

Università della Calabria

Dipartimento di Matematica e Informatica



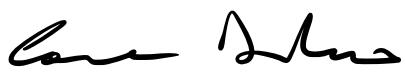
Corso di Laurea Triennale in Informatica

Tesi di Laurea

**Un Sistema Web per la
Rendicontazione e Allocazione
delle Risorse nei Dipartimenti
Universitari**

Relatore:

Prof. Carmine Dodaro



Candidato:

Aldo Gioia

Matricola 230731



Anno Accademico 2024/2025

*A mia Nonna Ines,
la luce nel mio cammino.*

Indice

Indice	2
1 Introduzione	4
1.1 Contesto e motivazioni	4
1.2 Attività svolte e risultati ottenuti	5
2 Tecnologie utilizzate	7
2.1 Figma	7
2.2 Java	7
2.3 TypeScript	8
2.4 Angular	8
2.5 Spring Boot	8
2.6 JPA e Hibernate	9
2.7 PostgreSQL	9
2.8 Lombok	10
2.9 Jackson	10
3 Descrizione del progetto realizzato	11
3.1 Livello Dati	12
3.2 Livello Applicazione - BackEnd	13
3.2.1 Strato Controller	14
3.2.2 Strato Service	20
3.2.3 Strato Repository	26
3.2.4 Sicurezza	30
3.3 FrontEnd (Livello Presentazione)	47

<i>INDICE</i>	3
3.3.1 Accesso	51
3.3.2 Reset Password	56
3.3.3 Area Personale	58
3.3.4 Elenco Progetti	63
3.3.5 Aggiungi Professore	65
3.3.6 Aggiungi Progetto	67
3.3.7 Dettagli Professore	68
3.3.8 Dettagli Progetto	69
3.3.9 Modifica Professore	70
3.3.10 Modifica Progetto	72
3.3.11 Assegnamenti	76
3.3.12 Rendicontazione	77
3.3.13 Dettaglio Annuale	80
3.3.14 Dettaglio Mensile	83
4 Conclusioni	86
Bibliografia	88

Capitolo 1

Introduzione

1.1 Contesto e motivazioni

La rendicontazione delle ore rappresenta uno degli aspetti più importanti nella gestione dei progetti accademici. In contesti del genere, in cui le attività di ricerca e collaborazione devono essere documentate per fini amministrativi, una rendicontazione precisa delle ore gioca un ruolo cruciale. All’interno delle università, i progetti coinvolgono diverse figure, tra cui: docenti ordinari, associati e ricercatori. Ognuno di loro dedica tempo e sforzi al fine di portare a termine tutte le sfide e gli obiettivi preposti dal progetto. Tenere traccia delle ore di lavoro, della distribuzione delle risorse e dei costi associati è essenziale per garantire la trasparenza necessaria.

Oltre agli aspetti burocratici, una rendicontazione accurata è determinante per il successo del progetto stesso. Essa permette di monitorare l’andamento e l’apporto di ogni “attore” che collabora al progetto, identificare le eventuali criticità che ovviamente possono presentarsi e ottimizzare l’allocazione delle risorse per far sì che si sfrutti al massimo il loro potenziale. Inoltre, un requisito fondamentale, spesso richiesto dagli enti finanziatori, è che i dati rispecchino la realtà oltre ad essere oggettivi e verificabili. Tutto questo è facilmente ottenibile tramite un sistema di rendicontazione, senza il quale la gestione diventerebbe problematica, evidenziando inefficienze e significativi sprechi di risorse. Nonostante la rendicontazione si trovi al centro di un’acca-

rata gestione dei progetti, molte università affidano ancora questo aspetto a metodi più o meno manuali o in ogni caso a strumenti poco strutturati, i quali presentano numerose criticità.

Uno dei principali problemi è rappresentato dall'elevato rischio di errore umano, che va a compromettere la qualità della rendicontazione. Questi metodi richiedono, tra l'altro, un notevole dispendio di tempo e risorse che potrebbero essere impiegate in altri aspetti. Il personale amministrativo è chiamato a dedicare diverso tempo a validare e correggere i dati forniti sulla rendicontazione. Inoltre, potrebbero sorgere anche problemi relativi alla consultazione dei dati oltre che alla loro condivisione in tempo reale. Questo va ad ostacolare il coordinamento dei membri del progetto rallentando sia i processi operativi che decisionali.

Alla luce di ciò, la necessità di strumenti avanzati che possano migliorare l'efficienza, facilitando la gestione e garantendo una rendicontazione affidabile, è più evidente che mai.

L'obiettivo di questa tesi è presentare lo sviluppo di un'applicazione web in grado di affrontare e risolvere le problematiche appena discusse. Attraverso un'interfaccia utente intuitiva e coinvolgente ed un meccanismo di autenticazione sicuro, i docenti e gli amministratori potranno registrare e consultare le informazioni che desiderano in modo rapido ed efficace.

L'applicazione proposta mira a ridurre al minimo gli errori che possono verificarsi migliorando l'affidabilità dei dati, infatti, grazie a meccanismi di validazione il sistema sarà in grado di evitare errori ed incongruenze ancor prima che queste possano essere generate. Inoltre, l'applicazione intende migliorare l'usabilità e l'accessibilità rispetto alle piattaforme esistenti.

1.2 Attività svolte e risultati ottenuti

L'applicazione web per la rendicontazione delle ore e la gestione dei progetti è stato pianificato per migliorare l'organizzazione e renderla più semplice e trasparente per favorire il resoconto delle attività accademiche. La realizz-

zazione di questo lavoro di tesi ha richiesto diverse fasi. Nella prima fase di analisi dei requisiti è stato studiato il contesto universitario, raccogliendo le informazioni necessarie, nel dipartimento, che risultassero utili ed esaustive per limitare le difficoltà ancora presenti e riscontrate. Successivamente, sono state definite le principali funzionalità dell'applicazione web, individuando gli strumenti necessari per semplificare il lavoro amministrativo e favorire una gestione più efficiente.

In particolare, l'applicazione web prevede due attori principali: l'amministratore e il docente. L'amministratore si occupa di caricare i dati dei progetti all'interno della piattaforma web, di assegnare i docenti, e le relative ore da rendicontare, a ogni progetto. Inoltre, ha la possibilità di definire le ore mensili e la distribuzione delle ore dei docenti all'interno di ogni progetto. Il docente, invece, può visionare i progetti a cui è stato assegnato e può caricare, all'interno della propria pagina personale, le ore mensili da dedicare a ogni progetto.

Capitolo 2

Tecnologie utilizzate

2.1 Figma

Come riporta il sito ufficiale di Figma in [Q]: “Figma Design è uno strumento che permette alle persone di creare, condividere e testare design per siti web, app mobile e altri prodotti ed esperienze digitali. È un tool molto popolare tra designer, product manager, writer e sviluppatori, e aiuta chiunque sia coinvolto nel processo di design a contribuire, dare feedback e prendere decisioni migliori in modo più rapido.”

2.2 Java

Come riporta il sito di Amazon Web Service in [B]: “Java è un linguaggio di programmazione ampiamente utilizzato per lo sviluppo di applicazioni web. È una scelta popolare tra gli sviluppatori da oltre due decenni, con milioni di applicazioni Java attualmente in uso. Java è un linguaggio multipiattaforma, orientato agli oggetti e incentrato sulle reti, tanto da poter essere considerato una piattaforma a sé stante. È un linguaggio di programmazione veloce, sicuro e affidabile, adatto allo sviluppo di tutto, dalle app mobili al software enterprise, fino alle applicazioni per big data e alle tecnologie server-side.”

2.3 TypeScript

Come riporta il sito Geekandjob in [7]: “TypeScript è un linguaggio di programmazione open source sviluppato da Microsoft. Più nello specifico, TypeScript è un superset di JavaScript, che aggiunge tipi, classi, interfacce e moduli opzionali al JavaScript tradizionale. Si tratta sostanzialmente di una estensione di JavaScript.”

Quindi, la particolarità di TypeScript è di essere tipizzato, quindi aggiunge definizioni di tipo statico. In questo modo, è possibile ridurre il numero di errori poiché alcune tipologie sono rilevate automaticamente e, inoltre, permette di documentare meglio il codice.

2.4 Angular

Come riporta il sito ufficiale di Angular in [8]: “Angular è una piattaforma e un framework per la creazione di applicazioni client a pagina singola utilizzando HTML e TypeScript. Angular è scritto in TypeScript. Implementa funzionalità di base e opzionali come un insieme di librerie TypeScript che è possibile importare nelle proprie applicazioni.”

Angular si basa sul concetto di componente, che definisce l’insieme degli elementi grafici dell’interfaccia. Inoltre, i componenti possono fare uso dei servizi che permettono di effettuare operazioni non direttamente legate alla visualizzazione.

2.5 Spring Boot

Come riporta il sito di IBM in [6]: “Java Spring Boot (Spring Boot) è uno strumento che rende lo sviluppo di applicazioni web e microservizi con Java Spring Framework più veloce e semplice.

Java Spring Framework (Spring Framework) è un framework open source di livello enterprise, molto popolare, per la creazione di applicazioni autonome e pronte per la produzione che vengono eseguite sulla Java Virtual Machine

(JVM). Spring Boot semplifica lo sviluppo con Spring Framework grazie a tre funzionalità principali: autoconfigurazione, un approccio strutturato alla configurazione e la possibilità di creare applicazioni autonome.”

2.6 JPA e Hibernate

Coem riporta il sito Turing in [1]: “ORM sta per Object-Relational Mapping, una tecnica di programmazione che consente agli ingegneri del software di scrivere in qualsiasi linguaggio di programmazione e di incapsulare il codice necessario per manipolare i dati, eliminando così la necessità di utilizzare SQL. Invece di scrivere query SQL, si interagisce direttamente con gli oggetti nello stesso linguaggio in cui si sta programmando. L’obiettivo dell’ORM è facilitare il lavoro con i database, permettendo agli sviluppatori di interagire con essi utilizzando concetti di programmazione orientata agli oggetti, come classi, oggetti e metodi, senza dover scrivere codice SQL a basso livello.”

ORM è strettamente legato alla specifica JPA (Jakarta Persistence API), che definisce uno standard per salvare gli oggetti Java all’interno del database e per creare oggetti Java a partire dal contenuto del database. Hibernate rappresenta l’implementazione più diffusa di JPA ed è uno dei framework open source più utilizzato in ambito enterprise.

2.7 PostgreSQL

Come riporta il sito di Amazon Web Service in [5]: “PostgreSQL è un database relazionale avanzato di livello enterprise che supporta sia SQL (relazionale) che JSON (non relazionale) per le query. È un sistema di gestione di database altamente stabile, supportato da oltre 20 anni di sviluppo da parte della comunità. Questo approccio collaborativo e approfondito ha contribuito alla sua elevata resilienza, integrità e correttezza.”

2.8 Lombok

Come riporta il sito Object Computing in [4]: “Il termine “Boilerplate” viene utilizzato per descrivere il codice ripetitivo presente in molte parti di un’applicazione con poche variazioni. Una delle critiche più comuni al linguaggio Java riguarda proprio la grande quantità di questo tipo di codice nei progetti. Questo problema è spesso causato da scelte progettuali in diverse librerie, ma è aggravato anche da alcune limitazioni del linguaggio stesso.

Project Lombok mira a ridurre la presenza di alcuni dei casi più problematici sostituendoli con un semplice set di annotazioni.”

2.9 Jackson

Come riporta il sito Medium in [2]: “Jackson è una libreria molto popolare per la gestione di JSON in Java. Fornisce una serie di strumenti per serializzare oggetti Java in JSON e deserializzare JSON in oggetti Java. Jackson è altamente personalizzabile e supporta diversi formati di dati, come XML, YAML, CSV e altri.”

Capitolo 3

Descrizione del progetto realizzato

L’architettura del progetto si basa su un modello Client-Server, che è un modello di comunicazione distribuita, in cui i client, tipicamente dispositivi o applicazioni, interagiscono con uno o più server, per ottenere delle risorse. Il principio su cui si basa questa architettura è quello della separazione dei ruoli.

Il client è il componente che richiede un servizio o dei dati al server, in questo caso è un applicativo web sviluppato in Angular. Il server, invece, è il componente che riceve le richieste da parte dei client, le elabora e successivamente risponde, restituendo il risultato, tipicamente dei dati. In questo caso il server è rappresentato da una API Rest sviluppata in Spring Boot.

Un’architettura basata sul modello Client-Server può essere implementata in vari modi, in particolare ho scelto di implementare l’architettura a tre livelli.

L’architettura a tre livelli suddivide l’applicazione in tre livelli separati tra di loro quali livello Presentazione, livello Applicazione e livello Dati.

- Livello Presentazione - FrontEnd: rappresenta l’interfaccia utente. Il suo principale scopo è far interagire l’utente col sistema, mostrando le informazioni ottenute dal livello Applicazione.
- Livello Applicazione - BackEnd: questo livello si frappone tra livello pre-

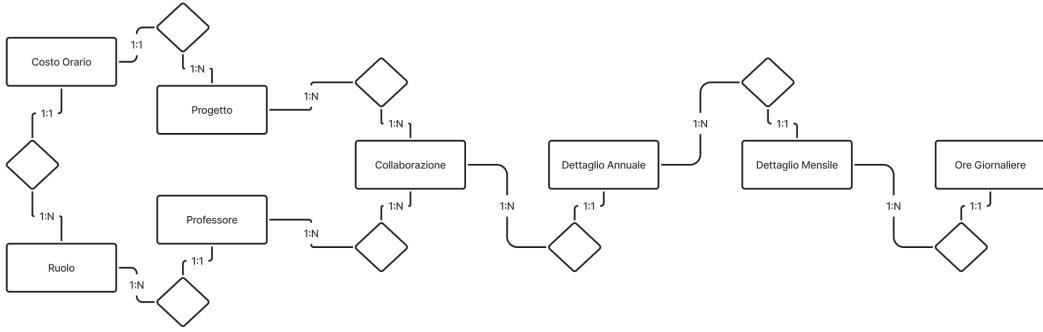


Figura 3.1: Modello entità/relazione relativo al database.

sentazione e livello dati, fungendo da ponte tra i due livelli che altrimenti non potrebbero comunicare. Il principale scopo del livello applicazione è elaborare i dati, secondo la logica al suo interno, prima di inviarli al livello presentazione o dei dati.

- **Livello Dati - Database:** è il responsabile della gestione dei dati dell'intero sistema, si occupa dell'archiviazione, manipolazione e recupero dei dati. Comunica direttamente con il livello applicazione. Questo livello è fondamentale per garantire persistenza, integrità e disponibilità dei dati.

3.1 Livello Dati

Il Livello Dati costituisce le fondamenta di tutta l'applicazione, in quanto si occupa della persistenza dei dati e della loro organizzazione.

Tale livello è strutturato in maniera tale da garantire consistenza e integrità, sfruttando un database relazionale basato su PostgreSQL. L'architettura del database è stata pensata e realizzata per garantire efficienza e flessibilità a possibili modifiche future. Viene seguito uno schema ben definito che rappresenta nel dettaglio tutte le entità e relazioni, come mostrato in Figura 3.1. Come si può vedere tutti gli elementi chiave dell'applicazione, come Progetto, Professore e Collaborazione, sono collegati tra di loro per tenere traccia del coinvolgimento dei professori nei vari progetti. Inoltre, ogni Collaborazione viene definita tramite una struttura gerarchica, che comprende il dettaglio an-

nuale, mensile e giornaliero, per garantire un alto livello di monitoraggio delle ore di lavoro di un professore e gestione di quelle che ci si aspetta che vengano impiegate sui vari progetti, oltre ad avere un'elevata trasparenza sui costi.

Inoltre, il concetto di Ruolo e Costo Orario definisce un ulteriore supporto decisionale orientato alla gestione degli assegnamenti dei professori ai progetti.

Quindi questa struttura permette di organizzare i dati in maniera scalabile, oltre a garantire efficienza nelle interrogazioni per analizzare l'allocazione delle risorse umane ed il contributo di ogni professore ai vari progetti.

3.2 Livello Applicazione - BackEnd

Il BackEnd è sviluppato in Java Spring Boot, secondo lo stile architettonico Rest, utile per la comunicazione tra sistemi distribuiti. Rest non si serve di un protocollo di comunicazione preciso, ma, tipicamente viene utilizzato il protocollo HTTP, scelta intrapresa anche per questa implementazione.

Per una giusta implementazione dello stile Rest, ci sono alcuni principi da seguire quali esporre ai client delle interfacce, rigorosamente progettate per far sì che ogni risorsa abbia un URI definito e che questa non sia troppo grande, client e server devono essere separati, indipendenti e le loro interazioni devono essere stateless, dunque ogni richiesta deve essere trattata come nuova, venendo accompagnata da tutto ciò che serve, come ad esempio informazioni relative all'autenticazione e alle autorizzazioni.

Con questo stile architettonico è possibile eseguire il caching dei dati delle risposte ed oltre alle classiche risposte in formato JSON o XML è possibile anche rispondere inviando codice eseguibile (sebbene quest'ultimo caso sia meno preferibile).

Rest permette di avere un'architettura a strati proprio come nel caso di questa implementazione del livello Applicazione, dove troviamo lo strato Controller, lo strato Service e lo strato Repository.

3.2.1 Strato Controller

Nello strato Controller troviamo le interfacce della nostra Rest API, dove vengono esposte le varie operazioni che è possibile effettuare sul Database.

Ogni interfaccia della Rest API è arricchita da queste annotazioni:

```

1 @RestController
2 @RequestMapping("/api/v1/professor")
3 @CrossOrigin(origins = "*")
4 @RequiredArgsConstructor
5 public class ProfessorController {
6     ...
7 }
```

dove `@RestController` indica che la classe è un Controller Rest, quindi gestisce le richieste HTTP e risponde in formato JSON.

`@RequestMapping` scandisce il percorso base per ogni richiesta in entrata sul controller specifico.

`@CrossOrigin(origins = "*")` consente le richieste Cross-Origin, che permettono a un dominio di fare richieste HTTP verso il server da un dominio diverso.

`@RequiredArgsConstructor` è un'annotazione fornita da Lombok e garantisce la generazione di un costruttore per tutti i campi “final” presenti all'interno della classe.

In ogni Rest Controller vengono gestite le richieste HTTP provenienti dal FrontEnd.

Il Controller riceve la richiesta, la valida e la delega al livello Service, che implementa la logica applicativa. Le principali richieste HTTP che possono sopraggiungere sono:

- GET: quando è necessario ottenere una o più risorse è necessaria questo tipo di richiesta;
- POST: viene effettuato questo tipo di richiesta quando bisogna inviare dei dati, ad esempio per aggiungere una risorsa;

- PUT/PATCH: sono entrambi tipi di richieste mirate ad aggiornare una specifica risorsa, nel caso della PUT la risorsa viene sostituita da una nuova, mentre nel caso della PATCH, la risorsa viene aggiornata parzialmente;
- DELETE: questa richiesta viene effettuata quando si vuole eliminare una specifica risorsa.

Questo che segue è il Controller Rest, relativo alle operazioni che è possibile effettuare sui singoli professori.

```

1 @RestController
2 @RequestMapping("/api/v1/Professor")
3 @CrossOrigin(origins = "*")
4 @RequiredArgsConstructor
5 public class ProfessorController {
6     private final ProfessorService professorService;
7
8     @PostMapping
9     @PreAuthorize("hasRole('ROLE_ADMIN')")
10    public ResponseEntity<Integer> createProfessor(
11        @Valid @RequestBody CreateProfessorDto createProfessorDto
12    ) {
13        Integer professorId = professorService.createProfessor(
14            createProfessorDto);
15        return ResponseEntity.status(HttpStatus.CREATED).body(
16            professorId);
17    }
18
19    @PatchMapping
20    @PreAuthorize("hasRole('ROLE_ADMIN')")
21    public ResponseEntity<Void> updateProfessor(
22        @Valid @RequestBody UpdateProfessorDto updateProfessorDto
23    ) {
24        professorService.updateProfessor(updateProfessorDto);
25        return ResponseEntity.ok().build();
26    }
27
28}
```

```

26     @GetMapping(value = "/{id}")
27     public ResponseEntity<ProfessorDto> getProfessor(
28         @PathVariable Integer id
29     ) {
30         return ResponseEntity.ok(professorService.getProfessor(
31             id));
32     }

```

In particolare, vengono esposti al FrontEnd, tre end-point.

Il primo serve a creare un nuovo professore, viene acceduto tramite una richiesta di tipo POST e nel corpo di questa dev'essere inviato l'oggetto `CreateProfessorDto` in formato JSON, l'annotazione `@Valid` indica che il DTO sarà validato in base alle annotazioni presenti sui parametri interni alla classe dell'oggetto. Se la creazione del professore avverrà con successo, verrà prodotta una risposta che presenterà: come corpo l'Id del professore appena creato e come stato della risposta il codice 201, indicando che la creazione è andata a buon fine.

Il secondo end-point viene acceduto tramite richiesta di tipo PATCH, con un oggetto `UpdateProfessorDto` in formato JSON all'interno del corpo della richiesta. Questo serve ad aggiornare parzialmente uno specifico professore. Se l'aggiornamento del professore avverrà con successo, verrà prodotta una risposta con stato 200, indicando che tutto è andato come previsto.

