Phd Angular: applicazione Angular 2+ per la gestione di dottorati

Elaborato per il corso Software Architectures and Methodologies

Alessandro Sestini
matricola: 6226094
alessandro.sestini@stud.unifi.it

Abstract

In questo elaborato é stata sviluppata un'user interface Angular 2+ per l'applicativo riguardante la gestione di dottorati; l'interfaccia si é basata su un back-end creato in precedenza che fornisce dei servizi REST fruibili da questo tipo di client. Il lavoro si é svolto in piú fasi: prima é stata fatta un'analisi dei requisiti in cui sono stati creati degli scenari e da questi dei casi d'uso, integrando quelli che esistevano giá; la seconda fase si é concentrata sullo sviluppo e l'implementazione in Angular di tutti gli scenari.

1. Introduzione

1.1. Architetture RESTful

Un'architettura REST (figura 1) é una particolare architettura software adatta a sviluppare delle applicazioni web. Il principio fondamentale si basa sull'esposizione di servizi da parte del back-end tramite un insieme di REST API fruibili dai vari client. Una REST API é a sua volta una collezione di REST service.

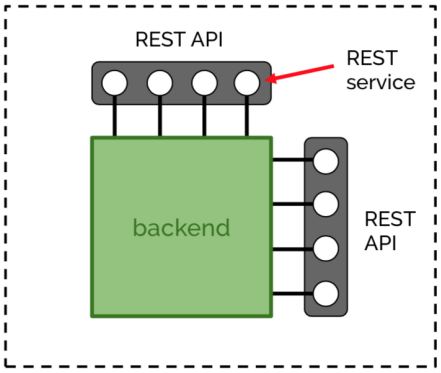


Fig. 1: architettura RESTful

Tutti i servizi associati ad una certa tipologia di risorsa sono gestiti da un Endpoint, una sorta di contenitore che organizza le risorse REST disponibili ai vari front-end. Ogni servizio attivo per un endpoint é associato ad una URI gerarchica, in cui la prima parte denota l'endpoint mentre quella finale corrisponde alla risorsa che si sta richiedendo. Per accedere ai servizi offerti dalle API, i client devono usare il protocollo HTTP (GET, POST, PUT, DELETE, PATCH, ...) assieme alle uri esposte dagli endpoint.

1.2. Back-end

In breve, il back-end (figura 2) é la parte di software che risiede totalmente lato server ed espone i servizi REST. Secondo il paradigma Client-Server, per poter comunicare in maniera adeguata con il front-end i dati scambiati con il back-end devono essere riferiti ad un 'contratto' che puó anche contenere l'indicazione sullo schema dei dati stessi. Tale accordo deve definire il formato: tipicamente, e anche in questo progetto, i dati saranno nel formato JSON. Una volta determinato, il back-end deve definire dei DTO, delle classi che modellano un oggetto software che rappresentano una versione semplificata o adeguata di oggetti riferiti al modello di dominio: le proprietá dei DTO saranno chiaramente collegate con il contratto definito.

1.3. Front-end

Il front-end (figura 3) é la parte di applicazione che usa i servizi REST e ne definisce la presentazione.

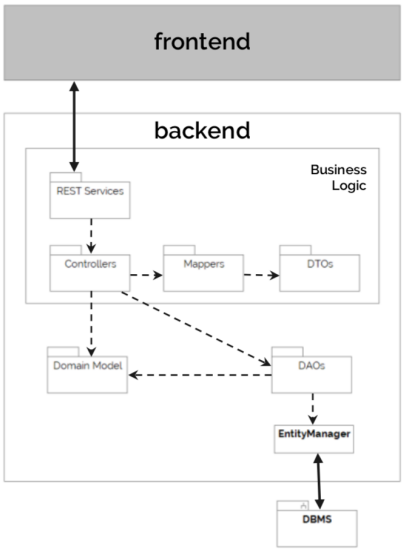


Fig. 2: Back-end di un'architettura RESTful

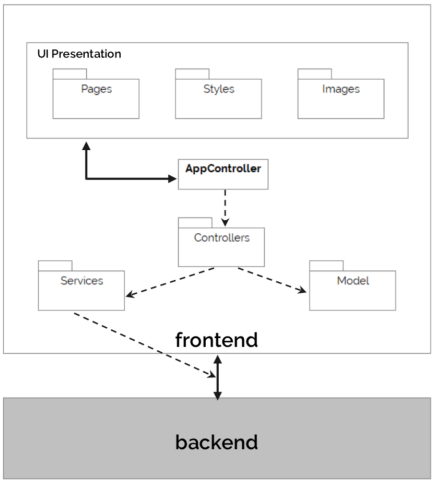


Fig. 3: Front-end di un'architettura RESTful

É possibile utilizzare diversi front-end (ad esempio, per diverse tipologie della stessa applicazione) per gli stessi servizi REST: esiste quindi un forte disaccoppiamento tra back-end e front-end nell'architettura REST.

1.4. Angular 2+

Angular 2+ [1] é un framework utilizzato per creare delle interfacce utente per applicazioni REST-based e single page, cioé in cui vengono caricate direttamente tutte le risorse e aggiunte dinamicamente alla pagina HTML. Segue il design pattern Model-View-Controller (figura 4).

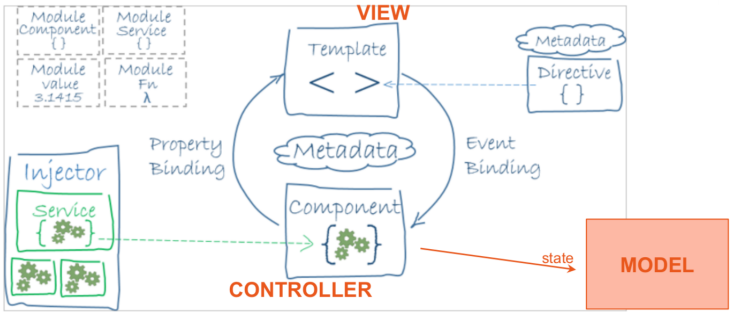


Fig. 4: Architettura e design pattern di un'applicazione Angular 2+

Le applicazioni sviluppate in Angular 2+ vengono eseguite interamente dal web browser dopo essere state scaricate dal web server: é stato progettato per fornire uno strumento facile e veloce per sviluppare su qualunque piattaforma, compresi smartphone e tablet. Il linguaggio di programmazione di Angular 2+ é il TypeScript, che estende il JavaScript aggiungendo classi, interfacce e moduli; puó essere inoltre sia implicitamente che esplicitamente tipato.

