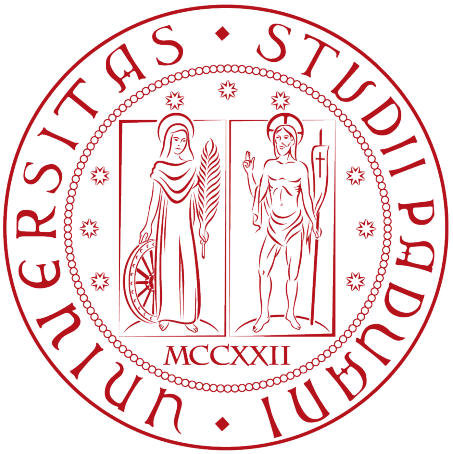
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



Sviluppo di un sistema di gestione prenotazioni con framework Angular e Spring Boot

Tesi di laurea

Relatore
Prof.Giovanni Da San Martino

Laureanda
Denisa Hida

Anno Accademico 2021-2022

Sommario

Il presente documento descrive l'attività di stage svolta presso l'azienda Ifin Sistemi Srl della durata di circa 320 ore. Lo scopo di questo progetto di stage era lo sviluppo di un sistema prenotazioni.
Con l'alternanza del lavoro in sede e in smart è nata l'esigenza di avere a disposizione un'applicazione che dia la possibilità ai lavoratori di prenotare il proprio posto in ufficio e al personale amministrativo di avere una panoramica dei lavoratori presenti in sede.
Il back-end del sistema è stato realizzato utilizzando il framework Spring Boot e il front-end realizzato con framework Angular.

"Yesterday I was clever so I wanted to change the world. Today I am wise, so I am changing myself."

— Rumi

Ringraziamenti

Innanzitutto, vorrei esprimere la mia gratitudine al Prof. Giovanni Da San Martino, relatore della mia tesi, per l'aiuto e il sostegno fornitomi durante la stesura del lavoro.

Vorrei ringraziare il mio tutor aziendale Claudio Mazzuco per tutti i consigli e il supporto duranto lo stage e il personale aziendale per avermi seguita ed aiutata.

Desidero ringraziare con affetto i miei genitori per il sostegno, il grande aiuto e per essermi stati vicini in ogni momento durante gli anni di studio.

Ho desiderio di ringraziare poi i miei amici per tutti i bellissimi anni passati insieme e le mille avventure vissute.

Padova, Dicembre 2022 Denisa Hida

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

1.1 L'azienda

Fondato nel 1981, Ifin Sistemi opera sull'intero mercato italiano occupandosi della gestione dei documenti digitali con soluzioni e servizi per trattare, convertire, archiviare e conservare a norma flussi di dati, informazioni e contenuti.
Con principale sede a Padova Ifin si specializza in:

- Archiviazione documentale
- Conservazione digitale
- Fatturazione elettronica

Il gruppo Ifin copre un ruolo molto importante nel mondo della dematerializzazione, con oltre 2500 clienti sia in aziende pubbliche che private.
L'azienda si divide in tre principali business unit:

- *Ifin Sistemi* che fornisce soluzioni di Document Management per le aziende private e per la pubblica amministrazione;
- *Ifin Med* che offre soluzioni di gestione documentale e conservazione digitale per il mondo clinico/sanitario;
- *Ifin Consulting* che eroga servizi di consulenza normativa e formazione.

Dal 2017 Ifin Sistemi ha lanciato il network TrustedChain, il primo network blockchain privato di Trusted Service Providers (TSP) europei. La società offre servizi blockchain ad altissima performance ed affidabilità.
Come prova della propria qualità, Ifin Sistemi ha ottenuto due certificazioni:

- **UNI CEI ISO/IEC 27001:2013** nel campo di applicazione acquisizione, trattamento finalizzato alla conservazione e gestione dei dati e documenti digitali, che rappresenta lo standard internazionale per gli aspetti legati alla sicurezza delle informazioni.
- **ISO 9001:2015** con campi di applicazione:
  - Progettazione, sviluppo, installazione di soluzioni ICT e relativi servizi di assistenza tecnica applicativa.

– Erogazione di servizi di consulenza in ambito tecnologico, organizzativo e gestionale.

Figura 1.1: Logo Ifin Sistemi s.r.l.

1.2 Il progetto

Numerose sono state le aziende che hanno introdotto lo *Smart Working* come nuova modalità di lavoro all'interno di specifici reparti aziendali. Già impiegato nelle aziende più all'avanguardia prima della pandemia di Covid-19, il lavoro in Smart Working si è affermato significativamente a seguito del protrarsi dell'emergenza pandemica. Come conseguenza si è visto il bisogno di avere una piattaforma che dia la possibilità ai lavoratori di prenotare il proprio posto in ufficio e al personale amministrativo di avere una panoramica dei lavoratori presenti in sede.
Gli obiettivi di questo stage erano:

– Analisi e progettazione della *web application*;

– Implementazione

– Verifica e collaudo del prodotto ultimato

1.3 Pianificazione del lavoro

Lo *stage* è stato svolto nel arco di 8 settimane per un totale di 320 ore. La divisione del lavoro è stato riportato nella tabella seguente:

Settimana	Attività
1	Formazione sullo *stack* operativo
2	Analisi e progettazione del *backend*
3	Progettazione della basi di dati e sviluppo
4	Implementazione di *backend* e *testing*
5	Sicurezza del applicativo con Spring Security
6	Formazione su JWT e implementazione
7	Formazione su framework Angular
8	Implementazione di *front-end*

Tabella 1.1: Tabella della pianificazione del lavoro

1.4 Organizzazione del testo

Il secondo capitolo riguarda l'analisi dettagliata del problema e del prodotto, e comprende la lista dei requisiti individuati;

Il terzo capitolo specifica quali strumenti e tecnologie sono stati analizzati e utilizzati;

Il quarto capitolo approfondisce la progettazione e codifica del prodotto *software*;

Il quinto capitolo descrive come sono state attuate le attività di verifica e validazione;

Il sesto capitolo riassume la valutazione personale del prodotto e le conoscenze acquisite.

1.5 Convenzioni tipografiche

Riguardo la stesura del testo, relativamente al documento sono state adottate le seguenti convenzioni tipografiche:

- gli acronimi, le abbreviazioni e i termini ambigui o di uso non comune menzionati vengono definiti nel glossario, situato alla fine del presente documento;

- i nomi di classi, funzioni o membri di una classe vengono evidenziati come `class`
- i termini in lingua straniera o facenti parti del gergo tecnico sono evidenziati con il carattere *corsivo*.

Capitolo 2

Descrizione del progetto

In questo capitolo vengono elencati e descritti i casi d'uso emersi nella fase di analisi del software e vengono tracciati i requisiti.

2.1 Introduzione

Lo scopo di questo progetto è la creazione di una web application che permetta ai dipendenti IFIN di poter prenotare (e gestire le prenotazioni) delle postazioni presenti nei vari uffici. Prima di progettare e sviluppare l'applicazione web, abbiamo effettuato un'analisi del problema con il tutor aziendale e gli *stakeholders*. Essendo un'applicazione esclusiva, solo i dipendenti dell'azienda ne possono avere accesso. Questo obbligo significa che gli utenti non possono registrarsi.
Si è pensato che nella prima versione del software gli utenti vengano inseriti nel *database* manualmente da un utente *admin*. Per poi aggiungere la possibilità di autenticazione tramite email aziendale.
L'interno dell'azienda è strutturato in uffici, ogni ufficio è composto da un determinato numero di postazioni. L'utente deve avere la possibilità di prenotare una postazione per un determinato periodo di tempo e deve avere la possibilità di liberare il posto prima della giornata scelta.
Una prenotazione consiste nell'assegnare un posto ad un utente per un insieme di giornate lavorative.
Un utente rappresenta un dipendente IFIN o un collaboratore che comunque è censito nei sistemi IFIN.
Una postazione rappresenta una scrivania in una stanza.
Una Stanza ha un nome identificativo ed un numero di posti totali di cui dispone.
Un'altra caratteristica del sistema è il periodo di prenotazione per una giornata. Un utente deve avere la possibilità di prenotare un posto anche per solo metà della giornata. Questo porta a definire i periodi di prenotazione in:

- Mattina
- Pomeriggio
- Giornata Intera

2.2 Requisiti

Di seguito si elencano i requisiti individuati nella fase di analisi. Ogni requisito è classificato e identificato univocamente attraverso un codice composto secondo il seguente schema:

`R[Priorità][Identificativo]`

Dove:

- **R** sta per requisito
- **Priorità** può assumere i seguenti valori:
  - **O**: obbligatorio;
  - **D**: desiderabile;
  - **F**: facoltativo.
- **Identificativo**: numero progressivo che identifica il requisito.