L'ultimo end-point esposto dal Controller Rest è il metodo per ottenere uno specifico professore e viene acceduto tramite una richiesta di tipo POST, aggiungendo al percorso dell'end-point l'id del professore di cui si vogliono ottenere i dati. Se l'Id fornito apparterrà ad un professore verrà prodotta una risposta che presenterà: come corpo l'oggetto `ProfessorDto` in formato JSON e come stato il codice 200.

Questo che segue è il Controller Rest relativo alle operazioni che è possibile effettuare sui singoli progetti.

```

1 @RestController
2 @RequestMapping("/api/v1/project")

```

```

3  @CrossOrigin(origins = "*")
4  @RequiredArgsConstructor
5  public class ProjectController {
6      private final ProjectService projectService;
7      @PostMapping
8      @PreAuthorize("hasRole('ROLE_ADMIN')")
9      public ResponseEntity<Long> createProject(
10          @Valid @RequestBody CreateProjectDto createProjectDto
11      ) {
12          return ResponseEntity
13              .status(HttpStatus.CREATED)
14              .body(projectService.createProject(
15                  createProjectDto));
16      }
17
18      @PatchMapping
19      @PreAuthorize("hasRole('ROLE_ADMIN')")
20      public ResponseEntity<Void> updateProject(
21          @Valid @RequestBody UpdateProjectDto updateProjectDto
22      ) {
23          projectService.updateProject(updateProjectDto);
24          return ResponseEntity.ok().build();
25      }
26
27      @GetMapping
28      public ResponseEntity<ProjectDto> getProject(@RequestParam
29          Long cup) {
30          return ResponseEntity.ok(projectService.getProject(cup))
31      }
32  }

```

Gli end-point esposti al FrontEnd sono tre e servono rispettivamente a creare, aggiornare e ad ottenere le informazioni di uno specifico progetto. Il funzionamento è analogo agli end-point offerti dal “ProfessorController”.

Questo che segue è il Controller Rest relativo alle operazioni che è possibile effettuare sulle collaborazioni. Sulla classe è presente l'annotazione

`@Validated` che permette di validare i parametri dei metodi prima che venga eseguita la logica al suo interno.

```

1  @RestController
2  @RequestMapping("/api/v1/collaborations")
3  @CrossOrigin(origins = "*")
4  @RequiredArgsConstructor
5  @Validated
6  public class CollaborationsController {
7      private final CollaborationsService collaborationsService;
8
9      @PostMapping(value = "/create")
10     @PreAuthorize("hasRole('ROLE_ADMIN')")
11     public ResponseEntity<Void> createCollaboration(
12         @Valid @RequestBody List<CreateCollaborationDto>
13         collaborationsDto
14     ) {
15         collaborationsService.createCollaborations(
16             collaborationsDto);
17
18         return ResponseEntity.status(HttpStatus.CREATED).build();
19     }
20
21
22     @GetMapping(value = "/professor")
23     public ResponseEntity<List<SummaryCollaborationProfessorDto>>
24         getCollaborationsByProfessor(
25             @ValidProfessorId @RequestParam Integer id
26         ) {
27
28         return ResponseEntity.ok(collaborationsService.
29             getCollaborationsByProfessorId(id));
30     }
31
32
33     @GetMapping(value = "/project")
34     public ResponseEntity<List<SummaryCollaborationProjectDto>>
35         getCollaborationsByProject(
36             @ValidProjectId @RequestParam Long cup
37         ) {
38
39         return ResponseEntity.ok(collaborationsService.

```

```

        getCollaborationsByProjectCup(cup));
    }

    @GetMapping(value = "/professors-hours")
    @PreAuthorize("hasRole('ROLE_ADMIN')")
    public ResponseEntity<List<ProfessorAssignedHoursDto>>
    getProfessorAssignedHours(
        @ValidProjectId @RequestParam Long cup
    ) {
        return ResponseEntity.ok(collaborationsService.
            getProfessorAssignedHours(cup));
    }
}

```

Sono esposti al FrontEnd quattro end-point: il primo è utile per la creazione di nuovi assegnamenti tra professori e progetti ed il funzionamento è analogo a quello dei metodi di creazione presenti nei precedenti Controller Rest.

Per quanto riguarda il secondo end-point, viene inviato tra i parametri della richiesta l'Id del professore di cui si vogliono ottenere i vari assegnamenti, che grazie all'annotazione `@Validated` viene validato secondo l'annotazione personalizzata `@ValidProfessorId`.

A volte, sorge la necessità di effettuare delle verifiche più approfondite rispetto alle validazioni offerte da Bean Validation, ad esempio di verificare la presenza di un'entità nel database. In questo caso, tramite `@ValidProfessorId` verifichiamo che l'Id ricevuto sia associato a un professore.

L'interfaccia dell'annotazione si presenta così:

```

1 @Constraint(validatedBy = ProfessorIdValidator.class)
2 @Target({ ElementType.FIELD, ElementType.METHOD, ElementType.
3         PARAMETER })
4 @Retention(RetentionPolicy.RUNTIME)
5 public @interface ValidProfessorId {
6     String message() default "Invalid Professor Id";
7     Class<?>[] groups() default {};
8     Class<? extends Payload>[] payload() default {};
9 }

```

dove `@Constraints` associa l'annotazione a un validatore specifico; `@Target` indica che l'annotazione è applicabile a campi, metodi e parametri; `@Retention` specifica che l'annotazione sarà disponibile a tempo di runtime.

Il validatore specificato con l'annotazione `@Constraints` è il seguente:

```

1 @RequiredArgsConstructor
2 public class ProfessorIdValidator implements
3     ConstraintValidator<ValidProfessorId, Integer> {
4     private final ProfessorDao professorDao;
5     @Override
6     public boolean isValid(Integer professorId,
7         ConstraintValidatorContext context) {
8         return professorDao.existsById(professorId);
9     }
10 }
```

Il metodo `isValid()` del Validatore verifica, tramite il “`ProfessorDao`”, l'esistenza di un entità professore con l'id specificato all'interno del Database, se così non fosse, il metodo `message()` restituirà il messaggio “Invalid Professor Id”.

Oltre ai Controller Rest mostrati, ne sono presenti di altri, che espongono tutte le altre operazioni che i vari client possono effettuare.

3.2.2 Strato Service

Il livello Service rappresenta il cuore della logica di applicativa. Il suo scopo principale è separare la logica applicativa dalla gestione delle richieste HTTP e dalla gestione dell'accesso ai dati rispettivamente gestiti da livello Controller e livello Dati, quindi opera da intermediario tra i due livelli garantendo come nell'architettura a tre livelli modularità, scalabilità e manutenibilità.

In breve, ogni Service incapsula la logica applicativa, riceve le richieste da parte dei Controller, elabora i dati e interagisce con il livello Repository.

Un ruolo molto importante in questo strato, è rappresentato dalla classe `ModelMapper` che aiuta a mantenere una chiara separazione tra gli oggetti uti-

lizzati all'interno dell'applicazione con quelli che vengono esposti all'esterno, semplificando quindi la trasformazione da Entity a DTO e viceversa.

Le Entity rappresentano direttamente il modello del database e non dovrebbero essere esposte direttamente all'esterno. Per evitare questo, si utilizzano i DTO (Data Transfer Object) dove all'interno si inseriscono solo i campi che è necessario esporre, evitando quindi quelli sensibili o non necessari.

Quindi, la classe `ModelMapper` si occupa di convertire in maniera automatica le Entity nei DTO che andiamo a specificare, senza dover quindi scrivere manualmente i metodi di conversione.

Qui di seguito, si riporta la configurazione della classe `ModelMapper` come Bean per poter essere iniettato ovunque sia necessario.

```

1 @Configuration
2 @RequiredArgsConstructor
3 public class ModelMapperConfig {
4     private final YearlyHoursDao yearlyHoursDao;
5     @Bean
6     public ModelMapper modelMapper() {
7         ModelMapper modelMapper = new ModelMapper();
8
9         // Mapping for ProfessorDto
10        modelMapper.addMappings(new PropertyMap<Professor,
11                               ProfessorDto>() {
12            @Override
13            protected void configure() {
14                map().setRole(source.getRole().getType());
15            }
16        });
17
18        // Mapping for CreateMonthlyHoursDto
19        modelMapper.addMappings(new PropertyMap<
20                               CreateMonthlyHoursDto, MonthlyHours>() {
21            @Override
22            protected void configure() {
23                using(ctx -> yearlyHoursDao.findById((String)
24                      ctx.getSource()).orElse(null))
25            }
26        });
27    }
28}
```

```

22         .map(source.
23             getCollaborationsHoursYearly(), destination.getYearlyHours()
24             .getId());
25     }
26
27     modelMapper.getConfiguration()
28         .setFieldMatchingEnabled(true)
29         .setFieldAccessLevel(org.modelmapper.config.
30             Configuration.AccessLevel.PUBLIC);
31
32     return modelMapper;
33 }
34 }
```

Nonostante ModelMapper esegua una conversione automatica, alcune volte è necessario aggiungere delle configurazioni specifiche. Ad esempio, qui si aggiunge una configurazione specifica per la trasformazione dell'Entity Professor in ProfessorDto, in particolare nell'Entity il campo `role` è un oggetto Role mentre nel DTO è un oggetto di tipo String. Quindi, aggiungendo questa configurazione, la classe ModelMapper ogni volta che dovrà trasferire un Professor in un ProfessorDto inserirà nel campo `role` il tipo dell'oggetto Role.

Nello strato Service tutte le operazioni che possono effettuare i vari Controller sono definite tramite delle interfacce, come mostrato di seguito.

```

1 public interface YearlyHoursService {
2     void createHoursYearly(List<CreateYearlyHoursDto>
3         createYearlyHoursDto);
4     void updateHoursYearly(List<UpdateYearlyHoursDto>
5         updateYearlyHoursDto);
6     List<YearlyDetailDto> getHoursYearly(Long projectCup);
7 }
```

Ogni interfaccia è poi implementata, insieme ai suoi metodi, da una classe che ne va a definire il comportamento, come mostrato di seguito.

```

1 @Service
2 @RequiredArgsConstructor
```

```
3 public class YearlyHoursServiceImpl implements
4     YearlyHoursService {
5
6     private final YearlyHoursDao yearlyHoursDao;
7     private final CollaborationDao collaborationDao;
8     private final ModelMapper modelMapper;
9
10    @Override
11    public void createHoursYearly(
12        List<CreateYearlyHoursDto> createYearlyHoursDto
13    ) {
14        yearlyHoursDao.saveAll(
15            createYearlyHoursDto.stream()
16                .map(dto -> {
17                    YearlyHours c = new YearlyHours();
18                    c.setCollaboration(collaborationDao
19                        .findById(dto.getCollaboration())
20                            .orElseThrow());
21                    c.setYear(dto.getYear());
22                    c.setYearExpectedHours(dto.
23                        getYearExpectedHours());
24
25                    return c;
26                })
27            .toList()
28        );
29    }
30
31    @Override
32    public void updateHoursYearly(List<UpdateYearlyHoursDto>
33        updateYearlyHoursDto) {
34        yearlyHoursDao.saveAll(
35            updateYearlyHoursDto.stream()
36                .map(dto -> {
37                    YearlyHours c = yearlyHoursDao.
38                        findById(dto.getId())
39                            .orElseThrow();
40                    c.setYearExpectedHours(dto.
41                        getYearExpectedHours());
42
43                    return c;
44                })
45            .toList()
46        );
47    }
48}
```

```

34     );
35 }
36
37 @Override
38 public List<YearlyDetailDto> getHoursYearly(Long projectCup)
39 ) {
40     return collaborationDao.findByProjectCup(projectCup)
41         .stream()
42         .map(c -> {
43             YearlyDetailDto dto = new YearlyDetailDto()
44             ;
45             dto.setCollaborationId(c.getId());
46             dto.setProfessor(modelMapper.map(c.
47                 getProfessor(), SummaryProfessorDto.class));
48             dto.setTotalExpectedHours(c.
49                 getExpectedHours());
50             dto.setCollaborationHoursYearly(
51                 yearlyHoursDao
52                     .
53                     findByCollaboration_Project_CupAndCollaboration_Professor_Id
54                     (c.getProject().get Cup(), c.getProfessor().getId())
55                     .stream().map(chy -> modelMapper.
56                     map(chy, YearlyHoursDto.class)).toList()
57                     );
58             return dto;
59         }).toList();
60     }
61 }
62 }
```

Questa classe implementa l’interfaccia `YearlyHoursService` definendo i comportamenti dei suoi tre metodi.

All’interno della classe sono iniettati la classe `ModelMapper` e i due Repository `YearlyHoursDao` e `CollaborationDao`.

Il primo metodo è utilizzato dai Controller per mappare una lista di DTO in delle Entity e successivamente delegare il salvataggio di queste nel database al Repository `YearlyHoursDao`.

Il secondo metodo si occupa di aggiornare per ogni DTO della lista passagli come parametro l'Entity corrispondente, delegando il salvataggio al Repository `YearlyHoursDao`.

L'ultimo metodo cerca nel database tutti gli assegnamenti relativi ad uno specifico progetto, restituendo per ognuno di essi il dettaglio annuale delle ore che ci si aspetta un professore lavori sul progetto, tramite il DTO `YearlyDetailDto`.

L'interfaccia riportata di seguito definisce, invece, le operazioni che si possono fare su tutti i professori.

```

1 public interface ProfessorsService {
2     Page<SummaryProfessorDto> getProfessors(Map<String, String>
3         sorting, Map<String, String> filtering, Pageable pageable);
4
5     List<SummaryProfessorDto> getAllProfessor();
}
```

La classe riportata di seguito implementa questa interfaccia.

```

1 @Service
2 @RequiredArgsConstructor
3 public class ProfessorsServiceImpl implements ProfessorsService {
4
5     private final ProfessorDao professorDao;
6     private final ModelMapper modelMapper;
7
8     @Override
9     public Page<SummaryProfessorDto> getProfessors(Map<String,
10         String> sorting, Map<String, String> filtering, Pageable
11         pageable) {
12
13         Sort sort = Sort.by(
14             Sort.Direction.fromString(sorting.getOrDefault(
15                 "direction", "ASC")),
16             sorting.getOrDefault("sortBy", "name"));
17
18         pageable = PageRequest.of(pageable.getPageNumber(),
19             pageable.getPageSize(), sort);
20     }
21 }
```

```

15     String role = filtering.getOrDefault("role", "");
16     String name = filtering.getOrDefault("name", "");
17
18     Specification<Professor> specification = Specification.
19     where(
20         ProfessorSpecifications.hasRole(role).and(
21             ProfessorSpecifications.hasNameContaining(name)));
22
23
24     @Override
25     public List<SummaryProfessorDto> getAllProfessor() {
26
27         return professorDao.findAll().stream()
28             .map(professor -> modelMapper.map(professor,
29                 SummaryProfessorDto.class))
30             .toList();
31     }
32 }
```

In questa classe, come nell'esempio precedente, vengono implementati i comportamenti dei metodi definiti nell'interfaccia. Sono presenti due metodi analoghi per ottenere tutti i professori, ma il primo implementa dei criteri di ordinamento, filtraggio e paginazione.

3.2.3 Strato Repository

L'ultimo strato è il Repository.

Questo strato opera da intermediario tra la Rest API e il Database, ed è il responsabile di tutte le operazioni che possono essere svolte sul Database. In particolare, sfruttando il framework Spring Data JPA permette di astrarre la logica di accesso ai dati evitando, nella maggior parte dei casi, di dover scrivere a mano le Query SQL usando un approccio orientato agli oggetti.

Un elemento centrale dello strato Repository sono le Entity, che rappresentano, sotto forma di classi, le tabelle del database. Ogni Entity è quindi una classe annotata con `@Entity` e mappata su una tabella tramite `@Table`, dove ogni attributo della classe rappresenta una colonna della Tabella nel database.

Le Entity vengono gestite tramite i Repository. Un esempio è riportato di seguito.

```

1  @Entity
2  @Table(name = "daily_hours_distribution")
3  @Data
4  @NoArgsConstructor
5  @EqualsAndHashCode(callSuper = true)
6  @EntityListeners(AuditingEntityListener.class)
7  public class DailyHours extends Auditable {
8      @Id
9      @UuidGenerator
10     private String id;
11
12     @Column(name = "day", nullable = false)
13     private Integer day;
14
15     @Column(name = "worked_hours", nullable = false)
16     private Integer workedHours;
17
18     @ManyToOne(fetch = FetchType.LAZY)
19     @JoinColumn(name = "collaboration_hours_monthly_id",
20     nullable = false)
21     private MonthlyHours monthlyHours;
}

```

Questa Entity rappresenta la tabella `daily hours distribution` all'interno del database. Viene gestita tramite il Repository riportato di seguito.

```

1  @Repository
2  public interface DailyHoursDao extends JpaRepository<DailyHours
   , String> {
3      List<DailyHours>
        findDailyHoursByMonthlyHours_YearlyHours_Collaboration_ProjectAndMonthlyHour
}

```

```

1   (Project project, Professor professor);

4

5     List<DailyHours>
6     findDailyHoursByMonthlyHours_MonthAndMonthlyHours_YearlyHours_YearAndMonthly
7     (
8       Month month,
9       Year year,
10      Long projectCup,
11      Integer professorId
12    );
13  }

```

Questo Repository estende la classe `JpaRepository`, dal quale eredità i metodi CRUD, inoltre definisce due metodi.

Il primo ricerca all'interno del database gli oggetti dell'Entity `DailyHours`, in base al progetto e al professore a cui sono associati.

Il secondo funziona analogamente al primo, con l'unica differenza che aggiunge come criteri di ricerca anche il mese e l'anno.

Di seguito si riporta un ulteriore esempio.

```

1 @EqualsAndHashCode(callSuper = true)
2 @Entity
3 @Table(name = "collaboration")
4 @Data
5 @NoArgsConstructor
6 @EntityListeners(AuditingEntityListener.class)
7 public class Collaboration extends Auditable {
8   @Id
9   @UuidGenerator
10  private String id;
11
12  @Column(name = "responsible", nullable = false)
13  private Boolean responsible;
14
15  @Column(name = "total expected hours", nullable = false)
16  private Integer expectedHours;
17

```

```

18     @ManyToOne(fetch = FetchType.LAZY)
19     @JoinColumn(name = "professor_id", nullable = false)
20     private Professor professor;
21
22     @ManyToOne(fetch = FetchType.LAZY)
23     @JoinColumn(name = "project_id", nullable = false)
24     private Project project;
25
26     @OneToMany(mappedBy = "collaboration", cascade =
27     CascadeType.ALL, orphanRemoval = true, fetch = FetchType.
28     LAZY)
29     @ToString.Exclude
30     private List<YearlyHours> yearlyHours;
31 }
```

Questa Entity rappresenta `Collaboration` e, in particolare, serve per la gestione degli assegnamenti dei professori nei progetti.

```

1 @Repository
2 public interface CollaborationDao extends JpaRepository<
3     Collaboration, String> {
4     List<Collaboration> findAllByProfessor_Id(Integer
5     professorId);
6     List<Collaboration> findAllByProject_Cup(Long projectCup);
7     List<Collaboration> findByResponsibleIsTrueAndProjectCup(
8     Long projectCup);
9 }
```

Nel Repository `CollaborationDao`, oltre ai metodi CRUD, sono stati definiti altri tre metodi per l'interrogazione del Database. Il primo per ottenere tutti gli assegnamenti di uno specifico professore tramite il suo Id. Il secondo per ottenere tutti gli assegnamenti di uno specifico professore a un progetto tramite il suo cup (codice unico di progetto).

Con l'ultimo si vogliono ottenere tutti gli assegnamenti di professori responsabili all'interno di un progetto tramite il suo cup.

Dunque JPA permette di evitare la scrittura di Query SQL tramite una naming convention, ma non sempre è possibile o risulta essere la scelta più giu-

sta, in questi casi possiamo utilizzare l'annotazione @Query per poter scrivere la nostra Query SQL, come mostrato nel codice riportato di seguito.

```

1 @Repository
2 public interface ProfessorDao extends JpaRepository<Professor,
   Integer>, JpaSpecificationExecutor<Professor> {
3
4     ...
5
6     @Query("SELECT p FROM Professor p WHERE p.id NOT IN (SELECT
7         c.professor.id FROM Collaboration c WHERE c.project.cup = :
8         projectCup)")
9     List<Professor> findProfessorsNotInProject(Long projectCup)
10    ;
11 }

```

In questo caso, l'utilizzo della naming convention avrebbe comportato l'obbligo di fare una doppia interrogazione al database, dunque si è scelto di scrivere direttamente una query per risolvere il problema. Tramite questa Query SQL si ottengono tutti i professori che non sono già stati assegnati ad un particolare progetto tramite il suo cup.

3.2.4 Sicurezza

Uno degli aspetti più importanti da considerare nello sviluppo di un'API, non può che essere la sicurezza.

