto la variante del *cooperative evaluation* che permette sia all'utente che al valutatore di fare domande o discutere dell'interfaccia [4]. A causa di forza maggiore ho dovuto abbandonare questa via per fare spazio alle euristiche.

Le euristiche sono delle regole, o linee guida, utilizzate per valutare o guidare la progettazione delle interfacce grafiche. Per il mio progetto di tirocinio ho utilizzato le euristiche di Nielsen [5] che riporto qui di seguito:

1. Visibilita dello stato del sistema
2. Corrispondenza tra il mondo reale e il sistema
3. Liberta e controllo da parte degli utenti
4. Coerenza e standard
5. Prevenzione degli errori
6. Riconoscere piuttosto che ricordare
7. Flessibilità ed efficienza d'uso
8. Design minimalista ed estetico
9. Aiutare gli utenti a riconoscere, diagnosticare e correggere gli errori
10. Aiutare e documentare

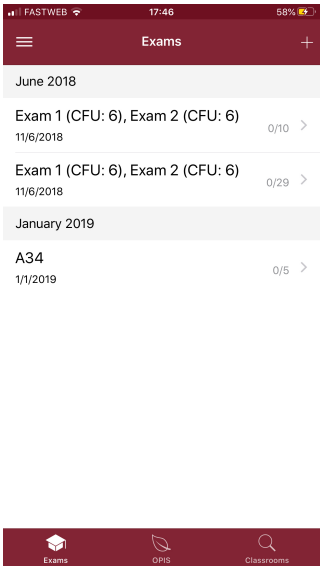
3.3.2 Iterazione 1

L'apertura di un verbale è una delle funzionalità base del sistema InfoStud, per questo è importante che venga messa in prima pagina proprio dove viene mostrata la lista degli esami aperti in passato. Un bottone con il simbolo "+" in alto a destra suggerisce l'aggiunta di un nuovo elemento alla lista. Se cliccato, l'applicazione propone una la lista degli insegnamenti tenuti dal professore e un'altra di *alias*. Gli alias sono etichette che si applicano ad uno o più insegnamenti in modo che se viene selezionato, allora vengono selezionati anche tutti gli insegnamenti ad esso collegati. Questo meccanismo va a sostituire quello che fa il "Duplica appello" di InfoStud desktop.

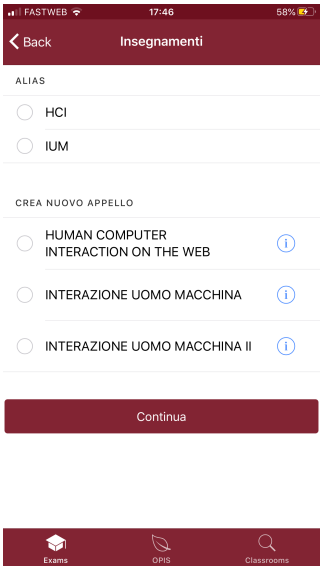
Inoltre accanto ad ogni insegnamento c'è un'icona blu "i" che se cliccata porta ad una nuova vista che mostra i corsi di studio in cui l'insegnamento è attivo. Questi elementi sono selezionabili e quando si clicca sul bottone "Continua" si arriva infine al form che richiede l'inserimento delle date di inizio e fine appello, inizio e fine prenotazioni, la necessità dell'OPIS ed eventualmente le note per gli studenti. La navigazione, così impostata, va a semplificare leggermente il form dividendolo in due pagine e quindi riducendo il carico di informazioni su una singola view. Il prototipo è visibile in figura 3.6.

Un altro elemento è quello del compilamento automatico delle date. L'idea è quella di selezionare la data di inizio dell'appello e l'app, ricordando le date che sono state utilizzate precedentemente, prevede quali saranno quelle che l'utente inserirà, facendolo per lui. Infatti se dalle interviste è emerso che i professori tendono a frammentarsi nel scegliere il numero di giorni dedicati all'appello e alle prenotazioni, d'altro canto la maggior parte è costante e utilizza sempre quelli. Dunque far ricordare all'applicazione le scelte fatte dall'utente in precedenza può essere una soluzione percorribile e dovrebbe velocizzare di molto l'apertura di un verbale.

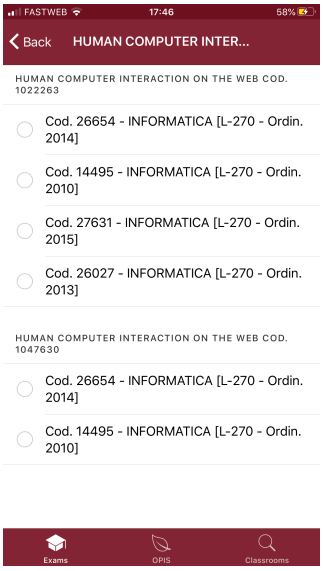
Nei test, ogni intervistato è sempre riuscito ad aprire il verbale senza difficoltà nell'ordine di pochissimi minuti. C'era qualche incertezza nel campo note, dove "Note per gli studenti" veniva spesso confuso con un placeholder. In pochi si sono interessati del tasto "i" accanto agli insegnamenti. Inoltre non avere nel form un'indicazione delle scelte fatte dall'utente nel corso della navigazione può essere una fonte di errore, quindi serve un modo per mostrare all'utente le scelte fatte nel corso della navigazione.