1.5. JsonServer

Per poter sviluppare l'user interface Angular 2+, non avendo a disposizione il back-end completato, é stato usato JsonServer [2], un semplice framework che permette di creare e gestire una REST API con operazioni di CRUD attraverso la definizione di un file JSON; non essendo peró una vera REST API, ci sono delle limitazioni nell'utilizzo. Nel corso della relazione verrá spiegato come questo framework é stato impiegato e quali sono i vincoli e i problemi derivanti in vista del collegamento tra il front-end e il back-end reale.

2. Analisi dei Requisiti

In questa sezione vengono riportati i risultati emersi dal colloquio per definire i requisiti del gestionale di Programmi di Dottorato. L'analisi é partita dalla definizione di alcuni scenari, da cui sono stati estrapolati i vari casi d'uso che sono stati aggiunti a quelli giá disponibili, completando la logica di dominio precedentemente creata.

2.1. Scenari

L'analisi dei requisiti é partita dalla definizione di alcuni scenari con lo scopo di scoprire i vari casi d'uso e i requisiti che l'applicativo deve soddisfare.

2.1.1 Scenario 1: anagrafica studenti e faculty

Un membro dello Staff pubblica nel sistema i PhD, i dati degli studenti e dei docenti e puó associare ad uno studente uno o piú advisor e/o un adjoint advisor (membro esterno alla faculty)

2.1.2 Scenario 2: un docente tiene un corso

Un coordinatore accede al sistema e aggiunge un corso al programma di dottorato. A questo punto gli studenti si possono registrare: un corso puó avere piú studenti iscritti e uno studente puó essere iscritto a piú corsi. Il docente puó vedere la lista degli studenti registrati al corso e puó interagire sia con loro che con il materiale delle lezioni. Al termine di un corso il docente registra i grades per ogni studente.

2.1.3 Scenario 3: uno studente carica eventi e report

Uno studente iscritto al sistema puó registrare degli eventi sulla sua pagina (es: pubblicazione articoli) e puó richiedere che vengano riconosciuti dei CFU in relazione a questi eventi. A questo punto l'advisor e il coordinatore approvano i CFU. Lo studente puó alla fine dell'anno caricare la relazione che verrá valutata dall'advisor e dal docente.

2.2. Requisiti

Dalla definizione degli scenari é possibile estrapolare dei requisiti e dei casi d'uso, che vanno a completare quelli giá esistenti nel precedente documento.

- Un membro dello staff amministrativo deve essere in grado di creare e gestire un PhDProgram
- Un membro dello staff amministrativo deve essere in grado di creare e gestire un Ciclo
- Un membro dello staff amministrativo deve essere in grado di creare e gestire uno Studente
- Un membro dello staff amministrativo deve essere in grado di creare e gestire un Faculty
- Un membro dello staff amministrativo deve nominare uno o piú advisor o adjoint advisor per uno Studente
- Un coordinatore deve essere in grado di creare e gestire un corso
- Un coordinatore deve abilitare altri docenti al corso
- Un docente deve essere in grado di vedere gli studenti iscritti ad un suo corso
- Un docente deve essere in grado di interagire con gli studenti iscritti
- Un docente deve essere in grado di caricare materiale per un determinato corso
- Un docente deve essere in grado di assegnare dei grades agli studenti iscritti ad un particolare corso
- Uno studente deve essere in grado di registrarsi ad un corso
- Uno studente deve essere in grado di gestire le registrazioni
- Uno studente deve essere in grado di scaricare materiale caricato per il corso
- Uno studente deve essere in grado di registrare degli eventi
- Uno studente deve poter proporre un riconoscimento in CFU per gli eventi registrati in precedenza
- Un advisor deve poter sostenere la richiesta per il riconoscimento dei CFU
- Un coordinatore deve poter riconoscere dei CFU a degli eventi registrati dagli studenti
- Uno studente deve essere in grado di caricare la relazione di fine anno
- Un advisor deve poter scaricare e valutare la relazione caricata dallo studente
- Un coordinatore deve poter scaricare e approvare la relazione caricata dallo studente

In seguito vengono riportati i requisiti scritti nel documento precedente ma non accennati al colloquio

- Un docente deve poter aggiungere o rimuovere studenti dal suo corso

- Uno studente deve poter rilasciare valutazioni su un corso da lui frequentato

Qui sotto vengono riportati i requisiti non funzionali:

- L'accesso al sistema deve avvenire tramite un'autenticazione
- Il sistema deve essere accessibile da tutti i tipi di dispositivi
- il sistema deve essere accessibile da piú utenti contemporaneamente

2.3. Casi d'uso

In questa sezione vengono riportati i casi d'uso emersi dalla specifica degli scenari e dei requisiti.

