



Università degli Studi di Milano Bicocca

Scuola di Scienze

Dipartimento di Informatica, Sistemistica e Comunicazione

Corso di laurea in Informatica

Web Development

Relatore: *Prof. Daniela Micucci*

Relazione della prova finale di:

Matteo Covelli

Matricola 861277

Anno Accademico 2022-2023

*Alla mia famiglia, per avermi supportato psicologicamente ed economicamente
in questi anni.*

*Ai miei compagni di corso Alaa e Niccolò, per aver reso questo percorso più
sereno e me più felice.*

Ai miei amici, per avermi supportato e sopportato nei momenti di difficoltà.

*A chiunque mi abbia aiutato e criticato, per avermi insegnato che di fronte alle
avversità non ci si deve mai arrendere.*

Introduzione

Il mondo del web e del suo sviluppo ha avuto una grandissima evoluzione negli ultimi vent'anni. Lo scopo di questa tesi è di trattare lo sviluppo web partendo dalla sua nascita, fino a raggiungere le tecnologie più recenti utilizzate in questo ambito al fine di creare una web app.

La relazione è organizzata come segue:

Nel capitolo 1 ci concentreremo sui fondamenti teorici, trattando la storia dello sviluppo web, per poi passare ai concetti base dell'architettura client server.

Nel capitolo 2 specificheremo i requisiti richiesti per lo sviluppo della web app che svilupperemo.

Nel capitolo 3 introdurremo diversi concetti generici legati alle web app.

Nel capitolo 4 andremo a descrivere le tecnologie e gli strumenti che vengono utilizzati per la creazione di una web app, trattando sia lo sviluppo front-end che lo sviluppo back-end.

Nel capitolo 5 applicheremo le tecnologie mostrate nel capitolo precedente andando a creare una web app.

Indice

1	Fondamenti Teorici	1
1.1	La nascita del web	1
1.2	L'evoluzione del web e la sua diffusione	4
1.2.1	Una tecnologia alla portata di tutti	5
1.3	Architettura client-server	6
1.3.1	Protocollo HTTP	6
2	Requisiti	7
3	Concetti generici di una web app	9
3.1	Che cos'è una web app	9
3.2	Architettura di una web app	9
4	Architettura della soluzione - Tecnologie	10
4.1	Pattern MVC	10
4.2	Back-end: Tecnologie utilizzate	11
4.2.1	Spring	11
4.2.2	Maven	13
4.2.3	Docker	15
4.2.4	JPA - Hibernate	17
4.2.5	Funzione crittografica di hash	18
4.3	Front-end: Tecnologie utilizzate	20
4.3.1	HTML	20
4.3.2	CSS	21
4.3.3	Bootstrap	22
4.3.4	JavaScript	22
4.3.5	Thymeleaf	23
5	Implementazione delle tecnologie	24
5.1	Implementazione back-end	24
5.1.1	Database	24
5.1.2	Classe Postit	26
5.1.3	Interfaccia e classe PostitService	27
5.1.4	Interfaccia PostitRepository	28
5.1.5	Login controller	28
5.1.6	Postit controller	29
5.1.7	Configurazione login	31

<i>Indice</i>	<i>Indice</i>
5.2 Implementazione front-end	32
5.2.1 Pagina di login	32
5.2.2 Homepage	33
5.2.3 Access denied page	35
5.3 Demo Postit App	36
Postfazione	40
Bibliografia	41

Capitolo 1

Fondamenti Teorici

1.1 La nascita del web

Tim Berners-Lee, uno scienziato britannico, mentre lavorava al CERN nel 1989 sollevò un enorme problema che riguardava la condivisione di informazioni. Nonostante al CERN ci fosse una buona struttura gestionale per il raggiungimento degli obiettivi, Tim si rese conto che a causa dell'elevato turnover del personale, l'introduzione dei nuovi dipendenti richiedeva una grande quantità di tempo e molti dettagli tecnici dei progetti venivano persi o recuperati solo grazie ad un'indagine investigativa di emergenza. Per risolvere questo problema, Tim propose nel marzo 1989 (Figura 1.1) lo sviluppo del World Wide Web, la cui idea alla base era quella di creare un potente sistema informativo globale facile da utilizzare, unendo le tecnologie in costante evoluzione del web, con i data networks e gli hypertext.

La proposta fu definita *"vaga ma eccitante"* dal suo capo, che nonostante lo scetticismo gli diede la possibilità di lavorarci nel settembre 1990.

Tim iniziò delineando le tre tecnologie che sono le fondamenta del web ancora oggi:

- HTML (HyperText Markup Language)
- URI (Uniform Resource Identifier): un indirizzo che identifica ogni risorsa nel web, solitamente chiamato URL.
- HTTP (HyperText Transfer Protocol): il protocollo che permette di recuperare le risorse all'interno del web.

Due mesi dopo, insieme all'ingegnere Robert Cailliau, la proposta venne formalizzata delineando i concetti principali e i termini importanti sul Web (Figura 1.2). Alla fine del 1990 Tim Berners-Lee aveva realizzato il primo browser e Web server funzionanti. La prima pagina web conteneva le informazioni del progetto WWW, con tutti i dettagli tecnici per la creazione di un web server e come collegarlo ad altri web server. La creazione di questo progetto risolse il problema del recupero delle informazioni, garantendone un facile accesso.^{[1][2]}

Information Management: A Proposal

Tim Berners-Lee, CERN

March 1989, May 1990

This proposal concerns the management of general information about accelerators and experiments at CERN. It discusses the problems of loss of information about complex evolving systems and derives a solution based on a distributed hypertext system.

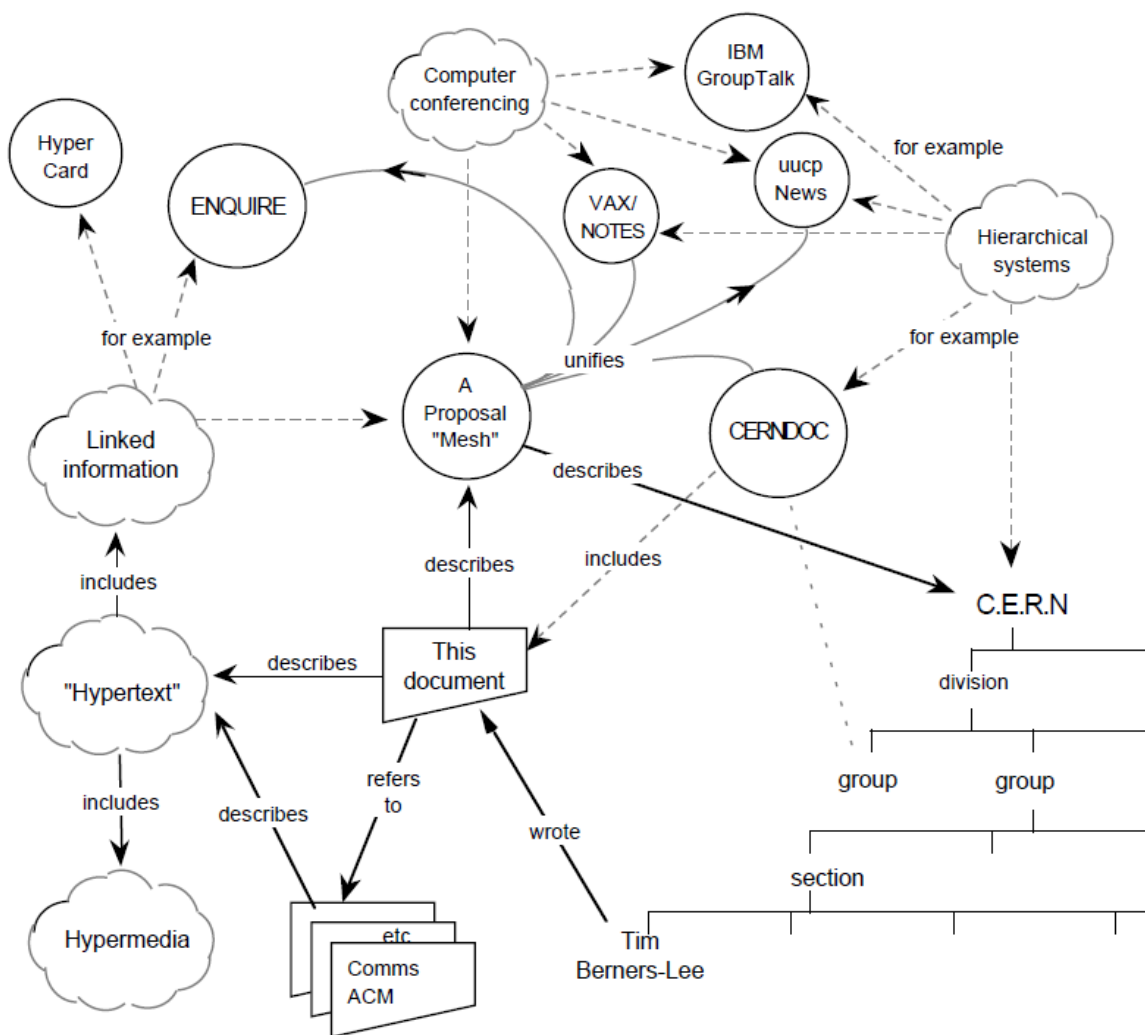


Figura 1.1: Prima pagina della proposta di Tim-Berners-Lee per il World Wide Web

[3]

WorldWideWeb

Proposal for a HyperText Project

To: P.G. Innocenti/ECP, G. Kellner/ECP, D.O. Williams/CN
Cc: R. Brun/CN, K. Gieselmann/ECP, R. Jones/ECP, T. Osborne/CN, P. Palazzi/ECP,
N. Pellow/CN, B. Pollermann/CN, E.M. Rimmer/ECP
From: T. Berners-Lee/CN, R. Cailliau/ECP
Date: 12 November 1990

The attached document describes in more detail a Hypertext project.