Codice	Descrizione
RO1	L'utente deve riuscire ad accedere con email e password
RO2	L'utente deve riuscire a selezionare una data
RO3	L'utente può visualizzare solo gli uffici liberi
RO4	L'utente deve riuscire a selezionare un ufficio libero
RO5	L'utente può visualizzare solo le postazioni liberi
RO6	L'utente deve riuscire a prenotare una postazione libera nell'ufficio scelto
RO7	L'utente deve riuscire a vedere le proprie prenotazioni
RO8	L'utente deve riuscire a eliminare una prenotazione in avvenire
RO9	L'utente non può eliminare una prenotazione del passato o del presente
RO10	L'utente deve riuscire a scollegarsi dal sistema
RD1	L'utente può scegliere la postazione e non solo la stanza
RD2	Test di unità per la parte back-end
RD3	Test di integrazione per back-end
RD4	Test per la parte front-end
RD5	Documentazione
RF1	Funzionalità aggiuntive per l'utente *admin*

Tabella 2.1: Tabella dei requisiti

2.3 Obiettivi formativi

Gli obiettivi formativi dell'attività di stage sono i seguenti:

- Apprendere come sviluppare una *web-app* seguendo le *best practice* utilizzate per l'implementazione di prodotti a livello *enterprise*;
  - *Agile methods*

  - Domain Driven Design (DDD)
  - Test Driven Design (TDD)

- Apprendere ad utilizzare le tecnologie necessarie per lo sviluppo di un'applicazione web

Capitolo 3

Strumenti e tecnologie

Analisi e descrizione delle tecnologie utilizzate

3.1 Strumenti utilizzati

3.1.1 Ambiente di sviluppo

Come ambiente di sviluppo per il backend si è utilizzato *IntelliJ IDEA*. *IntelliJ* è un Integrated Development Environment (IDE) intelligente e sensibile al contesto per lavorare con *Java* e altri linguaggi come *Kotlin*, *Scala* e *Groovy*, su tutti i tipi di applicazioni. Fornisce un rapido accesso a tutte le funzionalità e gli strumenti integrati importanti per lavorare, oltre a un'ampia gamma di opzioni di personalizzazione. Dà la possibilità di impostare scorciatoie, installare *plug-in*, personalizzare l'interfaccia e altro ancora. Una caratteristica molto utile è anche il completamento del codice e i suggerimenti intelligenti, questo perché utilizza l'apprendimento automatico per garantire che il suggerimento più pertinente sia in cima all'elenco.

Figura 3.1: IntelliJ logo

Per lo sviluppo del componente *front-end* si è utilizzato *Visual Studio Code*.

Figura 3.2: Visual Studio Code logo

3.1.2 Strumenti di versionamento

Come strumento per il controllo di versione l'azienda usa *Git*. Git è un sistema di controllo versione distribuito, gratuito e open source progettato per gestire i progetti con velocità ed efficienza.
Git è facile da imparare e supera gli altri strumenti di Version Control System (VCS) con le sue funzionalità di *branching* locale, aree di *staging* e flussi di lavoro multipli. In supporto a Git è stato adottato anche *Git Flow*, un flusso di lavoro che descrive un modello di diramazione ben preciso costruito intorno al concetto di *software release*. Rispetto allo sviluppo basato su *trunk*, Gitflow ha numerosi branch di più lunga durata e *commit* di maggiori dimensioni. Con questo modello, gli sviluppatori creano un branch di funzioni e ne posticipano il merge nel branch principale del trunk fino a quando la funzione non è completata.

Figura 3.3: Git logo

Come servizio di web-based hosting, è stato utilizzato *Bitbucket*, strumento che fa parte della suite *Atlassian*.

Figura 3.4: Bitbucket logo

3.1.3 Strumenti aziendali

Per scrivere documentazione relativa ai vari software, l'azienda usa *Confluence*, una piattaforma che fa parte della suite Atlassian. Confluence è un editor collaborativo che dà la possibilità di creare note di riunione, piani di progetto, requisiti di prodotto, mentre altri utenti stanno modificando e vedono tutte le modifiche contemporaneamente. È stato usato per la scrittura del manuale utente.
Un altro strumento molto utilizzato è stato anche *Jira*. Siccome Jira consente il *bug tracking* e la gestione dei progetti sviluppati con metodologie agili, ogni ticket aperto su Jira corrisponde ad un *branch(feature)* su BitBucket: in questo modo tutti i commit relativi ad un certo *ticket* vengono tracciati e risultano facilmente reperibili.

Figura 3.5: Confluence logo

Figura 3.6: Jira logo

Per quanto riguarda la comunicazione con il tutor e il personale aziendale, sono stati utilizzati Google Chat e Google Meet.

Figura 3.7: Google Chat logo

Figura 3.8: Google Meet logo

3.1.4 Altri strumenti

Docker

Docker è una piattaforma di *containerizzazione* che consente di creare, distribuire ed eseguire applicazioni comodamente con l'aiuto dei container. Si occupa fondamentalmente del confezionamento delle applicazioni con tutte le librerie richieste e altre dipendenze in un contenitore da parte dello sviluppatore. Docker è stato lanciato nel 2013 dalla società tecnologica americana Docker, Inc., precedentemente nota come dotCloud. Insieme ai contenitori, Docker ha anche molti altri componenti principali come: immagini Docker, file Docker, registri Docker, ecc. Con l'aiuto di Docker, gli sviluppatori sono in grado di scrivere codice o creare applicazioni senza preoccuparsi dell'ambiente. I vantaggi dell'utilizzo di Docker sono numerosi tra cui:

Ambiente coerente e isolato Si assume la responsabilità di isolare e segregare le app e le risorse in modo tale che ogni contenitore possa accedere a tutte le risorse richieste in modo isolato, cioè senza disturbare o dipendere da un altro contenitore. Alla fine consente di eseguire più contenitori contemporaneamente sullo stesso host. Inoltre, poiché ogni contenitore può accedere solo alle risorse assegnate, aiuta a ridurre il rischio di diversi potenziali problemi come tempi di inattività, ecc. Inoltre, è possibile rimuovere facilmente qualsiasi app eliminando il suo contenitore che non lascerà alcun file temporanei sul sistema.

Distribuzione rapida delle applicazioni Docker velocizza il processo di distribuzione dell'applicazione. Organizza in modo efficiente l'intero ciclo di vita dello sviluppo fornendo un ambiente di lavoro standardizzato agli sviluppatori. I contenitori Docker presentano i requisiti minimi di *runtime* dell'applicazione che consentono loro una distribuzione più veloce.

Garantisce scalabilità e flessibilità Le immagini Docker possono essere facilmente ordinate su più server. Ad esempio, se viene richiesto di eseguire un aggiornamento durante il rilascio dell'applicazione, si possono apportare le modifiche ai contenitori Docker, testarli e implementare nuovi contenitori. Oltre a questo, si può ripulire o riparare in modo efficiente l'applicazione senza rimuoverla completamente. Docker ha la capacità di essere distribuito in più server fisici, server di dati o piattaforme cloud. Inoltre, Docker consente di creare rapidamente repliche per motivi di ridondanza e consente di avviare e terminare tempestivamente l'applicazione o i servizi.

Migliore portabilità Un altro vantaggio arricchente di Docker è la portabilità! Le applicazioni create con i contenitori Docker sono portabili e possono essere

eseguiti su qualsiasi piattaforma, come Amazon EC2, Google Cloud Platform, VirtualBox, server Rackspace o qualsiasi altro, sebbene il sistema operativo host debba supportare Docker. Poiché l'applicazione e tutte le sue dipendenze sono raggruppate in un contenitore Docker, è facile distribuirla su qualsiasi sistema che supporti Docker e l'applicazione funzionerà in modo simile.