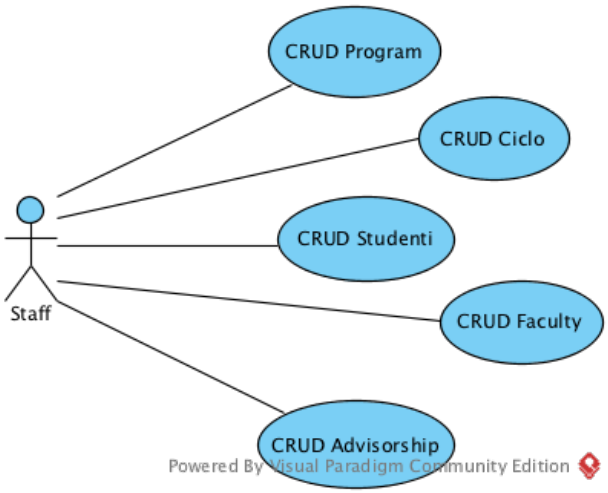


Fig. 5: Casi d'uso relativi allo scenario 1

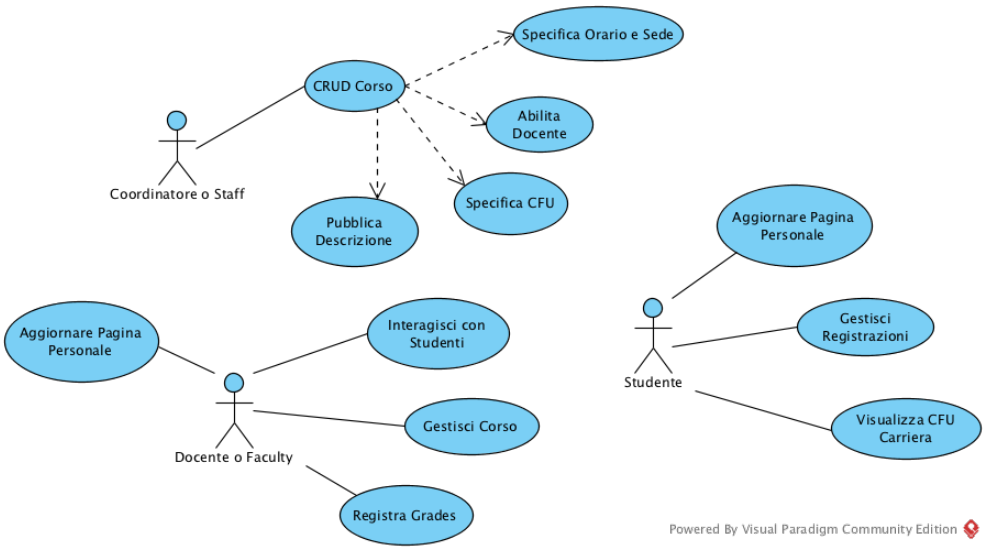


Fig. 6: Casi d'uso relativi allo scenario 2

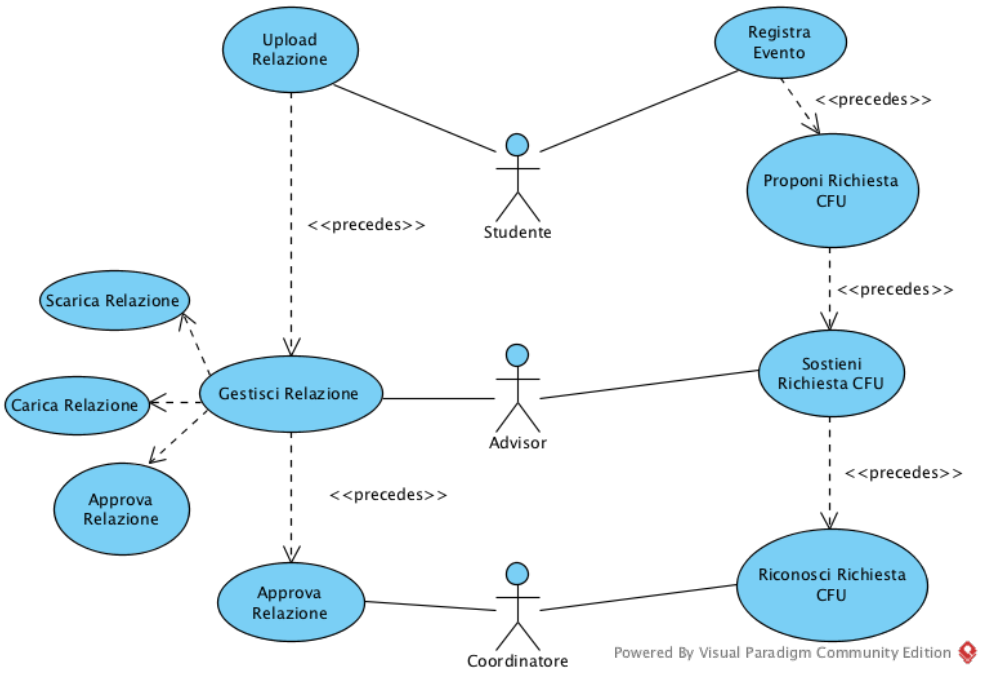


Fig. 7: Casi d'uso relativi allo scenario 3

Nel documento precedentemente definito veniva specificato come, dopo che lo studente fa richiesta di registrazione ad un corso, un membro dello staff deve procedere ad aggiungere lo studente a quel corso e viceversa, mentre **questo caso d'uso non é emerso dal colloquio**.

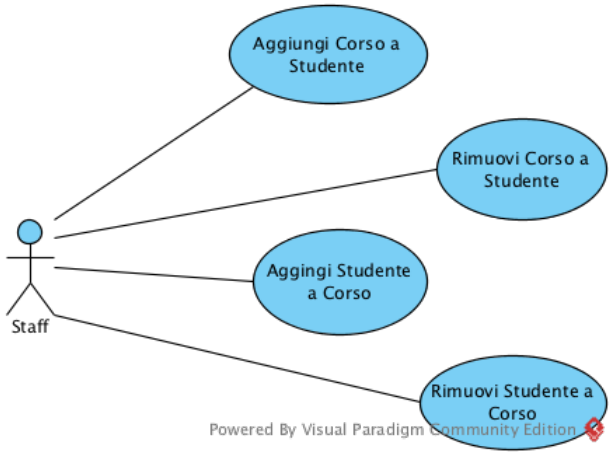


Fig. 8: Casi d'uso precedentemente definiti ma non emersi dal colloquio

2.4. Logica di Dominio

In questa sezione viene riportata la logica di dominio giá creata, da usare come base di partenza per l'applicazione Angular che dovrá soddisfare gli scenari qui descritti.