HyperText is a way to **link** and **access** information of various kinds as a web of nodes in which the user can **browse** at will. It provides a **single user-interface** to large classes of information (reports, notes, data-bases, computer documentation and on-line help). We propose a simple scheme incorporating **servers** already available at CERN.

The project has **two phases**: firstly we make use of existing software and hardware as well as implementing simple browsers for the user's workstations, based on an analysis of the requirements for information access needs by experiments. Secondly, we extend the application area by also allowing the users to add new material.

Phase one should take 3 months with the full manpower complement, phase two a further 3 months, but this phase is more open-ended, and a review of needs and wishes will be incorporated into it.

The **manpower** required is 4 software engineers and a programmer, (one of which could be a Fellow). Each person works on a specific part (eg. specific platform support).

Each person will require a state-of-the-art **workstation**, but there must be one of each of the supported types. These will cost from 10 to 20k each, totalling 50k. In addition, we would like to use **commercially available software** as much as possible, and foresee an expense of 30k during development for one-user licences, visits to existing installations and consultancy.

We will assume that the project can rely on some **computing support** at no cost: development file space on existing development systems, installation and system manager support for daemon software.



T. Berners-Lee



R. Cailliau

Figura 1.2: Prima pagina del documento di formalizzazione per il World Wide Web

[4]

1.2 L'evoluzione del web e la sua diffusione

Inizialmente solo pochi utenti avevano accesso alla piattaforma informatica su cui girava il primo browser, nel quale era possibile fare ricerche solo per parole chiave dato che i motori di ricerca non esistevano ancora. Perciò si decise di sviluppare un browser più semplice che funzionava solo in modalità di linea, affinché funzionasse su ogni sistema. Alla fine del 1991 negli Stati Uniti venne messo in rete il primo server web in un laboratorio di fisica.

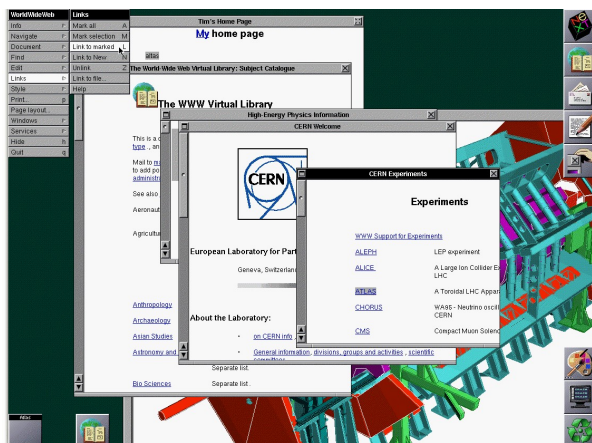
A questo punto esistevano due tipi di browser:

- La versione originale (Figura 1.3a), la quale funzionava solo sulle macchine NeXT.¹
- La versione in modalità di linea (Figura 1.3b), la quale era più facile da installare ed eseguire su qualsiasi piattaforma, ma limitata in termini di potenza e facilità d'uso.

Date le grandi dimensioni del progetto, il team del CERN non poteva fare tutto il lavoro da solo. Perciò Berners-Lee lanciò un appello via internet al fine di trovare nuovi sviluppatori che si unissero allo sviluppo di questa rivoluzionaria tecnologia.

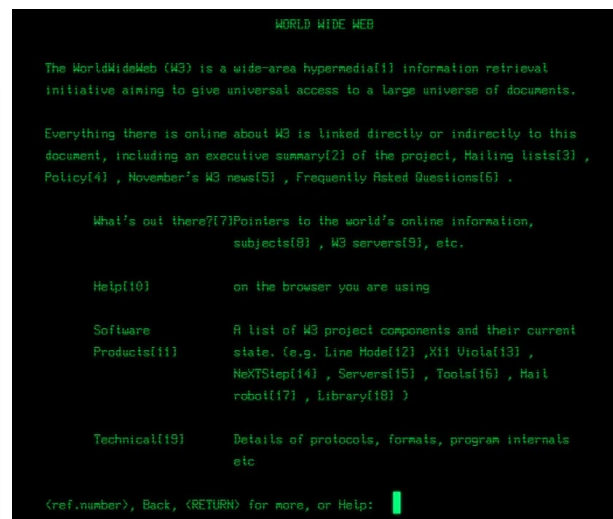
All'inizio del 1993 iniziarono a nascere diversi browser, che funzionavano sia in ambienti PC e Macintosh. La nascita di browser affidabili e facili da usare, comportarono una diffusione immediata del World Wide Web.

Il 1994 fu considerato *"l'anno del web"* e al suo termine il Web contava 10.000 server, di cui 2.000 commerciali, e 10 milioni di utenti.^[1]



(a) Browser originale con interfaccia grafica

[5]



(b) Browser a linea di comando

[6]

Figura 1.3: I primi due tipi di browser sviluppati nel 1991

¹Uno dei primi computer di Steve Jobs, sul quale veniva eseguito il primo browser web.

1.2.1 Una tecnologia alla portata di tutti

Gli sviluppatori al CERN decisero fin da subito che il web dovesse essere utilizzabile da tutti e che nessuno dovesse rinchiuderlo in un sistema proprietario. Per questo motivo il CERN decise di presentare una proposta, che fu subito accettata, alla commissione dell'Unione Europea al fine di formare un consorzio internazionale in collaborazione con il Massachusetts Institute of Technology (MIT).

Nel 1994 Tim Berners-Lee lasciò il CERN per unirsi al MIT e fondò l'International World Wide Consortium (W3S), il quale al giorno d'oggi definisce i vari standard nello sviluppo web e di applicazioni.

Ad oggi gli standard web del W3S garantiscono un'ottimizzazione per l'interoperabilità², la sicurezza e privacy dei dati, accessibilità e l'internazionalizzazione³.

"Web standards are blueprints or building blocks of a consistent and harmonious digitally connected world. They are implemented in browsers, blogs, search engines, and other software that power our experience on the web."^[9]

- W3S

²L'interoperabilità è, in ambito informatico, la capacità di un sistema o di un prodotto informatico di cooperare e di scambiare informazioni o servizi con altri sistemi o prodotti in maniera più o meno completa e priva di errori, con affidabilità e con ottimizzazione delle risorse.^[7]

³L'internazionalizzazione in economia, e per estensione in informatica e altri ambiti, è il processo di adattamento di una impresa, un prodotto, un marchio, pensato e progettato per un mercato o un ambiente definito, ad altri mercati o ambienti internazionali, in modo particolare altre nazioni e culture.^[8]

1.3 Architettura client-server

Un sistema client-server utilizza un'architettura nella quale un client, che è responsabile dell'interazione con l'utente e solitamente è un'interfaccia grafica di limitata complessità, si connette ad un server, il quale implementa la logica del sistema e tutte le tecniche per la gestione degli accessi, allocazione, rilascio e sicurezza delle risorse, per la fruizione di un determinato servizio, come ad esempio la condivisione di una certa risorsa.^{[10][11]}

I vantaggi di un'architettura di questo tipo sono:^[12]

- Accessibilità dei dati: dato che il server mantiene i dati in una posizione centralizzata, più utenti possono accedere e lavorare simultaneamente sui dati garantendone una maggiore condivisione.
- Scalabilità: è possibile aggiornare il server ad una macchina più potente, senza cambiamenti visibili all'utente. Ciò garantisce anche la possibilità di introdurre nuove tecnologie, sia hardware che software.
- Integrità dei dati: il server può usufruire e fornire di servizi che garantiscono la protezione dei dati, archiviazione crittografata di file.

1.3.1 Protocollo HTTP

Il protocollo HTTP, il quale utilizza uno schema client-server, è una delle tecnologie alla base del web. Quando usiamo un browser per caricare una pagina web, esso agisce come client HTTP comunicando con un server HTTP.^[13]

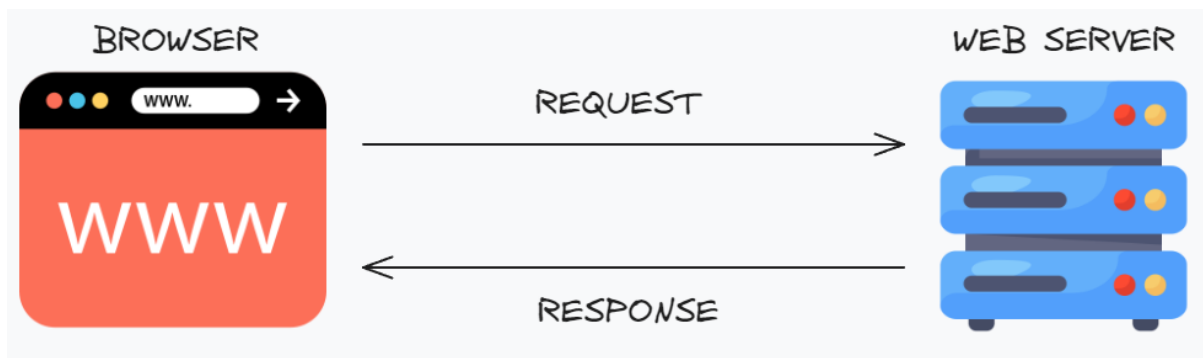


Figura 1.4: Comunicazione tra web-browser e web-server

Ad esempio, come vedremo più nello specifico nel capitolo 5, nel momento in cui un utente vuole caricare la pagina per visualizzare i postit (Figura 1.4):

1. Il browser aprirà una connessione verso il server, inviando una richiesta (request).
2. Ricevuta la richiesta, il server si occuperà di gestire la logica di recupero dei dati richiesti, inviandoli in risposta (response) a chi ha fatto la richiesta.
3. Il browser gestirà la risposta reindirizzando e caricando il contenuto nella pagina del sito.