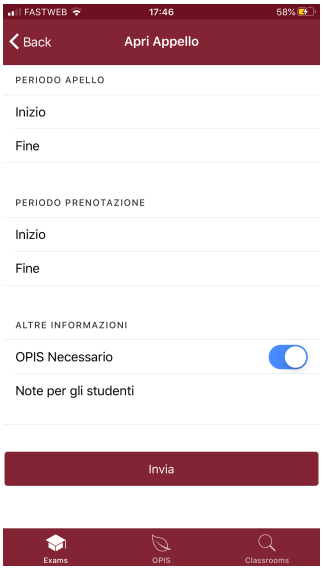
(a) La prima pagina dell'applicazione InfoProf.



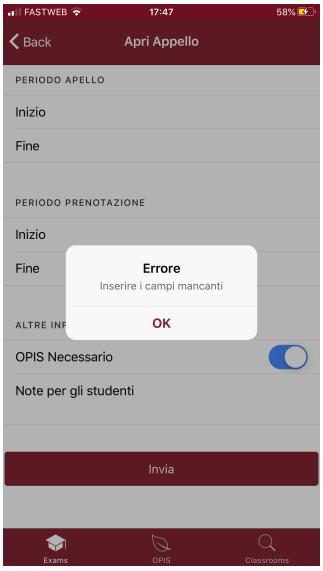
(b) Pagina di selezione degli insegnamenti.



(c) Pagina per la selezione dei corsi di studio.



(d) Il form per inserire le informazioni del verbale.



(e) Errore mostrato quando un utente non inserisce tutti i campi.

Figura 3.6. Prototipo della prima iterazione. I dati nelle immagini sono esemplificativi.

3.3.3 Iterazione 2

Il problema più grande dell'iterazione precedente è che la prima e la seconda view (nella quale si inseriscono le date) non sono legate in alcun modo, questo costringe il docente a ricordare le scelte fatte nella pagina precedente. Dato che la lista può essere complicata da gestire ed è importante aiutare l'utente a non fargli commettere errori, ho deciso di cambiare l'ordine della navigazione: un collegamento alla prima pagina infatti è stato messo all'interno del form, con un bottone "Scegli insegnamenti" che porta alla view della scelta degli insegnamenti e degli alias. Quando questi vengono selezionati, si torna indietro e sopra al bottone di prima compaiono gli insegnamenti scelti.

Altre modifiche all'interfaccia sono state quelle relative ai titoli delle navbar, che ora sono più esplicativi. Sono stati aggiunti dei messaggi di errore e il campo di testo ora è più chiaro ed ha un placeholder. In questa iterazione ho inoltre collegato le view al backend, in modo da lavorare su dei dati che strutturalmente rappresentassero i dati reali. Da qui ho implementato il meccanismo per sintetizzare la tabella degli insegnamenti, in cui parlerò più approfonditamente nella sezione 3.4. Per ora basti sapere che, mentre nell'interfaccia desktop abbiamo almeno due copie per ogni insegnamento (vecchio o presente che sia) [1], nell'app questo non avviene perché saranno racchiusi in una voce sola. Sarà poi l'app, in modo totalmente trasparente all'utente, a decidere quali corsi e insegnamenti andranno inviati al server.

Anche la grafica è cambiata, rendendola più simile a quella dell'aggiunta di un appuntamento sul calendario di iOS e Google. Non potendo fare più i test di usabilità, che hanno bisogno di essere fatti in presenza, mi sono affidato agli standard che hanno applicazioni più importanti. Infatti l'aggiunta di un appuntamento su un calendario non è tanto differente dall'apertura di un verbale, solo che bisogna gestire tipi differenti di dati.

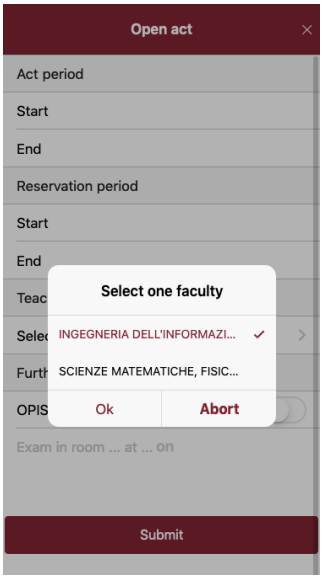
Infine, invece di utilizzare delle view, ho deciso di usare dei modali per far sparire i tab alla base delle pagine per impedire all'utente di navigare all'interno dell'applicazione dopo aver iniziato l'apertura di un verbale. Infatti è ragionevole pensare che egli non voglia farlo, semplicemente perché ho fatto in modo che tutto ciò che gli serve per aprire un verbale è visibile durante la navigazione. In più questo

[1] Gli insegnamenti infatti hanno tutti una voce per l'A.A. corrente e quello precedente.