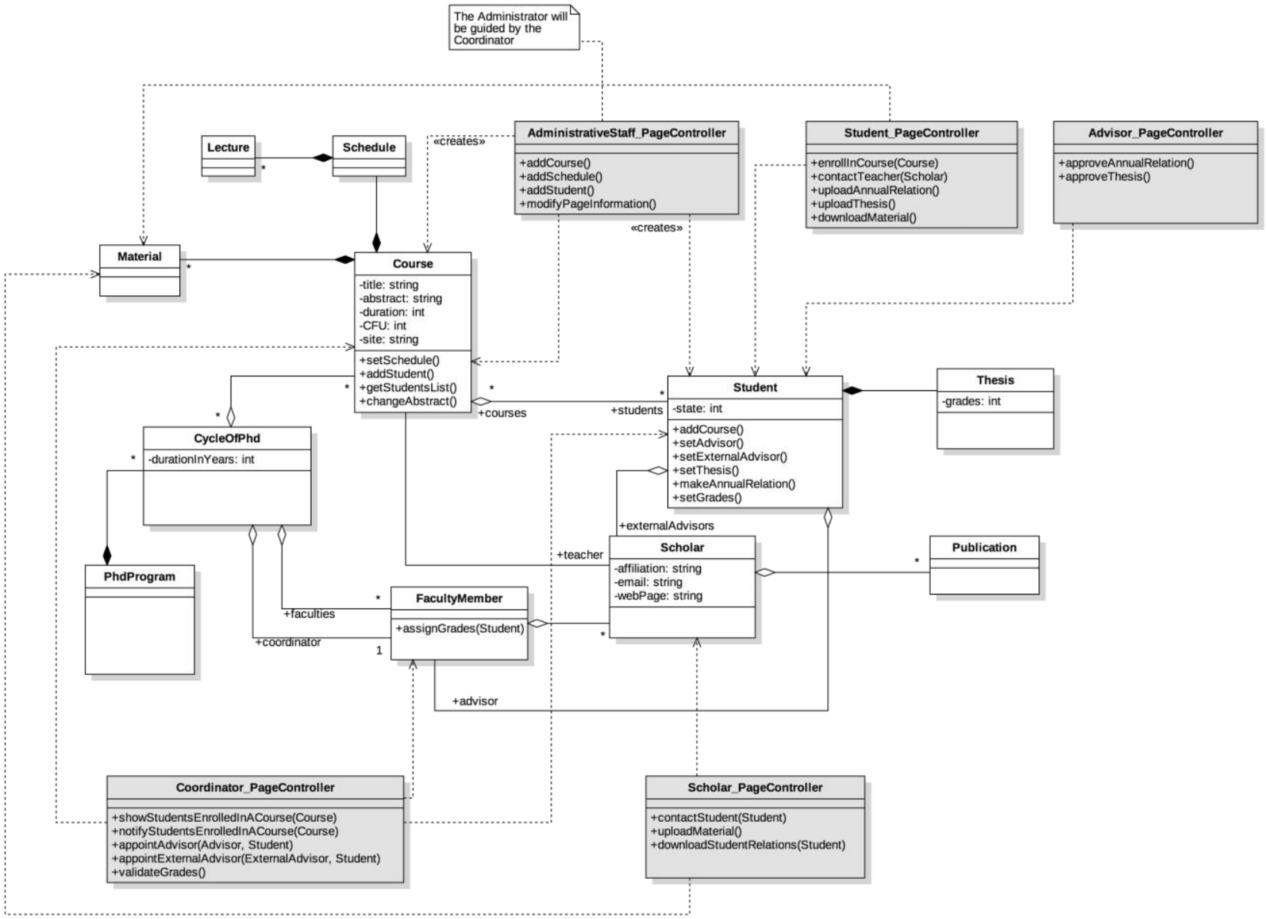


Fig. 9: UML: diagramma delle classi della logica di dominio

Rispetto a quella emersa dal colloquio ci sono un po' di differenze:

- non é presente nessuna classe Grade, e non é ben chiaro se sia un attributo della classe Student
- non é presente nessuna classe Registration, ma uno Student ha una collezione di Course e un Course ha una collezione di Student
- non é presente nessuna classe Collegio
- manca completamente la parte degli Eventi registrabili da uno studente

2.5. Commenti

Le differenze tra la logica definita in precedenza e quella definita nel colloquio sono minime, tranne che per alcuni dettagli:

- dai casi d'uso del documento si capisce come sia un membro dello Staff ad aggiungere uno studente ad un corso e viceversa, dopo richiesta dello studente stesso. Nel colloquio non é mai stato fatto riferimento a questo aspetto
- nel documento manca completamente la parte sia di dominio che di casi d'uso della registrazione di eventi (a cui é possibile collegare dei CFU) da parte dello studente, e la relativa accettazione da parte dell'advisor e del coordinatore

- dal documento sembra che sia il coordinatore ad associare un advisor ad uno studente, mentre dal colloquio é emerso come questo veniva fatto dal un membro dello staff

Il passo successivo é quello di definire un'interfaccia che possa soddisfare questi scenari: uno fondamentale sembra essere il numero 1, in quanto un membro dello staff deve poter prima di tutto creare un programma di dottorato e i vari membri che possono accedere a quest'ultimo (Coordinatori, Faculty, Studenti, ...). A quel punto si puó passare allo svolgimento degli altri due.

3. Implementazione

Come detto precedentemente, l'implementazione é partita dallo scenario 1, risultato essere basilare. Prima della stesura del codice sono stati fatti dei mock-up su carta (fig 10) per aiutare a definire il layout complessivo dell'applicazione: questi disegni sono serviti come punto di riferimento per l'implementazione vera e propria dell'applicativo.

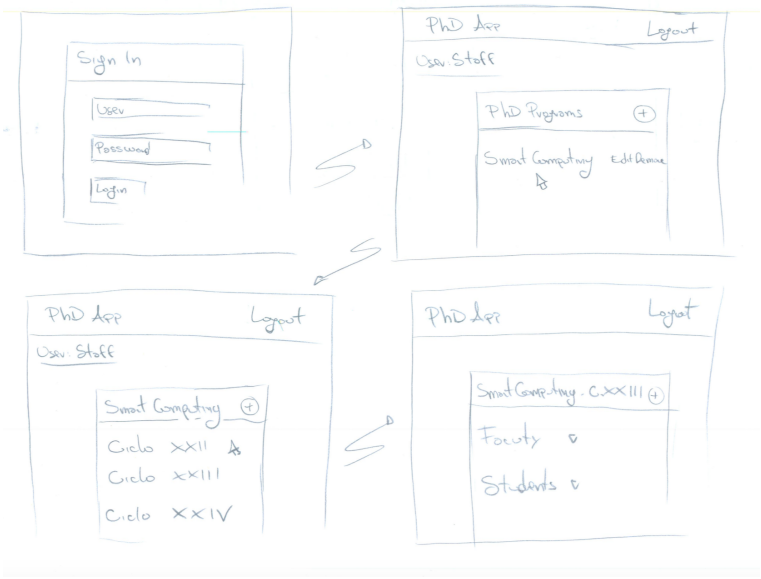


Fig. 10: Esempio di mock-up dell'applicazione

Per facilitare la creazione dell'interfaccia e la sua appetibilitá, sono stati usati i principi e gli elementi del Material Design [3] sviluppato da Google: é infatti presente un framework facilmente importabile in applicazioni Angular 2+ [4] che permette di usare componenti di questa libreria di design.