Capitolo 2

Requisiti

Al fine di applicare le tecnologie che tratteremo nel capitolo 4, mi è stato richiesto di sviluppare una web app. La web app deve implementare i seguenti requisiti:

- **Pagina di login:** pagina in cui l'utente può autenticarsi utilizzando il proprio username e password. La password deve essere crittografata nel database utilizzando la funzione crittografica di hash Bcrypt. Inoltre ad ogni utente deve essere associato un ruolo, che viene usato per gestire i permessi di accesso alle pagine della web app. Se l'utente non ha i permessi per accedere a un determinato endpoint, quest'ultimo deve essere mandato alla pagina di accesso negato. Mentre se possiede il ruolo "EMPLOYEE" viene mandato alla home page (postit-home). (Figura 2.1)
- **Homepage:** pagina in cui è presente una navbar, nella quale viene mostrato il titolo della web app, l'username e il ruolo dell'utente autenticato. All'interno della navbar devono essere presenti due bottoni, uno per la creazione dei postit, l'altro per effettuare il logout. Inoltre sotto la navbar vengono visualizzate delle note, chiamate postit, le quali sono costituite da un titolo e descrizione. Ogni postit deve avere la possibilità di essere modificato ed eliminato.

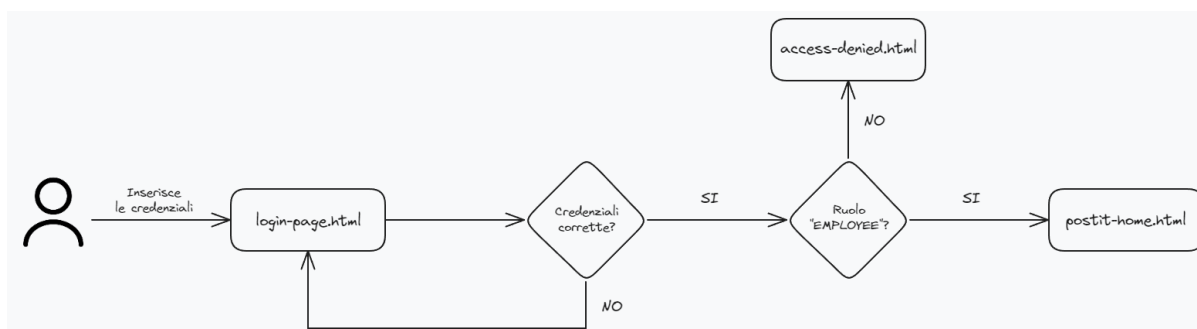


Figura 2.1: Pagine da implementare

Back-end

Il back-end deve essere implementato basandosi sul pattern MVC, utilizzando il framework Spring Boot, andando a implementare:

- Entità postit;
- Controller per login e home page;
- Configurazione per la gestione della sicurezza al fine di gestire l'autenticazione degli utenti e i permessi di accesso alle pagine di questi ultimi.

Front-end

Il front-end deve implementare le seguenti pagine:

- login-page.html, la quale deve avere uno stile css;
- postit-home.html, la quale deve avere uno stile css usando il framework bootstrap;
- access-denied.html

Database

I dati devono essere mantenuti all'interno di un database. Il database, PostgreSQL, deve essere inserito in un container utilizzando l'applicativo Docker, e deve implementare le seguenti tabelle:

- users
- ruoli
- postit

Al fine di recuperare i dati dal back-end e mostrarli nel front-end deve essere utilizzato il motore di template Thymeleaf.

Capitolo 3

Concetti generici di una web app

3.1 Che cos'è una web app

Una web app è un software applicativo eseguito su un server web, accessibile e utilizzabile da un utente tramite un browser.^[14] Un'applicazione web è costituita da:

- Back-end: si occupa, lato server, di tutta la logica dell'applicazione, della gestione dei dati e della comunicazione con il database.
- Front-end: si occupa, lato client, della presentazione delle informazioni all'utente garantendo l'interazione tra utente e applicazione web.

3.2 Architettura di una web app

Una web app possiede la seguente architettura (Figura 3.1):

1. L'utente, tramite un'interfaccia dell'applicazione, invia una richiesta al server;
2. Il client inoltra la richiesta dell'utente al server;
3. Il server esegue l'operazione richiesta, generando i risultati dei dati richiesti;
4. Il server invia i risultati al client, il quale si occuperà di visualizzarli all'utente.

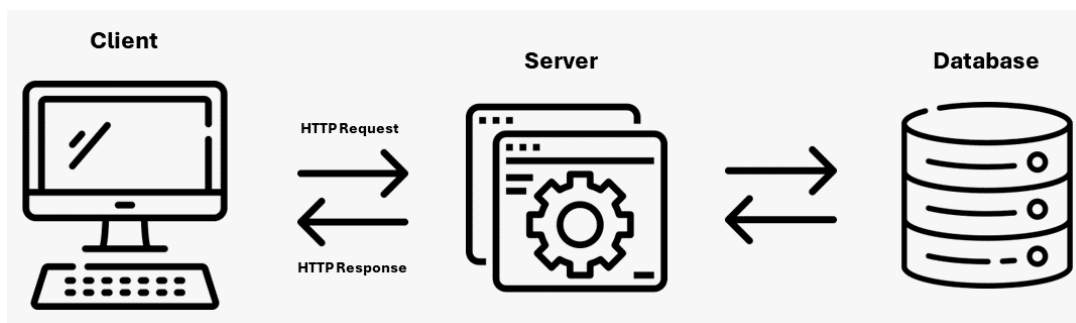


Figura 3.1: Funzionamento di un'applicazione web

Capitolo 4

Architettura della soluzione - Tecnologie

4.1 Pattern MVC

Per lo sviluppo della web app ho utilizzato il pattern MVC. Il Model-View-Controller è un pattern utilizzato per dividere il codice in blocchi che svolgono funzionalità distinte.

Esso è costituito da tre componenti principali^[15] (Figura 4.1):

- **Model:** si occupa di accedere ai dati necessari alla logica implementata nell'applicazione. In particolare nella nostra web app all'interno del model avremo la classe Postit.
- **View:** si occupano di creare l'interfaccia utilizzabile dall'utente, mostrando i dati richiesti da quest'ultimo. Nel nostro caso la view si occuperà di visualizzare, la pagina di login e tutti i postit nella homepage.
- **Controller:** si occupano di implementare la vera logica dell'applicazione utilizzando i due componenti precedenti, ricevendo gli input dell'utente, gestendo il model per la ricerca dei dati e la creazione delle view da restituire all'utente. Nel nostro caso usiamo due controller: uno per la gestione del login e uno per la gestione dei postit.

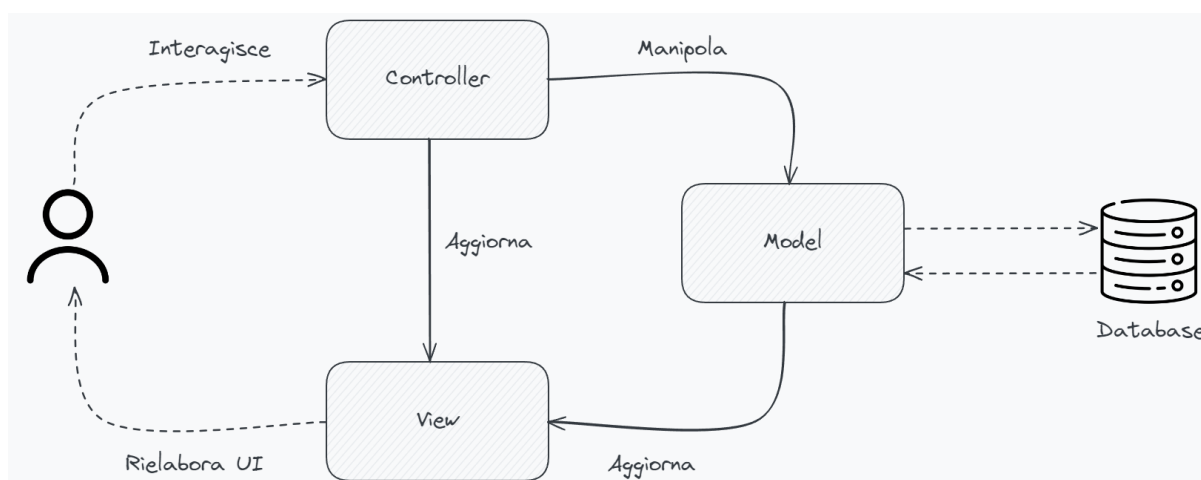


Figura 4.1: Funzionamento pattern MVC

4.2 Back-end: Tecnologie utilizzate

Per l'implementazione del back-end ho utilizzato diverse tecnologie:

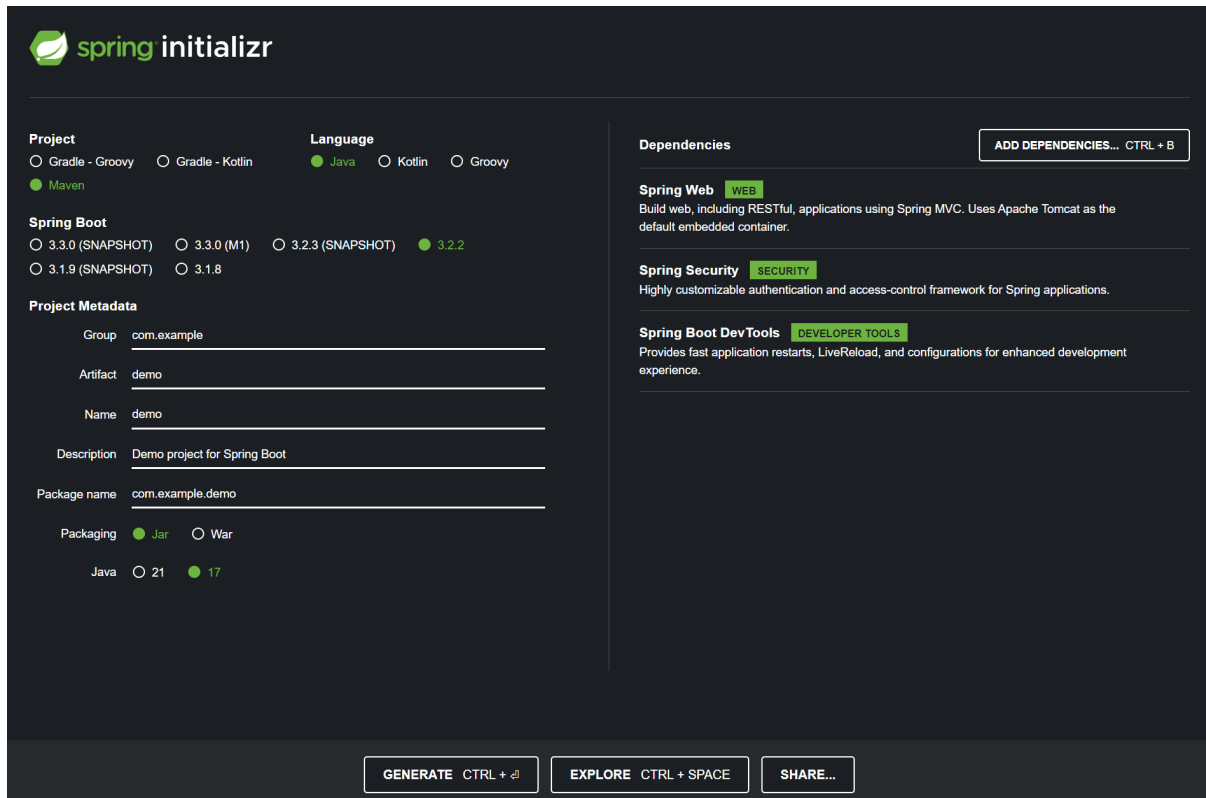
- Java Spring Boot: per la gestione della logica lato server;
- Maven: build automation tool;
- Docker: per dockerizzare in un container il DB di PostgreSQL;
- Bcrypt: algoritmo di hashing di password;

4.2.1 Spring

Spring è un framework di Java che viene utilizzato per la creazione di applicazioni autonome, adatte ad ambienti di produzione che vengono eseguite su Java Virtual Machine.

Spring Boot è uno strumento che semplifica e velocizza lo sviluppo delle applicazioni web che usano Spring. Spring fornisce una funzione di dependency-injection, che consente agli oggetti di definire le proprie dipendenze che successivamente il container Spring provvederà ad inserire. Ciò garantisce la possibilità di creare applicazioni modulari. Dato che Spring richiede un notevole dispendio di tempo e competenze per configurare, impostare e implementare applicazioni Spring, Spring Boot riduce questo sforzo usando tre funzionalità^[16]:

- Configurazione automatica: le applicazioni vengono inizializzate con dipendenze preimpostate in modo tale da non doverle configurare manualmente. Grazie alla configurazione automatica, Spring Boot è in grado di configurare sia le impostazioni di Spring sia i pacchetti di terze parti. Ciò garantisce la possibilità di iniziare velocemente a sviluppare le applicazioni, riducendo errori umani.
- Approccio categorico per le dipendenze: Spring è in grado di scegliere i pacchetti da installare e i valori predefiniti da utilizzare evitando allo sviluppatore la perdita di tempo nel prendere tutte queste decisioni. In particolar modo durante la creazione del progetto è possibile selezionare già le dipendenze necessarie, grazie allo Spring Boot Initializr compilando semplicemente un modulo web (Figura 4.2). Ad esempio per la creazione di un'applicazione web è possibile selezionare "Spring Web", per garantire la sicurezza è possibile selezionare "Spring Security", ecc...
- Applicazioni autonome: Spring Boot consente di creare applicazioni autonome che vengono eseguite automaticamente integrando un server web come ad esempio Tomcat nell'applicazione durante il processo di inizializzazione senza dover far affidamento su un server web esterno.



The image shows the Spring Initializr web interface for configuring a new project. The interface is dark-themed and organized into several sections:

- Project:** Includes radio buttons for **Gradle - Groovy**, **Gradle - Kotlin**, **Java** (selected), **Kotlin**, and **Groovy**. Below this is a radio button for **Maven** (selected).
- Spring Boot:** Includes radio buttons for versions: **3.3.0 (SNAPSHOT)**, **3.3.0 (M1)**, **3.2.3 (SNAPSHOT)**, **3.2.2** (selected), and **3.1.9 (SNAPSHOT)**. Below this is a radio button for **3.1.8**.
- Project Metadata:** A form with the following fields:
 - Group:** `com.example`
 - Artifact:** `demo`
 - Name:** `demo`
 - Description:** `Demo project for Spring Boot`
 - Package name:** `com.example.demo`
 - Packaging:** Radio buttons for **Jar** (selected) and **War**.
 - Java:** Radio buttons for **21** and **17** (selected).
- Dependencies:** A section on the right with a button **ADD DEPENDENCIES... CTRL + B**. It lists three dependencies:
 - Spring Web** (tag: **WEB**): Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
 - Spring Security** (tag: **SECURITY**): Highly customizable authentication and access-control framework for Spring applications.
 - Spring Boot DevTools** (tag: **DEVELOPER TOOLS**): Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

At the bottom of the interface are three buttons: **GENERATE CTRL + G**, **EXPLORE CTRL + SPACE**, and **SHARE...**

Figura 4.2: Modulo web Spring Initializr

4.2.2 Maven

Apache Maven è uno strumento di automazione build utilizzato per i progetti Java. Rende più facile gestire e mantenere grandi progetti fornendo una struttura coerente e una serie di convenzioni su come organizzare il progetto (Figura 4.3), aiutando gli sviluppatori ad automatizzare il processo di build, test e distribuzione del software.

Una delle caratteristiche di Maven è la capacità di gestire le dipendenze, tenendo traccia di tutte le librerie e altre dipendenze di cui un progetto ha bisogno, scaricandole automaticamente. Ciò aiuta gli sviluppatori senza il bisogno che debbano scaricare e gestire manualmente le dipendenze.^[17]

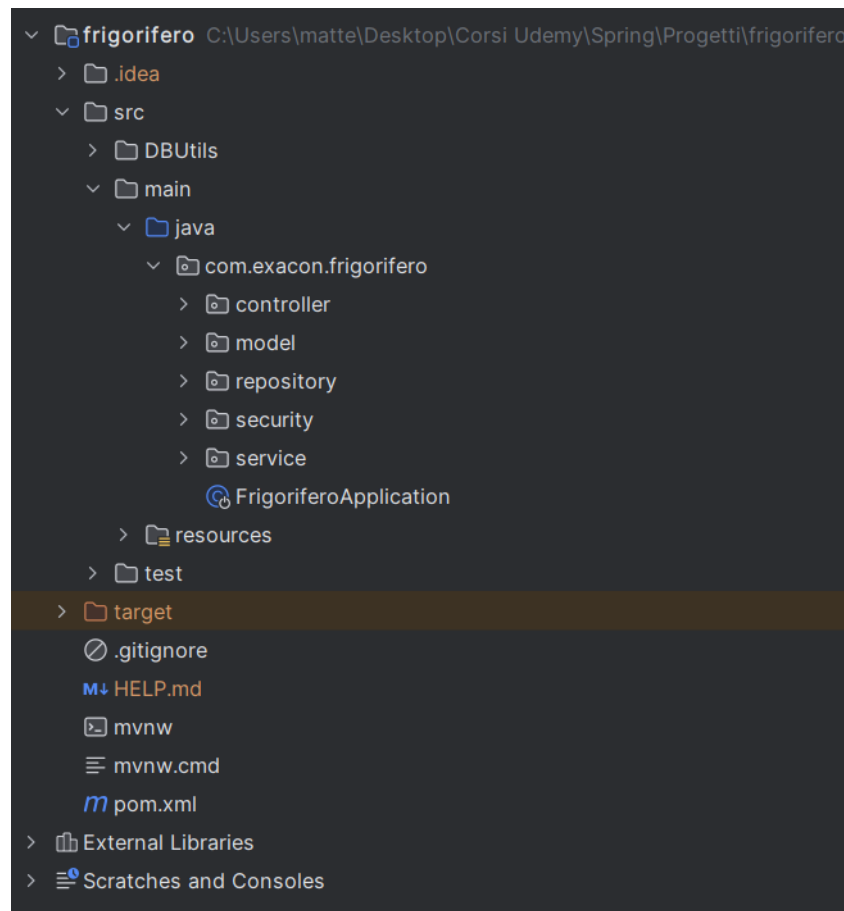


Figura 4.3: Struttura del progetto postit-app utilizzando Maven

Project Object Model

Per la gestione delle dipendenze del progetto, i plugin e la configurazione di build, Maven utilizza un file XML chiamato pom.xml (project object model).