Per individuare gli obiettivi di sicurezza di una API il modo migliore è studiare il contesto in cui si colloca l'API, concentrandosi sulle possibili minacce. Avendo ben chiare le minacce a cui può essere soggetta la nostra API, risulta poi più facile individuare gli obiettivi di sicurezza su cui concentrarsi.

Un API per poter essere considerata sicura, dunque, deve adottare dei meccanismi di difesa. I più comuni sono authentication, authorization, audit logging, rate limiting ed encryption.

Authentication

Per quanto riguarda l'authentication (autenticazione, in italiano), si vuole andare a verificare l'identità di colui che sta cercando di accedere a delle risorse. Per implementare questo aspetto di sicurezza è stato creato un meccanismo basato su JWT (Json Web Token). Un JWT è un formato per token self-contained, cioè con tutte le informazioni necessarie all'autenticazione contenute al suo interno.

```

1  @Component
2  @RequiredArgsConstructor
3  public class JwtHandler {
4
5      @Value("${jwt.secret}")
6      private String secret;
7
8      public String generateToken(Professor professor) {
9          Instant issuedAt = Instant.now().truncatedTo(ChronoUnit
10             .SECONDS);
11
12          JWTClaimsSet claims = new JWTClaimsSet.Builder()
13              .subject(professor.getEmail())
14              .claim("accessRole", professor.getAccessRole().
15                  name())
16              .issueTime(Date.from(issuedAt))
17              .expirationTime(Date.from(issuedAt.plus(24,
18                  ChronoUnit.HOURS)))
19              .build();
20
21          Payload payload = new Payload(claims.toJSONObject());
22
23          JWSObject jwsObject = new JWSObject(new JWSHeader(
24              JWSAlgorithm.HS256), payload);
25
26          try{
27              jwsObject.sign(new MACSigner(secret.getBytes()));
28          } catch (JOSEException e){
29              throw new RuntimeException("Error while generating
30              token", e);
31      }
32  }

```

```
24     }
25
26     return jwsObject.serialize();
27 }
28
29 public boolean isValidToken(String token) {
30     try {
31         SignedJWT signedJWT = SignedJWT.parse(token);
32         JWSVerifier verifier = new MACVerifier(secret.
33             getBytes());
34         if (!signedJWT.verify(verifier)) {
35             return false;
36         }
37         Date expiration = signedJWT.getJWTClaimsSet().
38             getExpirationTime();
39         return expiration == null || !expiration.before(new
40             Date());
41     } catch (Exception e) {
42         return false;
43     }
44 }
45
46 public String getJwtFromRequest(HttpServletRequest request)
47 {
48     String header = request.getHeader(HttpHeaders.
49 AUTHORIZATION);
50
51     if (header != null && header.startsWith("Bearer ")) {
52         return header.replace("Bearer ", "");
53     }
54
55     return "invalid";
56 }
57
58 public String getEmailFromToken(String token) {
59     try {
60         SignedJWT signedJWT = SignedJWT.parse(token);
61         return signedJWT.getJWTClaimsSet().getSubject();
62     } catch (Exception e) {
```

```

56         throw new RuntimeException("Invalid token");
57     }
58 }
59
60 public Date getExpirationDateFromToken(String token) {
61     try {
62         SignedJWT signedJWT = SignedJWT.parse(token);
63         return signedJWT.getJWTClaimsSet().
64             getExpirationTime();
65     } catch (Exception e) {
66         throw new RuntimeException("Invalid token");
67     }
68 }
```

Questa è la classe incaricata alla gestione dei token, gestione che concerne la creazione, validazione, e l'ottenimento dei vari claims al suo interno.

Quando andiamo a generare un JWT, partiamo creando delle informazioni da includere nel token come l'email e il ruolo d'accesso del professore, la data di creazione e la data di scadenza oltre il quale il token risulterà invalido. Successivamente, andiamo a creare un oggetto `Payload` incapsulando queste informazioni convertendole in formato JSON. Dopo ciò, andiamo a preparare il JWT, includendo nell'header l'algoritmo di firma e il `Payload` precedentemente creato. Una volta pronto, il JWT può essere firmato usando un `MACSigner`, che utilizza la chiave segreta. Infine il token firmato viene serializzato ed è quindi pronto per essere inviato al FrontEnd.

Di seguito si riporta l'implementazione del Rest Controller e del Service relativi all'autenticazione.

```

1 @RestController
2 @RequestMapping("/api/v1/auth")
3 @CrossOrigin(origins = "*", allowedHeaders = "*")
4 @RequiredArgsConstructor
5 public class AuthController {
6     private final AuthService authService;
7     private final ProfessorService professorService;
```

```

8     private final AuthenticationManager authenticationManager;
9
10    @GetMapping("/check-first-access")
11    public ResponseEntity<AccessDto> checkFirstAccess(
12        @RequestParam String email) {
13
14        return ResponseEntity
15            .status(HttpStatus.OK)
16            .body(authService.checkFirstAccess(email));
17
18    }
19
20    @PostMapping("/login")
21    public ProfessorDto login(@RequestBody AuthRequestDto
22        authRequestDto, HttpServletResponse response) {
23
24        authenticationManager.authenticate(new
25            UsernamePasswordAuthenticationToken(
26                authRequestDto.getEmail(),
27                authRequestDto.getPassword()));
28
29        String token = authService.login(authRequestDto.
30            getEmail());
31
32        response.addHeader(HttpHeaders.AUTHORIZATION, "Bearer "
33            + token);
34
35        return professorService.getProfessorDtoByEmail(
36            authRequestDto.getEmail());
37
38    }
39
40    @PostMapping("/logout")
41    public ResponseEntity<HttpStatus> logout(HttpServletRequest
42        request) {
43
44        authService.logout(request);
45
46        return ResponseEntity
47            .status(HttpStatus.OK)
48            .build();
49
50    }
51
52 }
```

```

1 @Service
2 @RequiredArgsConstructor
3 public class AuthServiceImpl implements AuthService {
4
5     private final ProfessorService professorService;
6     private final InvalidTokenService invalidTokenService;
7     private final JwtHandler jwtHandler;
```

```

7
8     @Override
9     public AccessDto checkFirstAccess(String email) {
10        Professor professor = professorService
11            .getProfessorByEmail(email);
12
13        if (professor == null)
14            throw new RuntimeException("Professor not found");
15
16        AccessDto accessDto = new AccessDto();
17        accessDto.setFirstAccess(professor.getPassword() ==
18        null || professor.getPassword().isBlank());
19        accessDto.setAccessRole(professor.getAccessRole().name
20        ());
21
22
23        return accessDto;
24    }
25
26
27    @Override
28    public String login(String email) {
29        Professor professor = professorService.
30        getProfessorByEmail(email);
31        return jwtHandler.generateToken(professor);
32    }
33
34
35    @Override
36    public void logout(HttpServletRequest request) {
37        invalidTokenService.invalidationToken(jwtHandler.
38        getJwtFromRequest(request));
39    }
40
41 }
```

Quando il FrontEnd farà una richiesta HTTP all'end-point `/login`, verrà richiamato il Service che provvederà ad autenticare l'utente in base all'email e alla password, a patto che sia presente nel database e, successivamente verrà generato un JWT che sarà incluso nell'Header della response.

Per il logout invece è stato usato un approccio a blacklist. La blacklist è

rappresentata dall'Entity `InvalidToken`, come riportato di seguito.

```

1 @Entity
2 @Table(name = "invalid_token")
3 @Data
4 @NoArgsConstructor
5 @EntityListeners(AuditingEntityListener.class)
6 public class InvalidToken {
7     @Id
8     @UuidGenerator
9     private String id;
10
11    @Column(name = "token", nullable = false, unique = true)
12    private String token;
13
14    @Column(name = "expiration_date", nullable = false)
15    private Date expirationDate;
16 }
```

Per verificare la presenza di un JWT all'interno della blacklist è stato definito un metodo all'interno del Repository `InvalidTokenDao`.

```

1 @Repository
2 public interface InvalidTokenDao extends JpaRepository<
3     InvalidToken, String> {
4     Boolean existsByToken(String token);
5 }
```

L'interfaccia `InvalidTokenService` espone i metodi di ciò che può essere eseguito sulla blacklist.

```

1 public interface InvalidTokenService {
2     void invalidationToken(String jwt);
3     Boolean isTokenInvalidated(String jwt);
4 }
```

Il Service `InvalidTokenServiceImpl` implementa l'interfaccia `InvalidTokenService` definendo il comportamento dei suoi metodi.

```

1 @Service
2 @NoArgsConstructor
```

```

3 public class InvalidTokenServiceImpl implements
4     InvalidTokenService {
5
6     private final InvalidTokenDao invalidTokenDao;
7
8     private final JwtHandler jwtHandler;
9
10    @Override
11
12    public void invalidationToken(String jwt) {
13
14        try {
15
16            InvalidToken invalidToken = new InvalidToken();
17
18            invalidToken.setToken(jwt);
19
20            invalidToken.setExpirationDate(jwtHandler.
21                getExpirationDateFromToken(jwt));
22
23            invalidTokenDao.save(invalidToken);
24
25        }
26
27        catch (Exception e) {
28
29            throw new RuntimeException("Invalid token");
30
31        }
32
33    }
34
35
36    @Override
37
38    public Boolean isTokenInvalidated(String jwt) {
39
40        return invalidTokenDao.existsByToken(jwt);
41
42    }
43
44 }
```

Quando il FrontEnd farà una richiesta HTTP all'end-point `/logout`, il JWT ricevuto verrà invalidato tramite il suo inserimento all'interno della blacklist.

Ogni volta che dal FrontEnd perviene una richiesta HTTP, questa viene intercettata dalla classe `JwtFilter`. Il compito di questa classe è di intercettare tutte le richieste, verificare la presenza e la validità del JWT e, se tutto è corretto, di autenticare l'utente nel contesto di sicurezza di Spring Boot.

```

1 @Component
2 @NonNullApi
3 @RequiredArgsConstructor
4 public class JwtFilter extends OncePerRequestFilter {
5
6
7     private final JwtHandler jwtHandler;
8
9     private final UserDetailsService userDetailsService;
```

```

8     private final InvalidTokenService invalidTokenService;
9
10
11    @Override
12    protected void doFilterInternal(HttpServletRequest request,
13                                    HttpServletResponse response, FilterChain filterChain)
14    throws ServletException, IOException {
15
16        String jwt = jwtHandler.getJwtFromRequest(request);
17
18        if (!jwt.equals("invalid") && jwtHandler.isValidToken(jwt) && !invalidTokenService.isTokenInvalidated(jwt)) {
19
20            String email = jwtHandler.getEmailFromToken(jwt);
21
22            UserDetails userDetails = userDetailsService.
23                loadUserByUsername(email); //considero l'email come username
24
25            UsernamePasswordAuthenticationToken authentication
26            = new UsernamePasswordAuthenticationToken(userDetails, null,
27                userDetails.getAuthorities());
28
29            authentication.setDetails(new
30                WebAuthenticationDetailsSource().buildDetails(request));
31
32            SecurityContextHolder.getContext().
33                setAuthentication(authentication);
34
35        }
36
37        filterChain.doFilter(request, response);
38
39    }
40
41 }

```

Nello specifico, nel metodo `doFilterInternal` viene estratto il JWT dalla richiesta e si verifica se questo è effettivamente presente nella richiesta, scaduto oppure nella blacklist. Dopodiché, si estraе l'email dell'utente dal JWT e con questa viene recuperato l'utente dal database. Infine, viene creato l'oggetto `UsernamePasswordAuthenticationToken`, inserendo utente e permessi, per poi essere registrato nel `SecurityContextHolder` così che Spring possa riconoscere l'utente per quella richiesta.

Nei casi in cui il JWT della richiesta non superi i controlli, quindi un utente sta cercando di accedere in modo non legittimo ad un risorsa protetta, Spring Boot reindirizza automaticamente la richiesta alla classe `JwtEntryPoint`.

```

1 @Component
2 @RequiredArgsConstructor

```

```

3 public class JwtEntryPoint implements AuthenticationEntryPoint
4 {
5
6     @Override
7     public void commence(HttpServletRequest request,
8             HttpServletResponse response, AuthenticationException
9             authException) throws IOException {
10
11         response.setStatus(HttpServletResponse.SC_UNAUTHORIZED)
12
13         response.getWriter().write("Unauthorized:
14             Authentication token is missing or invalid");
15         response.getWriter().flush();
16     }
17 }
```

In particolare viene richiamato il metodo `commence` che va ad impostare come stato della response il codice 401, cioè “Unauthorized”, oltre ad un messaggio di errore che indica l’invalidità del JWT. Con questo approccio andiamo a gestire tutte le richieste in entrata, autorizzando solo quelle legittime.

Authorization

Per la gestione dei livelli di accesso alle risorse da parte di ogni utente possono essere intraprese differenti strade, in questo caso è stato implementato un meccanismo basato sui ruoli. Ad ogni utente è assegnato un ruolo che va a specificare quali risorse può o non può accedere.

```

1 @Configuration
2 @EnableWebSecurity
3 @EnableMethodSecurity
4 @RequiredArgsConstructor
5 public class SecurityConfig {
6
7     private final JwtFiler jwtFiler;
8     private final JwtEntryPoint jwtEntryPoint;
9
10    @Bean
```

```
11     public AuthenticationManager authenticationManager( AuthenticationConfiguration authenticationConfiguration)
12         throws Exception {
13
14             return authenticationConfiguration.
15         getAuthenticationManager();
16
17     }
18
19
20     @Bean
21
22     public PasswordEncoder passwordEncoder() {
23
24         return new BCryptPasswordEncoder(12);
25
26     }
27
28
29     @Bean
30
31     public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
32
33         http
34
35             .cors(cors -> cors.configurationSource(request
36             -> {
37
38                 CorsConfiguration corsConfiguration = new
39                 CorsConfiguration();
40
41                 corsConfiguration.setAllowedOrigins(List.of(
42                     ("*")));
43
44                 corsConfiguration.setAllowedMethods(Arrays.
45                     asList("GET", "POST", "PATCH", "PUT", "DELETE", "OPTIONS"));
46
47                 corsConfiguration.setAllowedHeaders(Arrays.
48                     asList("Authorization", "Content-Type", "Accept"));
49
50                 corsConfiguration.setExposedHeaders(Arrays.
51                     asList("Access-Control-Allow-Origin", "Access-Control-Allow-
52                     Credentials", "Authorization", "Access-Control-Request-
53                     Method", "Access-Control-Request-Headers"));
54
55                 return corsConfiguration;
56             })
57
58             .csrf(AbstractHttpConfigurer::disable)
59
60             .sessionManagement(session -> session.
61                 sessionCreationPolicy(SessionCreationPolicy.STATELESS))
62
63             .authorizeHttpRequests(auth -> auth
```

```

34         .requestMatchers("/api/v1/auth/login",
35             "/api/v1/auth/check-first-access").permitAll()
36             .requestMatchers("/api/v1/password-set
37             /**").permitAll()
38             .anyRequest().authenticated()
39         )
40         .exceptionHandling(ex -> ex.
41             authenticationEntryPoint(jwtEntryPoint))
42             .addFilterBefore(jwtFiler,
43                 UsernamePasswordAuthenticationFilter.class);
44
45     return http.build();
46 }
47 }
```

In questa classe viene specificato quali sono gli end-point accessibili senza la necessità di avere un token, cioè: login, verifica del primo accesso e l'end-point per il cambio password. Per tutti gli altri end-point bisogna inserire nell'Header della richiesta un JWT valido. Per gestire quali end-point sono esclusivamente accessibili dagli Admin è stato utilizzata l'annotazione `@PreAuthorized`.

```

1 @RestController
2 @RequestMapping("/api/v1/professor")
3 @CrossOrigin(origins = "*")
4 @RequiredArgsConstructor
5 public class ProfessorController {
6     private final ProfessorService professorService;
7
8     @PostMapping
9     @PreAuthorize("hasRole('ROLE_ADMIN')")
10    public ResponseEntity<Integer> createProfessor(@Valid
11        @RequestBody CreateProfessorDto createProfessorDto) {
12        Integer professorId = professorService.createProfessor(
13            createProfessorDto);
14
15        return ResponseEntity.status(HttpStatus.CREATED).body(
16            professorId);
17    }
18
19    @PatchMapping
```

```

16     @PreAuthorize("hasRole('ROLE_ADMIN')")
17     public ResponseEntity<Void> updateProfessor(@Valid
18         @RequestBody UpdateProfessorDto updateProfessorDto) {
19
20         professorService.updateProfessor(updateProfessorDto);
21
22         return ResponseEntity.ok().build();
23     }
24
25
26 }
```

In questo Controller è presente `@PreAuthorize("hasRole('ROLE_ADMIN')")` sugli end-point per la creazione e per la modifica di un professore, mentre sull'end-point per ottenere uno specifico professore non è presente. Questo va a specificare che i primi due end-point sono accessibili solo da utenti con ruolo Admin.

Audit Logging

Un aspetto di sicurezza molto importante è garantire la non ripudiabilità delle azioni che vengono effettuate. Per fare ciò è stato implementato un meccanismo di audit logging, che permette di registrare chi ha fatto cosa e quando ciò è accaduto.

Questa implementazione sfrutta Spring Data JPA Auditing che, consente di registrare i dettagli delle operazioni che avvengono sugli oggetti persistenti in maniera automatica, evitando di dover scrivere manualmente queste informazioni in ogni entità.

```

1 @Configuration
2 @RequiredArgsConstructor
3 public class AuditorAwareConfig {
4
5     private final AuditorAwareImpl auditorAware;
6
7     @Bean
```

```

6     public AuditorAware<String> auditorProvider() {
7
8         return auditorAware;
9     }

```

Questa appena mostrata è la classe di configurazione del meccanismo di audit logging, dove viene registrato un `@Bean` che fornisce l'email dell'utente che sta eseguendo l'operazione. Per fare ciò viene utilizzata la classe `AuditorAwareImpl`, mostrata di seguito.

```

1 @Component
2 @RequiredArgsConstructor
3 @NonNullApi
4 public class AuditorAwareImpl implements AuditorAware<String> {
5
6     private final UserDetailsServiceImpl userDetailsService;
7
8     @Override
9     public Optional<String> getCurrentAuditor() {
10
11         try {
12             return Optional.of(userDetailsService.
13                 getCurrentUserEmail());
14         } catch (Exception e) {
15             return Optional.of("system");
16         }
17     }
18 }

```

La classe `AuditorAwareImpl` prova a ricavare l'utente autenticato, se non dovesse riuscirci allora l'operazione è effettuata dal sistema, dunque restituisce il testo `system`.

Per salvare le informazioni di auditing nel database è stata definita la classe astratta `Auditale`, mostrata di seguito.

```

1 @Getter
2 @Setter
3 @MappedSuperclass
4 @EntityListeners(AuditingEntityListener.class)
5 public abstract class Auditable {
6
7     @CreatedBy

```

```

7     private String createdBy;
8
9     @LastModifiedBy
10    private String lastModifiedBy;
11
12    @CreatedDate
13    @Temporal(TemporalType.TIMESTAMP)
14    private Date createdDate;
15
16    @LastModifiedDate
17    @Temporal(TemporalType.TIMESTAMP)
18    private Date lastModifiedDate;
19 }
```

Questa classe viene ereditata da tutte le Entity che devono essere tracciate. Le annotazioni `@MappedSuperclass` e `@EntityListeners(AuditingEntityListener.class)` specificano rispettivamente che la classe non ha una propria tabella sul database e che l'auditing automatico è attivo.

Un esempio di Entity che eredità dalla classe astratta `Audit` è riportata di seguito.

```

1 @EqualsAndHashCode(callSuper = true)
2 @Entity
3 @Table(name = "role")
4 @Data
5 @NoArgsConstructor
6 @EntityListeners(AuditingEntityListener.class)
7 public class Role extends Audit {
8     @Id
9     @UuidGenerator
10    private String id;
11
12    @Column(name = "type", nullable = false, unique = true)
13    private String type;
14
15    @OneToMany(mappedBy = "role", cascade = CascadeType.ALL,
16    orphanRemoval = true, fetch = FetchType.LAZY)
17    @ToString.Exclude
```

```

17     private List<Professor> professors;
18 }
```

Rate Limiting

In una strategia di sicurezza di una Rest API implementare il Rate Limiting risulta essere essenziale. Il Rate limiting protegge la Rest API da abusi come il “Denial of Service” (DoS), dove, ad esempio, vengono effettuate molteplici richieste mirate a sovraccaricare il sistema rendendolo inaccessibile.