3.1. Struttura dell'applicazione

L'applicazione é strutturata seguendo lo schema riportato nella figura 11.

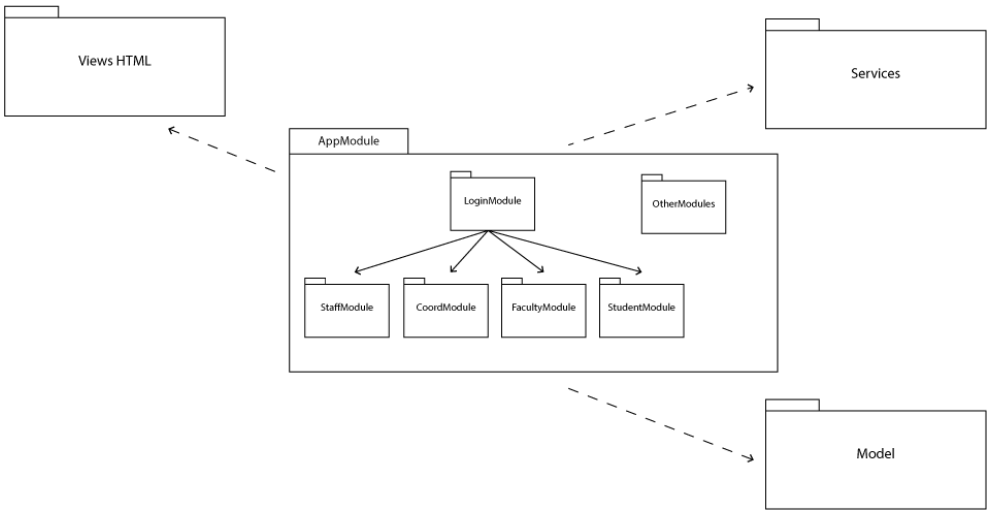


Fig. 11: Struttura dell'applicazione

- É presente un modulo principale chiamato AppModule. All'interno di questo sono stati creati altri moduli, ognuno dei quali contiene i componenti relativi all'interfaccia di un particolare attore, che sono stati individuati nella stesura dei casi d'uso: membro dello staff, coordinatore, faculty/docente e studente;
- Il primo modulo che l'utente si troverá di fronte é peró il LoginModule, in cui sará richiesto di inserire le credenziali per poter entrare all'interno del sistema; da questo si passerá poi alla view relativa al particolare attore;
- Gli altri moduli definiti e visibili in figura racchiudono quei componenti che sono usati all'interno di uno o piú moduli descritti in precedenza;
- La parte dei Service racchiude tutti quelli che in Angular 2+ vengono chiamati servizi: quelle classi che definiscono i metodi di interfaccia tra l'applicazione e i servizi REST offerti dal back-end;
- La parte del Model invece racchiude le classi relative al modello: queste cercano di rappresentare il piú fedelmente possibile la logica di dominio definita nella sezione 2 e descrivono poi quella che sará la struttura delle entitá dei DTO che saranno creati nel back-end;

Nel proseguo della relazione entreremo piú nel dettaglio di tutti gli elementi che compongono la struttura del programma.

3.2. Model e DTO

Per definire le classi del modello in Angular é stato preso come punto di riferimento il diagramma delle classi della logica di dominio in figura 9. In figura 12 é quindi possibile vedere tutte le classi che sono state implementate.

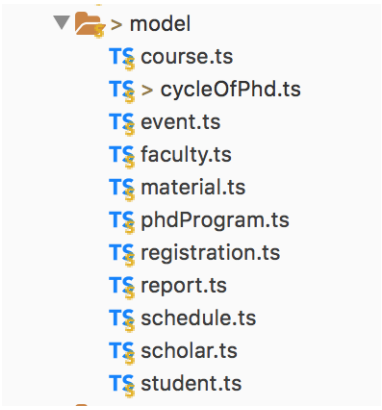


Fig. 12: Classi del modello in Angular

A questo punto si é stato necessario ragionare sulle relazioni tra le varie entitá di questo modello: per fare un esempio concreto, una possibile domanda potrebbe essere se un corso di dottorato deve contenere il riferimento ai cicli di cui é composto. L'approccio usato in questo elaborato é quello di essere piú 'component driven' possibile: si cerca di definire delle entitá che abbiano solo gli attributi necessari alla visione dei componenti che costituiscono l'interfaccia. Questo porta a far si che, se in tutti i componenti in cui vengono usati i programmi di dottorato non é necessario far vedere all'utente le informazioni relative ai cicli di cui é composto, questa relazione non viene esplicitata nel modello, portando a definire una semplice classe come quella nell'esempio 1.

Listing 1: PhdProgram

```
export class PhdProgram {
        id: number;
        name: string;
}
```

Mentre i cicli avranno un riferimento all'id del programma di dottorato in cui é definito, in modo da riuscire a recuperare il suo PhD, come é possibile vedere nell'esempio 2.

Listing 2: CycleOfPhd

```
export class CycleOfPhd {
        id: number;
        num: string;
        numNumber: number;
        durationInYears: number;
        phdProgramId: number;
}
```

É stato scelto questo approccio principalmente per 2 motivi:

- per non caricare il modello di troppe informazioni, che dovranno necessariamente essere richieste ad un back-end, cercando dunque minimizzare la quantitá di dati che vengono passati tra le due parti;
- per rendere piú coerente lo stato dell'intera interfaccia con quello del back-end: se un utente andasse a caricare subito tutti i cicli di un programma di dottorato, un'eventuale modifica da parte di un altro utilizzatore non verrá visionata dal primo. Tramite l'approccio 'component driven', le informazioni vengono caricate solamente quando richieste dall'interfaccia, cercando di minimizzare questo problema.

Ci sono peró dei casi in cui é necessario mettere a schermo contemporaneamente le informazioni di piú entitá relazionate tra di loro. Ad esempio, nella pagina di un docente verranno visualizzati tutti gli studenti di cui é advisor e gli eventi che questi hanno registrato: tutti questi dati dovranno essere disponibili nella stessa vista. In questo caso dunque le informazioni necessarie riguardano anche gli eventi, ed é quindi risultato piú utile esplicitare la relazione che lega gli studenti agli eventi all'interno del modello, ottenendo una classe come quella dell'esempio 3.