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
4   <modelVersion>4.0.0</modelVersion>
5
6   <parent>
7     <groupId>org.springframework.boot</groupId>
8     <artifactId>spring-boot-starter-parent</artifactId>
9     <version>3.2.2</version>
10    <relativePath/> <!-- lookup parent from repository -->
11  </parent>
12  <groupId>com.example</groupId>
13  <artifactId>demo</artifactId>
14  <version>0.0.1-SNAPSHOT</version>
15  <name>demo</name>
16  <description>Demo project for Spring Boot</description>
17  <properties>
18    <java.version>17</java.version>
19  </properties>
20
21  <dependencies>
22
23    <dependency>
24      <groupId>org.springframework.boot</groupId>
25      <artifactId>spring-boot-starter-web</artifactId>
26    </dependency>
27
28    <dependency>
29      <groupId>org.postgresql</groupId>
30      <artifactId>postgresql</artifactId>
31      <scope>runtime</scope>
32    </dependency>
33
34    <dependency>
35      <groupId>org.springframework.boot</groupId>
36      <artifactId>spring-boot-starter-test</artifactId>
37      <scope>test</scope>
38    </dependency>
39
40  </dependencies>
41
42  <build>
43    <plugins>
44      <plugin>
45        <groupId>org.springframework.boot</groupId>
46        <artifactId>spring-boot-maven-plugin</artifactId>
47      </plugin>
48    </plugins>
49  </build>
50
51 </project>

```

Listing 4.1: Esempio di pom.xml, nel quale sono state aggiunte le dipendenze web e database.

4.2.3 Docker

Docker è una piattaforma open source che consente agli sviluppatori di creare, implementare, eseguire, aggiornare e gestire container, i quali sono delle unità eseguibili di software in cui viene impacchettato il codice applicativo, insieme alle sue librerie e dipendenze in modo da poter essere eseguito ovunque. I container sono piccoli, veloci e portabili perché, diversamente da una Virtual Machine (VM), non hanno bisogno di includere un sistema operativo guest in ogni istanza, ma possono invece sfruttare le funzioni e le risorse del sistema operativo host.^[18]

Differenza tra container e virtual machine

Ogni VM contiene un sistema operativo guest, una copia virtuale dell'hardware di cui il sistema operativo ha bisogno per essere eseguito, insieme a un'applicazione e alle relative librerie e dipendenze associate. I container, a differenza delle VM, virtualizzano il sistema operativo, in modo che ogni singolo container includa solo l'applicazione e le relative librerie e dipendenze.^[19] L'utilizzo dei container porta diversi vantaggi:

- Leggero: i container condividono il kernel del sistema operativo della macchina, eliminando la necessità di un'istanza del sistema operativo completa per ogni applicazione e rendendo i file container piccoli e poco gravosi sulle risorse. Ciò garantisce un'esecuzione più rapida.
- Portabile: i container portano con loro tutte le loro dipendenze, in modo tale che il software debba essere scritto una sola volta e eseguito in diversi ambienti senza doverlo riconfigurare ogni volta.

Creazione database in un container

Per gestione del database della web app ho inserito in un container il database di PostgreSQL. Per inserire il db in un container ho eseguito all'interno del terminale il seguente comando:

```
1 docker run --name postit-app-db ^
2   -e POSTGRES_USER=utente ^
3   -e POSTGRES_DB=postit-app-db ^
4   -e POSTGRES_PASSWORD=password ^
5   -e PGDATA=/var/lib/postgresql/data/postitData ^
6   -p 5434:5432 ^
7   -v C:\Docker_Data:/var/lib/postgresql/data ^
8   postgres:14.1-alpine
```

Docker Desktop

Tramite l'utilizzo dell'applicazione Docker Desktop, è possibile gestire i container ed eseguirli. In Figura 4.4 è possibile vedere che il container "postit-app-db" è in esecuzione.

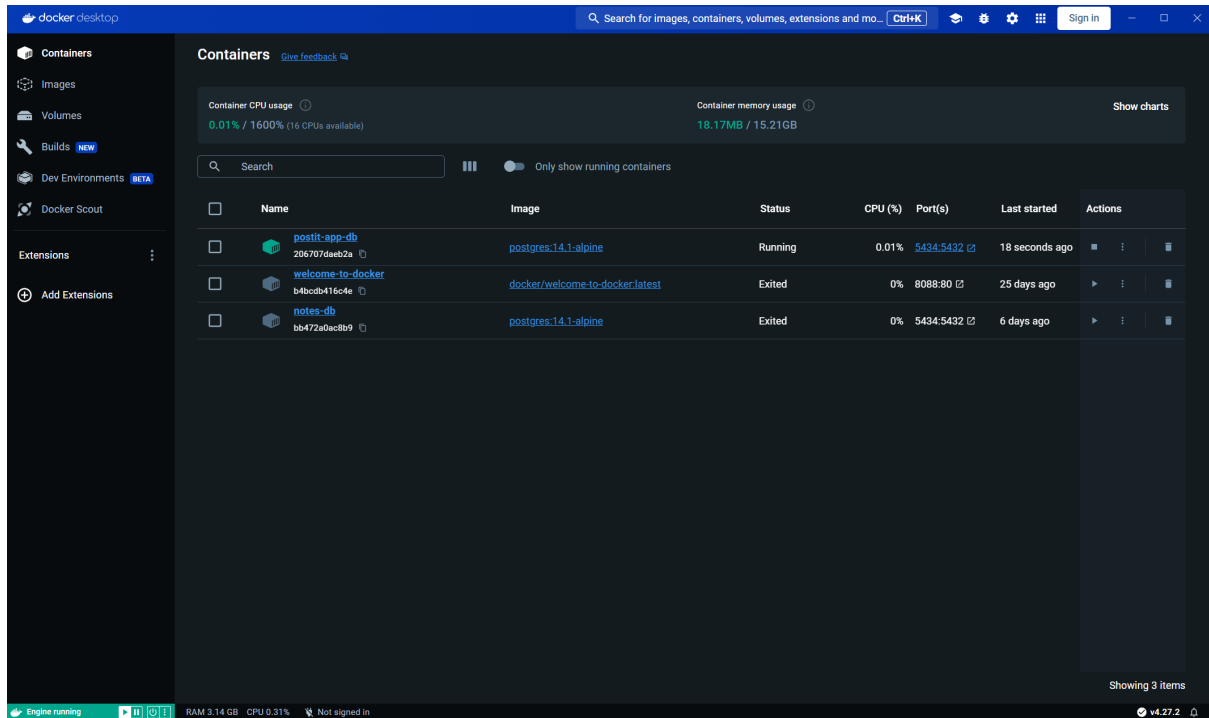


Figura 4.4: Interfaccia di Docker Desktop

4.2.4 JPA - Hibernate

JPA

Jakarta Persistent API (JPA) è un'interfaccia di programmazione che consente agli sviluppatori di lavorare con i database utilizzando Java. Viene utilizzato per recuperare e aggiornare i dati in un database. JPA si basa sulla tecnica di programmazione ORM (Object-Relational Mapping) e può essere implementato da vari framework ORM come Hibernate (Figura 4.6) ed essere anche usato in combinazione con tecnologie Java come Spring.^[20]

Hibernate

Hibernate ORM è un framework Java utilizzato per mappare modelli di dominio orientati agli oggetti su un database relazionale (Figura 4.5). Sostanzialmente Hibernate viene utilizzato per rendere persistenti i dati dall'ambiente Java al database. Hibernate implementa le specifiche JPA per la persistenza dei dati¹.^[22]

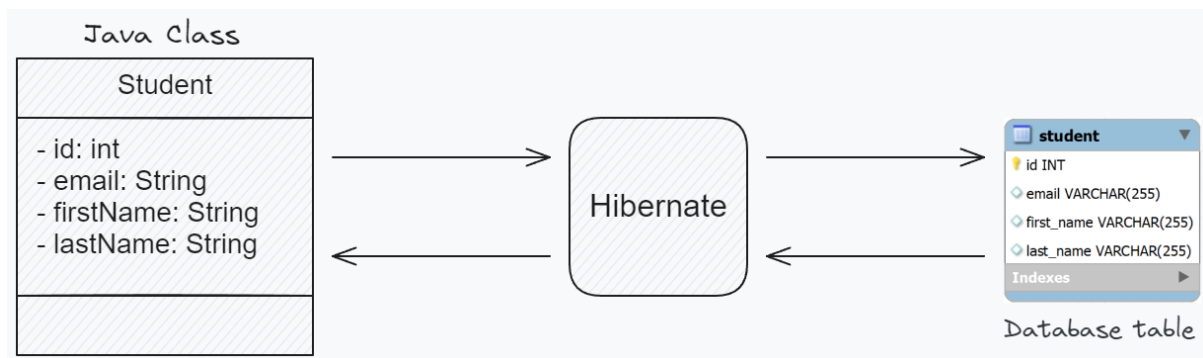


Figura 4.5: Funzionamento di Hibernate

JDBC

JDBC (Java Database Connectivity) è un'API che permette di eseguire query SQL all'interno di un'applicazione Java, consentendo a queste ultime di connettersi e interrogare dei database come PostgreSQL o MySQL.^[23]

¹In informatica, il concetto di persistenza si riferisce alla caratteristica dei dati di un programma di sopravvivere all'esecuzione del programma stesso che li ha creati. La persistenza si riferisce in particolare alla possibilità di far sopravvivere delle strutture dati all'esecuzione di un singolo programma. In ogni caso questa possibilità è raggiunta salvando i dati in uno storage non volatile, come su un file system o su un database.^[21]

Object-Relational Mapping

L'Object-Relational Mapping (ORM) è una tecnica di programmazione utilizzata per memorizzare, richiamare, aggiornare ed eliminare dati da un database all'interno di programmi object-oriented solitamente scritti utilizzando linguaggi di programmazione orientata agli oggetti (OOP), come Java o C#. L'ORM si applica ai database relazionali, conosciuti come RDBMS (Relational Database Management System), e si occupa di convertire e tradurre tutti i dati che non potrebbero altrimenti coesistere tra database e linguaggi OOP.^[24]

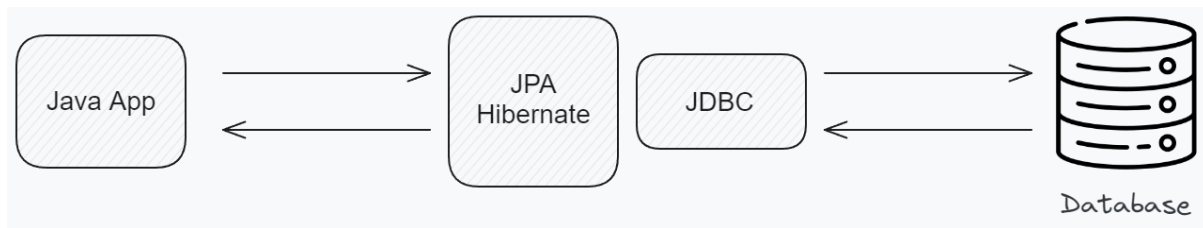


Figura 4.6: Funzionamento di JPA con il framework ORM Hibernate.

4.2.5 Funzione crittografica di hash

Una funzione crittografica di hash è un algoritmo che mappa dei dati di lunghezza arbitraria in una stringa binaria di dimensione fissa chiamata valore di hash, ma spesso chiamata digest. Tale funzione di hash è progettata per essere unidirezionale (one-way), ovvero una funzione difficile da invertire: l'unico modo per ricreare i dati di input dall'output di una funzione di hash ideale, è quello di tentare una ricerca di forza bruta di possibili input per vedere se vi è un corrispondenza.^[25]

La funzione crittografica di hash ideale deve avere alcune proprietà fondamentali:

- Deve identificare univocamente il messaggio, non è possibile che due messaggi differenti abbiano lo stesso valore di hash;
- Deve essere deterministica, in modo che lo stesso messaggio si traduca sempre nello stesso hash;
- Il calcolo di un valore hash da un qualunque tipo di dato deve essere semplice e veloce;
- Deve essere molto difficile o quasi impossibile generare un messaggio dal suo valore hash se non provando tutti i messaggi possibili.

Bcrypt

Bcrypt è una funzione crittografica di hash progettata per l'hashing delle password e l'archiviazione di queste ultime nel back-end delle applicazioni in modo tale da essere meno vulnerabile ad attacchi informatici. Bcrypt esegue un processo di hasing complesso, durante il quale la password di un utente viene trasformata in un thread di caratteri a lunghezza fissa (Figura 4.7). Utilizza una funzione di hash unidirezionale, il che significa che una volta che la password è stata sottoposta ad hashing, non può essere ripristinata alla sua forma originale. Ogni volta che l'utente accede al proprio account, Bcrypt esegue nuovamente l'hasing della password e confronta il nuovo valore hash con la versione archiviata nella memoria del sistema per verificare se le password corrispondono.^[26]

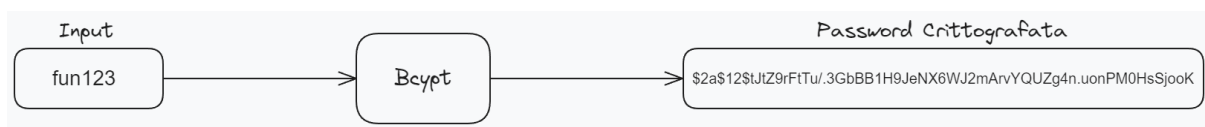


Figura 4.7: Funzionamento Bcrypt

4.3 Front-end: Tecnologie utilizzate

Per l'implementazione del front-end ho utilizzato diverse tecnologie:

- HTML
- CSS
- Bootstrap
- JavaScript
- Thymeleaf

4.3.1 HTML

HTML (Hypertext Markup Language) rappresenta la struttura portante delle pagine web: su questa struttura si possono aggiungere modifiche grafiche, grazie ai fogli di stile CSS ed elementi dinamici, grazie a JavaScript. HTML è un linguaggio di markup gerarchico strutturato ad albero: esistono collegamenti gerarchici fra gli elementi, che rendono uno il genitore dell'altro ovvero il figlio. HTML consente di descrivere semanticamente la struttura di un documento web attraverso tag, paragrafo, elenco, collegamento, citazione, o altri elementi.^[27]

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale
6         =1.0">
7     <title>Document</title>
8 </head>
9 <body>
10     <p>Hello World</p>
11
12 </body>
13 </html>
```