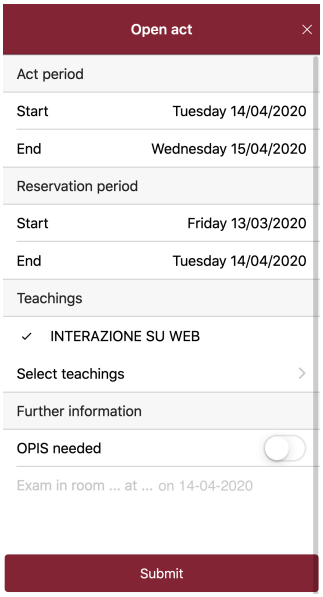
permette di risparmiare un po' di spazio per evitare lo scroll. Nell'immagine 3.7 è possibile vedere tutte le pagine.

Come anticipato prima 3.3.1, valuterò questa iterazione tramite le euristiche di Nielsen. Possiamo dire che l'interfaccia, arrivata a questo punto, le soddisfa almeno per buona parte. Per quanto riguarda la prima regola, "Visibilita dello stato del sistema" è soddisfatta da tutti gli elementi tramite dei titoli esplicativi e pulsanti ben visibili. La seconda, "Corrispondenza tra sistema e mondo reale", è soddisfatta pienamente anche lei in quanto i termini "Appello", "Corso", "Insegnamento" e "Facoltà" sono gli stessi utilizzati nel gergo comune, inoltre non essendo cambiati dall'iterazione precedente ed avendo fatto dei test, si può assumere che non siano fonte di ambiguità. Anche "Controllo e libertà" viene rispettata tramite l'aggiunta degli alias, che permettono una scorciatoia agli utenti esperti e la scelta dei corsi dà una certa libertà nell'apertura di un appello. L'uniformità dei termini usati garantisce la "consistenza"; mentre la selezione delle date, che deve rispettare determinati vincoli [2] ed è fatta automaticamente, unita al collasso degli insegnamenti in meno voci all'interno della tabella, garantiscono la prevenzione degli errori, l'efficienza d'uso ed un design minimalista. La regola "riconoscere invece che ricordare" è stata già discussa ad inizio sezione anche se, quando gli alias sono vuoti, non c'è alcuna indicazione su cosa sono e come vengono creati. Per quanto riguarda gli errori, per ora ci sono solo quelli che vengono lanciati quando l'utente preme "ok" senza selezionare una facoltà all'interno dell'alert e quando clicca su "invia" senza aver messo le date e almeno un insegnamento. Come si vede dalle immagini, però, sono stati creati ad arte per essere ben diagnosticabili. Le condizioni per cui un utente può lanciare un errore è solo in questi due casi comunque, quindi sono veramente poche. Vedremo nella terza iterazione come sono riuscito ad abbassare le condizioni a solo un caso, cioè quello dei campi obbligatori mancanti.

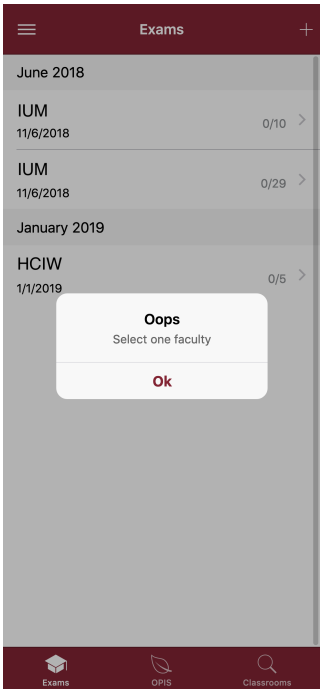
[2] Non è possibile selezionare date nel passato e non devono accavallarsi tra di loro.



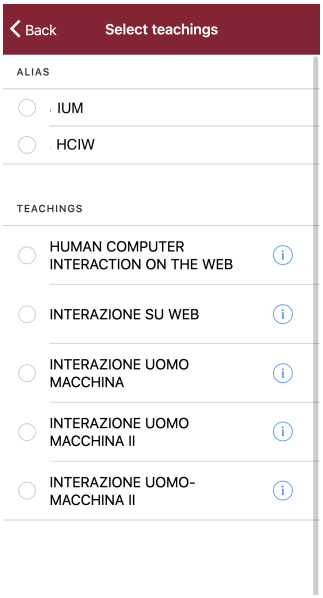
(a) Alert per la selezione della facoltà.



(b) Form con le date inserite ed un insegnamento selezionato.



(c) Messaggio di errore.



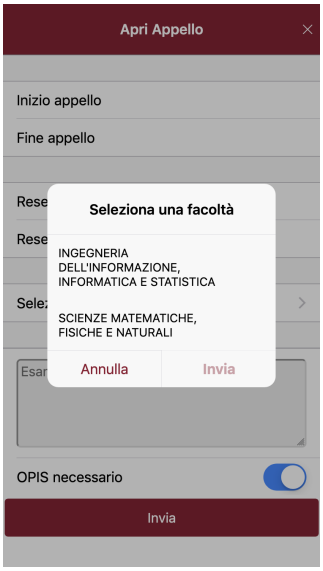
(d) Lista degli insegnamenti.

Figura 3.7. Seconda iterazione.