Figura 3.9: Docker logo

PostgreSQL

PostgreSQL è un RDMS open source, simile a MySQL. È un database orientato agli oggetti, ma si possono elaborare sia dati strutturati che non strutturati.
Il motore di database PostgreSQL funziona su vari sistemi operativi, tra cui Windows, Mac OS X e Linux. Fornisce inoltre tipi di dati avanzati e funzionalità di ottimizzazione delle prestazioni per archiviare e ridimensionare complessi carichi di lavoro di database.

Figura 3.10: PostgreSQL logo

Flyway

È stato usato **Flyway** come strumento di migrazione *database*. Flyway permette di:

- Ricreare un database da zero;
- Sapere lo stato che si trova il database;
- Migrare in un modo determinato, dalla versione corrente del database a una nuova versione.

Quando si aggiunge al progetto, Flyway crea una tabella chiamata `flyway_schema_history` che tiene traccia dello stato del database.

installed_rank	version	description	type	script	checksum	installed_by	installed_on	execution_time	success
1	1	Initial Setup	SQL	V1__Initial_Setup.sql	1996767037	axel	2016-02-04 22:23:00.0	546	true
2	2	First Changes	SQL	V2__First_Changes.sql	1279644856	axel	2016-02-06 09:18:00.0	127	true
3	2.1	Refactoring	JDBC	V2_1__Refactoring		axel	2016-02-10 17:45:05.4	251	true

Figura 3.11: Flyway history table

Ad ogni migrazione la tabella si aggiorna come nella figura 3.11.

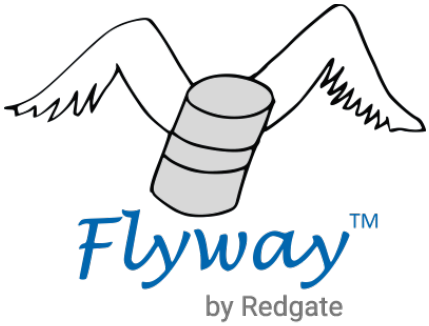


Figura 3.12: Flyway logo

3.2 Tecnologie utilizzate

3.2.1 Linguaggi

Java

Il linguaggio utilizzato per la parte *back-end* del progetto è stato Java, versione 17. Java è un linguaggio orientato agli oggetti, concorrente, fortemente tipizzato e basato su classi. Normalmente è compilato nel set di istruzioni bytecode e nel formato binario definito nella specifica Java Virtual Machine (JVM).

Typescript

TypeScript estende JavaScript e consente agli sviluppatori di aggiungere *type safety* ai propri progetti. Inoltre, TypeScript fornisce varie altre funzionalità, come interfacce, alias di tipo, classi astratte, *overloading* di funzioni, generici, ecc.

Figura 3.13: Java logo

Figura 3.14: TypeScript logo

3.2.2 Framework

Spring boot

Spring è un progetto *open source* che fornisce un approccio modulare semplificato per la creazione di app con Java. La famiglia di progetti Spring è iniziata nel 2003 come risposta alle complessità dello sviluppo Java iniziale e fornisce supporto per lo sviluppo di app Java. La funzionalità principale di Spring è la Dependency Injection (DI), la quale è una tecnica di progettazione usata per ottenere l'inversione del controllo (IoC).
Spring Boot è un modulo specifico, creato come estensione del *framework* Spring. Consente di creare applicazioni Spring autonome che possono essere eseguite immediatamente senza necessità di annotazioni, configurazione XML o scrittura di una quantità elevata di codice aggiuntivo. Spring Boot configura automaticamente Spring e altri *framework* di terze parti in base al principio predefinito "Convention over configuration".
Tra i molti moduli del *framework* Spring, per questo progetto ho studiato e usato Spring Data Jakarta Persistence API (JPA) e Spring Security.

Spring Data JPA Spring Data JPA fornisce supporto di *repository* per l'Application Program Interface (API) Jakarta Persistence (JPA). L'interfaccia centrale nell'astrazione del *repository* è `Repository`. Richiede la classe di dominio da gestire e il tipo di ID della classe di dominio come argomenti di tipo. Questa interfaccia funge principalmente da interfaccia *marker* per acquisire i tipi con cui lavorare e per aiutare a scoprire interfacce che estendono questa. Le interfacce `CrudRepository` e `ListCrudRepository` forniscono funzionalità Create Read Update Delete (CRUD) per la classe di entità gestita.

```
public interface CrudRepository<T, ID> extends Repository<T,
    ID> {

  <S extends T> S save(S entity);       (1)

  Optional<T> findById(ID primaryKey); (2)

  Iterable<T> findAll();                (3)

  long count();                         (4)

  void delete(T entity);                (5)

  boolean existsById(ID primaryKey);    (6)

  //     more functionality omitted.
}
```

(1) Salva l'entità data.

(2) Restituisce l'entità identificata dall'ID specificato.

(3) Restituisce tutte le entità.

(4) Restituisce il numero di entità.

(5) Elimina l'entità data.

(6) Indica se esiste un'entità con l'ID specificato.

Spring Security *Spring Security* fornisce servizi di sicurezza completi per applicazioni software aziendali basate su Jakarta EE (J2EE). Le due aree principali di *Spring Security* sono autenticazione e autorizzazione. A livello di autenticazione, Spring Security supporta un'ampia gamma di modelli di autenticazione. La maggior parte di questi modelli di autenticazione sono forniti da terze parti o sono sviluppati da organismi di standardizzazione. Inoltre, Spring Security fornisce il proprio set di funzionalità di autenticazione. Nello specifico, supporta l'integrazione dell'autenticazione con le tecnologie menzionate sotto e altro ancora:

- Intestazioni di autenticazione HTTP BASIC (uno standard basato su IEFT RFC)
- Intestazioni di autenticazione HTTP Digest (uno standard basato su IEFT RFC)
- Scambio di certificati client HTTP X.509 (uno standard basato su IEFT RFC)
- LDAP (un approccio molto comune alle esigenze di autenticazione multi-piattaforma, specialmente in ambienti di grandi dimensioni)
- Autenticazione basata su moduli (per semplici esigenze di interfaccia utente)
- Autenticazione OpenID
- Autenticazione basata su intestazioni di richiesta prestabilite (come Computer Associates Siteminder)
- Servizio di autenticazione e autorizzazione Java (JAAS)

Il diagramma seguente mostra come vengono elaborate le richieste di autenticazione:

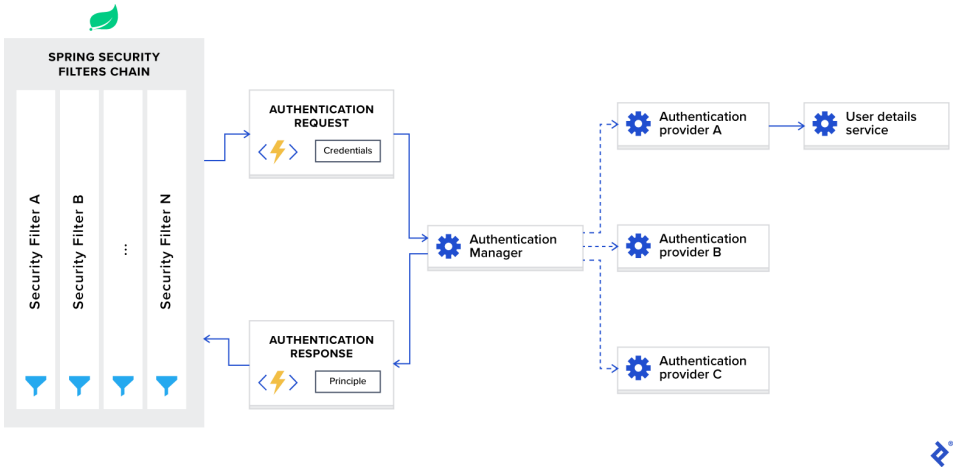


Figura 3.15: Architettura Spring Security

Angular

Angular è un framework *JavaScript* per sviluppare un Single Page Application (SPA). A differenza di altri framework che offrono data binding unidirezionale, Angular fornisce data binding bidirezionale. Sincronizza i dati tra Modello e Vista. Pertanto, quando i dati vengono modificati, questi due componenti vengono aggiornati automaticamente in tempo reale.
L'architettura di Angular è basata su componenti, ciò porta a un metodo efficiente per testare le unità separatamente. Ogni componente ha:

- Un modello HTML che dichiara ciò che viene visualizzato nella pagina.
- Classe Typescript che definisce il comportamento.
- Un selettore CSS che definisce come il componente viene utilizzato in un modello.
- Stili CSS applicati al modello.