Listing 3: Student

```
import { Faculty } from './faculty';
import { Scholar } from './scholar';
import { Event } from './event';
import { Report } from './report';

export class Student {
        id: number;
        facultyId = [];
        firstname: string;
        lastname: string;
        webpage: string;
        email: string;
        faculties: Faculty[];
        externalAdvisor: Scholar = new Scholar();
        role = 'student';
        username: string;
        password: string;
        cycleOfPhdId: number;
        phdProgramId: number;
        events: Event[];
        reports: Report[];
}
```

La definizione del modello in Angular prevede quindi anche la scelta di quali e quante informazioni dovranno essere trasportate dal back-end al front-end e viceversa. Questo é fondamentale nella definizione dei DTO all'interno delle classi Java: verosimilmente, queste dovranno rispecchiare quelle definite in TypeScript in questo progetto, e sará poi compito dei Mapper mappare le entitá della logica di dominio in quelle dei DTO.

3.3. Servizi

Un servizio in Angular 2 é una classe che implementa funzionalitá condivise dai vari elementi di un'applicazione, siano essi componenti che altri servizi; generalmente mette a disposizione dei metodi per la gestione di una tipologia di risorsa (quindi una classe del modello di dominio). Le classi 'service' sono solitamente le uniche del framework a comunicare con i servizi REST esposti dal backend; per questo motivo possono essere considerate dei DAO (Data Access Object). All'interno di questa tipologia di classe ci saranno dunque dei metodi che rispecchiano le chiamate HTTP, come é possibile vedere dall'esempio 4. Ció vuol dire che é necessario inserire come parametro d'ingresso l'indirizzo del servizio REST da cui estrarre i dati; gli indirizzi di questi service rispecchiano la logica di dominio in figura 9, con la differenza che JsonServer vuole i nomi plurali: ad esempio, per estrarre i cicli di un PhD, si interrogherá *'phdPrograms/ID/cycleOfPhds'*, oppure per avere gli eventi di uno studente, useremo *'students/ID/events'*. Sará poi compito del back-end esporre gli endpoint dipendenti da un determinato URI e dall'HTTP method. I service sono di solito dei Singleton che vengono 'iniettati' all'interno degli altri componenti, in modo da essere disponibili in tutti gli elementi dell'applicazione.

Listing 4: PhdProgramService

```
export class PhdProgramService {

getPhdPrograms() {
return this.http.get<PhdProgram[]>('${this.general.uri}/phdPrograms')
.pipe(catchError(error => of(this.general.setError(true, error))));
}

addPhd(phd) {
return this.http.post('${this.general.uri}/phdPrograms', phd)
.pipe(catchError(error => of(this.general.setError(true, error))));
}

deletePhd(id) {
return this.http.delete('${this.general.uri}/phdPrograms/${id}')
.pipe(catchError(error => of(this.general.setError(true, error))));
}
...
}
```

3.4. JsonServer e relative limitazioni

Come anticipato nell'introduzione, per rendere lo sviluppo dell'interfaccia indipendente dal completamento effettivo del back-end, é stato usato il framework JsonServer per creare una finta REST API. Questo approccio porta peró delle limitazioni:

- JsonServer permette l'uso di nested resources solo di 1 livello di profonditá: ció vuol dire che é possibile chiamare l'indirizzo 'phdPrograms/id/cycleOfPhds', ma non andare piú a fondo. Questo problema é stato risolto attraverso il re-mapping di determinate route;
- JsonServer mette a disposizione un meccanismo di Delete in cascata che elimina anche i figli di primo livello di una particolare entitá, ma non quelli di livelli piú profondi;
- JsonServer non permette l'uso di relazioni many-to-many: ció vuol dire che, ad esempio, é possibile mettere in relazione un faculty a piú studenti, ma non uno studente a piú faculty. Anche questo problema é stato risolto tramite l'uso del re-mapping di specifiche route usate in combinazione a dei filtri di ricerca.

Tutte queste problematiche peró non si dovrebbero presentare al momento dell'implementazione dei veri end-point Java, in quanto essi faranno riferimento alla vera logica di dominio e potranno avere un comportamento scelto totalmente dallo sviluppatore.

3.5. LoginModule

Per implementare un meccanismo di autenticazione, é stato usato il framework JsonWebToken [5] assieme a JsonServer: una volta inserite le credenziali, Angular fará una chiamata Post a JsonServer, che controllerá l'esistenza di un utente con quelle referenze. In caso positivo, manderá all'applicazione tutti i suoi dati, insieme al suo ruolo e ad un token di accesso, che dovrá essere creato lato back-end; queste informazioni saranno poi salvate sul Local Storage del web browser. Una volta autenticato, l'utente potrá vedere su schermo i componenti relativi al modulo descritto dal suo ruolo: *staff, coordinator, faculty, scholar* e *student*. A questo punto vengono usati due meccanismi di controllo: ogniqualvolta una vista viene caricata, si controllerá che l'utente salvato abbia il ruolo giusto per usare e visualizzare quel componente; ogniqualvolta si effettua una chiamata HTTP, tramite un oggetto