Listing 4.2: Esempio di una pagina html

4.3.2 CSS

CSS (Cascading Style Sheets) è un linguaggio usato per implementare lo stile di documenti scritti in un linguaggio di markup, come HTML e XML. Con cascading in CSS si intende che i fogli di stile si applicano a cascata: quando un elemento è soggetto a diverse regole, tutte le regole sono valide, ma prevale sempre l'ultima regola.^[28] Per applicare lo stile a un determinato elemento nella pagina HTML si utilizzano i selettori.

Esistono diversi tipi di selettori e vi sono delle priorità di applicazione di quest'ultimi (1 alta priorità, 6 bassa priorità):

1. Dichiarazione con !important
2. Inline style
3. ID (#) selector
4. Class (.) selector, pseudo-class (:) selector
5. Element selector
6. Universal selector (*)

```

1  * {
2    margin: 0;
3    padding: 0;
4    box-sizing: border-box;
5  }
6
7  p {
8    font-family: sans-serif;
9    color: black;
10   font-size: 22px;
11 }
12
13 .author {
14   font-style: italic;
15   font-size: 18px;
16 }
17
18 #author-text {
19   font-size: 20px;
20 }
```

Listing 4.3: Esempio di utilizzo dei selettori usando CSS

4.3.3 Bootstrap

Bootstrap è un framework di sviluppo web open source che permette la creazione di siti web responsive e mobile-first², facendo in modo che tutti gli elementi di un sito web funzionino in modo ottimale su tutte le dimensioni dello schermo.^[30] Per fare ciò vengono utilizzati dei componenti, che sono recuperabili dalla documentazione.

```

1 <div class="card" style="width: 18rem;">
2   
3   <div class="card-body">
4     <h5 class="card-title">Card title</h5>
5     <p class="card-text">Some quick example text to build on the
      card title and make up the bulk of the card's content.</p>
6     <a href="#" class="btn btn-primary">Go somewhere</a>
7   </div>
8 </div>

```

Listing 4.4: Esempio di un componente di Bootstrap (card)

4.3.4 JavaScript

JavaScript è un linguaggio di programmazione che viene utilizzato per di creare interazioni dinamiche quando si sviluppano pagine web, semplici e complesse, applicazioni, server.^[31] JavaScript ha diversi vantaggi:

- Semplicità: avere una struttura semplice rende JavaScript più facile da imparare e funziona più velocemente di altri linguaggi.
- Immediatezza: è eseguibile direttamente da un browser senza set-up aggiuntivi.
- Carico del server: operando lato client riduce le richieste inviate al server.
- Aggiornamenti: le community di sviluppatori aggiornano e creano nuovi framework e librerie.

```

1 document.querySelector(".btn-add").addEventListener("click", clear);
2
3 function clear(e) {
4   document.querySelector("#id-postit").value = "";
5   document.querySelector("#titolo-postit").value = "";
6   document.querySelector("#descrizione-postit").value = "";
7 }

```

Listing 4.5: Esempio di script JavaScript

²Il termine Mobile First è una metodologia che promuove un modo di realizzare un'esperienza web pensata e creata nativamente per i display di dimensioni ridotte, come quelli degli smartphone e che viene progressivamente adattata e integrata per i display più grandi, come quelli dei computer desktop e delle TV.^[29]

4.3.5 Thymeleaf

Thymeleaf è un motore di template, una libreria scritta in Java, che consente agli sviluppatori di definire un modello di pagina HTML e in seguito riempirlo con i dati per generare la pagina finale. Pertanto realizza una parte Model-View del Model-View-Controller.^[32] In particolar modo all'interno della classe Java avremo un oggetto di tipo Model, che fa da contenitore. Ad esempio nella classe che fa da controller utilizziamo il model, e al suo interno possiamo inserire una lista di oggetti, a cui accederemo dalla pagina HTML per poi popolarla dinamicamente.

```

1 @RequestMapping(value = "/test")
2 public String showCheckbox(Model model) {
3     boolean myBooleanVariable = false;
4     model.addAttribute("myBooleanVariable", myBooleanVariable);
5
6     return "sample-checkbox";
7 }

```

Listing 4.6: Esempio di utilizzo del contenitore Model in un metodo del controller

```

1 <input
2   type="checkbox"
3   name="myBooleanVariable"
4   th:checked="${myBooleanVariable}"/>

```

Listing 4.7: Esempio di utilizzo di un attributo del Model all'interno della pagina HTML

Capitolo 5

Implementazione delle tecnologie

In questo capitolo ci occuperemo di applicare le tecnologie trattate nel capitolo 4, andando a sviluppare la web app "Postit-app", nella quale sarà possibile effettuare il login con il proprio account e creare, aggiornare ed eliminare delle note (postit) nella home page del sito. Ogni postit è costituito da un titolo e una descrizione.