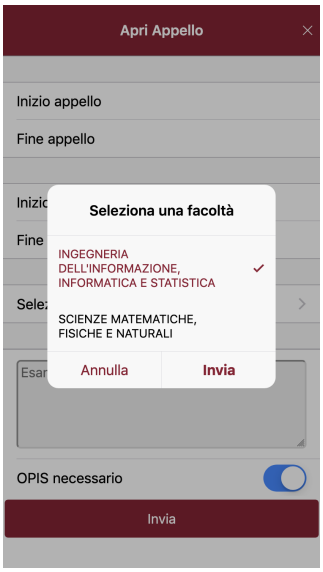
3.3.4 Iterazione 3

La scelta dei corsi di studio è un componente che ho voluto tenere fino ad ora per non togliere libertà di scelta all'utente. Da quanto è emerso nelle interviste e nei test, però, è un elemento che crea confusione e gli utenti sono riluttanti nell'usare tale strumento. Pertanto in questa iterazione abbiamo deciso di oscurare i pulsanti per accedervi e verificare se fosse una funzionalità richiesta. Ma se l'ipotesi che non serva fosse corretta allora avremmo tolto una fonte di distrazione e avremmo alleggerito l'interfaccia.

Altri miglioramenti sono stati apportati all'alert per la scelta della facoltà dove ora il bottone per confermare la facoltà è disabilitato finché non se ne seleziona una (questo elimina un triggher per gli errori). Inoltre le etichette agli input del form sono cambiate, adesso quelle per ogni singolo elemento sono più esplicative e ho tolto i titoli nelle varie sezioni che occupavano solo spazio. Anche il textbox ora è più grigio in modo tale da essere ben visibile all'utente. Infine è stato aggiunto un messaggio che inviti l'utente ad utilizzare gli alias e il perché esistono. Nell'immagine 3.8 si può vedere l'interfaccia finale.

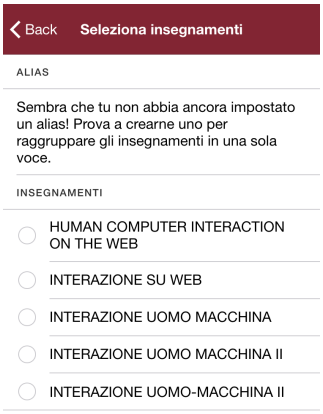


(a) Alert di selezione facoltà con bottone "Invia" disabilitato.



(b) Alert di selezione facoltà, con facoltà selezionata.

(c) Grafica del form aggiornata.



(d) Lista degli insegnamenti aggiornata.

Figura 3.8. Terza iterazione

3.4 Implementazione

Ottenuta l'interfaccia e valutate le sue funzionalità, adesso spiegherò come ho implementato il tutto. In questa sezione andrò più nel dettaglio tecnico di ogni pagina e spiegherò i ruoli che ogni componente ha.

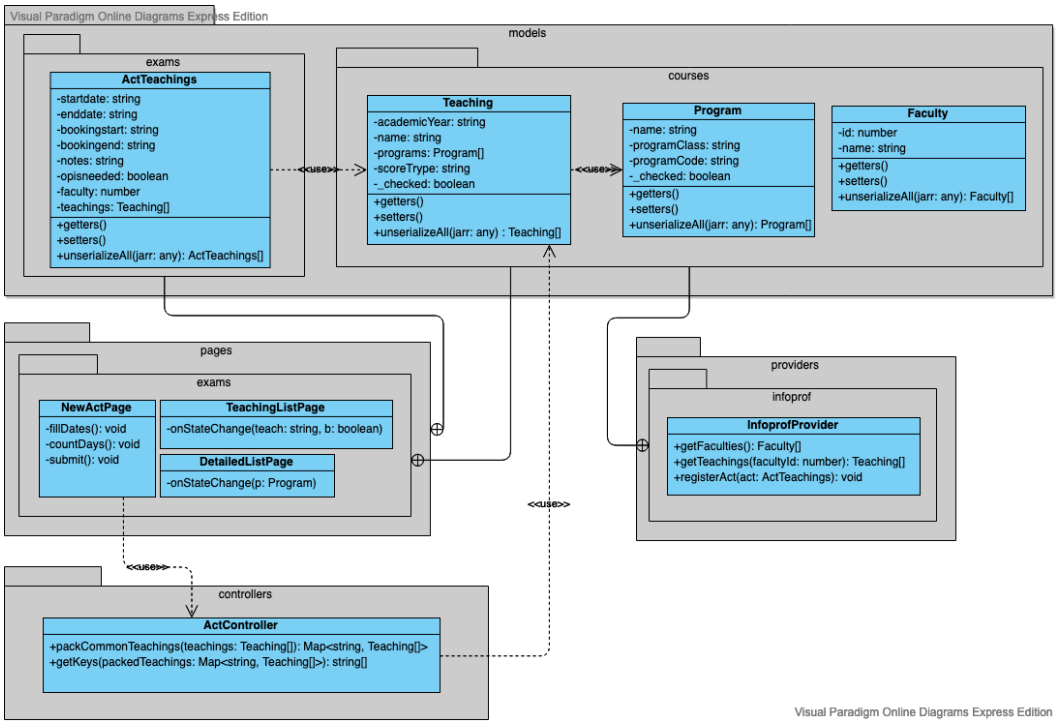
3.4.1 Le classi

Per realizzare ciò che è stato detto nella sezione precedente, ho bisogno delle seguenti classi:

- **ActTeachings**: Rappresenta un appello;
- **Teaching**: Rappresenta un insegnamento;
- **Program**: Rappresenta un corso di studi;
- **Faculty**: Rappresenta una facoltà;
- **ActController**: Ha il compito di raccogliere gli insegnamenti con lo stesso nome in una sola voce;
- **NewActPage**: Rappresenta la UI del form;
- **TeachingListPage**: Mostra la UI per la selezione degli insegnamenti;
- **DetailedListPage**: Mostra la UI per la selezione dei corsi di studio collegati ad un insegnamento;
- **InfoprofProvider**: Classe che accede ai dati del server.