In Angular, la DI consente di creare un oggetto utilizzando altri oggetti. Allo stesso tempo, le modifiche alla definizione degli oggetti utilizzati non influiscono in alcun modo sull'oggetto creato. Il nucleo di Angular ha la propria implementazione del pattern Dependency Injection.
Angular ha un'interfaccia a riga di comando integrata che semplifica la creazione di nuovi progetti. Con i seguenti comandi si avvia un nuovo progetto:

```
npm install -g @angular/cli

ng new my-first-project

cd my-first-project

ng serve
```

Figura 3.16: Angular logo

Capitolo 4

Progettazione e codifica

In questo capitolo viene descritta l'architettura del back-end e del front-end, le funzionalità, la loro progettazione e codifica.

4.1 Progettazione back-end

4.1.1 Architettura

Dopo aver definito i requisiti nella fasi di analisi, con l'aiuto del tutor aziendale abbiamo iniziato a pensare sull'architettura del back-end.
È stato deciso di adottare l'*onion architecture*. L'onion architecture mira di separare il *"business logic"* dall'interfaccia utente. Il centro del applicazione non è più il database ma la logica di business.

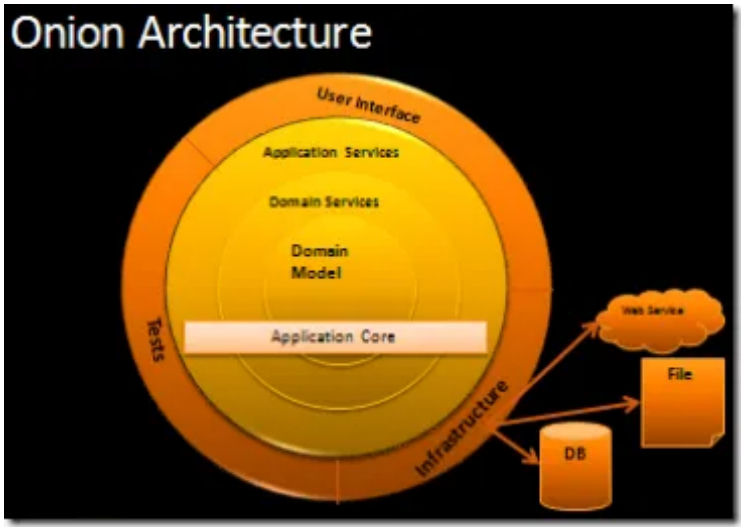


Figura 4.1: Onion Architecture

Questa architettura controlla l'accoppiamento. Tutto il codice può dipendere da strati più centrali, ma il codice non può dipendere da strati più lontani dal nucleo. Cioè, tutti gli accoppiamenti sono verso il centro. Questa architettura è sbilanciata verso la programmazione orientata agli oggetti e mette gli oggetti prima di tutti gli

altri.
Nella figura 4.1, al centro si trova il Modello di Dominio. Intorno al modello di dominio ci sono altri livelli con più comportamento. Il numero di livelli nel core dell'applicazione varierà, ma ricorda che il modello di dominio è il vero centro e poiché tutti gli accoppiamenti sono verso il centro, il modello di dominio è accoppiato solo a se stesso.
L'architettura *Onion* si basa fortemente sul principio di inversione delle dipendenze. Il nucleo dell'applicazione richiede l'implementazione delle interfacce principali e se quelle classi di implementazione risiedono ai margini dell'applicazione, abbiamo bisogno di un meccanismo per iniettare quel codice in fase di esecuzione in modo che l'applicazione possa fare qualcosa di utile.

4.1.2 Modello di dominio

Con l'architettura scelta, abbiamo iniziato a modellare il dominio. Le entità fondamentali individuate all'inizio per lo sviluppo di questa applicazione erano:

- `User`: utente che utilizza il sistema;
- `Office`: ufficio con un identificativo unico che appartiene ad una specifica aerea, contiene un numero determinato di postazioni;
- `Booking`: la prenotazione effettuata da un utente identificato per una giornata determinata in un ufficio definito;
- `Area`: area con un identificativo unico che contiene un numero determinato di uffici.

Con questo modello, l'intenzione era di non specificare la postazione dentro l'ufficio. L'utente potrà solo scegliere l'ufficio preferito e occupare un posto non definito. Questo significa che la postazione non ha nessun importanza.
Prima di continuare con la modellazione si è discusso con i stakeholders e il personale aziendale questa prima versione. Si è vista una preferenza sulla possibilità di scegliere la postazione. Inoltre per questa prima versione dell'applicazione si è deciso di non includere le aree siccome gli uffici per come sono nominati, possono essere identificati al di fuori del contesto di un'aerea.
Si elencano così le entità:

- `User`
- `Office`
- `Desk`
- `Booking`

4.1.3 Struttura

Seguendo le regole dell'onion architecture, il back-end si struttura in:

- *Domain Layer*

- *Infrastructure Layer*
- *Application Layer*

Domain Layer Il *domain layer* contiene le classi del modello del dominio individuate nella sezione precedente e le interfacce `Repository` e `Service`.

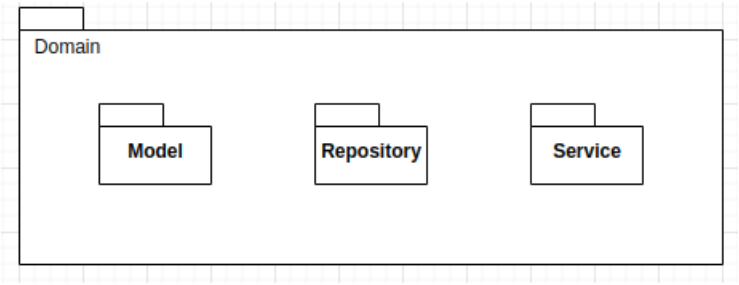


Figura 4.2: Diagramma dei package: Domain layer

Nel *package* `Model` si trovano gli aggregati, i loro *value object* e i *child entity*. Un aggregato è composto da almeno un'entità: la radice dell'aggregato, detta anche entità radice o entità primaria. Inoltre, un aggregato può avere più entità figli e oggetti di valore, con tutte le entità e gli oggetti che lavorano insieme per implementare il comportamento e le transazioni richieste. Un oggetto valore è un oggetto senza identità concettuale che descrive un aspetto di dominio.

- `Booking` : la classe che modella l'aggregato *Booking*, la qualle contiene gli elencati membri:
  - `bookingId: BookingId`, dove `BookingId` è la classe che rappresenta l'identificatore unico per la prenotazione, composta da un `UUID`;
  - `bookingDay: Day`, `Day` è un **Value Object** che rappresenta una data e l'intervallo della giornata: `Day{bookingDate: LocalDate, dayInterval: DayInterval}`. `DayInterval` è l'enumerazione che rappresenta la divisione della giornata in intervalli di tempo:
    - `AFTERNOON`
    - `FULL_DAY`
    - `MORNING`
  - `userId: UserId`, `UserId` è una classe che rappresenta l'identificazione unica del utente che ha prenotato.
  - `deskNumber: Desk`, `Desk` è la classe che rappresenta la postazione scelta, è composta da:
    - `deskNumber: Integer`
    - `officeId: OfficeId`, dove `OfficeId` è una classe che rappresenta l'identificatore unico dell'ufficio, composta da un `UUID`.
- `User`: la classe che rappresenta l'aggregato `User`, è composta da:

  - `userId: UserId`, dove `UserId` è una classe che rappresenta l'identificatore unico di un utente.
  - `email: String`
  - `password: String`
  - `role: String`, ruolo dell'utente che ha acceduto al sistema.