Il Rate Limiting impone dei limiti di richieste per ogni client, in questo caso 10 al secondo, per far sì che attacchi di tipo DoS siano inefficaci. Oltre al meccanismo di protezione, il Rate Limiting garantisce sia una distribuzione equa delle risorse per ogni utente che un mantenimento delle performance del sistema.

```

1 @Aspect
2 @Component
3 @RequiredArgsConstructor
4 public class RateLimitingAspect {
5
6     private final ConcurrentHashMap<Method, RateLimiter>
7         rateLimiterMap = new ConcurrentHashMap<>();
8
9     @Around("@within(org.springframework.web.bind.annotation.
10          RestController)")
11     public Object limitRequestRate(ProceedingJoinPoint
12         joinPoint) throws Throwable {
13         Method method= ((MethodSignature) joinPoint.
14             getSignature()).getMethod();
15         RateLimit rateLimited = method.getAnnotation(RateLimit.
16             class);
17         if (rateLimited == null) {
18             rateLimited = joinPoint.getTarget().getClass().
19                 getAnnotation(RateLimit.class);
20         }
21         if (rateLimited != null) {
```

```

16         final double permitsPerSecond = rateLimited.
17             permitsPerSecond();
18
19             RateLimiter rateLimiter = rateLimiterMap.
20             computeIfAbsent(method, key -> RateLimiter.create(
21                 permitsPerSecond));
22
23             if (rateLimiter.tryAcquire()) {
24                 return joinPoint.proceed();
25             } else {
26                 throw new RuntimeException("Too many requests")
27             ;
28         }
29     }
30 }
```

L'implementazione del Rate Limiting sfrutta il concetto di programmazione orientata agli aspetti per intercettare tutte le chiamate ai Controller Rest controllando il numero di richieste consentite. Ogni qual volta il metodo di un Controller Rest viene eseguito, l'aspetto verifica se è presente l'annotazione personalizzata `RateLimit`, mostrata di seguito:

```

1 @Retention(RetentionPolicy.RUNTIME)
2 @Target(ElementType.TYPE)
3 public @interface RateLimit {
4     double permitsPerSecond();
5 }
```

Tramite l'uso di questa annotazione sui Controller Rest possiamo specificare il limite di richieste consentite al secondo. Se l'annotazione è presente, allora viene utilizzata una mappa per associare un'istanza di `RateLimiter` al metodo chiamato. Durante l'esecuzione l'aspetto tenta di ottenere un permesso dal `RateLimiter`, se viene negato vuol dire che il limite di richieste è stato raggiunto, altrimenti il metodo continua la sua naturale esecuzione.

Il Rate Limiter quindi opera da filtro, evitando un eccessivo afflusso di

richieste con conseguente sovraccarico del sistema che andrebbe ad abbattere prestazioni e disponibilità delle risorse.

3.3 FrontEnd (Livello Presentazione)

Il livello Presentazione è stato realizzato in Angular e si occupa dell’interazione tra utente e livello Applicazione, sfruttando gli strumenti offerti da Angular. Per la comunicazione tra i due livelli, viene usato il modulo `HttpClient`, che consente di effettuare richieste HTTP alle API Rest offerte dal livello applicazione, recuperando così, i dati da mostrare o inviare quelli forniti dall’utente.

Un esempio di richieste HTTP effettuate grazie all’utilizzo dell’`HttpClient` è contenuto nell’`AuthService`, il cui codice è mostrato di seguito.

```

1  @Injectable({
2      providedIn: 'root'
3  })
4  export class AuthService {
5      private apiUrl = 'http://localhost:8080/api/v1/auth';
6
7      constructor(private http: HttpClient, private router: Router)
8      {
9
10
11      checkFirstAccess(email: string) {
12          return this.http.get<AccessDto>(this.apiUrl + '/check-first
13          -access', {params: {email}});
14
15      login(authRequest: AuthRequestDto) {
16          return this.http.post<Professor>(this.apiUrl + '/login',
17          authRequest, {observe: 'response'});
18
19      logout() {

```

```

20     if (this.getAccessToken() !== null) {
21
22         const headers = new HttpHeaders({
23             'Content-Type': 'application/json',
24             'Authorization': 'Bearer ' + this.getAccessToken()
25         })
26
27         return this.http.post(this.apiUrl + '/logout', null, {
28             headers, observe: 'response'
29         })
30
31     else throw new Error('Authentication required to logout');
32 }
33
34 makeLogout() {
35
36     this.logout().subscribe({
37         next: () => {
38             this.removeItemFromStorage();
39             this.router.navigate(['login']).then();
40         }
41     });
42 }
```

Per garantire, che le risorse vengano fruite solo dagli utenti legittimi, è stato implementato un meccanismo di controllo accessi tramite `AuthGuard`.

Il compito dell'`AuthGuard` è quello di verificare che l'utente sia autenticato prima di consentire l'accesso ad una determinata pagina dell'applicazione.

Le rotte alle varie pagine dell'applicazione sono mostrate di seguito.

```

1 export const routes: Routes = [
2     {path: 'set-password', component: ResetPasswordComponent},
3     {path: 'login', component: LoginComponent},
4     {path: '', component: LeftbarComponent, children: [
5         {path: '', redirectTo: 'personal-area', pathMatch: 'full'}
6     ]},
```

```

6      { path: 'projects', component: ProjectsComponent,
7        canActivate: [authGuard] },
8
9      { path: 'project', component: ProjectComponent,
10        canActivate: [authGuard] },
11
12     { path: 'add-project', component: AddProjectsComponent,
13       canActivate: [authGuard] },
14
15     { path: 'professors', component: ProfessorsComponent,
16       canActivate: [authGuard] },
17
18     { path: 'professor', component: ProfessorComponent,
19       canActivate: [authGuard] },
20
21     { path: 'add-professor', component:
22       AddProfessorsComponent, canActivate: [authGuard] },
23
24     { path: 'collaboration', component:
25       CollaborationComponent, canActivate: [authGuard] },
26
27     { path: 'yearly-detail', component: YearlyHoursComponent,
28       canActivate: [authGuard] },
29
30     { path: 'monthly-detail', component: MonthHoursComponent,
31       canActivate: [authGuard] },
32
33     { path: 'calendar', component: CalendarComponent,
34       canActivate: [authGuard] },
35
36     { path: 'personal-area', component:
37       CustomizationComponent, canActivate: [authGuard] }
38   }
39 ];

```

Come si può notare, escluse le pagine per effettuare l'accesso, tutte le rotte sono protette dall'AuthGuard, definita attraverso il codice riportato di seguito.

```

1 export const authGuard: CanActivateFn = (route, state) => {
2
3   const router = inject(Router);
4
5   const authService = inject(AuthService);
6
7
8   if (!authService.getAccessToken()) {
9     router.navigate(['/login'], {queryParams: {returnUrl: state
10 .url}}).then();
11
12     return false;
13   }
14
15   return true;

```

10 };

In particolare, l'`AuthGuard` intercetta le richieste di accesso alle varie rotte, verificando che l'utente che vuole compiere tale azione abbia a disposizione un JWT, se così non fosse, questo verrà reindirizzato alla schermata per effettuare l'accesso. Se l'utente effettuerà l'accesso con successo, allora verrà automaticamente reindirizzato alla pagina associata alla rotta che stava cercando di accedere.

Un altro componente fondamentale è l'`Interceptor`. Questo svolge il suo compito parallelamente all'`AuthGuard`. Se con l'`AuthGuard` viene verificata la presenza del JWT, con l'`Interceptor` si provvede ad intercettare tutte le richieste HTTP, arricchendo gli Header delle informazioni relative all'autenticazione dell'utente che le sta effettuando.

```

1 export function tokenInterceptor(req: HttpRequest<unknown>,
2   next: HttpHandlerFn): Observable<HttpEvent<any>> {
3   const authService = inject(AuthService);
4   const router = inject(Router);
5
6   if (authService.getAccessToken() && !req.headers.has(
7     'Authorization')) {
8     req = req.clone({
9       setHeaders: {
10         Authorization: "Bearer " + authService.getAccessToken()
11       }
12     });
13
14   return next(req).pipe(
15     catchError((error: HttpErrorResponse) => {
16       if (error.status === 401) {
17         authService.removeItemFromStorage()
18         router.navigate(['/login']).then();
19       }
20       return throwError(() => error);
21     })
22   );
23 }
```

22 }

Oltre ad intercettare le risposte provenienti dal livello Applicazione, l'`Interceptor`, gestisce in maniera centralizzata gli eventuali errori di tipo “Unauthenticated”. In tal caso l'`Interceptor` provvede alla rimozione di tutte le informazioni contenute nel “LocalStorage”, reinderizzando l’utente alla schermata di accesso.

Grazie all’approccio modulare di Angular, basato su componenti, mi è stato possibile (insieme ad un’oculata progettazione, basata sulle regole di UI e UX, delle varie schermate utilizzando Figma), creare interfacce facilmente estendibili e riutilizzabili oltre che a migliorare la scalabilità complessiva del sistema.

3.3.1 Accesso

La schermata di accesso, riportata in Figura 3.2, è ciò che viene mostrato ad un utente che non ha un JWT valido per poter entrare nel sistema. La schermata è stata pensata per offrire all’utente un meccanismo di accesso a due passaggi. Il primo passaggio consiste nel fornire la propria email.

Con un click sul pulsante “Avanti” verrà eseguito il metodo riportato di seguito.

```

1 initLogin(){
2     this.email = this.form.get('email')!.value;
3
4     this.authService.checkFirstAccess(this.email).subscribe({
5         next: (response) => {
6             if (response.firstAccess)
7                 this.firstAccess = true
8             else this.toPassword = true;
9         },
10        error: () => {
11            this.isError = true;
12            this.message = "Errore, riprova piu' tardi.";
13            this.showToast = true;
14        }

```

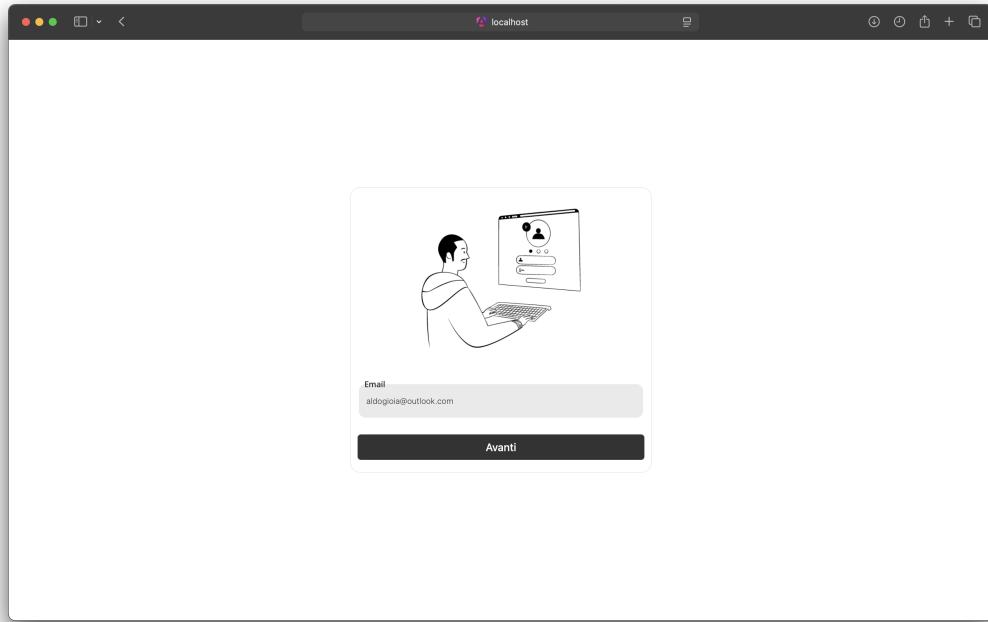


Figura 3.2: Inserimento dell'email nella schermata di login.

```

15     });
16
17     setTimeout(() => {
18       this.showToast = false;
19     }, 3000);
20   }

```

Il metodo `initLogin()` effettuerà una richiesta HTTP con cui si invia l'email fornita dall'utente per verificare se si tratta della prima volta in cui tenta di accedere al sistema.

Quindi, il sistema verifica che l'utente associato all'email abbia già effettuato altre volte l'accesso al sistema. Se così fosse, gli verrà chiesto di inserire la propria password, come mostrato nella Figura 3.3.

Premendo su “Accedi” viene eseguito il metodo `login()`, riportato di seguito.

```

1 login() {
2   const email: string = this.email;
3   const password: string = this.form.get('password')!.value;
4