Ho dunque realizzato il diagramma delle classi visibile in figura 3.9.

Figura 3.9. Diagramma delle classi.



3.4.2 InfoprofProvider

InfoprofProvider è la classe che si occupa di utilizzare le API per accedere ai dati nel server, quando questi metodi sono dei "getter" il server risponde con dei dati in formato JSON che andranno deserializzati con gli appositi metodi che ogni classe di tipo "entity" fornisce nel costruttore o, se trattiamo un array di JSON, allora con il metodo uniserializeAll() fornito sempre dalle classi entity. La funzionalità di apertura di un verbale ha dunque bisogno di comunicare in 2 occasioni con il server, per un totale di 3 chiamate: la prima e la seconda vengono chiamate quando l'utente accede al form di apertura del verbale e scarica prima le facoltà, **getFaculties()**, in cui l'utente lavora e poi gli insegnamenti corrispondenti alla facoltà scelta, con **getTeachings()**. L'ultimo metodo, **registerAct()**, invece viene chiamato quando bisogna registrare l'appello, ovvero quando l'utente clicca sul pulsante "Invia".

```
1 public registerAct(act: ActTeachings): Promise<void> {
2   if (this.professorId == "0000000") {
3     return new Promise((resolve, reject) => {
4       resolve();
```

```
    });
  }
  return new Promise<void>((resolve, reject) => {
    this.http.setDataSerializer("json");
    this.http.put(
      this.env.getInfoprofAPI() + "/acts/",
      act,
      this.getOptions()
    ).then(data => {
      this.accessToken = data.headers["x-access-token"] != null ? data.
          headers["x-access-token"] : this.accessToken;
      resolve();
    }).catch((err) => this.errorHandler(err, reject));
  });
}
```

```
public getFaculties(): Promise<Faculty[]> {
  return new Promise((resolve, reject) => {
    this.http.setDataSerializer("urlencoded");
    this.http.get(
      this.env.getInfoprofAPI() + "/me/acts/faculties/",
      {},
      this.getOptions()
    ).then(data => {
      this.accessToken = data.headers["x-access-token"] != null ? data.
          headers["x-access-token"] : this.accessToken;
      try {
        resolve(Faculty.unserializeAll(JSON.parse(data.data)));
      } catch (e) {
        console.error("Exception during parse: " + e.toString());
        reject(new ApiError(ApiError.SERVER_ERROR, e.toString()));
      }
    }).catch((err) => this.errorHandler(err, reject));
  });
}
```

3.4.3 ActController

Prima di mostrare come funzionano le pagine è opportuno presentare anche l'ActController che ha il compito di gestire gli insegnamenti in strutture dati diverse dall'array con cui ci vengono dati dal provider. I metodi più importanti sono **packCommonTeachings()** e **getKeys()**.

Il primo metodo si occupa di raccogliere tutti gli insegnamenti con lo stesso nome in un dizionario dove le chiavi rappresentano il nome dell'insegnamento e i

valori invece sono array contenenti gli insegnamenti che condividono quel nome. Infatti, possiamo vedere che gli appellativi che vengono dati rispettano uno schema: nella prima parte c'è il nome effettivo dell'insegnamento, mentre nella seconda ci sono le informazioni aggiuntive racchiuse tra parentesi. Ecco un esempio **HUMAN COMPUTER INTERACTION ON THE WEB (cod. 1047630 - INF/01 - cfu: 6, canale: NESSUNA CANALIZZAZIONE)** e **HUMAN COMPUTER INTERACTION ON THE WEB (cod. 1047630 - INF/01 - cfu: 6, canale: NESSUNA CANALIZZAZIONE)**. Il codice taglia la stringa tra le parentesi per ricavare la chiave e raccogliere tutte le stringhe che matchano con essa in un dizionario.

```
1  public packCommonTeachings(teachings: Teaching[]): Map<string, Teaching[]> {
2    // All teachings with the same names are collected under the same key.
3    let map: Map<string, Teaching[]> = new Map<string, Teaching[]>();
4
5    for (let teaching of teachings) {
6      let key = teaching.getName().slice(0, teaching.getName().indexOf("(") -
           1);
7      if (map.has(key)) {
8        map.get(key).push(teaching);
9      } else {
10       let t: Teaching[] = [teaching];
11       map.set(key, t);
12     }
13   }
14   return map;
15 }
```