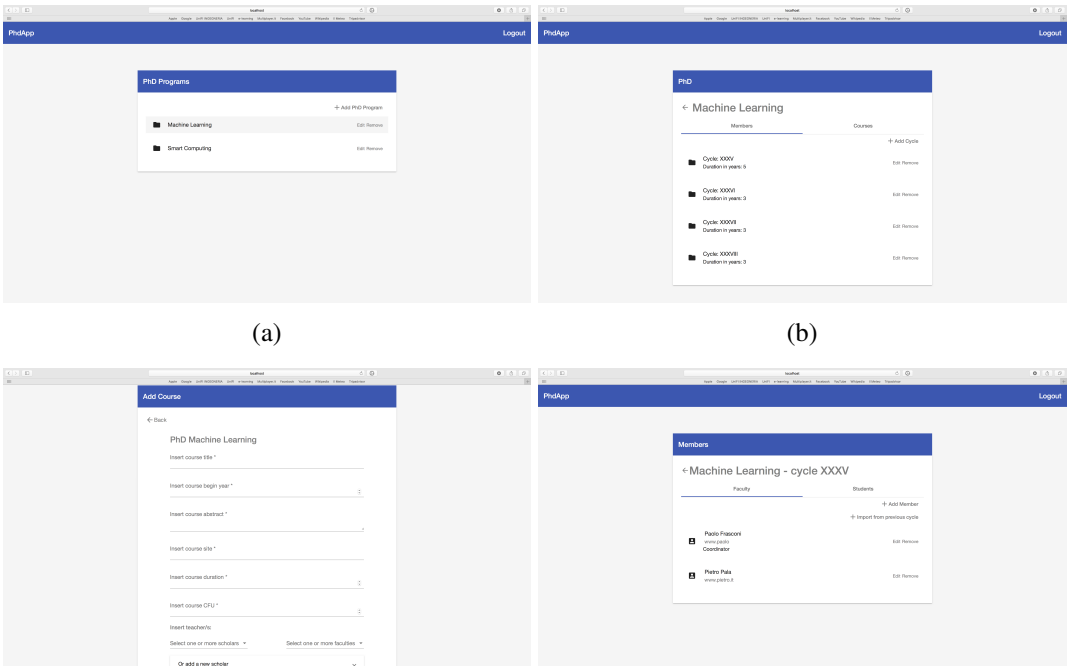
JwtInterceptor si aggiunge un header con il token salvato in precedenza; sará poi compito del server controllare che il token sia ancora valido.

3.6. Interfaccia e funzionalitá

Ora analizzeremo brevemente le funzionalitá previste dall'interfaccia facendo riferimento agli scenari e ai requisiti descritti nella sezione di analisi.

3.6.1 Scenario 1: anagrafica

L'attore principale dello scenario é il membro dello staff: appena autenticato, l'utente di questo tipo si troverá di fronte la lista dei PhD presenti nel sistema (figura 13a), con l'opportunitá di crearne uno nuovo. Selezionando un programma, sará possibile vedere l'elenco dei cicli (figura 13b) e dei corsi appartenenti al particolare PhD, oppure aggiungerne di nuovi. Inserendo un nuovo corso (figura 13c), l'applicazione offre la possibilitá di abilitare un docente creato in precedenza ad entrare nella pagina relativa. Accedendo ai cicli l'utente staff potrá visionare sia i faculty sia gli studenti appartenenti (figura 13d); la creazione di qualsiasi membro richiederá l'inserimento di username e password per poterlo abilitare all'accesso del sistema. É inoltre possibile autorizzare un faculty ad essere un coordinatore del PhD e abilitare uno o piú faculty ad essere advisor per uno studente. Qualunque entitá del modello puó essere modificata premendo il tasto *'Edit'* posto accanto ad ogni nome.



(a) (b)

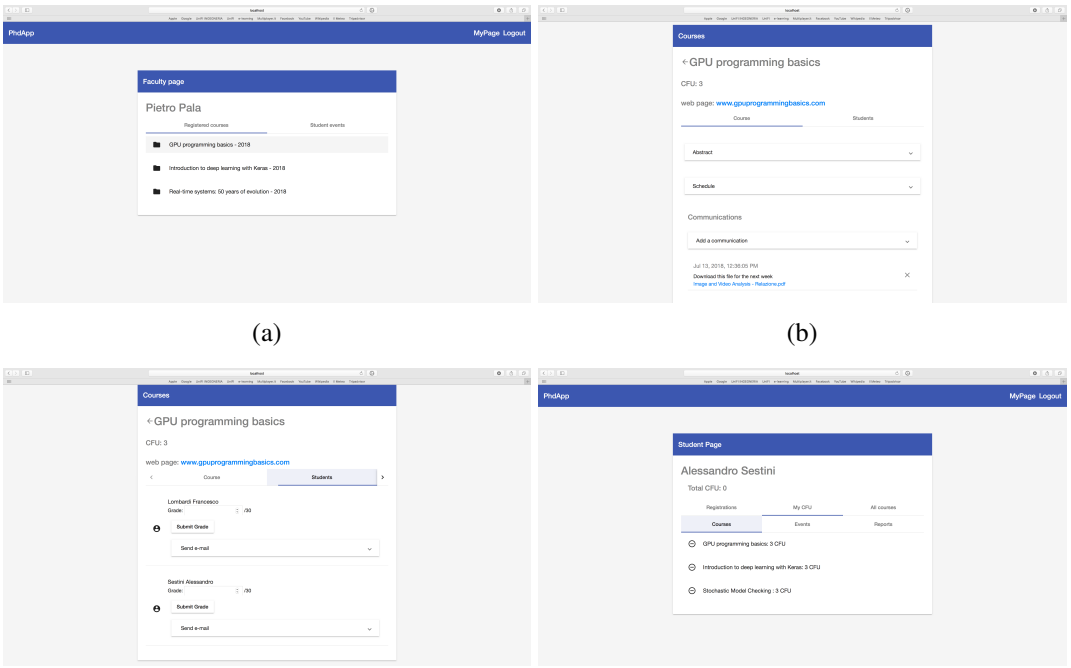
(c) (d)

Fig. 13: Alcuni screenshot che descrivono lo scenario 1

3.6.2 Scenario 2: un docente tiene un corso

Il componente relativo alla gestione dei corsi di un dottorato sará disponibile anche nella pagina dell'utente di ruolo coordinatore, che potrá interagire con tutti quelli appartenenti al suo PhD. Autenticandosi invece con credenziali di tipo *faculty* o *scholar*, apparirá su schermo il componente contenente i corsi per i quali il docente é stato abilitato (figura 14a). Entrando su uno di questi, sará possibile aggiungere nuovi orari o nuove comunicazioni,

con l'opportunitá di inserire in quest'ultime dei file da caricare sul server, di modo che siano visibili a chiunque possa entrare in questa view (figura 14b). L'utente di ruolo *student*, una volta autenticato nel sistema, potrá vedere tutti i corsi all'interno del suo PhD e potrá registrarsi a qualsiasi di essi; una volta effettuata la registrazione, questo nuovo corso apparirá nella lista di quelli a cui é iscritto, con la possibiliá di entrare nella pagina relativa, vedere le comunicazioni inserite dai docenti e scaricare il materiale caricato. L'insegnate potrá dunque consultare ed interagire con la lista di tutti gli studenti (figura 14c) iscritti ad uno dei suoi corsi. Al termine delle lezioni, sempre nel componente contenente l'elenco degli studenti, il docente potrá assegnare un voto ad ogni persona. Una volta fatta questa operazione, nella pagina dello studente verrá notificata l'effettiva conclusione del corso tramite un'icona posta accanto al nome, mentre verrá aggiornato il conto totale dei CFU. Nella pagina attinente ai crediti, sará poi possibile consultare tutti i corsi sostenuti e da sostenere, i CFU relativi ed eventualmente il voto di completamento (figura 14d). Il coordinatore, essendo anche lui un faculty, potrá effettuare anche tutte le operazioni relative al ruolo di docente.