```

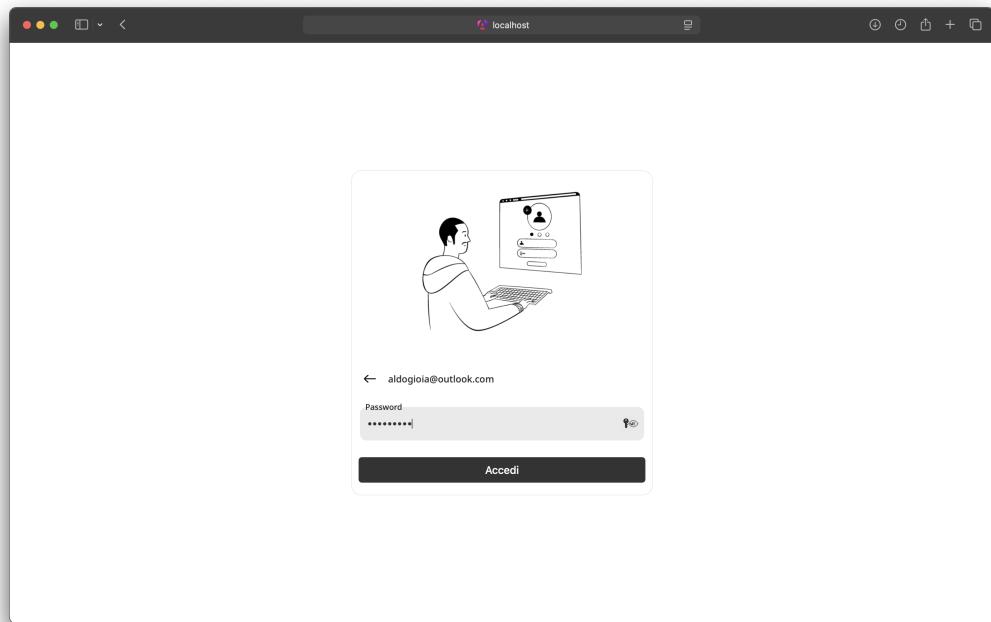


Figura 3.3: Inserimento della password nella schermata di login.

```
5      this.authService.login(  
6          new AuthRequestDto({email, password})  
7      ).subscribe(  
8          next: (response) => {  
9              const authorization = response.headers.get('  
Authorization')  
10  
11              if (authorization) {  
12                  const accessToken = authorization.replace('Bearer ',  
'');  
13                  this.authService.setItemInStorage(accessToken,  
response.body!);  
14              }  
15  
16              this.router.navigateByUrl(  
17                  this.route.snapshot.queryParams['returnUrl'] || '/'  
18              ).then();  
19          },  
20          error: () => {  
21              this.form.get('password')!.setValue('');
```

```

22         this.isError = true;
23
24         this.message = "Email o password errati. Riprova grazie
25         .";
26
27         this.showToast = true;
28     }
29
30     });
31
32     setTimeout(() => {
33
34         this.showToast = false;
35     }, 3000);
36
37 }
38
39 }
40
41 }
42
43 }
44
45 }
46
47 }
48
49 }
50
51 }
52
53 }
54
55 }
56
57 }
58
59 }
60
61 }
62
63 }
64
65 }
66
67 }
68
69 }
70
71 }
72
73 }
74
75 }
76
77 }
78
79 }
80
81 }
82
83 }
84
85 }
86
87 }
88
89 }
90
91 }
92
93 }
94
95 }
96
97 }
98
99 }
100
101 }
102
103 }
104
105 }
106
107 }
108
109 }
110
111 }
112
113 }
114
115 }
116
117 }
118
119 }
120
121 }
122
123 }
124
125 }
126
127 }
128
129 }
130
131 }
132
133 }
134
135 }
136
137 }
138
139 }
140
141 }
142
143 }
144
145 }
146
147 }
148
149 }
150
151 }
152
153 }
154
155 }
156
157 }
158
159 }
160
161 }
162
163 }
164
165 }
166
167 }
168
169 }
170
171 }
172
173 }
174
175 }
176
177 }
178
179 }
180
181 }
182
183 }
184
185 }
186
187 }
188
189 }
190
191 }
192
193 }
194
195 }
196
197 }
198
199 }
200
201 }
202
203 }
204
205 }
206
207 }
208
209 }
210
211 }
212
213 }
214
215 }
216
217 }
218
219 }
220
221 }
222
223 }
224
225 }
226
227 }
228
229 }
230
231 }
232
233 }
234
235 }
236
237 }
238
239 }
240
241 }
242
243 }
244
245 }
246
247 }
248
249 }
250
251 }
252
253 }
254
255 }
256
257 }
258
259 }
259
260 }
261
262 }
263
264 }
265
266 }
267
268 }
269
270 }
271
272 }
273
274 }
275
276 }
277
278 }
279
280 }
281
282 }
283
284 }
285
286 }
287
288 }
289
290 }
291
292 }
293
294 }
295
296 }
297
298 }
299
299
300 }
301
302 }
303
304 }
305
306 }
307
308 }
309
309
310 }
311
312 }
313
314 }
315
316 }
317
318 }
319
319
320 }
321
322 }
323
324 }
325
326 }
327
328 }
329
329
330 }
331
332 }
333
334 }
335
336 }
337
338 }
339
339
340 }
341
342 }
343
344 }
345
346 }
347
348 }
349
349
350 }
351
352 }
353
354 }
355
356 }
357
358 }
359
359
360 }
361
362 }
363
364 }
365
366 }
367
368 }
369
369
370 }
371
372 }
373
374 }
375
376 }
377
378 }
379
379
380 }
381
382 }
383
384 }
385
386 }
387
388 }
389
389
390 }
391
392 }
393
394 }
395
396 }
397
398 }
399
399
400 }
401
402 }
403
404 }
405
406 }
407
408 }
409
409
410 }
411
412 }
413
414 }
415
416 }
417
418 }
419
419
420 }
421
422 }
423
424 }
425
426 }
427
428 }
429
429
430 }
431
432 }
433
434 }
435
436 }
437
438 }
439
439
440 }
441
442 }
443
444 }
445
446 }
447
448 }
449
449
450 }
451
452 }
453
454 }
455
456 }
457
458 }
459
459
460 }
461
462 }
463
464 }
465
466 }
467
468 }
469
469
470 }
471
472 }
473
474 }
475
476 }
477
478 }
479
479
480 }
481
482 }
483
484 }
485
486 }
487
488 }
489
489
490 }
491
492 }
493
494 }
495
496 }
497
498 }
499
499
500 }
501
502 }
503
504 }
505
506 }
507
508 }
509
509
510 }
511
512 }
513
514 }
515
516 }
517
518 }
519
519
520 }
521
522 }
523
524 }
525
526 }
527
528 }
529
529
530 }
531
532 }
533
534 }
535
536 }
537
538 }
539
539
540 }
541
542 }
543
544 }
545
546 }
547
548 }
549
549
550 }
551
552 }
553
554 }
555
556 }
557
558 }
559
559
560 }
561
562 }
563
564 }
565
566 }
567
568 }
569
569
570 }
571
572 }
573
574 }
575
576 }
577
578 }
579
579
580 }
581
582 }
583
584 }
585
586 }
587
588 }
589
589
590 }
591
592 }
593
594 }
595
596 }
597
598 }
599
599
600 }
601
602 }
603
604 }
605
606 }
607
608 }
609
609
610 }
611
612 }
613
614 }
615
616 }
617
618 }
619
619
620 }
621
622 }
623
624 }
625
626 }
627
628 }
629
629
630 }
631
632 }
633
634 }
635
636 }
637
638 }
639
639
640 }
641
642 }
643
644 }
645
646 }
647
648 }
649
649
650 }
651
652 }
653
654 }
655
656 }
657
658 }
659
659
660 }
661
662 }
663
664 }
665
666 }
667
668 }
669
669
670 }
671
672 }
673
674 }
675
676 }
677
678 }
679
679
680 }
681
682 }
683
684 }
685
686 }
687
688 }
689
689
690 }
691
692 }
693
694 }
695
696 }
697
698 }
699
699
700 }
701
702 }
703
704 }
705
706 }
707
708 }
709
709
710 }
711
712 }
713
714 }
715
716 }
717
718 }
719
719
720 }
721
722 }
723
724 }
725
726 }
727
728 }
729
729
730 }
731
732 }
733
734 }
735
736 }
737
738 }
739
739
740 }
741
742 }
743
744 }
745
746 }
747
748 }
749
749
750 }
751
752 }
753
754 }
755
756 }
757
758 }
759
759
760 }
761
762 }
763
764 }
765
766 }
767
768 }
769
769
770 }
771
772 }
773
774 }
775
776 }
777
778 }
779
779
780 }
781
782 }
783
784 }
785
786 }
787
788 }
789
789
790 }
791
792 }
793
794 }
795
796 }
797
798 }
799
799
800 }
801
802 }
803
804 }
805
806 }
807
808 }
809
809
810 }
811
812 }
813
814 }
815
816 }
817
818 }
819
819
820 }
821
822 }
823
824 }
825
826 }
827
828 }
829
829
830 }
831
832 }
833
834 }
835
836 }
837
838 }
839
839
840 }
841
842 }
843
844 }
845
846 }
847
848 }
849
849
850 }
851
852 }
853
854 }
855
856 }
857
858 }
859
859
860 }
861
862 }
863
864 }
865
866 }
867
868 }
869
869
870 }
871
872 }
873
874 }
875
876 }
877
878 }
879
879
880 }
881
882 }
883
884 }
885
886 }
887
888 }
889
889
890 }
891
892 }
893
894 }
895
896 }
897
898 }
899
899
900 }
901
902 }
903
904 }
905
906 }
907
908 }
909
909
910 }
911
912 }
913
914 }
915
916 }
917
918 }
919
919
920 }
921
922 }
923
924 }
925
926 }
927
928 }
929
929
930 }
931
932 }
933
934 }
935
936 }
937
938 }
939
939
940 }
941
942 }
943
944 }
945
946 }
947
948 }
949
949
950 }
951
952 }
953
954 }
955
956 }
957
958 }
959
959
960 }
961
962 }
963
964 }
965
966 }
967
968 }
969
969
970 }
971
972 }
973
974 }
975
976 }
977
978 }
979
979
980 }
981
982 }
983
984 }
985
986 }
987
988 }
989
989
990 }
991
992 }
993
994 }
995
996 }
997
998 }
999
999
1000 }
1001
1002 }
1003
1004 }
1005
1006 }
1007
1008 }
1009
1009
1010 }
1011
1012 }
1013
1014 }
1015
1016 }
1017
1018 }
1019
1019
1020 }
1021
1022 }
1023
1024 }
1025
1026 }
1027
1028 }
1029
1029
1030 }
1031
1032 }
1033
1034 }
1035
1036 }
1037
1038 }
1039
1039
1040 }
1041
1042 }
1043
1044 }
1045
1046 }
1047
1048 }
1049
1049
1050 }
1051
1052 }
1053
1054 }
1055
1056 }
1057
1058 }
1059
1059
1060 }
1061
1062 }
1063
1064 }
1065
1066 }
1067
1068 }
1069
1069
1070 }
1071
1072 }
1073
1074 }
1075
1076 }
1077
1078 }
1079
1079
1080 }
1081
1082 }
1083
1084 }
1085
1086 }
1087
1088 }
1089
1089
1090 }
1091
1092 }
1093
1094 }
1095
1096 }
1097
1098 }
1099
1099
1100 }
1101
1102 }
1103
1104 }
1105
1106 }
1107
1108 }
1109
1109
1110 }
1111
1112 }
1113
1114 }
1115
1116 }
1117
1118 }
1119
1119
1120 }
1121
1122 }
1123
1124 }
1125
1126 }
1127
1128 }
1129
1129
1130 }
1131
1132 }
1133
1134 }
1135
1136 }
1137
1138 }
1139
1139
1140 }
1141
1142 }
1143
1144 }
1145
1146 }
1147
1148 }
1149
1149
1150 }
1151
1152 }
1153
1154 }
1155
1156 }
1157
1158 }
1159
1159
1160 }
1161
1162 }
1163
1164 }
1165
1166 }
1167
1168 }
1169
1169
1170 }
1171
1172 }
1173
1174 }
1175
1176 }
1177
1178 }
1179
1179
1180 }
1181
1182 }
1183
1184 }
1185
1186 }
1187
1188 }
1189
1189
1190 }
1191
1192 }
1193
1194 }
1195
1196 }
1197
1198 }
1199
1199
1200 }
1201
1202 }
1203
1204 }
1205
1206 }
1207
1208 }
1209
1209
1210 }
1211
1212 }
1213
1214 }
1215
1216 }
1217
1218 }
1219
1219
1220 }
1221
1222 }
1223
1224 }
1225
1226 }
1227
1228 }
1229
1229
1230 }
1231
1232 }
1233
1234 }
1235
1236 }
1237
1238 }
1239
1239
1240 }
1241
1242 }
1243
1244 }
1245
1246 }
1247
1248 }
1249
1249
1250 }
1251
1252 }
1253
1254 }
1255
1256 }
1257
1258 }
1259
1259
1260 }
1261
1262 }
1263
1264 }
1265
1266 }
1267
1268 }
1269
1269
1270 }
1271
1272 }
1273
1274 }
1275
1276 }
1277
1278 }
1279
1279
1280 }
1281
1282 }
1283
1284 }
1285
1286 }
1287
1288 }
1289
1289
1290 }
1291
1292 }
1293
1294 }
1295
1296 }
1297
1298 }
1299
1299
1300 }
1301
1302 }
1303
1304 }
1305
1306 }
1307
1308 }
1309
1309
1310 }
1311
1312 }
1313
1314 }
1315
1316 }
1317
1318 }
1319
1319
1320 }
1321
1322 }
1323
1324 }
1325
1326 }
1327
1328 }
1329
1329
1330 }
1331
1332 }
1333
1334 }
1335
1336 }
1337
1338 }
1339
1339
1340 }
1341
1342 }
1343
1344 }
1345
1346 }
1347
1348 }
1349
1349
1350 }
1351
1352 }
1353
1354 }
1355
1356 }
1357
1358 }
1359
1359
1360 }
1361
1362 }
1363
1364 }
1365
1366 }
1367
1368 }
1369
1369
1370 }
1371
1372 }
1373
1374 }
1375
1376 }
1377
1378 }
1379
1379
1380 }
1381
1382 }
1383
1384 }
1385
1386 }
1387
1388 }
1389
1389
1390 }
1391
1392 }
1393
1394 }
1395
1396 }
1397
1398 }
1399
1399
1400 }
1401
1402 }
1403
1404 }
1405
1406 }
1407
1408 }
1409
1409
1410 }
1411
1412 }
1413
1414 }
1415
1416 }
1417
1418 }
1419
1419
1420 }
1421
1422 }
1423
1424 }
1425
1426 }
1427
1428 }
1429
1429
1430 }
1431
1432 }
1433
1434 }
1435
1436 }
1437
1438 }
1439
1439
1440 }
1441
1442 }
1443
1444 }
1445
1446 }
1447
1448 }
1449
1449
1450 }
1451
1452 }
1453
1454 }
1455
1456 }
1457
1458 }
1459
1459
1460 }
1461
1462 }
1463
1464 }
1465
1466 }
1467
1468 }
1469
1469
1470 }
1471
1472 }
1473
1474 }
1475
1476 }
1477
1478 }
1479
1479
1480 }
1481
1482 }
1483
1484 }
1485
1486 }
1487
1488 }
1489
1489
1490 }
1491
1492 }
1493
1494 }
1495
1496 }
1497
1498 }
1499
1499
1500 }
1501
1502 }
1503
1504 }
1505
1506 }
1507
1508 }
1509
1509
1510 }
1511
1512 }
1513
1514 }
1515
1516 }
1517
1518 }
1519
1519
1520 }
1521
1522 }
1523
1524 }
1525
1526 }
1527
1528 }
1529
1529
1530 }
1531
1532 }
1533
1534 }
1535
1536 }
1537
1538 }
1539
1539
1540 }
1541
1542 }
1543
1544 }
1545
1546 }
1547
1548 }
1549
1549
1550 }
1551
1552 }
1553
1554 }
1555
1556 }
1557
1558 }
1559
1559
1560 }
1561
1562 }
1563
1564 }
1565
1566 }
1567
1568 }
1569
1569
1570 }
1571
1572 }
1573
1574 }
1575
1576 }
1577
1578 }
1579
1579
1580 }
1581
1582 }
1583
1584 }
1585
1586 }
1587
1588 }
1589
1589
1590 }
1591
1592 }
1593
1594 }
1595
1596 }
1597
1598 }
1599
1599
1600 }
1601
1602 }
1603
1604 }
1605
1606 }
1607
1608 }
1609
1609
1610 }
1611
1612 }
1613
1614 }
1615
1616 }
1617
1618 }
1619
1619
1620 }
1621
1622 }
1623
1624 }
1625
1626 }
1627
1628 }
1629
1629
1630 }
1631
1632 }
1633
1634 }
1635
1636 }
1637
1638 }
1639
1639
1640 }
1641
1642 }
1643
1644 }
1645
1646 }
1647
1648 }
1649
1649
1650 }
1651
1652 }
1653
1654 }
1655
1656 }
1657
1658 }
1659
1659
1660 }
1661
1662 }
1663
1664 }
1665
1666 }
1667
1668 }
1669
1669
1670 }
1671
1672 }
1673
1674 }
1675
1676 }
1677
1678 }
1679
1679
1680 }
1681
1682 }
1683
1684 }
1685
1686 }
1687
1688 }
1689
1689
1690 }
1691
1692 }
1693
1694 }
1695
1696 }
1697
1698 }
1699
1699
1700 }
1701
1702 }
1703
1704 }
1705
1706 }
1707
1708 }
1709
1709
1710 }
1711
1712 }
1713
1714 }
1715
1716 }
1717
1718 }
1719
1719
1720 }
1721
1722 }
1723
1724 }
1725
1726 }
1727
1728 }
1729
1729
1730 }
1731
1732 }
1733
1734 }
1735
1736 }
1737
1738 }
1739
1739
1740 }
1741
1742 }
1743
1744 }
1745
1746 }
1747
1748 }
1749
1749
1750 }
1751
1752 }
1753
1754 }
1755
1756 }
1757
1758 }
1759
1759
1760 }
1761
1762 }
1763
1764 }
1765
1766 }
1767
1768 }
1769
1769
1770 }
1771
1772 }
1773
1774 }
1775
1776 }
1777
1778 }
1779
1779
1780 }
1781
1782 }
1783
1784 }
1785
1786 }
1787
1788 }
1789
1789
1790 }
1791
1792 }
1793
1794 }
1795
1796 }
1797
1798 }
1799
1799
1800 }
1801
1802 }
1803
1804 }
1805
1806 }
1807
1808 }
1809
1809
1810 }
1811
1812 }
1813
1814 }
1815
1816 }
1817
1818 }
1819
1819
1820 }
1821
1822 }
1823
1824 }
1825
1826 }
1827
1828 }
1829
1829
1830 }
1831
1832 }
1833
1834 }
1835
1836 }
1837
1838 }
1839
1839
1840 }
1841
1842 }
1843
1844 }
1845
1846 }
1847
1848 }
1849
1849
1850 }
1851
1852 }
1853
1854 }
1855
1856 }
1857
1858 }
1859
1859
1860 }
1861
1862 }
1863
1864 }
1865
1866 }
1867
1868 }
1869
1869
1870 }
1871
1872 }
1873
1874 }
1875
1876 }
1877
1878 }
1879
1879
1880 }
1881
1882 }
1883
1884 }
1885
1886 }
1887
1888 }
1889
1889
1890 }
1891
1892 }
1893
1894 }
1895
1896 }
1897
1898 }
1899
1899
1900 }
1901
1902 }
1903
1904 }
1905
1906 }
1907
1908 }
1909
1909
1910 }
1911
1912 }
1913
1914 }
1915
1916 }
1917
1918 }
1919
1919
1920 }
1921
1922 }
1923
1924 }
1925
1926 }
1927
1928 }
1929
1929
1930 }
1931
1932 }
1933
1934 }
1935
1936 }
1937
1938 }
1939
1939
1940 }
1941
1942 }
1943
1944 }
1945
1946 }
1947
1948 }
1949
1949
1950 }
1951
1952 }
1953
1954 }
1955
1956 }
1957
1958 }
1959
1959
1960 }
1961
1962 }
1963
1964 }
1965
1966 }
1967
1968 }
1969
1969
1970 }
1971
1972 }
1973
1974 }
1975
1976 }
1977
1978 }
1979
1979
1980 }
1981
1982 }
1983
1984 }
1985
1986 }
1987
1988 }
1989
1989
1990 }
1991
1992 }
1993
1994 }
1995
1996 }
1997
1998 }
1999
1999
2000 }
2001
2002 }
2003
2004 }
2005
2006 }
2007
2008 }
2009
2009
2010 }
2011
2012 }
2013
2014 }
2015
2016 }
2017
2018 }
2019
2019
2020 }
2021
2022 }
2023
2024 }
2025
2026 }
2027
2028 }
2029
2029
2030 }
2031
2032 }
2033
2034 }
2035
2036 }
2037
2038 }
2039
2039
2040 }
2041
2042 }
2043
2044 }
2045
2046 }
2047
2048 }
2049
2049
2050 }
2051
2052 }
2053
2054 }
2055
2056 }
2057
2058 }
2059
2059
2060 }
2061
2062 }
2063
2064 }
2065
2066 }
2067
2068 }
2069
2069
2070 }
2071
2072 }
2073
2074 }
2075
2076 }
2077
2078 }
2079
2079
2080 }
2081
2082 }
2083
2084 }
2085
2086 }
2087
2088 }
2089
2089
2090 }
2091
2092 }
2093
2094 }
2095
2096 }
2097
2098 }
2099
2099
2100 }
2101
2102 }
2103
2104 }
2105
2106 }
2107
2108 }
2109
2109
2110 }
2111
2112 }
2113
2114 }
2115
2116 }
2117
2118 }
2119
2119
2120 }
2121
2122 }
2123
2124 }
2125
2126 }
2127
2128 }
2129
2129
2130 }
2131
2132 }
2133
2134 }
2135
2136 }
2137
2138 }
2139
2139
2140 }
2141
2142 }
2143
2144 }
2145
2146 }
2147
2148 }
2149
2149
2150 }
2151
2152 }
2153
2154 }
2155
2156 }
2157
2158 }
2159
2159
2160 }
2161
2162 }
2163
2164 }
2165
2166 }
2167
2168 }
2169
2169
2170 }
2171
2172 }
2173
2174 }
2175
2176 }
2177
2178 }
2179
2179
2180 }
2181
2182 }
2183
2184 }
2185
2186 }
2187
2188 }
2189
2189
2190 }
2191
2192 }
2193
2194 }
2195
2196 }
2197
2198 }
2199
2199
2200 }
2201
2202 }
2203
2204 }
2205
2206 }
2207
2208 }
2209
2209
2210 }
2211
2212 }
2213
2214 }
2215
2216 }
2217
2218 }
2219
2219
2220 }
2221
2222 }
2223
2224 }
2225
2226 }
2227
2228 }
2229
2229
2230 }
2231
2232 }
2233
2234 }
2235
2236 }
2237
2238 }
2239
2239
2240 }
2241
2242 }
2243
2244 }
2245
2246 }
2247
2248 }
2249
2249
2250 }
2251
2252 }
2253
2254 }
2255
22
```

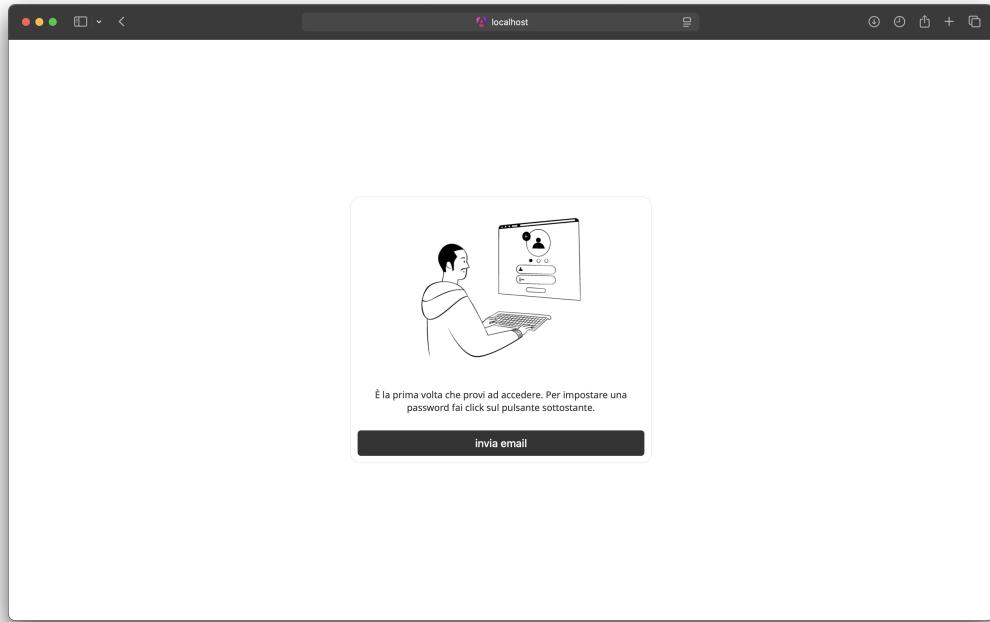


Figura 3.4: Primo accesso nel sistema.

```

9     error: () => {
10       this.isError = true;
11       this.message = "Errore, riprova piu' tardi.";
12       this.showToast = true;
13     }
14   );
15
16   setTimeout(() => {
17     this.showToast = false;
18   }, 3000);
19 }
```

Tale metodo effettua una richiesta HTTP, inviando l'email fornita dall'utente alla Rest API, se la risposta ha uno status impostato su 200, viene mostrato un popup indicando l'avvenuto invio dell'email, altrimenti il messaggio sarà riferito all'errore che si è verificato.

L'email inviata all'utente è mostrata nella Figura 3.5, che è provvista di link con token, per il reindirizzamento alla pagina per impostare la password. Il token, contenuto nel link, è associato all'utente che ha avviato la procedura



Figura 3.5: Contenuto della email inviata all’utente.

per impostare la password. Questo token ha una durata breve, di 5 minuti, per evitare che possa essere rubato e utilizzato da terzi.

3.3.2 Reset Password

Un utente che ha avviato la procedura per impostare la password, tramite il link ricevuto con l'email, sarà reindirizzato alla seguente schermata riportata nella Figura 3.6.

Qui l’utente è invitato a inserire la password che desidera usare per accedere al sistema e ripeterla al fine di evitare possibili errori di battitura. Attraverso un click sul pulsante “Conferma” verrà eseguito il metodo riportato di seguito.

```

1 submit() {
2   if (this.form.valid) {
3     const newPassword = this.form.get('newPassword')!.value;
4     const confirmPassword = this.form.get('confirmPassword')!.
5       value;

```

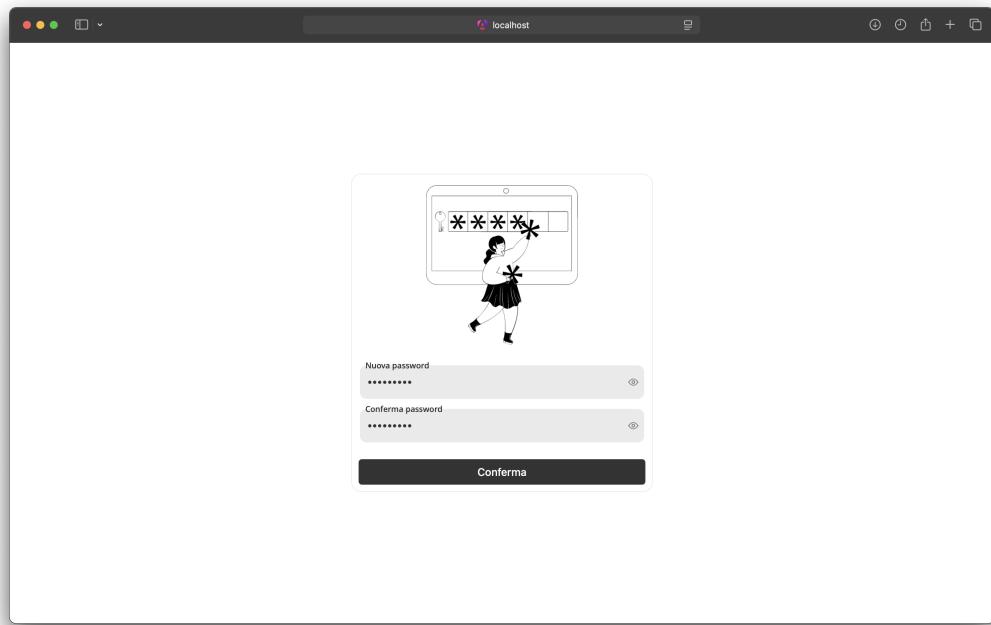


Figura 3.6: Schermata di reset della password.

```
6     if (newPassword === confirmPassword && this.token != null)
7     {
8         const requestSetPasswordDto = new RequestSetPasswordDto(
9             this.token, newPassword);
10        this.passwordSetService.setPassword(requestSetPasswordDto)
11            .subscribe({
12                next: () => {
13                    this.isError = false;
14                    this.message = "Password modificata con successo.";
15                    this.showToast = true;
16                },
17                error: () => {
18                    this.isError = true;
19                    this.message = "Errore durante il reset della
20                        password.";
21                    this.showToast = true;
22                }
23            });
24    } else {
25        this.isError = true;
```

```

22     this.message = "Le password non coincidono.";
23     this.showToast = true;
24   }
25   setTimeout(() => {
26     this.showToast = false;
27   }, 3000);
28 }
29 }
```

L'esecuzione dal metodo consiste nel verificare che le due password inserite dall'utente combacino e che il token ricevuto con il link sia effettivamente presente. Se così fosse allora verrà effettuata una richiesta HTTP al fine di rendere persistente la nuova password dell'utente.

3.3.3 Area Personale

Quando l'utente accede al sistema viene reindirizzato nella sua area personale. In questa schermata l'utente può visualizzare i propri dati (Figura 3.7) oppure personalizzare la propria esperienza all'interno del sistema (Figura 3.8).