All'interno del DOM è molto complesso lavorare con i dizionari, quindi avere un metodo che all'occorrenza estrae le chiavi in un array e le mostra all'utente è comodo e per questo ho scritto il metodo **getKeys()**.

```
1  public getKeys(packedTeachings: Map<string, Teaching[]>): string[] {
2    let keys: string[] = [];
3    packedTeachings.forEach((value, key) => {
4      keys.push(key);
5    });
6    return keys;
7  }
```

3.4.4 NewActPage

Passando invece alle pagine, NewActPage è la prima in ordine di navigazione e contiene il form nel quale inserire le varie informazioni necessarie al successo dell'operazione. Il primo metodo è **fillDates()** e, come il nome suggerisce, è il responsabile della compilazione automatica delle date. Il metodo in prima battuta controlla che non ci sia già una data in uno dei campi preposti, a parte actStart. Se la condizione è verificata, si occupa di compilare le date in automatico, altrimenti si ferma per non sovrascrivere quello che l'utente ha fatto in precedenza. Le variabili **actPeriod**, **emptyPeriod** e **reservationPeriod** rappresentano rispettivamente la differenza espressa in giorni tra la fine dell'appello e l'inizio, l'inizio dell'appello e la fine delle prenotazioni e la fine delle prenotazioni e l'inizio delle stesse (da questo momento chiamerò queste variabili **periods**). Questi valori vengono presi dallo storage interno del telefono nel costruttore della classe e assumono "-1" se non esistono valori precedentemente salvati. In quest'ultimo caso, la compilazione prenderà dei valori di default che sono un mese per le prenotazioni e l'appello, e 1 giorno di differenza trai due.

La seconda parte del metodo si occupa di decidere se le variabili periods andranno salvate nello storage. Prima di questo però bisogna tenere presente che ci sono dei vincoli che le date devono rispettare [3]:

1. Il minimo valore che può assumere resStart[4] è "oggi" (chiamato minResStart);
2. Il minimo valore che può assumere actStart[5] è resStart + 1 giorno (chiamato minActStart);
3. Il minimo valore che può assumere resEnd è minResStart;
4. Il massimo valore che può assumere resStart è actStart - 1 giorno (chiamato maxResStart);
5. Il massimo valore che può assumere resEnd è actStart - 1 giorno (chiamato maxResEnd);

A questo punto, se alla fine del calcolo, i valori assunti dalle date violano uno dei vincoli posti, il sistema va ad aggiustare le date interessate in modo che li rispettino

[3] Il DOM si occupa di controllare che tutti i vincoli siano soddisfatti.
[4] resStart è l'inizio delle prenotazioni, resEnd è la loro fine.
[5] actStart è l'inizio dell'appello, actEnd è la sua fine.

tutti. Se si verificasse questa condizione le variabili "periods" assumerebbero valori calcolati dal sistema e non dall'utente, quindi non vanno salvati nello storage ma rimangono quelli che c'erano precedentemente.

```
1  private fillDates() {
2    if (this.actEnd == undefined && this.resStart == undefined && this.resEnd
         == undefined) {
3      let actStart = moment(this.actStart);
4      if (this.actPeriod >= 0) {
5        this.actEnd = actStart.add(this.actPeriod, 'd').format("YYYY-MM-DD");
6      } else {
7        this.actEnd = actStart.add(1, 'M').format("YYYY-MM-DD");
8      }
9      actStart = moment(this.actStart);
10     if (this.emptyPeriod >= 0) {
11       this.resEnd = actStart.subtract(this.emptyPeriod, 'd').format("YYYY-MM-
             DD");
12     } else {
13       this.resEnd = actStart.subtract(1, 'd').format("YYYY-MM-DD");
14     }
15     let resEnd = moment(this.resEnd);
16     if (this.reservationPeriod >= 0) {
17       this.resStart = resEnd.subtract(this.reservationPeriod, 'd').format("
             YYYY-MM-DD");
18     } else {
19       this.resStart = actStart.subtract(1, 'M').format("YYYY-MM-DD");
20     }
21
22     actStart = moment(this.actStart);
23     this.maxResEnd = actStart.subtract(1, "d").format("YYYY-MM-DD");
24     this.maxResStart = this.maxResEnd;
25
26     if (moment(this.resStart).isBefore(this.minResStart)) {
27       this.resStart = this.minResStart;
28       this.savePeriods = false;
29     } else if (moment(this.resStart).isAfter(this.maxResStart)) {
30       this.resStart = this.maxResStart
31       this.savePeriods = false;
32     }
33     if (moment(this.resEnd).isBefore(this.resStart)) {
34       this.resEnd = this.resStart;
35       this.savePeriods = false;
36     } else if (moment(this.resEnd).isAfter(this.maxResEnd)) {
37       this.resEnd = this.maxResEnd;
38       this.savePeriods = false;
39     }
40   }
```

```
41 }
```

Questo metodo invece semplicemente calcola le variabili "periods" esattamente come ho spiegato poco sopra.

```
1  private countDays() {
2    let actStart = moment(this.actStart);
3    let actEnd = moment(this.actEnd);
4    let resStart = moment(this.resStart);
5    let resEnd = moment(this.resEnd);
6
7    this.actPeriod = actEnd.diff(actStart, 'd');
8    this.reservationPeriod = prenEnd.diff(prenStart, 'd');
9    this.emptyPeriod = actStart.diff(prenEnd, 'd');
10 }
```