5.1 Implementazione back-end

5.1.1 Database

Per prima cosa mi sono occupato dell'implementazione del database, e per fare ciò l'ho creato utilizzando docker eseguendo il seguente comando:

```
1 docker run --name postit-app-db ^
2   -e POSTGRES_USER=utente ^
3   -e POSTGRES_DB=postit-app-db ^
4   -e POSTGRES_PASSWORD=password ^
5   -e PGDATA=/var/lib/postgresql/data/postitData ^
6   -p 5434:5432 ^
7   -v C:\Docker_Data:/var/lib/postgresql/data ^
8   postgres:14.1-alpine
```

Listing 5.1: Creazione del DB "postit-app-db"

Successivamente al fine di gestire i dati nel db e creare le diverse tabelle necessarie alla web app, ho utilizzato DBeaver, un'applicazione software client SQL e uno strumento di amministrazione di database. Una volta effettuata la connessione al DB ho creato diverse tabelle:

- users
- roles
- postit

Tabella users

All'interno della tabella users ho inserito gli utenti con le loro rispettive password criptate.

```
1 CREATE TABLE users (  
2     user_id varchar(50) not null primary key ,  
3     pw char(68) not null ,  
4     active int not null );
```

Listing 5.2: Creazione tabella users

```
1 INSERT INTO users VALUES  
2     ( 'john' , '{bcrypt}$2a$10$qeS0HEh7urweMojsnwNAR .  
        vcXJeXR1UcMRZ2WcGQI9YeuspUdgF.q' , 1 ) ,  
3     ( 'mary' , '{bcrypt}$2a$10$qeS0HEh7urweMojsnwNAR .  
        vcXJeXR1UcMRZ2WcGQI9YeuspUdgF.q' , 1 ) ,  
4     ( 'susan' , '{bcrypt}$2a$10$qeS0HEh7urweMojsnwNAR .  
        vcXJeXR1UcMRZ2WcGQI9YeuspUdgF.q' , 1 );
```

Listing 5.3: Inserimento dati degli utenti in users

Tabella roles

Al fine di gestire l'accesso a determinate pagine della web app, ad ogni utente è associato un ruolo.

```
1 CREATE TABLE roles (  
2     user_id varchar(50) references users(user_id) ,  
3     ruolo varchar(50) not null );
```

Listing 5.4: Creazione tabella roles

```
1 INSERT INTO roles VALUES  
2     ( 'john' , 'ROLE_EMPLOYEE' ) ,  
3     ( 'mary' , 'ROLE_EMPLOYEE' ) ,  
4     ( 'susan' , 'ROLE_EMPLOYEE' );
```

Listing 5.5: Inserimento dei ruoli nella tabella

Sequence per gestione dell'id

Al fine di garantire un incremento dell'id numerico di ogni postit ho creato una sequence, che ho poi utilizzato nella creazione della tabella postit.

```
1 create sequence seq_postit  
2 increment by 1  
3 start with 1  
4 minvalue 1  
5 no maxvalue;
```

Listing 5.6: Creazione della sequence per l'id del postit

Tabella postit

All'interno della tabella postit vengono salvati tutti i dati delle note e l'utente che ha creato queste ultime.

```

1 CREATE TABLE postit (
2     id bigint primary key default nextval ('seq_postit'),
3     body varchar,
4     title varchar,
5     "timestamp" timestamp with time zone default current_timestamp,
6     user_id varchar(50) references users(user_id));

```

Listing 5.7: Creazione tabella postit

5.1.2 Classe Postit

Al fine di far sì che i record salvati all'interno della tabella postit, possano essere trasformati in un oggetto Java ho creato la classe Postit e per garantire questa comunicazione ho utilizzato diverse annotation:

- **@Entity**: per far sì che un oggetto Java possa interfacciarsi con il db;
- **@Table**: per associare la classe Postit alla tabella postit salvata nel db;
- **@Column**: per associare la colonna della tabella alla variabile della classe;
- **@Id**: per indicare che la variabile è un id;
- **@SequenceGenerator**: per associare la sequence del db alla variabile della classe, garantendo l'incremento automatico.

```

1 @Entity
2 @Table(name = "postit")
3 @Getter
4 @Setter
5 @NoArgsConstructor
6 @AllArgsConstructor
7 @Builder
8 public class PostIt {
9
10     @Id
11     @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "seq_postit")
12     @SequenceGenerator(name = "seq_postit", sequenceName = "seq_postit", allocationSize = 1)
13     @Column(name = "id")
14     private Long id;
15
16     @Column(name = "body")
17     private String body;
18
19     @Column(name = "title")
20     private String title;
21
22     @Column(name = "timestamp")
23     @UpdateTimestamp
24     private LocalDateTime timestamp;
25
26     @Column(name = "user_id")
27     private String userId;
28
29 }

```

Listing 5.8: Classe PostIt.java

5.1.3 Interfaccia e classe PostitService

In mezzo alla comunicazione tra template, controller e repository vi è la classe Service la quale si occupa di chiamare i metodi della classe repository dopo l'esecuzione di un metodo del controller. Prima di implementare la classe PostitService ho definito un'interfaccia con al suo interno tutti i metodi utilizzati.

```

1 public interface PostitService {
2
3     public List<PostIt> findByUser_id( String userId );
4
5     public void save( PostIt thePostit );
6
7     public void deleteById( Long id );
8
9 }

```

Listing 5.9: Interfaccia PostitService.java

Successivamente ho implementato la classe PostitService e ho utilizzato l'annotation @Service per indicare che la classe fa da service. All'interno di questa classe come attributo vi è l'oggetto che ci permette di riferirci alla repository e quindi a comunicare con il DB.

Il metodo findByUser_id permette di cercare nel database tutti i postit che hanno come user id il valore passato come parametro, restituendo una lista di postit.

Il metodo save viene utilizzato per salvare un nuovo postit oppure uno modificato.

Il metodo deleteById permette di eliminare un postit specificando il suo id.

```

1 @Service
2 public class PostitServiceImpl implements PostitService {
3
4     @Autowired
5     private PostitRepository repo;
6
7     @Override
8     public List<PostIt> findByUser_id( String userId ) {
9         return repo.findById(userId);
10    }
11
12    @Override
13    public void save( PostIt thePostit ) {
14        repo.save(thePostit);
15    }
16
17    @Override
18    public void deleteById( Long id ) {
19        repo.deleteById(id);
20    }
21 }

```

Listing 5.10: Classe PostitServiceImpl.java

5.1.4 Interfaccia PostitRepository

L'interfaccia `PostitRepository` si occupa di interfacciarsi con il DB eseguendo le diverse operazioni richieste da parte del service. Per indicare che un'interfaccia o una classe è una repository ho usato l'annotation `@Repository`. In Spring Boot, grazie a JPA, è possibile estendere questa interfaccia con la classe `JpaRepository<PostIt, Long>` la quale fornisce automaticamente tutte le operazioni CRUD, senza la necessità di dover scrivere metodi che le implementino. In particolar modo tra le parentesi angolate come primo attributo va inserita la classe che voglio utilizzare e come secondo attributo specifico il tipo dell'id della classe utilizzata, ovvero la classe `PostIt`.

```
1 @Repository
2 public interface PostitRepository extends JpaRepository<PostIt, Long>
3 {
4     List<PostIt> findByUserId(String nome);
5 }
6 }
```

Listing 5.11: Interfaccia `PostitRepository` che estende `JpaRepository`

5.1.5 Login controller

Al fine di gestire le richieste da parte del browser per effettuare il login, ho creato la classe `LoginController`, nella quale ho il metodo `showMyLoginPage`, che una volta ricevuta la richiesta permette la visualizzazione della pagina di login.

Il metodo `accessDenied` visualizza la pagina di accesso negato, nel caso in cui un utente non possedesse i permessi per visualizzare la homepage.

In questa classe ho usato due annotation:

- `@Controller`: per indicare che la classe è un controller.
- `@GetMapping`: per gestire la richiesta del browser. Ad esempio se nell'url è presente l'endpoint `/login`, il controller si occuperà di mostrare la pagina di login eseguendo il metodo `showMyLoginPage`.

```
1 @Controller
2 public class LoginController {
3
4     @GetMapping("/login")
5     public String showMyLoginPage() {
6
7         return "login-page";
8     }
9
10    @GetMapping("/access-denied")
11    public String accessDenied() {
12
13        return "access-denied";
14    }
15
16 }
```

Listing 5.12: Login controller

5.1.6 Postit controller

La classe `PostItController` si occupa di gestire tutte le richieste della homepage. Nello specifico si occupa di garantire il corretto funzionamento delle funzionalità CRUD (Create, Read, Update, Delete). Inoltre in ogni metodo viene utilizzato il `PostitService`, che permette di comunicare con l'interfaccia `PostitRepository`, che a sua volta comunica con il db.

All'interno di questa classe è possibile trovare tre metodi:

- `getPersonalPostits`: permette di recuperare tutti i postit dell'utente corrente, inserendoli all'interno del model il quale verrà utilizzato all'interno della pagina html per visualizzare i postit.
- `addPostit`: permette di gestire la richiesta di creazione e modifica di un postit.
- `deleteById`: permette di gestire la richiesta di eliminazione di un postit.

```
1 @Controller
2 @RequestMapping("/postit")
3 @RequiredArgsConstructor
4 public class PostItController {
5
6     private final PostitService service;
7
8     @GetMapping("/all")
9     public String getPersonalPostits(Authentication authentication,
10         Model theModel) {
11         List<PostIt> personalPostits = service.findByUser_id(
12             authentication.getName());
13         PostIt p = new PostIt();
14         theModel.addAttribute("postits", personalPostits);
15         theModel.addAttribute("postitSingolo", p);
16
17         return "postit-home";
18     }
19
20     @PostMapping("/add/single")
21     public String addPostit(@ModelAttribute("postit") PostIt
22         thePostit, Authentication authentication) {
23         thePostit.setUserId(authentication.getName());
24         service.save(thePostit);
25
26         return "redirect:/postit/all";
27     }
28
29     @GetMapping("/delete/single")
30     public String deleteById(@RequestParam("postitId") Long theId) {
31         service.deleteById(theId);
32
33         return "redirect:/postit/all";
34     }
35 }
```

Listing 5.13: Postit Controller

5.1.7 Configurazione login

Per la gestione del login ho utilizzato Spring Security e ho creato una classe di configurazione al fine di configurare la corretta autenticazione di un utente.

Per indicare che la classe si tratta di una configurazione ho usato l'annotation `@Configuration`.