Le principali operazioni, oltre alla semplice visualizzazione delle informazioni, sono eseguite il logout dal sistema, personalizzazione (font, modalità tema e layout) e modifica delle informazioni dell'utente.

Per quanto riguarda la modifica delle informazioni dell'utente, premendo il pulsante contenente l'icona della matita, si viene reindirizzati alla pagina “Modifica Professore”.

```

1 goToUpdate() {
2   this.router.navigate(['/add-professor'], { state: { id: this.
3   professor.id } }).then();
}
```

Il metodo `goToUpdate()`, tramite il modulo `Router` fornito da Angular, reindirizza alla pagina associata alla rotta `/add-professor`, aggiungendo nello stato della navigazione l'id dell'utente, che sarà gestito adeguatamente dalla pagina di destinazione.

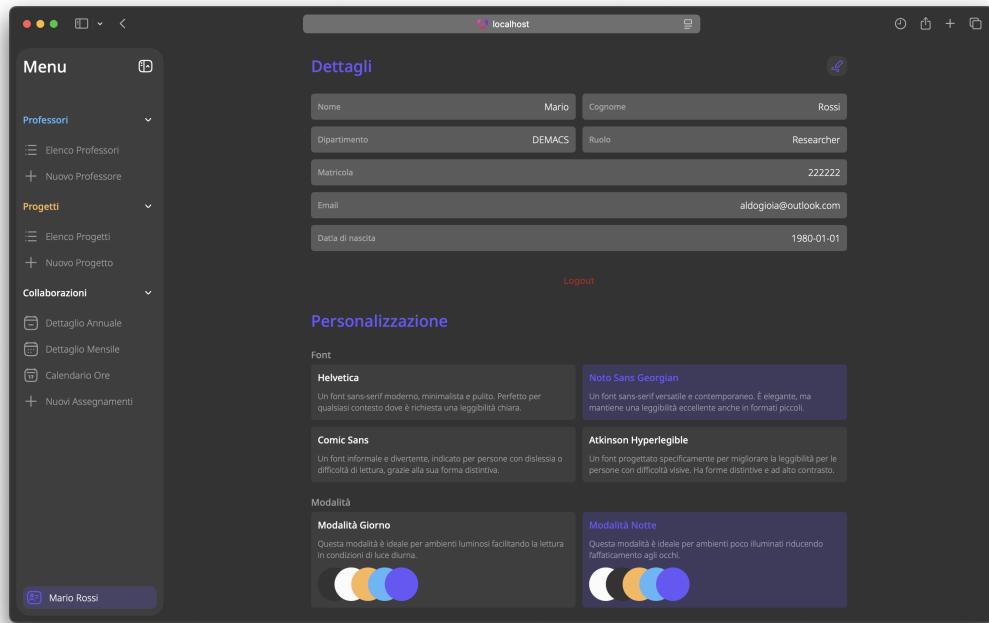


Figura 3.7: Schermata con le informazioni dell'utente.

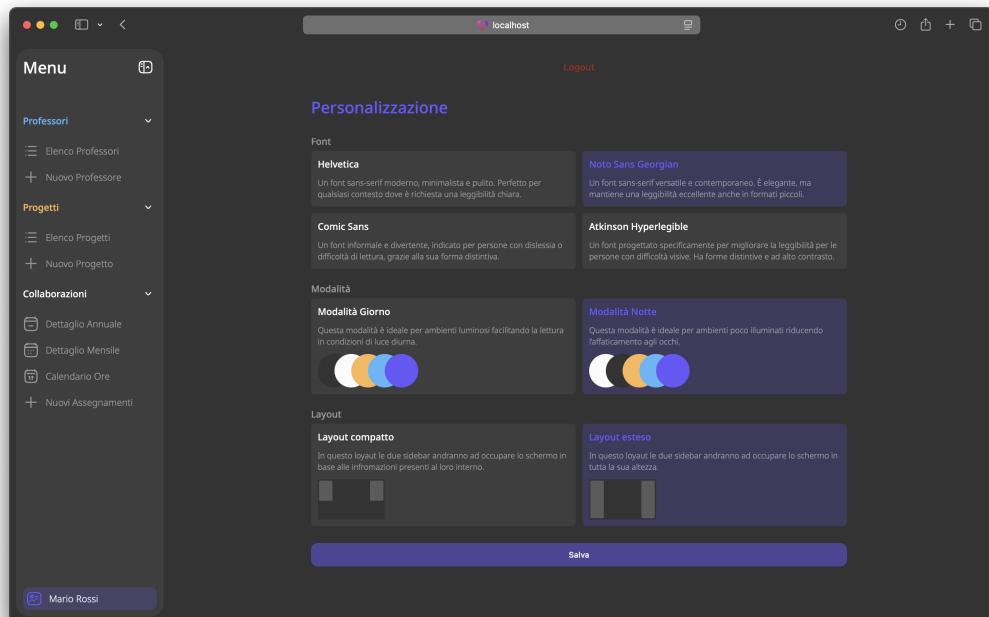


Figura 3.8: Schermata con le personalizzazioni dell'utente.

Se un utente invece è intendo ad eseguire il logout, premendo su “Logout” manderà in esecuzione tale metodo, riportata di seguito.

```

1  logout() {
2      if (this.getAccessToken() !== null) {
3          const headers = new HttpHeaders({
4              'Content-Type': 'application/json',
5              'Authorization': 'Bearer ' + this.getAccessToken()
6          })
7
8          return this.http.post(this.apiUrl + '/logout', null, {
9              headers, observe: 'response'
10         })
11     }
12
13     else throw new Error('Authentication required to logout');
14 }
15
16 makeLogout() {
17     this.logout().subscribe({
18         next: () => {
19             this.removeItemFromStorage();
20             this.router.navigate(['login']).then();
21         }
22     });
23
24     removeItemFromStorage() {
25         localStorage.removeItem('accessToken');
26         localStorage.removeItem('accessRole');
27         localStorage.removeItem('professor');
28     }
29 }
```

Eseguendo il logout viene inviato alle API Rest, tramite una richiesta HTTP, il JWT al fine di invalidarlo. Una volta invalidato il JWT, si provvede a rimuovere le informazioni d’accesso dal “LocalStorage” e, infine, si viene reindirizzati alla pagina d’accesso.

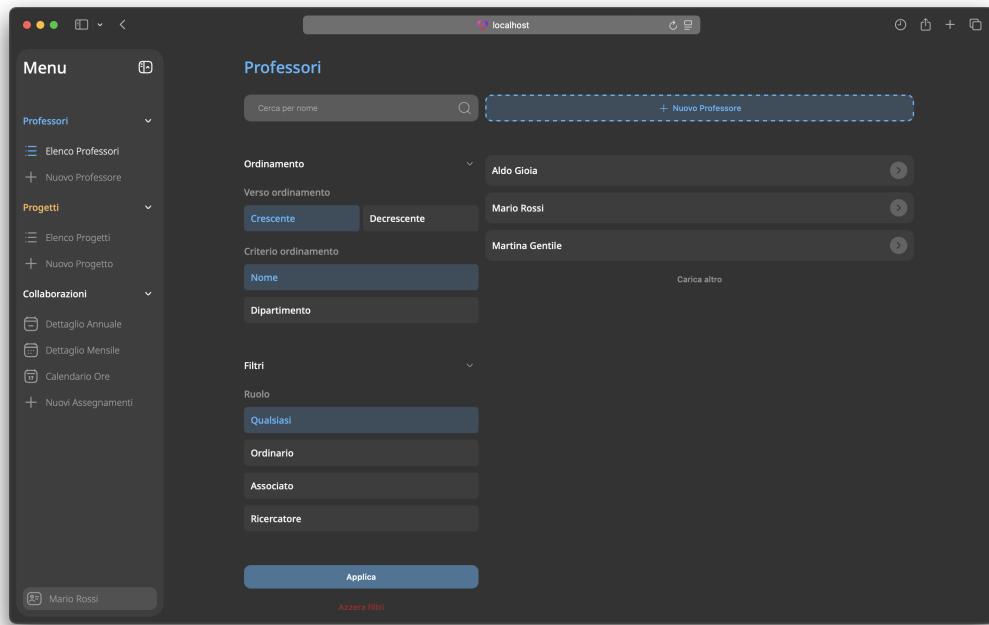


Figura 3.9: Schermata con l’elenco professori.

Elenco Professori

Se l’utente che ha effettuato l’accesso con un account Admin, ha intenzione di visualizzare l’elenco di tutti i professori registrati tramite la barra menù può selezionare “Elenco professori” (Figura 3.9).

In questa schermata è presente un sistema di ricerca, ordinamento e filtraggio e l’elenco dei professori mostrati in base ai criteri selezionati.

Per quanto riguarda la ricerca si sfrutta una barra di ricerca in cui si può inserire il nome del professore che si vuole cercare. L’ordinamento dei professori può essere eseguito in ordine crescente o decrescente e, anche, in base al nome o al dipartimento di riferimento dei professori. Per quanto riguarda il filtraggio, si può applicare alla ricerca un filtro basato sul ruolo del professore, in particolare tra *qualsiasi*, *ordinario*, *associato* o *ricercatore*.

L’elenco dei professori è basato sui criteri scelti nel sistema di ricerca, ordinamento e filtraggio.

Per applicare i filtri, si può cliccare sul tasto “invia” della tastiera oppure su “Applica”. In entrambi i casi, verrà eseguito il metodo `addFilter()`, descritto

di seguito.

```
1 addFilter() {
2     this.professors = []
3     this.loadData()
4 }
5
6 private loadData(pageNumber: number = 0, pageSize: number = 10)
7 {
8     this.loading = true
9
10    const sorting = { ['direction']: this.ordinamento, ['sortBy']:
11        this.criterio }
12    const filtering = { ['role']: this.ruolo, ['name']: this.
13        filter.get('name')?.value }
14
15    this.professorService.getProfessors(
16        sorting,
17        filtering,
18        pageNumber,
19        pageSize
20    ).subscribe({
21        next: page => {
22            this.page = page
23
24            const newProfessors = page.content.filter(professor
25            =>
26                !this.professors.some(existingProfessor =>
27                    existingProfessor.id === professor.id)
28            );
29
30            this.professors = [...this.professors, ...
31            newProfessors]
32
33            this.loading = false
34        },
35        error: error => {
36            this.loading = false
37        }
38    })
39}
```

```

31     }
32   })
33 }
```

Ciò che viene fatto dal metodo è effettuare una richiesta HTTP, specificando i criteri con cui si vuole ottenere la lista di professori da dover visualizzare.

Cliccando su “Azzera filtri”, invece, viene eseguita la funzione `resetFilter()`, il cui compito è riportare ai valori di default i criteri di ricerca ordinamento e filtraggio, come mostrato di seguito.

```

1 resetFilter() {
2   this.ordinamento = "ASC"
3   this.criterio = "name"
4   this.ruolo = ""
5   this.filter.get('name')?.setValue("")
6   this.addFilter()
7 }
```

Nella schermata “Elenco Professori” è presente anche il pulsante “Aggiungi Professore”, che reindirizza alla pagina associata alla rotta `/add-professor`.

Analogamente premendo un elemento dell’elenco dei professori viene eseguito il metodo `navigate()` che reindirizza alla pagina per visualizzare i dettagli ed i progetti a cui è assegnato tale professore, come mostrato di seguito.

```

1 navigate(id: number) {
2   this.router.navigate(['/professor'], { state: { id: id } })
3 }
```

3.3.4 Elenco Progetti

Analogamente alla schermata “Elenco Professori”, è presente la schermata “Elenco Progetti” (Figura 3.10). Queste due schermate sono uguali per aspetto e funzionalità, l’unica differenza è ciò che mostrano, la prima una lista di professori e la seconda una lista di progetti.

Di seguito si riporta il codice delle operazioni che è possibile effettuare.

```

1 addFilter() {
```

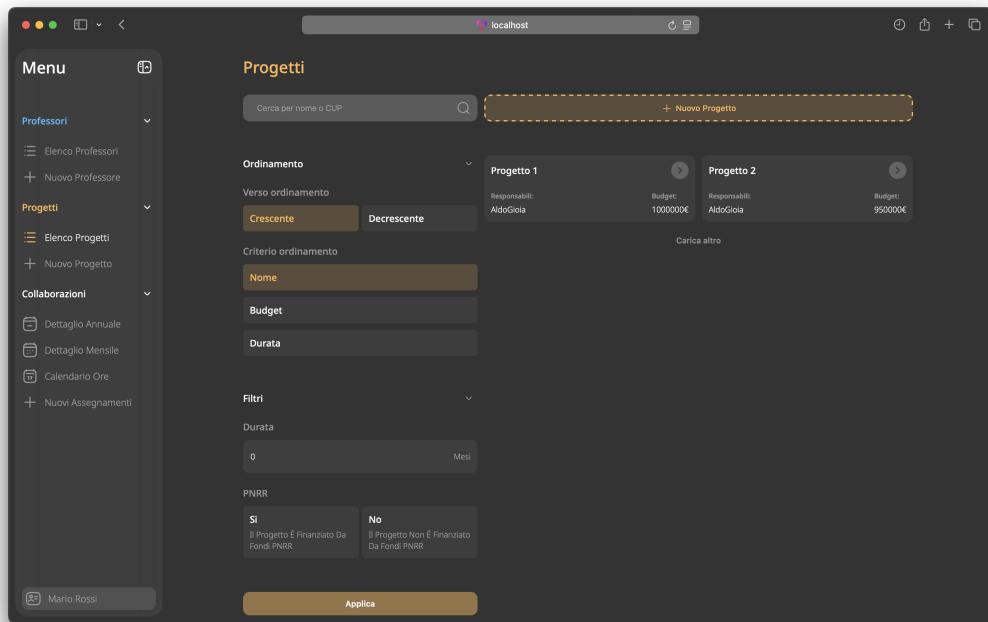


Figura 3.10: Schermata di elenco progetti.

```
2   this.projects = []
3
4 }
5
6 private loadData(pageNumber: number = 0, pageSize: number = 10)
7 {
8
9   this.loading = true
10
11
12   this.projectsService.getProjectsCriteria(
13     this.ordinamento,
14     this.criterio,
15     this.form.get('name')?.value,
16     this.form.get('duration')?.value,
17     this.pnrr
18   ).subscribe({
19     next: page => {
20       this.page = page
21
22       const newProfessors = page.content.filter(project =>
23         !this.projects.some(existingProfessor =>
```

```

existingProfessor.cup === project.cup)
);
;

this.projects = [...this.projects, ...newProfessors]

this.loading = false
},
error: error => {
  this.loading = false
}
}

})
}

resetFilter() {
  this.ordinamento = "ASC"
  this.criterio = "name"
  this.pnrr = "";
  this.form.get('name')?.setValue("")
  this.form.get('duration')?.setValue(0)
  this.addFilter()
}

navigate(id: number) {
  this.router.navigate(['/project'], { state: { id: id }})
}

```

3.3.5 Aggiungi Professore

L’utente che ha effettuato l’accesso con un account Admin ha la possibilità di registrare nel sistema nuovi professori. La registrazione di un nuovo professore all’interno del sistema avviene tramite la schermata “Aggiungi Professore” (Figura 3.11).

In tale schermata si possono aggiungere i vari dettagli del professore, partendo dal suo nome fino ad arrivare al suo ruolo.

```

addProfessor() {
  if (this.professorForm.valid) {
    let professor = new Professor(this.professorForm.value);
  }
}

```

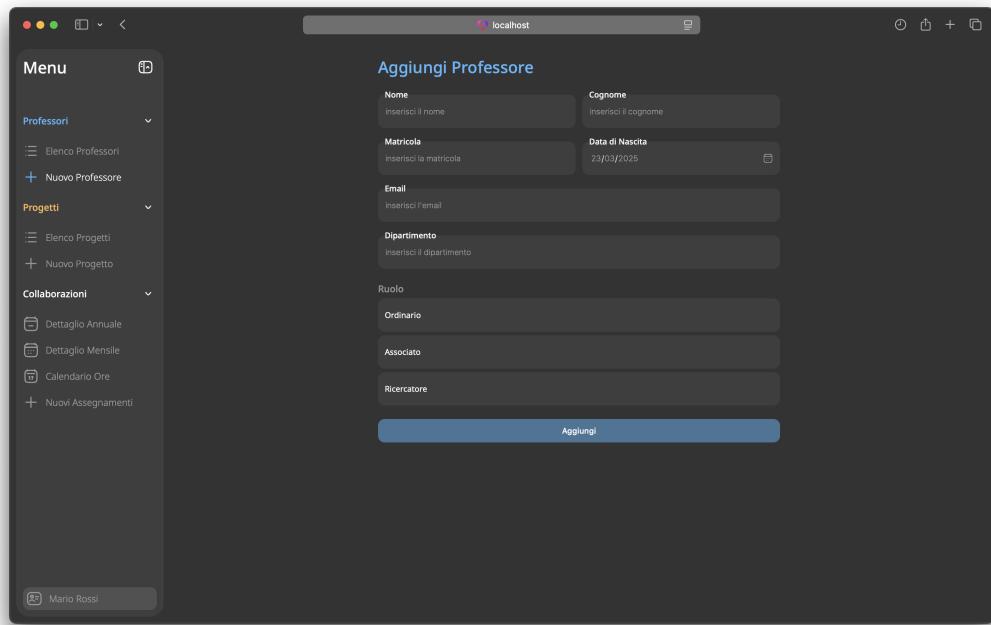


Figura 3.11: Schermata di aggiunta di un professore.

```
4   professor.role = this.role;
5
6   this.professorService.addProfessor(professor).subscribe(
7     { next: () => {
8       this.isError = false;
9       this.showToast = true;
10      this.message = 'Professor added successfully';
11      this.professorForm.reset();
12      this.role = 'ordinario';
13    },
14    error: () => {
15      this.isError = true;
16      this.showToast = true;
17      this.message = 'An error occurred'
18    }
19  }
20)
21 setTimeout(() => {
22   this.showToast = false;
23 }, 3000);
```

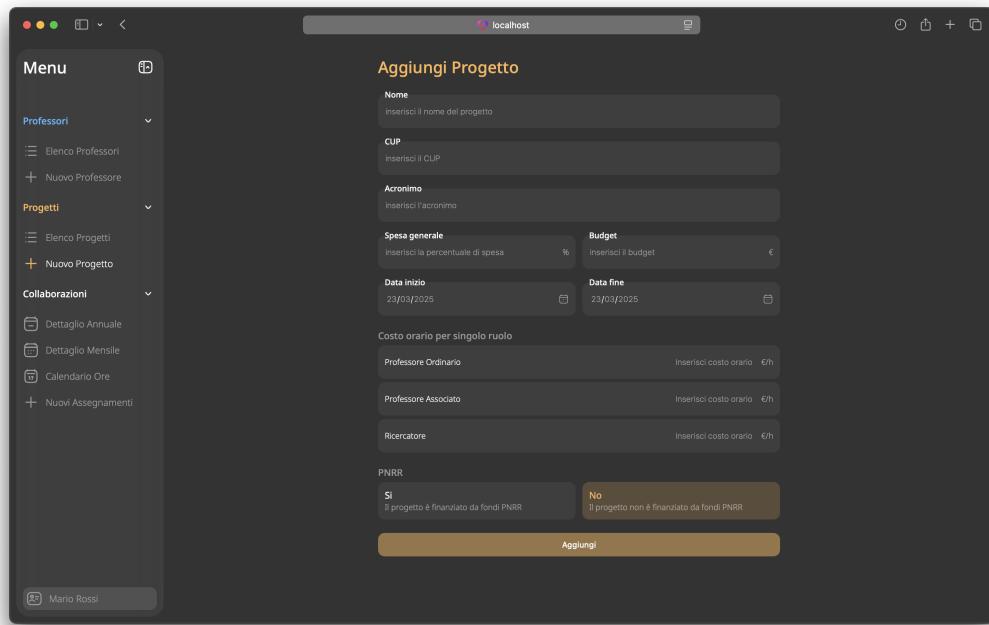


Figura 3.12: Schermata di aggiunta di un progetto.

```
24     }
25 }
```

Cliccando sul pulsante “Aggiungi”, viene eseguita la funzione `addProfessor()`, che ottiene tutti i dettagli del professore dal form per poi creare un oggetto `Professor`.

Tale oggetto verrà inviato alle API Rest, tramite richiesta HTTP, che risponderà in base all'esito delle operazioni. La risposta delle API Rest sarà poi mostrata tramite popup.