Infine c'è submit, che viene chiamato quando l'utente clicca sul pulsante "Invia" e si occupa di controllare che tutti i campi obbligatori siano stati riempiti (la correttezza dei dati invece è già verificata dal DOM). Se lo sono, controlla se si sta aprendo un verbale dopo il 1 gennaio in modo tale da eliminare dalla selezione gli insegnamenti dell'anno accademico passato, poiché non più accettati dal sistema InfoStud desktop. Infine crea un'istanza della classe ActTeachings e chiama il metodo di InfoprofProvider **registerAct()** il quale invia l'appello appena creato al server. Al successo dell'operazione le preferenze sui periods vengono salvate nello storage del dispositivo.

```
1  private submit(isRetryCall?: boolean): void {
2    if (this.actStart != undefined && this.actEnd != undefined && this.
         prenStart != undefined && this.prenEnd != undefined && this.
         checkedTeachings.length > 0) {
3      let loader = this.loadingCtrl.create({
4        spinner: 'crescent',
5        content: this.txt["RECORD_IN_PROGRESS"],
6      });
7      loader.present();
8
9      // If the start of the act is after the 31-01 of the current academic
           year, the teachings of the previous
10     // academic year will be deselected automatically.
11     if (this.actStart > moment(moment().year() + "01" + "31").format("YYYY-MM
           -DD")) {
12       let checkedTeachings = []
13       checkedTeachings.push(...this.checkedTeachings)
14
15       for (let t of checkedTeachings) {
```

```
16          if (t.getAcademicYear().slice(t.getAcademicYear().indexOf("/") + 1) <
                moment().format("YYYY")) {
17            let i = this.checkedTeachings.indexOf(t);
18            if (i > -1) {
19              this.checkedTeachings.splice(i, 1);
20            }
21          }
22        }
23      }
24
25      this.ip.registerAct(
26        new ActTeachings({
27          "startdate": this.actStart,
28          "enddate": this.actEnd,
29          "bookingstart": this.prenStart,
30          "bookingend": this.prenEnd,
31          "notes": this.notes,
32          "opisneeded": this.opis,
33          "faculty": this.facultyId,
34          "teachings": this.checkedTeachings
35        })).then(() => {
36          // Save the user's preferencies before closing the modal.
37          this.countDays();
38          this.storage.set("actPeriod", this.actPeriod);
39
40          if (this.savePeriods) {
41            this.storage.set("reservationPeriod", this.reservationPeriod);
42            this.storage.set("emptyPeriod", this.emptyPeriod);
43          }
44          loader.dismiss();
45          this.events.publish('refreshPage');
46          this.navCtrl.pop();
47        }).catch((error: ApiError) => {
48          // Tutte le varie condizioni d'errore, che sono state omesse qui
49        });
50    } else {
51      let alert = this.alertCtrl.create({
52        title: this.txt["ERROR"],
53        subTitle: this.txt["INSERT_DATES_AND_TEACHINGS"],
54        buttons: ['OK']
55      });
56      alert.present();
57    }
58  }
```

3.4.5 TeachingListPage

Questa pagina si occupa, per l'appunto, di gestire la grande tabella degli insegnamenti. Come abbiamo visto nelle sezioni precedenti dedicate alla progettazione della UI, l'intento era di raccogliere tutti i duplicati che popolavano la tabella in un'unica voce. In fondo l'utente, da quel che ho potuto constatare, non è mai interessato a fare distinzioni tra anni accademici, corsi, insegnamenti vecchi che hanno cambiato nome o altro e quindi abbiamo assunto che la tabella espressa in modo estensivo serva solo in casi particolari. All'utente spetterà solo l'onere di selezionare per quali gruppi di insegnamenti l'appello dovrà essere aperto. Per l'appunto ho pensato di raccogliere questi gruppi in un dizionario, come visto nella sezione 3.4.3.

Il metodo **ionViewWillEnter()** fa parte del ciclo di vita delle pagine di Ionic. Viene chiamato quando la pagina sta per terminare la transizione da quella precedente [6]. Il compito che ha in questo caso è di far rispettare il seguente vincolo "Un insegnamento è selezionato se e solo se c'è almeno un corso al suo interno che è selezionato", vincolo che su InfoStud desktop non è rispettato. Come è possibile vedere infatti il codice mette a *true* la variabile *b* non appena trova un corso che è "checkato", altrimenti rimane in *false*. Questo metodo ha quindi il solo scopo di essere attivato quando la pagina da cui si proviene è **DetailedListPage** (sezione 3.4.6) e aggiornare la tabella coerentemente a come si sono manipolati i corsi nella precedente view.

```
1  ionViewWillEnter() {
2    this.packedTeachings.forEach((teachings) => {
3      let b = false;
4      for (let t of teachings) {
5        for (let p of t.getPrograms()) {
6          if (p.isChecked()) {
7            b = true;
8          }
9        }
10       t.setChecked(b);
11       this.checked[this.keys.indexOf(t.getName().slice(0, t.getName().indexOf
             ("(") - 1))] = b;
12     }
13   });
14 }
```