- `Office`: rappresenta l'aggregato ufficio ed è composta da:
  - `officeId: OfficeId`, dove `OfficeId` è la classe composta da un `UUID` per identificare l'ufficio.
  - `officeName: String`, il nome dell'ufficio.
  - `available: boolean`, in casi eccezionali dove l'ufficio non si può prenotare, diventa *true*.

- `Desk`: l'entità figlio dell'ufficio, che rappresenta una postazione, composta da:
  - `deskNumber: Integer`
  - `officeId: OfficeId`
  - `available: boolean`, in casi eccezionali dove la postazione non si può prenotare, diventa *true*.

La scelta di usare classi personalizzate per rappresentare l'identificatore di un entità invece che usare direttamente un `UUID` per esempio, o un `String`, è perché, se in un futuro ci sarà il bisogno di cambiare l'identificativo, sarà molto più facile modificare solo una classe invece che modificare ogni singola classe.

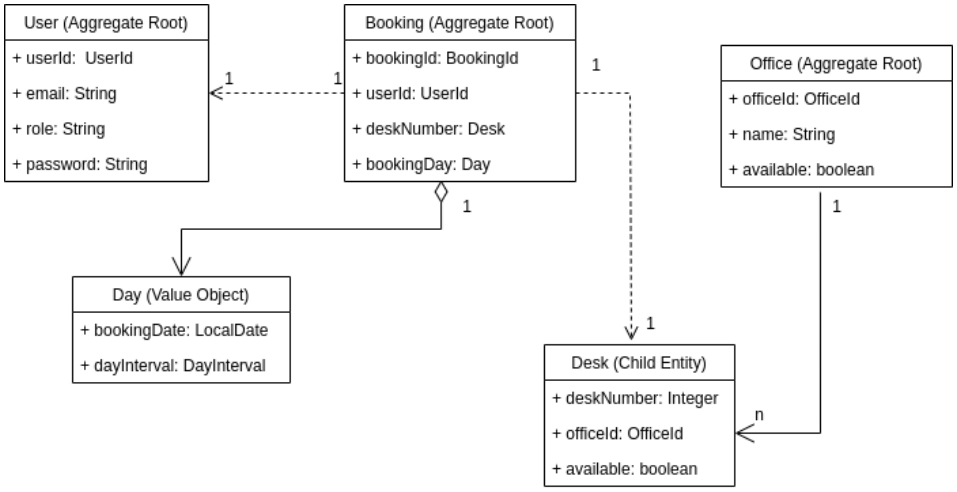


Figura 4.3: Diagramma delle classi per il modello del dominio

Il pacchetto `Repository` contiene le interfacce:

- `BookingRepository`
- `UserRepository`

- `OfficeRepository`

Il pacchetto `Service` invece contiene le interfacce:

- `BookingService`
- `UserService`
- `OfficeService`

Queste interfacce definiscono le funzionalità che servono al dominio.

Infrastructure Layer I componenti di persistenza dei dati forniscono l'accesso ai dati ospitati all'interno del database. In questo strato non viene implementato nessuna logica di *business* ma solo le classi *Repository*. I repository sono classi o componenti che incapsulano la logica richiesta per accedere alle origini dati. Centralizzano le funzionalità di accesso ai dati comuni, fornendo una migliore manutenibilità e disaccoppiando l'infrastruttura o la tecnologia utilizzata per accedere ai database dal livello del modello di dominio.
Per questo strato si è scelto di modellare il dominio in modo che sia compatibile con le regole di Sprig Data Jpa. Siccome abbiamo usato la repository `CrudRepository <T , ID >`, ci servono classi che modellano il dominio senza aver bisogno di modificare il dominio. Perciò abbiamo implementato il modello dello strato *infrastructure*, le classi di cui possono essere annottati con `@Entity`, indicando così che sonno delle entità JPA.

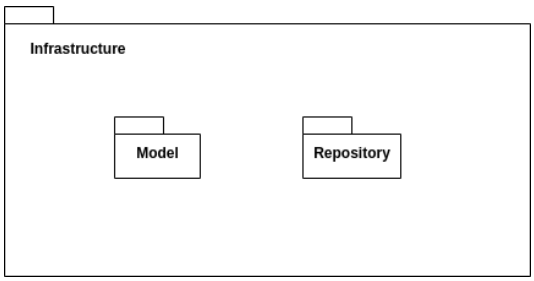


Figura 4.4: Diagramma dei package: Infrastructure layer

Il modello in questo strato non è altro che il modello del dominio con funzionalità che permettono di modificare i membri, e membri più semplificati; esempio le classi `UserId, BookingId, OfficeId` diventano dei semplici UUID.
Per ogni aggregato o radice di aggregazione, abbiamo creato una classe di repository che implementa l'interfaccia repository del dominio e usa la classe che estende `CrudRepository`. Di seguito il diagramma Unified Modeling Language (UML) che descrive meglio il caso con `BookingRepository`.

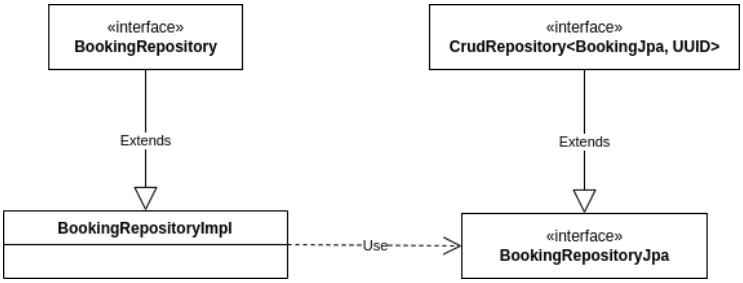


Figura 4.5: Esempio delle classi repository

Application Layer Lo strato dove abbiamo implementano le classi *Service* e *Controller*. Le classi *Controller* sono l'API del applicativo. Ricevono le richieste e ottengono un risultato. Il risultato è l'esecuzione corretta della richiesta o una eccezione. In caso di eccezione lo stato del sistema rimane invariato. Un *Controller* esegue i seguenti passaggi:

- Riceve la richiesta HTTP
- Convalida che la richiesta è valida; per farlo abbiamo aggiunto una dipendenza nel Project Object Model (POM):

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-validation</artifactId
    >
</dependency>
```

- Crea un'istanza della radice aggregata che è la destinazione della richiesta HTTP corrente.
- Esegue il metodo sull'istanza della radice aggregata, ottenendo i dati richiesti dalla richiesta HTTP.
- Rende persistente il nuovo stato dell'aggregato nel relativo database.

Per ogni aggregato abbiamo implementato il proprio *Controller* annottato con `@RestController`:

- `BookingController`
- `OfficeController`
- `UserController`

I *service* sono servizi responsabili solo dell'orchestrazione dei passaggi per le richieste e non implementano alcuna logica aziendale.
In questo strato vengono implementati anche i Data Transfer Object (DTO) che si trovano nel pacchetto **model**.

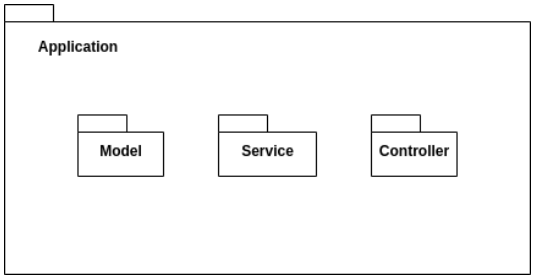


Figura 4.6: Diagramma dei package: Application Layer

4.1.4 Database

Per il database abbiamo usato un contenitore Docker. Per prima si esegue il pull dell'immagine pubblica postgres disponibile su https://hub.docker.com/_/*postgres* per poi eseguire il contenitore Docker utilizzando il comando seguente:

```
$ docker run -itd -e POSTGRES_USER=admin -e POSTGRES_PASSWORD=
    password -p 5432:5432  --name postgresql postgres
```

Le tabelle nel database corrispondono alle entità:

- Booking
- Office
- Desk
- User

4.1.5 Testing

Durante lo sviluppo delle nuove funzionalità abbiamo seguito le regole del TDD:

1. Scrivere il codice di produzione solo per superare un *unit test* fallito.
2. Non scrivere più di un *unit test* sufficiente per fallire.
3. Non scrivere più codice di produzione del necessario per superare l'unico *unit test* non riuscito.

Per quanto riguarda le librerie usate, abbiamo usato:

- JUnit
- TestContainers

TestContainers consiste in una libreria Java che, integrandosi con JUnit, permette l'utilizzo di container Docker "usa e getta".La libreria permette di automatizzare la creazione e la distruzione dei container necessari ad eseguire i test.

4.2 Sicurezza

Dopo l'implementazione della struttura del applicativo abbiamo aggiunto la sicurezza. Aggiungendo la dipendenza Spring Security nel POM:

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

otteniamo un'autenticazione *basic* implementando nel seguente modo la classe `WebSecurityConfigurer`:

```
@Configuration
@EnableWebSecurity
public class CustomWebSecurityConfigurer {


    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http)
       throws Exception {
        http.authorizeRequests()
            .antMatchers("/login")
            .permitAll()
            .anyRequest()
            .authenticated()
            .and()
            .httpBasic()
            .authenticationEntryPoint(authenticationEntryPoint
               );
        http.addFilterAfter(new CustomFilter(),
           BasicAuthenticationFilter.class);
        return http.build();
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}
```

Per far funzionare la configurazione ci sono due metodi:

- Autenticazione *In Memory*
- Implementare `UserDetailsService` e creare la classe `ApplicationUserDetails` che estende `UserDetails` o `org.springframework.security.core.userdetails.User` per abilitare la autenticazione usando il database

`UserDetailsService` contiene solo un metodo:

```
public interface UserDetailsService {
    UserDetails loadUserByUsername(String username) throws
       UsernameNotFoundException;
}
```

Poi abbiamo creato la classe `ApplicationUserDetails` come segue:

```
public class ApplicationUserDetails extends org.
    springframework.security.core.userdetails.User
{
     private final UserId userId;

     public ApplicationUserDetails(User user) {
          super(user.getEmail(), user.getPassword(),
                                createAuthorities(user.getRoles()))
              ;
          this.userId = user.getId();
     }
     public UserId getUserId() {
          return userId;
     }

     private static Collection<SimpleGrantedAuthority>
        createAuthorities(Set<UserRole> roles) {
     return roles.stream().map(
          userRole -> new SimpleGrantedAuthority(userRole.name()
              ))
          .collect(Collectors.toSet());
     }
}
```

Con questo configurazione le richieste HTTP provenienti devono essere autenticate altrimenti viene lanciata una eccezione. L'autenticazione si fa inserendo *username* e *password*.

4.2.1 JWT

JSON Web Tokens (JWT) è uno standard comunemente usato per rendere sicure le API REST. Nel processo di autenticazione JWT, il front end invia per primo alcune credenziali da utilizzare per l'autenticazione (nel nostro caso, username e password). Il server (back-end) poi, controlla le credenziali ricevute, se sono valide genera il JWT e lo restituisce.

Dopo questo passaggio, il client deve inviare questo token nell'intestazione di Autorizzazione della richiesta, nel modulo "Bearer TOKEN". Il back-end controllerà la validità del token e autorizzerà o meno la richiesta. Il token contiene informazioni sull'utente e le relative autorizzazioni in base al suo ruolo.

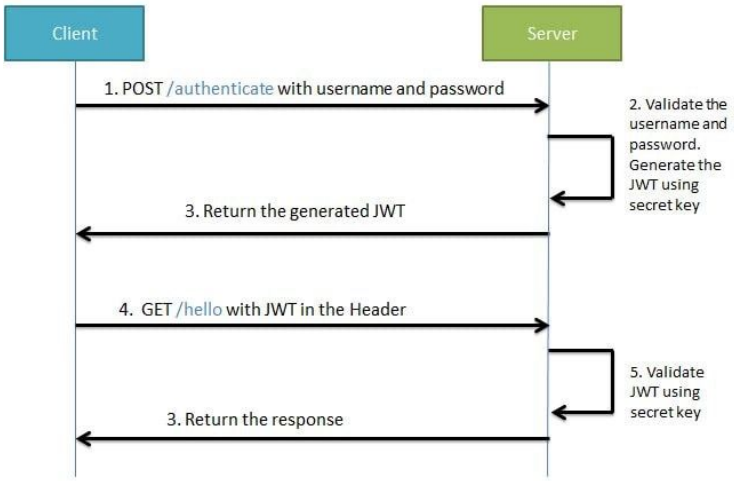


Figura 4.7: Esempio di come JWT funziona

Per implementare questo metodo di autenticazione abbiamo aggiunto la dipendenza:

```
<dependency>
  <groupId>com.auth0</groupId>
  <artifactId>java-jwt</artifactId>
  <version>4.2.1</version>
</dependency>
```

La documentazione della libreria **authO** si trova nel seguente link: `https://github.com/auth0/java-jwt` e per una maggiore variazione di librerie si consulta `https://jwt.io/libraries`.
Dopo aver aggiunto le librerie abbiamo creato le classi:

- `JwtProvider` per la gestione del token JWT
- `AuthorizationFilter` per l'autorizzazione delle chiamate HTTP in ingresso
- `SecurityConfig` per tutte le altre configurazioni richieste da Spring Security

4.3 Progettazione front-end

4.3.1 Architettura

Essendo Angular il framework usato per lo sviluppo del front-end, l'architettura è Model View Controller (MVC).
Avendo pronta la struttura del back-end il front-end è stato più facile da progettare.

Model Il modello è composto dagli oggetti DTO che vengono usati per mandare le richieste e ricevere le risposte dal server. Questi oggetti sono delle classi/interfacce simili alle entità create nel back-end.

View La View è composta da i componenti di Angular.I componenti sono il blocco di base dell'interfaccia utente di un'app Angular. Usando la CLI è molto facile avere il *template* di un componente per poi aggiungere o modificare quello che serve.

Controller Il Controller è composto dalle classi servizio. Si crea una classe servizio per ogni aggregato, in modo che le funzionalità che un servizio offre rispettino il principio di *Single Responsability*. I servizi sono utili per attività come il recupero di dati dal server, la convalida dell'input dell'utente o l'accesso diretto alla console. Definendo tali attività di elaborazione in una classe di servizio iniettabile, si rendono tali attività disponibili a qualsiasi componente.

UI/UX

L'interfaccia grafica è stata implementata cercando di rendere l'esperienza dell'utente più semplice possibile.

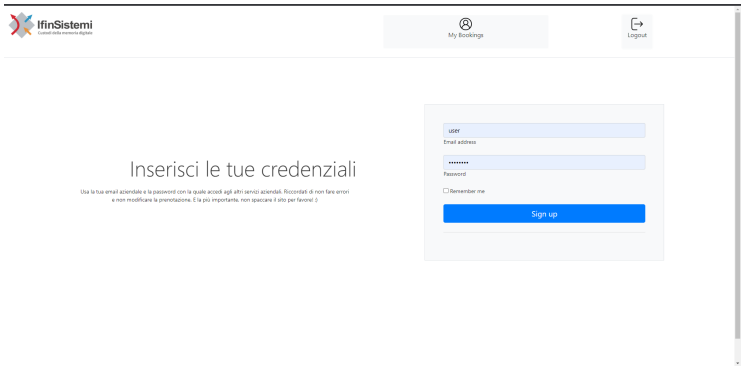


Figura 4.8: Visualizzazione della pagina di login

Una volta che l'utente accede al sito, visualizza un calendario. Nel calendario si trovano le prenotazioni effettuate, sia quelle del passato che quelle a venire.

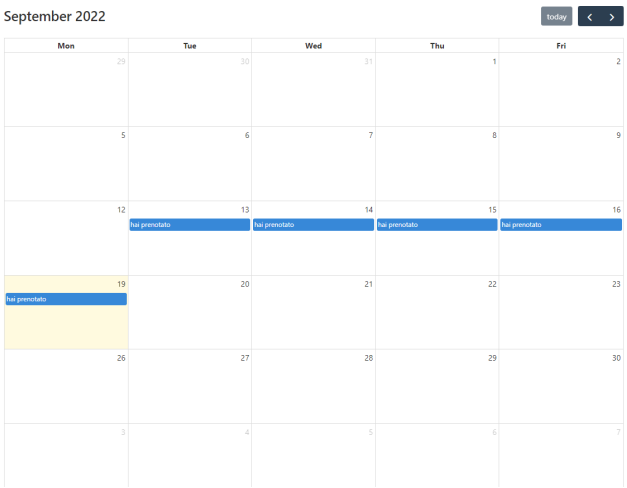


Figura 4.9: Visualizzazione della pagina del calendario

Per l'implementazione del calendario si è usato una libreria chiamata FullCalendar (https://fullcalendar.io/docs/angular).
L'utente può cliccare su una giornata e visualizza un modulo con i dettagli della prenotazione:

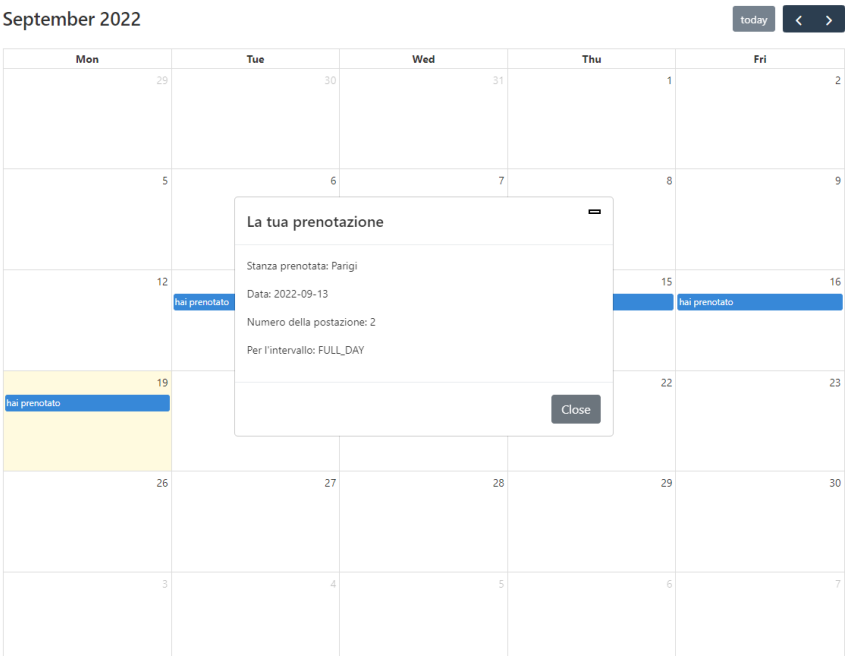


Figura 4.10: Visualizzazione del modulo della prenotazione

Se la prenotazione era nel passato l'utente non può eliminarla, invece se è una prenotazione da venire, l'utente ha la possibilità di eliminarla. Quando si è effettuato una prenotazione Full Day per una giornata definita, l'utente non può effettuare un'altra. Invece nel caso dove la prenotazione è solo per metà giornata, è possibile effettuare una prenotazione per l'altra metà. In più è stato aggiunto una pagina dove l'utente può trovare le proprie prenotazioni e visualizzarli nello stesso stile di prima.

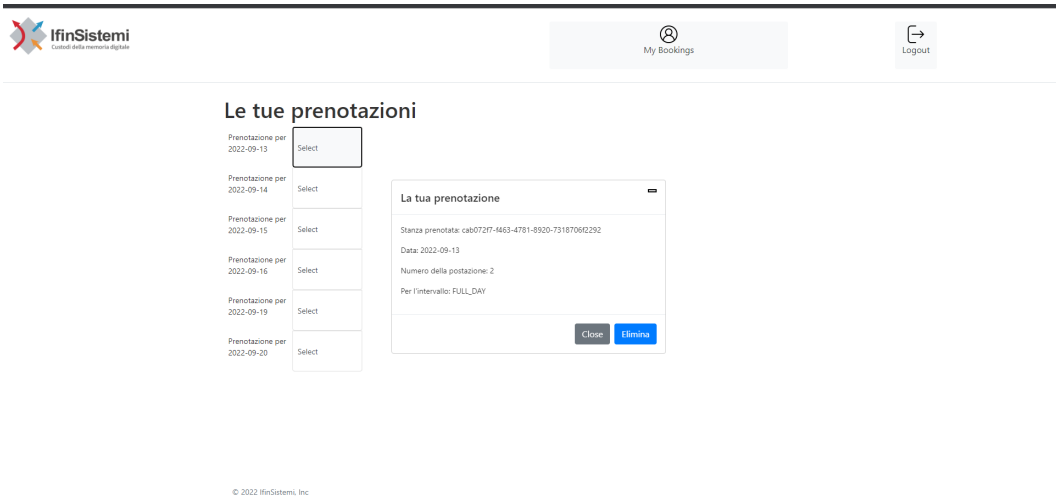


Figura 4.11: Visualizzazione della pagina delle prenotazioni

Capitolo 5

Verifica e validazione

5.1 Analisi statica

Essendo IntelliJ l'IDE usato per sviluppare il back-end, non sono stati utilizzati altri strumenti per l'analisi statica. In IntelliJ IDEA è disponibile una serie di ispezioni del codice che rilevano e correggono il codice anomalo nel progetto prima di compilarlo. L'IDE può trovare ed evidenziare vari problemi, individuare *dead code*, trovare probabili bug, problemi di ortografia e migliorare la struttura generale del codice.
Nella parte front-end, si è verificato principalmente che le classi del modello siano conforme ai **DTO**, per non avere errori nella comunicazione tra server(back-end) e client(front-end).

5.2 Analisi dinamica

Durante lo sviluppo del codice di produzione sono stati sviluppati molti test di unità. Questo anche perché abbiamo cercato di seguire le regole del **TDD**.
Per quanto riguarda i test di integrazione, abbiamo usato la libreria TestContainers come già menzionato nella sezione 4.1.5.
Con l'integrazione tra il front-end e back-end non abbaimo incontrato errori, la comunicazione avveniva con facilità.
Un utente esistente nel database poteva accedere al sistema e prenotare o visualizzare le proprie prenotazioni. Le funzionalità implementate, descritte nella tabella 6.1 e elencate nel capitolo 6, risultano implementate correttamente e senza errori.
L'unica cosa che si era potuto ampliare erano le eccezioni.Ne sono state personalizzati alcune, per avere una descrizione più precisa degli errori. Ulteriori personalizzazioni delle eccezioni sono possibili.

Capitolo 6

Conclusioni

6.1 Raggiungimento degli obiettivi

Durante il progetto di stage sono stati soddisfatti tutti i requisiti obbligatori.
Il tempo pianificato per la formazione iniziale e la realizzazione del sistema è stato meno di quanto auspicabile, molte delle tecnologie utilizzate erano nuove e la parte front-end non è stato pienamente completata.
Non abbiamo potuto implementare le funzionalità per l'utente amministratore e i test per la parte front-end. Siamo riusciti a soddisfare alcuni dei requisiti desiderabili, come la possibilità di scegliere una postazione specifica e non solo la stanza, testare il back-end con test di unità e test di integrazione.
Vengono elencati di seguito gli obiettivi soddisfati e quelli non soddisfatti, seguendo la tabella 6.1:

Codice	Descrizione	Stato
RO1	L'utente deve riuscire ad accedere con email e password	Soddisfatto
RO2	L'utente deve riuscire a selezionare una data	Soddisfatto
RO3	L'utente può visualizzare solo gli uffici liberi	Soddisfatto
RO4	L'utente deve riuscire a selezionare un ufficio libero	Soddisfatto
RO5	L'utente può visualizzare solo le postazioni liberi	Soddisfatto
RO6	L'utente deve riuscire a prenotare una postazione libera nell'ufficio scelto	Soddisfatto
RO7	L'utente deve riuscire a vedere le proprie prenotazioni	Soddisfatto
RO8	L'utente deve riuscire a eliminare una prenotazione in avvenire	Soddisfatto
RO9	L'utente non può eliminare una prenotazione del passato o del presente	Soddisfatto
RO10	L'utente deve riuscire a scollegarsi dal sistema	Soddisfatto
RD1	L'utente può scegliere la postazione e non solo la stanza	Soddisfatto
RD2	Test di unità per la parte back-end	Soddisfatto
RD3	Test di integrazione per back-end	Soddisfatto
RD4	Test per la parte front-end	Non soddisfatto
RD5	Documentazione	Parzialmente soddisfatto
RF1	Funzionalità aggiuntive per l'utente *admin*	Non soddisfatto

Tabella 6.1: Soddisfazione dei requisiti

6.2 Conoscenze acquisite e valutazione personale

Durante le otto settimane ho imparato a mettere in pratica le conoscenze acquisite durante il percorso universitario. Essere in grado di mettere i pezzi insieme era molto soddisfacente. Ho imparato come riuscire a non scoraggiarsi dalle numerose nuove tecnologie che dovevo imparare ed usare.