3.3.6 Aggiungi Progetto

Analogamente alla schermata “Aggiungi Professore”, è presente la schermata “Aggiungi Progetto” (Figura 3.12). Il codice per l’invio della richiesta HTTP alle API Rest è mostrato di seguito.

```
1 private addProject() {
2     if (this.projectForm.valid) {
3         let project: Project = new Project(this.projectForm.value)
4     }
5 }
```

```

4     project.pnrr = this.pnrr;
5     project.state = 'Attivo';
6     project.remunerations =
7       new Remuneration({roleType: 'Full', amount: this.
projectForm.get('amountFull')?.value}),
8       new Remuneration({roleType: 'Associate', amount: this.
projectForm.get('amountAssociate')?.value}),
9       new Remuneration({roleType: 'Researcher', amount: this.
projectForm.get('amountResearcher')?.value})
10    ];
11
12    this.projectsService.addProject(project).subscribe({
13      next: () => {
14        this.isError = false;
15        this.message = 'Project added successfully';
16        this.showToast = true;
17        this.projectForm.reset();
18      },
19      error: () => {
20        this.isError = true;
21        this.message = 'Project not added';
22        this.showToast = true;
23      }
24    });
25
26    setTimeout(() => {
27      this.showToast = false;
28    }, 3000);
29  }
30}

```

3.3.7 Dettagli Professore

La schermata mostrata in Figura 3.13 è accessibile cliccando su un elemento della lista di professori nella schermata “Elenco Professori”. Qui sono presenti tutti i dettagli di un professore oltre che l’elenco di progetti a cui

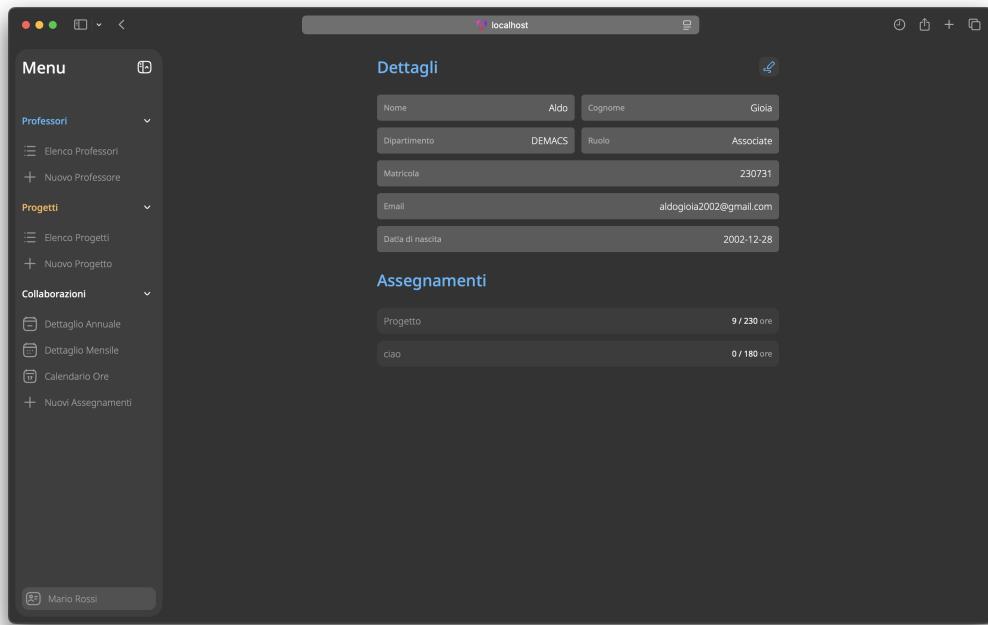


Figura 3.13: Schermata dei dettagli di un professore.

il professore è associato. Come la schermata di elenco dei professori, anche questa di dettaglio è accessibile solo dagli utenti amministratori.

I dettagli del professore sono modificabili, infatti, cliccando sul pulsante con l’icona della matita, si viene reindirizzati alla schermata adibita alla modifica.

```

1 goToUpdate() {
2   this.router.navigate(['/add-professor'], { state: { id: this.
3   professor.id } }).then();
}
```

Si può notare che la rotta è quella associata alla schermata “Aggiunta Professore” ma, nello stato della navigazione, viene passato l’id dell’utente.

3.3.8 Dettagli Progetto

La schermata riportata in Figura 3.14 è accessibile cliccando su un elemento della lista di progetti presente in “Elenco Progetti”. In questa schermata, è possibile visualizzare i dettagli del progetto, i professori assegnati, insieme alle ore già lavorate e quelle attese in tutta la durata del progetto.

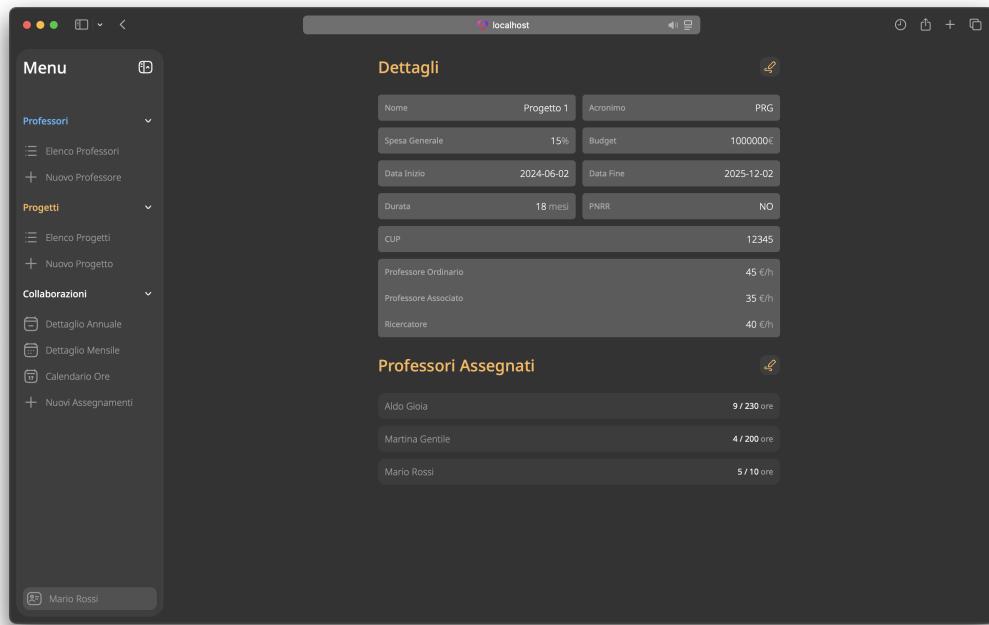


Figura 3.14: Schermata dei dettagli di un progetto.

Sia i dettagli che gli assegnamenti sono modificabili cliccando sul pulsante con l'icona della matita presente nelle rispettive sezioni.

```

1 goToUpdate() {
2   this.router.navigate(['/add-project'], { state: { id: this.id } }).then();
3 }
4
5 goToCollaboration() {
6   this.router.navigate(['/collaboration']).then();
7 }
```

3.3.9 Modifica Professore

La schermata “Modifica Professore” (Figura 3.15) riutilizza la schermata “Aggiungi Professore”, con la differenza sostanziale che, in questa schermata, i campi sono tutti già compilati e alcuni campi di testo non sono modificabili. Ogni volta che la schermata “Aggiunta Professore” viene generata, nel suo costruttore si prova a recuperare dallo stato della navigazione l’id. Se l’id è

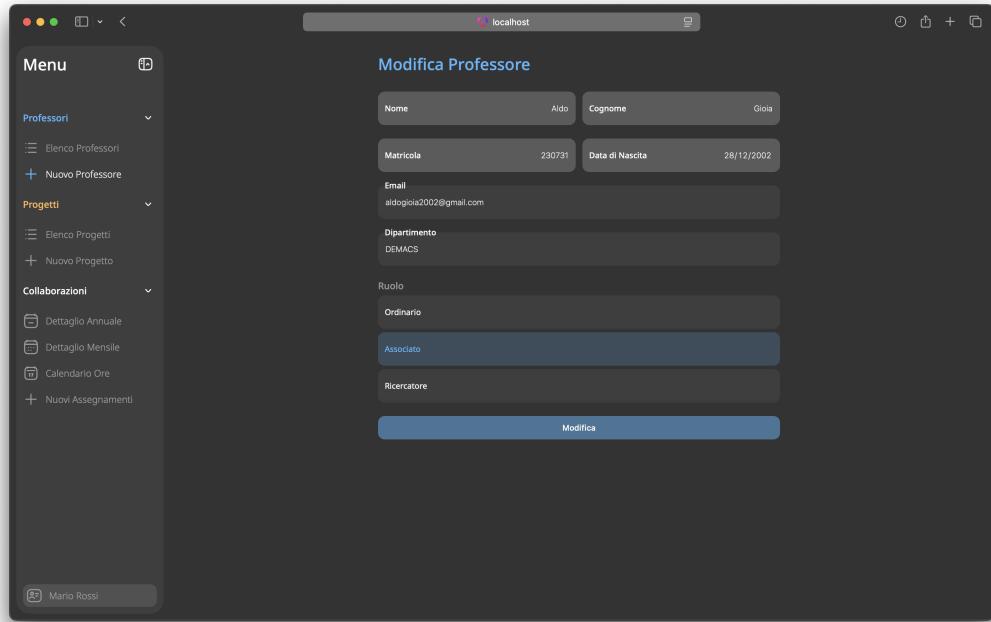


Figura 3.15: Schermata di modifica di un professore.

presente allora `modify` viene impostato a `true` e viene effettuata una richiesta HTTP alle API Rest per ottenere i dettagli del professore da dover mostrare.

```

1 constructor(
2   private formBuilder: FormBuilder,
3   private professorService: ProfessorService,
4   private router: Router
5 ) {
6   ...
7
8   const navigation = this.router.getCurrentNavigation();
9   if (navigation?.extras && navigation.extras.state) {
10     this.id = navigation.extras.state['id'];
11   }
12 }
13
14 ngOnInit(): void {
15   if (this.id !== undefined) {
16     this.loadProfessor(this.id);
17     this.modify = true;
18 }
```

19 }

Le modifiche effettuate vengono rese persistenti utilizzando il metodo descritto di seguito.

```

1 updateProfessor() {
2     if (this.professorForm.valid) {
3         let professor = new UpdateProfessorDto(this.professorForm.
4             value);
5             professor.role = this.role;
6
7         this.professorService.updateProfessor(professor).subscribe
8         ({
9             next: () => {
10                 this.isError = false;
11                 this.showToast = true;
12                 this.message = 'Professor updated successfully';
13             },
14             error: () => {
15                 this.isError = true;
16                 this.showToast = true;
17                 this.message = 'An error occurred'
18             }
19         )
20
21         setTimeout(() => {
22             this.showToast = false;
23         }, 3000);
24     }
}

```

3.3.10 Modifica Progetto

La schermata riportata in Figura 3.16 permette di modificare i dettagli di un progetto.

Come nel caso della modifica dei professori, anche qui è utilizzata la pagina “Aggiungi Progetto” che cambia il proprio aspetto con lo stesso me-

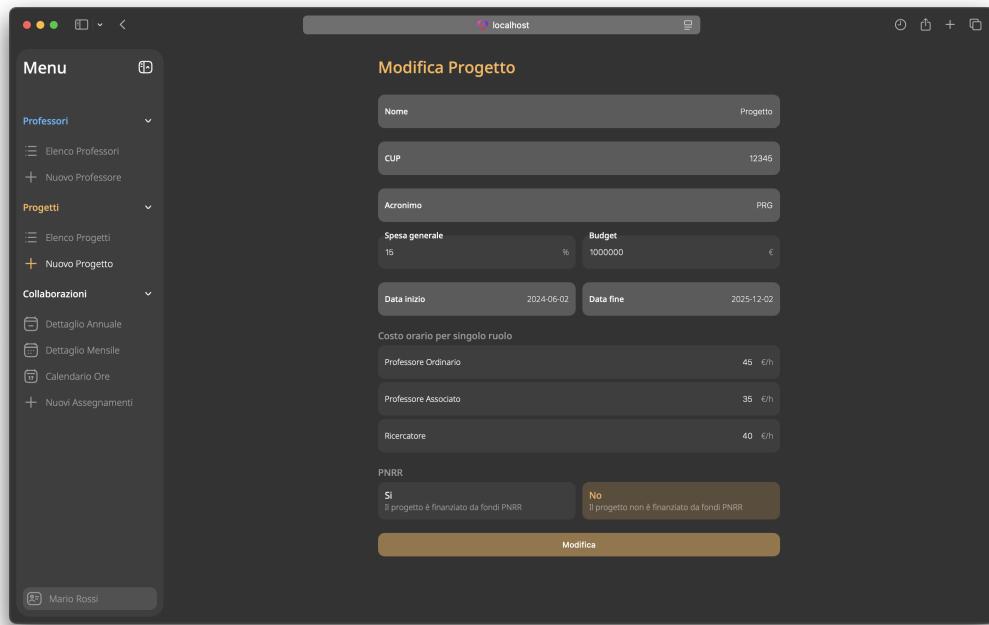


Figura 3.16: Schermata di modifica progetto.

canismo spiegato precedentemente. Di seguito si riporta il codice responsabile alla gestione di questo meccanismo.

```

1 constructor(
2     private formBuilder: FormBuilder,
3     private projectsService: ProjectsService,
4     private router: Router
5 ) {
6     ...
7
8     const navigation = this.router.getCurrentNavigation();
9     if (navigation?.extras && navigation.extras.state) {
10         this.id = navigation.extras.state['id'];
11     }
12 }
13
14 ngOnInit(): void {
15     if (this.id !== undefined) {
16         this.loadProject(this.id);
17         this.modify = true;
18     }

```

19 }

Cliccando sul pulsante “Modifica”, il progetto con i propri dettagli aggiornati viene reso persistente tramite una richiesta HTTP alle API Rest, come mostrato di seguito.

```
1 private updateProject() {
2     if (this.projectForm.valid) {
3         let project = new UpdateProjectDto(this.projectForm.value);
4
5         project.pnrr = this.pnrr;
6         project.state = 'Attivo';
7
8         if (this.project !== null) {
9             project.remunerations = [
10                 new UpdateRemunerationDto({
11                     id: this.project.remunerations.find(r => r.roleType
12 === 'Full')?.id,
13                     amount: this.projectForm.get('amountFull')?.value
14                 }),
15                 new UpdateRemunerationDto({
16                     id: this.project.remunerations.find(r => r.roleType
17 === 'Associate')?.id,
18                     amount: this.projectForm.get('amountAssociate')?.value
19                 }),
20                 new UpdateRemunerationDto({
21                     id: this.project.remunerations.find(r => r.roleType
22 === 'Researcher')?.id,
23                     amount: this.projectForm.get('amountResearcher')?.value
24                 })
25             ];
26         }
27
28         this.projectsService.updateProject(project).subscribe({
29             next: () => {
30                 this.toastr.success('Progetto aggiornato con successo');
31             },
32             error: (err) => {
33                 this.toastr.error('Errore durante l\'aggiornamento del progetto');
34             }
35         });
36     }
37 };
```

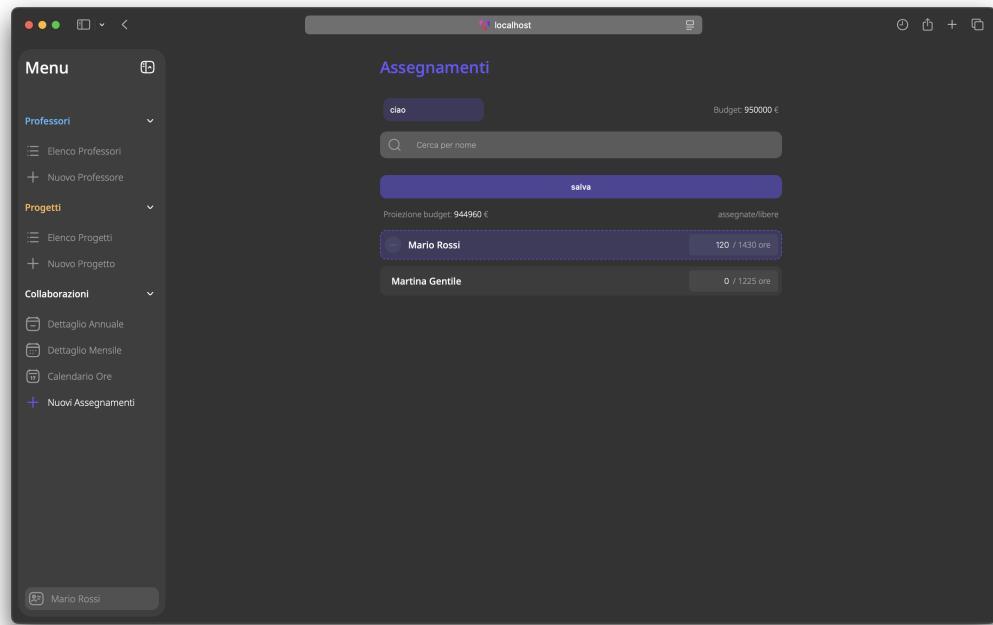


Figura 3.17: Schermata di assegnamento dei professori a un progetto.

```
27     this.isError = false;
28
29     this.message = 'Project updated successfully';
30     this.showToast = true;
31   },
32   error: () => {
33     this.isError = true;
34     this.message = 'An error occurred';
35     this.showToast = true
36   }
37
38   setTimeout(() => {
39     this.showToast = false;
40   }, 3000);
41
42 }
```

3.3.11 Assegnamenti

Un utente Admin può assegnare dei professori ad un progetto attraverso la schermata “Assegnamenti”. Qui sono presenti tre elementi principali:

1. Scelta progetto, con cui l’utente può selezionare, tra i vari progetti presenti nel sistema, il progetto a cui assegnare dei professori. Una volta selezionato il progetto viene anche mostrato il budget di quest’ultimo.
2. Barra di ricerca, con cui l’utente può esemplificare le operazioni cercando un professore specifico da assegnare al progetto.
3. Lista dei professori, in cui vengono mostrati, uno dopo l’altro, tutti i professori a patto che questi non siano già assegnati al progetto selezionato e, soprattutto, abbiano delle ore libere disponibili all’assegnamento.

Una volta scelto il progetto e i professori da assegnare, allocando le ore minime che ogni professore deve lavorare sul progetto, si può procedere al salvataggio. Per salvare è sufficiente cliccare sul pulsante apposito “Salva”, che eseguirà il metodo `save()` che, a sua volta, effettuerà una richiesta HTTP, per rendere persistenti gli assegnamenti, come mostrato di seguito.

```

1 save() {
2   const collaborations = this.professorToAdd
3     .map(i => this.collaborations[i]);
4
5   this.collaborationService.addCollaboration(collaborations).
6     subscribe({
7       next: () => {
8         this.isError = false;
9         this.message = "Professori assegnati con successo";
10        this.showToast = true;
11        this.resetAfterSave()
12      },
13      error: () => {
14        this.isError = true;
15        this.message = "Errore durante l'assegnazione dei
professori, riprovare.";
```

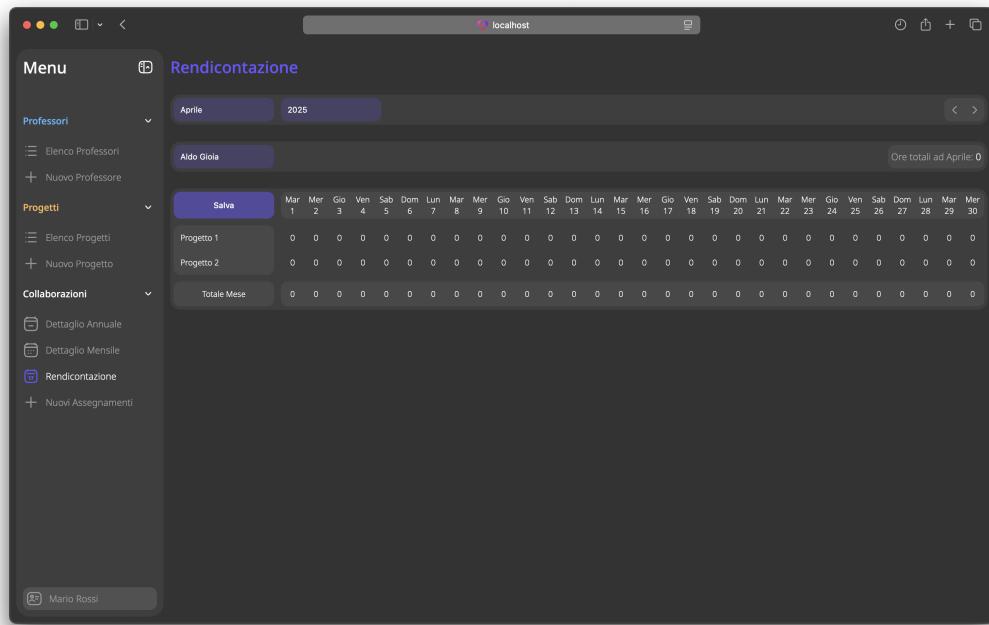


Figura 3.18: Schermata di rendicontazione per l'Admin.

```

15     this.showToast = true;
16   }
17 }
18 setTimeout(() => {
19   this.showToast = false;
20 }, 3000);
21 }
```

3.3.12 Rendicontazione

La schermata “Rendicontazione”, riportata in Figura 3.18, consente ai professori di dichiarare le ore lavorate su uno specifico progetto. In particolare, è possibile selezionare il mese e l’anno in cui si vogliono dichiarare le ore lavorate. Inoltre, agli Admin è consentito di selezionare il professore di cui si vogliono dichiarare le ore e visualizzarle. Infine, è presente una tabella per caricare le ore nel mese e nell’anno selezionati. In particolare, nelle colonne sono presenti tutti i giorni relativi al mese ed all’anno scelto, sulle righe sono presenti tutti i progetti in cui il professore selezionato presenta un assegna-

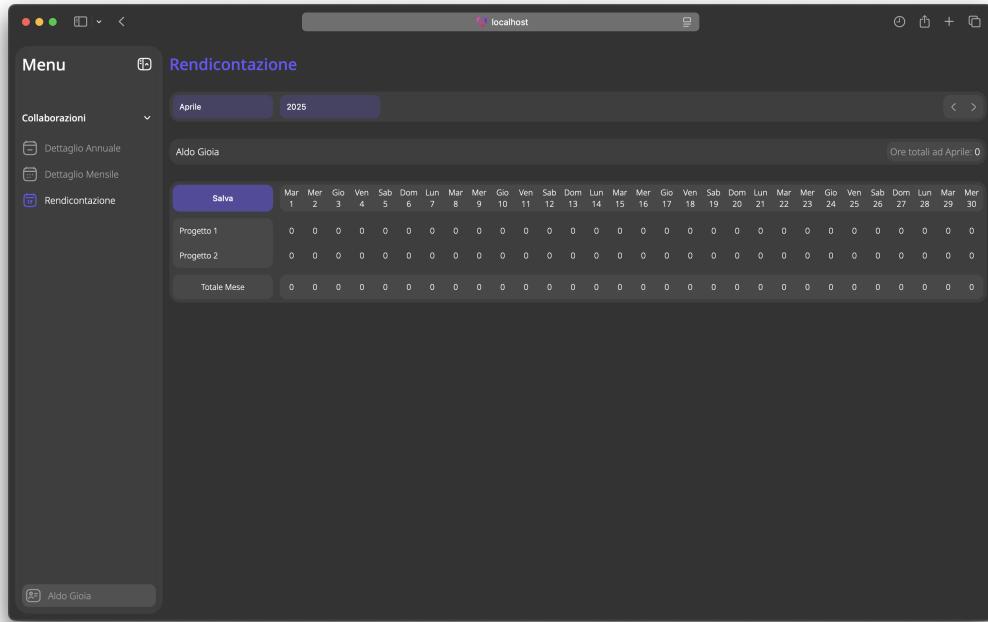


Figura 3.19: Schermata di rendicontazione per il professore.

mento e nelle intersezioni tra giorno e progetto sono presenti le ore lavorate, in quel giorno e per quel progetto, dal professore selezionato.