(a) (b)

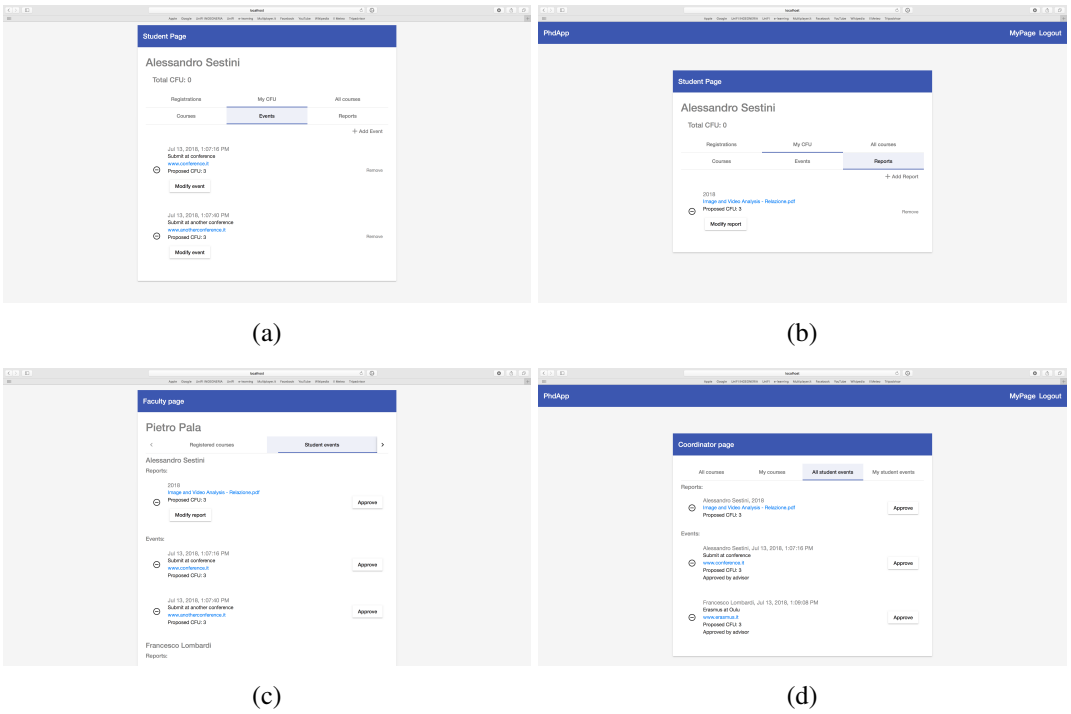
(c) (d)

Fig. 14: Alcuni screenshot che descrivono lo scenario 2

3.6.3 Scenario 3: uno studente carica eventi e report

Nella pagina relativa ai CFU della carriera di uno studente, l'utente di questo ruolo potrá aggiungere e gestire eventi, indicandone una descrizione e un link in cui é possibile vedere il materiale attinente alla circostanza, e potrá proporre dei crediti da riconoscere. Dunque, uno dei faculty che é advisor di questo studente potrá sostenere la proposta nella lista relativa agli eventi di tutti i suoi allievi. L'ultimo passaggio é l'accettazione dei crediti da parte del coordinatore del PhD, che vedrá nella propria pagina tutti gli eventi registrati e sostenuti dagli advisor all'interno del programma di dottorato: una volta approvato, l'evento sparirá dalla pagina del coordinatore, ma in quella dello studente verrá notificata, nel componente dei CFU, la relativa approvazione e verrá aggiornato il conteggio totale dei crediti. Lo stesso procedimento avviene per la gestione dei report di fine anno: uno studente, nella sezione apposita, potrá caricare il file, l'anno e la proposta di CFU della relazione, che a quel punto sará

disponibile anche al suo advisor; quest'ultimo potrá scaricare, correggere e ricaricare l'elaborato nel server e, quando sará pronto, sostenere il riconoscimento dei CFU. Infine, il coordinatore deciderá se approvare la richiesta relativa al report di fine anno nello stesso modo in cui approvava gli eventi.



(a) (b)

(c) (d)

Fig. 15: Alcuni screenshot che descrivono lo scenario 3

4. Conclusione e sviluppi futuri

In conclusione, in questo elaborato é stata sviluppata un'interfaccia utente per la gestione di dottorati basata sulla logica di dominio precedentemente creata ma non ancora conclusa. Sono state implementate tutte le funzionalitá richieste ed emerse durante la definizione degli scenari nella fase di analisi dei requisiti. Per rendere lo sviluppo indipendente dal completamento del back-end, é stato usato il framework JsonServer; questo ha portato peró a delle limitazioni, che sono state facilmente risolte e non dovrebbero creare problemi durante la definizione degli endpoint. I prossimi passi dello sviluppo potrebbero essere:

- aspettare il completamento del back-end per sviluppare gli endpoint di modo da fornire i servizi REST necessari al reale funzionamento dell'applicazione;
- aumentare l'usabilitá dell'interfaccia generale, ad esempio aggiungendo una barra di navigazione laterale con le azioni piú frequenti per ogni utente, o migliorare in generale il flow del programma;
- definire nuovi scenari di utilizzo del sistema per aumentare il numero di funzioni dell'applicativo: ad esempio la gestione di tesi di fine dottorato da parte degli studenti, presente nella logica di dominio ma non emersa durante il colloquio.

References

[1] Angular: *https://angular.io*

[2] JsonServer: *https://github.com/typicode/json-server*

[3] Material Design: *https://material.io/design/*

[4] Angular Material: *https://material.angular.io*

[5] JsonWebToken: *https://github.com/auth0/node-jsonwebtoken*