Inoltre ho imparato come si lavora in un'azienda e ho acquisito non solo *hard skills* ma anche *soft skills*, che mi hanno aiutato a comunicare in modo più efficace non solo le mie idee ma anche le mie difficoltà, in modo che gli altri potessero aiutarmi e consigliarmi.
Da questa esperienza ho potuto approfondire la mia conoscenza del linguaggio Java. Ho potuto imparare il linguaggio Typescript, i framework Spring Boot ed Angular. Imparare ad analizzare e progettare concentrandosi sul dominio applicativo. E alla fine approfondire e implementare le *best practice* architetturali e di sviluppo.
In conclusione, valuto molto positivamente tutto il percorso effettuato durante il mio stage universitario.

Acronimi e abbreviazioni

API Application Program Interface. 14, 36

CRUD Create Read Update Delete. 14, 36

DDD Domain Driven Design. 8, 36

DI Dependency Injection. 14, 16, 36

DTO Data Transfer Object. 23, 27, 36

IDE Integrated Development Environment. 9, 36

J2EE Jakarta EE. 15, 37

JPA Jakarta Persistence API. 14

JVM Java Virtual Machine. 13, 37

JWT JSON Web Tokens. 26

MVC Model View Controller. 27

POM Project Object Model. 23, 25

SPA Single Page Application. 16

TDD Test Driven Design. 8, 24, 37

UML Unified Modeling Language. 22, 37

VCS Version Control System. 10, 37

Glossario

API in informatica con il termine *Application Programming Interface API* (ing. interfaccia di programmazione di un'applicazione) si indica ogni insieme di procedure disponibili al programmatore, di solito raggruppate a formare un set di strumenti specifici per l'espletamento di un determinato compito all'interno di un certo programma. La finalità è ottenere un'astrazione, di solito tra l'hardware e il programmatore o tra software a basso e quello ad alto livello semplificando così il lavoro di programmazione. 35

CRUD sono le quattro operazioni basilari della gestione persistente dei dati.. 35

DDD La progettazione basata sul dominio è un importante approccio alla progettazione del software, incentrato sulla modellazione del software in modo che corrisponda a un dominio in base all'input degli esperti di quel dominio. Domain Driven Design promuove la modellazione basata sulla realtà aziendale come rilevante per i casi d'uso. Nel contesto della creazione di applicazioni, DDD parla di problemi come domini. Descrive aree problematiche indipendenti come contesti delimitati (ogni contesto delimitato è correlato a un microservizio) e sottolinea un linguaggio comune per parlare di questi problemi.. 35

DI è una tecnica di progettazione usata per ottenere l'inversione del controllo (IoC). Nella programmazione orientata a oggetti come Java gli oggetti che dipendono da altri oggetti vengono definiti dipendenze. L'oggetto ricevente o dipendente viene in genere definito client e l'oggetto da cui il client dipende viene definito servizio. L'inserimento delle dipendenze passa quindi il servizio al client o "inserisce" la dipendenza usando codice definito injector. Grazie all'inserimento delle dipendenze, il client non deve specificare quale servizio usare: l'injector gestisce tale aspetto per il client.. 35

DTO è un oggetto che trasporta dati tra processi. La motivazione del suo utilizzo è che la comunicazione tra i processi avviene solitamente ricorrendo a interfacce remote (es. servizi web), dove ogni chiamata è un'operazione costosa. Poiché la maggior parte del costo di ciascuna chiamata è legata al tempo di andata e ritorno tra il client e il server, un modo per ridurre il numero di chiamate è utilizzare un oggetto (il DTO) che aggreghi i dati che sono state trasferite dalle diverse chiamate, ma che è servita da una sola chiamata.. 35

IDE è un'applicazione software che aiuta i programmatori a sviluppare codice software in modo efficiente. Aumenta la produttività degli sviluppatori combinando

funzionalità come la modifica, la creazione, il test e il packaging del software in un'applicazione di facile utilizzo.. 35

J2EE è una piattaforma Java che fornisce uno standard per lo sviluppo di applicazioni enterprise a più livelli.. 35

JVM è un ambiente del tempo di esecuzione che è possibile aggiungere in un browser Web o in qualsiasi sistema operativo, come IBM® i. La JVM (Java virtual machine) esegue istruzioni generate da un compilatore Java. Essa consiste in un interpreter bytecode e tempo di esecuzione che consente di eseguire i file di classe Java su qualsiasi piattaforma, indipendentemente dalla piattaforma su cui sono stati sviluppati in origine. Il programma di caricamento classi e il responsabile della riservatezza, che fanno parte del tempo di esecuzione Java, isolano il codice che proviene da un'altra piattaforma. Essi possono anche limitare le risorse di sistema cui può accedere ogni classe caricata.. 35

TDD è una tipologia di sviluppo software che mette al primo posto la fase di testing. Secondo questa metodologia, il programmatore scrive per prima cosa dei test automatici sulla base della funzionalità richiesta, test che ovviamente falliranno perché la funzionalità ancora manca. Viene poi sviluppato il codice minimo che consenta il passaggio del test, codice che passa poi attraverso azioni di refactoring. In questo modo lo sviluppo dell'applicazione procede per cicli, guidati dai test, che favoriscono lo sviluppo di codice minimale, di elevata qualità. . 35

UML in ingegneria del software *UML, Unified Modeling Language* (ing. linguaggio di modellazione unificato) è un linguaggio di modellazione e specifica basato sul paradigma object-oriented. L'*UML* svolge un'importantissima funzione di "lingua franca" nella comunità della progettazione e programmazione a oggetti. Gran parte della letteratura di settore usa tale linguaggio per descrivere soluzioni analitiche e progettuali in modo sintetico e comprensibile a un vasto pubblico. 35

VCS è la gestione di versioni multiple di un insieme di informazioni: gli strumenti software per il controllo versione sono ritenuti molto spesso necessari per la maggior parte dei progetti di sviluppo software o documentali gestiti da un team collaborativo di sviluppo o redazione.. 35

Bibliografia

Riferimenti bibliografici

Deblauwe, Wim. *Practical Guide To Building An API Back-end With Spring Boot*. InfoQ.com, 2018.

Siti web consultati

Documentazione Angular. URL: https://angular.io/docs.

Documentazione Flyway. URL: https://flywaydb.org/documentation/getstarted/.

Documentazione Java. URL: https://docs.oracle.com/en/java/.

Documentazione Spring. URL: https://docs.spring.io.

Documentazione Spring Data Jpa. URL: https://docs.spring.io/spring-data/jpa/docs/current/reference/html/.

Documentazione Spring Security. URL: https://spring.io/projects/spring-security.

Documentazione Testcontainers. URL: https://www.testcontainers.org/.

FullCalendar for Angular. URL: https://fullcalendar.io/docs/angular.

Online Courses. URL: https://www.udemy.com/.

PostgreSQL. URL: https://www.postgresql.org/docs/.

The Onion Architecture. URL: https://jeffreypalermo.com/2008/07/the-onion-architecture-part-1/.

Using JWT with Spring Security OAuth. URL: https://www.baeldung.com/spring-security-oauth-jwt.

Validation in Spring Boot. URL: https://www.baeldung.com/spring-boot-bean-validation.