La schermata riportata in Figura 3.19 mostra la stessa schermata precedente ma dal punto di vista del professore. La differenza principale rispetto alla schermata precedente è l'assenza della scelta del professore, in quanto è implicito.

Nell'intersezione tra un progetto e un giorno è presente l'elemento HTML riportato di seguito.

```
1 <input class="cell-mod" type="number" *ngFor="let d of days;
  let j = index" [formControlName]="j" [class]={`${'error':``}}`:
  checkInvalid(i,j), 'to-add': toAdd.includes(i*this.days.length + j)}" (focusout)="add(i, j, d, dd)" >
```

Quando viene scaturito l'evento `focusout`, quindi quando l'utente ha inserito il dato delle ore lavorate, viene eseguito il seguente metodo riportato di seguito.

```
1 add(i: number, j: number, s: string, detail: DailyDetailDto) {
  2   const control = this.getControl(i, j)
```

```

3     if(!control.invalid) this.toAdd.push(i*this.days.length+j)
4   else {
5     control.reset()
6     control.setValue(this.getWorkedHours(s, detail))
7     this.toAdd = this.toAdd.filter(e => e !== i*this.days.
8       length+j)
9   }

```

Il metodo verifica che il dato inserito sia valido. In caso affermativo, viene aggiunto ad una lista di oggetti da dover inviare alle API Rest, che dopo le giuste verifiche li renderà persistenti.

Quando invece si clicca sul tasto “Salva” viene eseguito il metodo descritto di seguito.

```

1 save() {
2   const toCreate: DailyHours[] = []
3   const toUpdate: UpdateDailyHoursDto[] = []
4
5   for (let i = 0; i < this.dailyDetail.length; i++) {
6     for (let j = 0; j < this.days.length; j++) {
7       const x = i * this.days.length + j
8
9       if (this.toAdd.includes(x)) {
10         const dailyHours = this.dailyDetail[i]
11           .dailyHours.find(d => d.day === Number(this.days[j]
12             .substring(4)))
13         if (dailyHours) {
14           toUpdate.push({
15             id: dailyHours.id,
16             workedHours: this.getControl(i, j).value
17           })
18         } else {
19           toCreate.push({
20             monthlyHours: this.dailyDetail[i].monthlyHours.id
21             ,
22             day: Number(this.days[j].substring(4)),
23             workedHours: this.getControl(i, j).value
24           })
25       }
26     }
27   }
28 }

```

```

22         })
23     }
24   }
25 }
26 }

27

28   this.dailyHoursService.createDailyHours(toCreate).pipe(
29     switchMap(() => this.dailyHoursService.updateDailyHours(
30       toUpdate))
31   ).subscribe({
32     next: () => {
33       this.toAdd = []
34       this.isError = false
35       this.showToast = true
36       this.message = 'Salvataggio completato con successo'
37     },
38     error: () => {
39       this.isError = true
40       this.showToast = true
41       this.message = 'Errore durante il salvataggio'
42     }
43   })
44   setTimeout(() => {
45     this.showToast = false
46   }, 3000);
}

```

Il metodo suddivide gli oggetti da inviare alla Rest API in `toCreate` e `toUpdate`. Successivamente esegue le chiamate HTTP alle API Rest, concatenate una dopo l'altra, al fine di rendere persistenti tutti gli oggetti all'interno del database.

3.3.13 Dettaglio Annuale

Ogni assegnamento prevede che le ore minime da lavorare sul progetto siano adeguatamente distribuite per ogni anno compreso nella durata del progetto. Distribuire le ore di ogni assegnamento per tutta la durata del progetto permette una gestione più accurata e avanzata delle risorse umane. Ad esem-

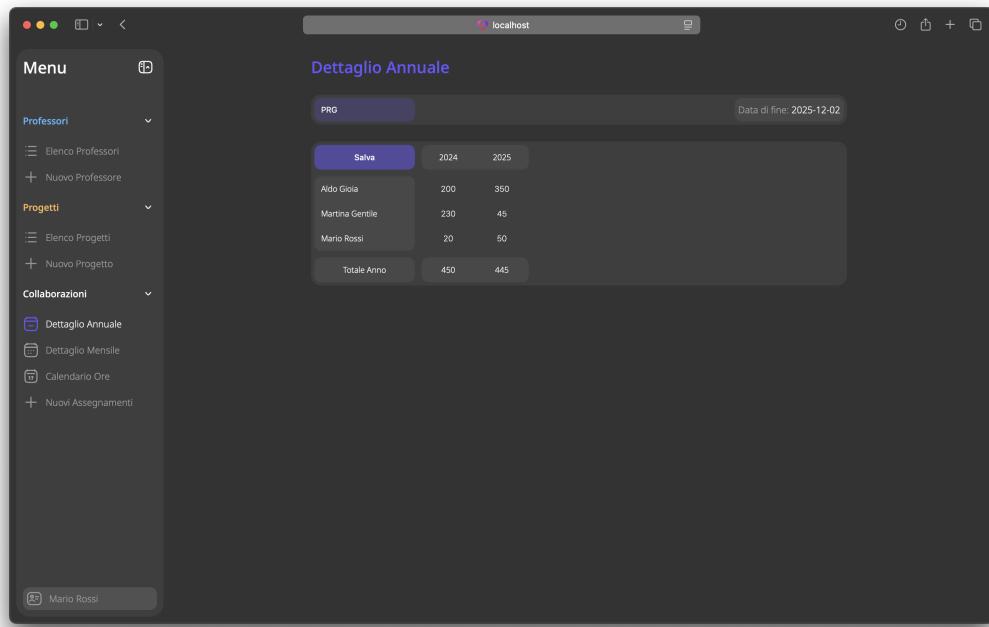


Figura 3.20: Schermata con il dettaglio annuale delle ore su un progetto.

pio, un professore particolarmente specializzato in un certo ambito potrebbe iniziare a lavorare su un particolare progetto, solo dopo che lo sviluppo di esso abbia già raggiunto alcuni obiettivi.

La schermata “Dettaglio Annuale”, riportata in Figura 3.20, favorisce proprio questo tipo di gestione. Il funzionamento è analogo a quello della schermata “Rendicontazione”. Quindi, una volta inserite le ore, per ogni professore/anno, si può procedere a salvare le modifiche cliccando sul pulsante “Salva”.

Il codice per rendere persistenti le ore da lavorare anno per anno per ogni professore/progetto è riportato di seguito.

```

1 save() {
2
3     const toCreate: YearlyHours[] = []
4     const toUpdate: UpdateYearlyHoursDto[] = []
5
6     for (let i = 0; i < this.yearlyDetail.length; i++) {
7         for (let j = 0; j < this.years.length; j++) {
8             const x = i*this.years.length + j
9             if (!this.yearlyDetail[i].years[j]) {
10                 toCreate.push({id: x, professorId: this.years[j].professorId, year: this.years[j].year, hours: this.yearlyDetail[i].hours})
11             } else {
12                 toUpdate.push({id: x, professorId: this.years[j].professorId, year: this.years[j].year, hours: this.yearlyDetail[i].hours})
13             }
14         }
15     }
16 }
```

```
9         if (this.toAdd.includes(x)) {
10             const yearlyHours = this.yearlyDetail[i]
11                 .collaborationHoursYearly.find(c => c.year.toString
12 () == this.years[j])
13
14             if (yearlyHours) {
15                 toUpdate.push({
16                     id: yearlyHours.id,
17                     yearExpectedHours: this.getControl(i, j).value,
18                 })
19             }
20             else {
21                 toCreate.push({
22                     collaboration: this.yearlyDetail[i].
23                     collaborationId,
24                     year: parseInt(this.years[j]),
25                     yearExpectedHours: this.getControl(i, j).value,
26                 })
27             }
28         }
29
30         this.yearlyHoursService.createYearlyHours(toCreate).pipe(
31             switchMap(() => this.yearlyHoursService.updateYearlyHours
32             (toUpdate))
33             .subscribe({
34                 next: () => {
35                     this.toAdd = []
36                     this.isError = false
37                     this.showToast = true
38                     this.message = 'Salvataggio completato con successo'
39                 },
40                 error: () => {
41                     this.isError = true
42                     this.showToast = true
43                     this.message = 'Errore nel salvataggio'
```

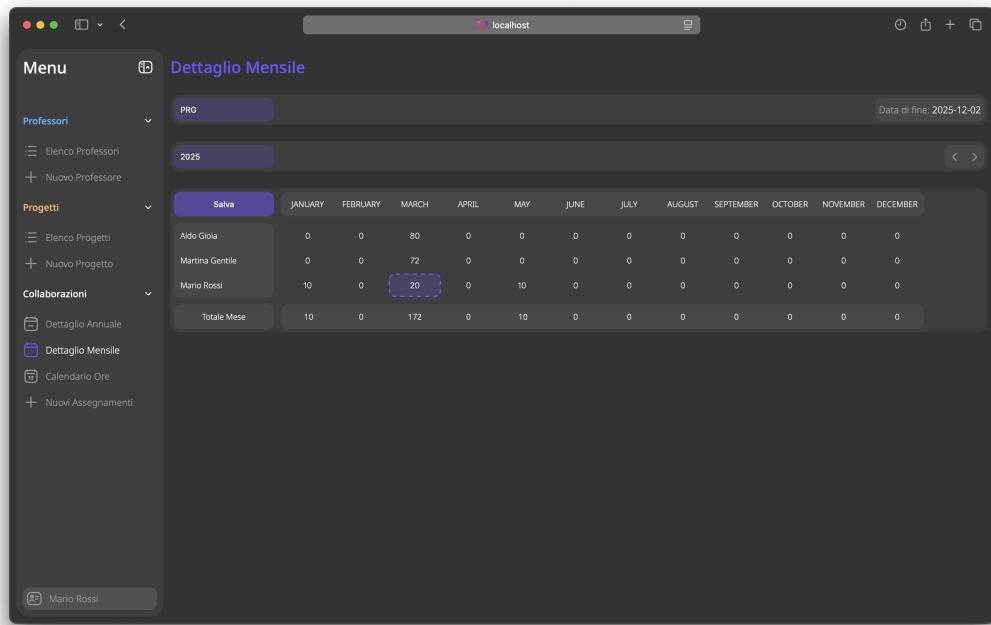


Figura 3.21: Schermata con il dettaglio mensile delle ore su un progetto.

```

43     }
44   })
45   setTimeout(() => {
46     this.showToast = false
47   } , 3000);
48 }
```

3.3.14 Dettaglio Mensile

Per garantire un’organizzazione ancora più granulare, le ore, oltre che distribuite anno per anno, possono anche essere distribuite mese per mese. A tal fine, la schermata “Dettaglio Mensile” (Figura 3.21) riesce a gestire questo aspetto.

Il funzionamento e l’organizzazione delle informazioni sono analoghi a quelli delle precedenti schermate. In particolar modo, il funzionamento segue questi passaggi:

1. si sceglie il progetto;

2. si sceglie l'anno, compreso nella durata del progetto, di cui si vogliono distribuire le ore assegnate;
3. si distribuiscono le ore per ogni professore/mese;
4. si salvano le modifiche tramite il pulsante “Salva”.

Con il pulsante “Salva”, viene eseguito il metodo `save()` che si occupa di inviare i dati, alle API Rest, da dover rendere persistenti.

```

1 save() {
2   const toCreate: MonthlyHours[] = []
3   const toUpdate: UpdateMonthlyHoursDto[] = []
4
5   for (let i = 0; i < this.monthlyDetail.length; i++) {
6     for (let j = 0; j < this.yearsMonths[this.currentYear].
7       months.length; j++) {
8       const x = i*this.yearsMonths[this.currentYear].months.
9         length + j
10
11      if (this.toAdd.includes(x)) {
12        const monthlyHours = this.monthlyDetail[i]
13          .collaborationHoursMonthly.find(c => c.month.toString()
14          () == this.yearsMonths[this.currentYear].months[j])
15
16        if (monthlyHours) {
17          toUpdate.push({
18            id: monthlyHours.id,
19            monthExpectedHours: this.getControl(i, j).value
20          })
21        } else {
22          toCreate.push({
23            collaborationsHoursYearly: this.monthlyDetail[i].
24            collaborationHoursYearly.id,
25            month: this.yearsMonths[this.currentYear].months[j],
26            monthExpectedHours: this.getControl(i, j).value
27          })
28        }
29      }
30    }
31  }
32}
```

```
25     }
26   }
27 }
28
29 this.monthlyHoursService.createMonthlyHours(toCreate).pipe(
30   switchMap(() => this.monthlyHoursService.updateMonthlyHours
31     (toUpdate))
32   ).subscribe({
33     next: () => {
34       this.toAdd = []
35       this.isError = false
36       this.showToast = true
37       this.message = 'Salvataggio completato con successo'
38     },
39     error: () => {
40       this.isError = true
41       this.showToast = true
42       this.message = 'Errore durante il salvataggio'
43     }
44   })
45   setTimeout(() => {
46     this.showToast = false
47   }, 3000);
48 }
```

Capitolo 4

Conclusioni

Nel seguente lavoro di tesi, è stata sviluppata un'applicazione web orientata alla gestione delle risorse umane (professori) a disposizione e dei loro assegnamenti nei vari progetti di ricerca che ogni anno vengono affrontati all'interno delle università. Durante lo sviluppo, sono state affrontate diverse sfide improntate a garantire che il sistema fosse efficiente, scalabile, ma soprattutto sicuro, oltre che a pensare e realizzare una struttura dati adeguata, che permetesse di tenere traccia del contributo, in ore, di ogni professore nei diversi progetti a cui esso è assegnato. L'integrazione tra il FrontEnd e il BackEnd ha richiesto una particolare attenzione, per far sì che comunicassero in sicurezza. A tal fine sono state implementate diverse strategie di sicurezza, come un robusto meccanismo di Rate Limiting per evitare abusi, la gestione dell'autenticazione con autorizzazione basata su ruoli, oltre ad un meccanismo di Audit Logging che garantisse la responsabilità e il non ripudio.

Nonostante i risultati ottenuti, il sistema può essere ulteriormente migliorato, anche grazie all'alto livello di scalabilità offerto dalle scelte implementative adottate. Un possibile sviluppo futuro dell'applicazione, potrebbe essere l'integrazione di sistemi di intelligenza artificiale, ad esempio quelli basati su Answer Set Programming , per supportare ad esempio l'allocazione automatica delle risorse umane sui progetti. In particolare, questi strumenti andrebbero a supportare gli utenti a effettuare delle scelte, suggerendo automaticamente quali professori assegnare a un determinato progetto e definire le ore da alloca-

re sia a livello generale, che annuale e mensile, in base a vincoli e disponibilità delle risorse disponibili.

Bibliografia

- [1] Descrizione di Hibernate. URL: <https://www.turing.com/kb/jpa-for-database-access>. Consultato in data 12/03/2024.
- [2] Descrizione di Jackson. URL: <https://rameshfadatare.medium.com/jackson-library-in-java-eef28ab9fb40>. Consultato in data 12/03/2024.
- [3] Descrizione di Java. URL: <https://aws.amazon.com/what-is/java>. Consultato in data 12/03/2024.
- [4] Descrizione di lombok. URL: <https://objectcomputing.com/resources/publications/sett/january-2010-reducing-boilerplate-code-with-project-lombok>. Consultato in data 12/03/2024.
- [5] Descrizione di Postgresql. URL: <https://aws.amazon.com/it/rds/postgresql/what-is-postgresql/>. Consultato in data 12/03/2024.
- [6] Descrizione di spring boot. URL: <https://www.ibm.com/think/topics/java-spring-boot>. Consultato in data 12/03/2024.
- [7] Descrizione di Typescript. URL: <https://www.geekandjob.com/wiki/typescript>. Consultato in data 12/03/2024.
- [8] Pagina ufficiale di Angular. URL: <https://v17.angular.io/guide/architecture>. Consultato in data 12/03/2024.
- [9] Pagina ufficiale di Figma. URL: <https://help.figma.com/hc/en-us/articles/14563969806359-What-is-Figma>. Consultato in data 12/03/2024.

Ringraziamenti

Ai miei genitori, Fabrizio e Lavinia;

Se oggi sono la persona che sono, che spero voi possiate guardare con orgoglio, e se ho avuto la possibilità di studiare e poter ogni giorno coltivare i miei sogni, oltre ad avere sempre l'appoggio che ogni figlio vorrebbe nelle proprie scelte è merito vostro. Vi voglio bene, grazie per tutto quello che avete fatto e continuerete a fare.

Ai miei nonni, Emilio e Ines;

Un grande grazie va a voi per la vostra costante presenza in ogni giorno importante della mia vita, per il supporto che non è mai mancato, per i consigli sempre pieni di saggezza che non avete mai esitato a darmi e per l'amore profondo con cui mi avete cresciuto. Avrei voluto condividere quest'altro traguardo anche con te, nonna, ma i ricordi che mi legano alla tua persona saranno sempre forti e vivi dentro il mio cuore.

A mia nonna Ilde;

Un grazie doveroso va anche a te, cara nonna, per tutta la stima e l'amore che mi dimostri ogni giorno, da sempre. Le tue parole di incoraggiamento, il tuo affetto e i tuoi gesti mi riempiono di gioia e serenità.

Alle mie cugine, Paola, Maria e Veronica;

Dai nostri primi passi, al giorno della laurea, siete sempre state il porto sicuro in cui rifugiarmi in caso di un problema. Crescere insieme, è stato il dono più bello che la vita potesse regalarmi, vi voglio bene.

Ai miei zii, zie, cugini e cugine; Un grazie speciale va a voi, per l'affetto, l'incoraggiamento e la vicinanza. Sapere di avere una famiglia pronta sempre ad appoggiarti è un valore che mi appaga.

Ai colleghi, Francesco, Pierpaolo e Giuseppe;
Grazie per aver reso le giornate all'università, più leggere, anche solo con una battuta, e per avermi sempre strappato un sorriso. Ogni momento che ha descritto la nostra crescita, lo custodirò sempre nel cuore.

Ai miei amici, Gianmaria, Giuseppe D. e Giuseppe P.;
È sempre bastato farvi solo uno squillo, per avervi vicino, anche quando eravate lontani. In un attimo, mi passa davanti tutto quello che abbiamo vissuto e mi scende una lacrima sapendo che voi oggi, siete ancora qui con me, certo che ci sarete ancora.

A tutti gli altri miei amici;
Grazie per ogni momento passato e presente, a modo vostro, con piccoli gesti, parole o semplici ricordi, avete reso più ricco, il mio cammino.

Alla mia ragazza, Martina;
Sei arrivata nella mia vita nell'ultimo anno di università, grazie a te ho avuto la forza di dare sempre di più e di non arrendermi mai, mi hai insegnato ad affrontare le difficoltà col sorriso ed osservare il mondo da una prospettiva differente, mi hai aiutato a diventare una persona migliore oltre che a "colorare di felicità" tutte le mie giornate, grazie amore mio.