Questo metodo viene chiamato non appena un utente clicca su un insegnamento per selezionarlo. Quello che fa è mettere a *true* il valore booleano di ogni insegna-

mento che ha come chiave un elemento in *keys*, all'interno del dizionario *packedTeachings*. Inoltre si occupa di gestire gli Alias, che altro non sono che delle stringhe con tutti i nomi completi degli insegnamenti separati da una virgola. Quando all'argomento *teach* viene passato un alias, si fa uno split sulla virgola che separa ogni elemento e sulle stringhe risultanti si toglie la parte tra parentesi. A questo punto da *packedTeachings* si ricavano tutti gli insegnamenti collegati alle chiavi che devono essere selezionati. In questo modo, anche se l'input ha due formati differenti, *teach* viene normalizzato in un array *toCheck* che nel primo caso è composto da un solo elemento, mentre nel caso dell'alias da tanti elementi quanti sono quelli contenuti in esso.

```
1  private onStateChange(teach: string, b: boolean) {
2    // Check the teachings as 'b' and so ALL the programs contained in
3    // the teaching object must be checked as 'b'. If an alias is clicked,
4    // then check as 'b' each teaching contained in it and so all the programs.
5
6    let keys = teach.split(", ");   // splits the teachings when coming from an
         alias
7    let toCheck: Teaching[] = [];
8
9    for (let k of keys) {
10     if (k.indexOf("(") >= 0) {
11       k = k.slice(0, k.indexOf("(") - 1);
12     }
13     if (k.indexOf('Canale') >= 0) {
14       continue;
15     }
16     toCheck.push(...this.packedTeachings.get(k.toUpperCase()));
17   }
18
19   this.packedTeachings.forEach((teachings) => {
20     for (let t of teachings) {
21       if (t == toCheck[toCheck.indexOf(t)])
22       t.setChecked(b);
23       for (let p of t.getPrograms()) {
24         p.setChecked(b);
25       }
26     }
27   });
28 }
```

3.4.6 DetailedListPage

Sebbene questa pagina sia stata disabilitata, non abbiamo scartato il codice in quanto in futuro potrebbe tornare. Probabilmente è la view più semplice, perché di fatto sono le altre a occuparsi di tutto: se immaginiamo la struttura *packedTeachings* come un albero, questa pagina deve preoccuparsi di gestire solo le foglie, cioè i corsi. Saranno poi *TeachingListPage* e *NewActPage* a preoccuparsi di assegnare dei valori true o false agli insegnamenti, come visto nelle sezioni precedenti. Infatti l'unico metodo qui disponibile è *onStateChange()*, che viene attivato dal DOM non appena l'utente clicca su un corso, prendendo come argomento il corso interessato e cambiando il suo valore booleano in modo coerente.

```
1 private onStateChange(p: Program) {
2   p.isChecked() ? p.setChecked(false) : p.setChecked(true);
3 }
```

Capitolo 4

Conclusioni

Durante il mio tirocinio ho cercato di dare il meglio, dedicando molto tempo alla crescita delle due applicazioni risolvendo problemi e aggiungendo funzionalità che vedranno la luce presto. Ritengo che il lavoro fatto all'interno del Gamification Lab migliori, nel suo piccolo, il nostro ateneo e la comunità degli studenti e dei professori fornendo degli strumenti più moderni e facili da utilizzare costruiti dalla comunità stessa.

L'applicazione InfoProf è ancora in una fase iniziale del suo sviluppo e per questo sono stato felice di aver contribuito ad un progetto con una funzionalità base del sistema, la quale verrà utilizzata molto se ben realizzata. Chiaramente ci sono ancora dei problemi che vanno risolti, come ad esempio il fatto che il sistema non permette di strutturare in modo adeguato le date degli esami. Ragionando insieme infatti siamo arrivati alla conclusione che sarebbe opportuno progettare e sviluppare un modo per aggiungere un numero arbitrario di esami associati ad un appello. Con il sistema attuale il giorno dell'esame che gli studenti vedono in fase di prenotazione è determinato dalla data di inizio appello. In realtà però un appello può avere molteplici date d'esame: ci sono insegnamenti che hanno sia lo scritto che l'orale, hanno più date per gli orali o addirittura hanno più di uno scritto. Questo dato nel sistema attuale può essere rappresentato solo nel campo note oppure nei siti web dei rispettivi docenti, creando dati non strutturati e addirittura frammentati tra più piattaforme. Dunque grazie a questa funzionalità i professori potrebbero aggiungere le date, l'orario e anche l'aula in cui l'esame si svolgerà, tramite il sistema di ricerca che già è implementato.

Bibliografia

[1] **Progressive Web App.** https://web.dev/what-are-pwas/

[2] **Angulare modules.** https://angular.io/guide/architecture-modules

[3] **Thinking Aloud: The #1 Usability Tool.** Jakob Nielsen (2012). https://www.nngroup.com/articles/thinking-aloud-the-1-usability-tool/

[4] **Cooperative usability testing: Complementing usability tests with user-supported interpretation sessions.** Kasper Hornbæk, Erik Frøkjær (2005).

[5] **10 Usability Heuristics for User Interface Design.** Jakob Nielsen (1994). https://www.nngroup.com/articles/ten-usability-heuristics/

[6] **Ionic page lifecycle** https://ionicframework.com/docs/angular/lifecycle