In questa classe sono presenti due metodi:

- `userDetailsManager`: si occupa di gestire gli utenti all'interno del database per recuperare l'id dell'utente, la password e il ruolo associato, al fine di garantire una corretta autenticazione dell'utente.
- `filterChain`: si occupa di gestire gli accessi alle pagine statiche e template a seconda del ruolo associato. Inoltre viene gestito il login in modo tale che ad un utente, dopo essersi autenticato, venga visualizzata la homepage. Oltre al login vengono gestite le eccezioni e il logout.

Nel nostro caso, un utente a cui è associato il ruolo di "EMPLOYEE" può visualizzare la homepage e accedere a tutte le funzionalità della web app. Nel caso un utente avesse un ruolo diverso da "EMPLOYEE", verrebbe mandato alla pagina di accesso negato.

```

1  @Configuration
2  public class SecurityConfig {
3
4      @Bean
5      public UserDetailsManager userDetailsManager(DataSource dataSource) {
6
7          JdbcUserDetailsManager jdbcUserDetailsManager = new JdbcUserDetailsManager(dataSource);
8
9          jdbcUserDetailsManager.setUsersByUsernameQuery(
10             "SELECT user_id, pw, active FROM users WHERE user_id=?");
11
12          jdbcUserDetailsManager.setAuthoritiesByUsernameQuery(
13             "SELECT user_id, ruolo FROM roles WHERE user_id=?");
14
15          return jdbcUserDetailsManager;
16      }
17
18      @Bean
19      public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
20
21          http.authorizeHttpRequests(configurer ->
22              configurer
23                  .requestMatchers(PathRequest.toStaticResources().atCommonLocations()).permitAll()
24                  .requestMatchers("/login").permitAll()
25                  .requestMatchers("/error").permitAll()
26                  .requestMatchers("/postit/**").hasAnyRole("EMPLOYEE")
27                  .anyRequest().authenticated()
28              )
29              .formLogin(form ->
30                  form
31                      .loginPage("/login")
32                      .defaultSuccessUrl("/postit/all")
33                      .loginProcessingUrl("/authenticateTheUser")
34                      .permitAll()
35                  )
36              .logout(LogoutConfigurer::permitAll)
37              .exceptionHandling(configurer ->
38                  configurer.accessDeniedPage("/access-denied")
39              );
40
41          return http.build();
42      }
43  }
44
45
46
47

```

Listing 5.14: Classe SecurityConfig.java

5.2 Implementazione front-end

5.2.1 Pagina di login

All'interno della login-page.html è presente un form in cui l'utente può inserire l'username e la password per autenticarsi. Se le credenziali di accesso sono corrette spring security si occuperà di mandare l'utente alla homepage. In caso contrario viene visualizzato un messaggio di errore.

```

1 <!DOCTYPE html>
2 <html lang="en" xmlns:th="http://www.thymeleaf.org">
3   <head>
4     <meta charset="UTF-8" />
5     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6     <link
7       rel="stylesheet"
8       href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.2/css/all.min.css"
9     />
10    <link th:href="@{/css/login.css}" type="text/css" rel="stylesheet" />
11
12    <title>Login Page</title>
13  </head>
14
15  <body>
16    <div class="container1" id="container">
17      <div class="form-container sign-in">
18        <form action="#" th:action="@{/authenticateTheUser}" method="POST">
19          <h1>Sign In</h1>
20
21          <div>
22            <!-- Check for login error -->
23            <div th:if="{param.error}">
24              <div>
25                <p class="invalid-username">Invalid username or password.</p>
26              </div>
27            </div>
28
29            <div th:if="{param.logout}">
30              <div class="alert alert-success col-xs-offset-1 col-xs-10">
31                <p class="logout">You have been logged out.</p>
32              </div>
33            </div>
34          </div>
35
36          <input type="text" name="username" placeholder="Username" />
37          <input type="password" name="password" placeholder="Password" />
38          <button>Sign In</button>
39        </form>
40      </div>
41      <div class="toggle-container">
42        <div class="toggle">
43          <div class="toggle-panel toggle-right">
44            <h1>Welcome to the Postit App</h1>
45            <p>Sign In to se your personal postits!</p>
46          </div>
47        </div>
48      </div>
49    </div>
50  </body>
51 </html>

```

Listing 5.15: login-page.html

5.2.2 Homepage

Nella pagina `postit-home.html` è presente una navbar in cui viene visualizzato il nome dell'utente e il suo ruolo. Inoltre sono presenti i bottoni che permettono di creare un nuovo postit ed effettuare il logout. Al di sotto della navbar vengono visualizzati i postit, i quali hanno un titolo, una descrizione e i bottoni che permettono di modificare o eliminare il postit selezionato. I postit vengono visualizzati grazie all'utilizzo di un `foreach`, il quale scorre la lista di postit creati dall'utente e per ognuno crea una card nella quale vengono inseriti i corrispettivi dati. La lista di postit creati dall'utente viene recuperata grazie all'utilizzo del contenitore `model` di Thymeleaf.

```

1 <!DOCTYPE html>
2 <html
3   lang="en"
4   xmlns:th="http://www.thymeleaf.org"
5   xmlns:sec="http://www.thymeleaf.org/extras/spring-security"
6 >
7   <head>
8     <meta charset="UTF-8" />
9
10    <!-- bootstrap -->
11    <link
12      href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css"
13      rel="stylesheet"
14      integrity="sha384-T3c6CoIi6uLrA9TneNEoa7RxnatzjcDSCmG1MXxSR1GAsXEV/Dwvykc2MPK8M2HN"
15      crossorigin="anonymous"
16    />
17    <link
18      rel="stylesheet"
19      href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.2/css/all.min.css"
20    />
21    <link
22      rel="stylesheet"
23      href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.2/css/all.min.css"
24    />
25    <link th:href="@{/css/home.css}" type="text/css" rel="stylesheet" />
26
27    <title>Postit App</title>
28  </head>
29
30  <body>
31    <!-- navbar start -->
32    <nav class="navbar navbar-expand-lg bg-body-tertiary naviga">
33      <div class="container-fluid">
34        <a class="navbar-brand titolo-app" href="#">Postit App</a>
35        <div class="container d-flex align-items-center ms-3 ps-5">
36          <p class="m-0 utente">
37            Hello ,
38            <span class="utente" sec:authentication="principal.username"></span>
39          </p>
40          <p class="ps-2 m-0 utente">
41            Your role is :
42            <span
43              class="utente"
44              sec:authentication="principal.authorities"
45            ></span>
46          </p>
47        </div>
48      </div>
49
50      <div
51        class="container d-flex align-items-center m-0 justify-content-end"
52      >
53        <!-- Button trigger modal -->
54        <button
55          type="button"
56          class="btn btn-add"
57          data-bs-toggle="modal"
58          data-bs-target="#exampleModal"
59        >
60          ADD POSTIT
61        </button>
62
63        <form action="#" th:action="@{/logout}" method="POST" class="ps-3">
64          <button type="submit" class="btn btn-logout">Logout</button>
65        </form>
66      </div>
67
68    <!-- Modal -->
69    <div
70      class="modal fade modale"
71      id="exampleModal"
72      tabindex="-1"
73      aria-labelledby="exampleModalLabel"
74      aria-hidden="true"
75    >
76      <div class="modal-dialog">
77        <div class="modal-content">

```

```

78     <div class="modal-header">
79       <h1 class="modal-title fs-5" id="exampleModalLabel">
80         Postit Modal
81       </h1>
82       <button
83         type="button"
84         class="btn-close"
85         data-bs-dismiss="modal"
86         aria-label="Close"
87       ></button>
88     </div>
89     <div class="modal-body">
90       <form
91         id="postit-form"
92         action="#"
93         th:action="@{/postit/add/single}"
94         th:object="{postitSingolo}"
95         method="post"
96       >
97         <div class="mb-3">
98           <label class="form-label">Title</label>
99
100           <input
101             type="hidden"
102             class="form-control id-postit"
103             th:field="*{id}"
104           />
105
106           <input
107             type="text"
108             class="form-control titolo-postit"
109             th:field="*{title}"
110           />
111         </div>
112         <div class="mb-3">
113           <label class="form-label">Body</label>
114           <input
115             type="text"
116             class="form-control descrizione-postit"
117             th:field="*{body}"
118           />
119         </div>
120       </form>
121     </div>
122     <div class="modal-footer">
123       <button
124         type="button"
125         class="btn btn-secondary btn-chiudi"
126         data-bs-dismiss="modal"
127       >
128         Close
129       </button>
130       <button
131         type="submit"
132         form="postit-form"
133         class="btn btn-success btn-save"
134       >
135         Save
136       </button>
137     </div>
138   </div>
139 </div>
140 <!-- modal end -->
141 </div>
142 </nav>
143 <!-- navbar end -->
144
145 <div class="container-fluid p-5" sec:authorize="hasRole('EMPLOYEE')">
146   <div class="row row-cols-5 m-0 p-0">
147     <!-- card start -->
148     <div class="col m-4 p-0" th:each="tempPostit : ${postits}">
149       <div class="card carta border-0">
150         <div class="card-body p-0 m-0 carta-contenuto text-center">
151           <h5
152             hidden
153             class="n-postit"
154             type="hidden"
155             th:text="*{tempPostit.id}"
156           />
157           <h5
158             class="card-title text-center titolo p-3 m-0"
159             th:text="*{tempPostit.title}"
160           >
161             Your Title
162           </h5>
163           <hr class="m-0 p-0" />
164           <p
165             class="card-text text-start m-4 body-postit"
166             th:text="*{tempPostit.body}"
167           >
168             Your description
169           </p>

```



```

173     <div
174       class="container d-flex justify-content-around align-items-center pb-3"
175     >
176       <a href="#" class="btn btn-aggiorna modifica mb-2">UPDATE</a>
177       <a
178         href="#"
179         th:href="@{/postit/delete/single(postitId=${tempPostit.id})}"
180         class="btn btn-danger btn-elimina mb-2 modifica"
181         onclick="if (!confirm('Are you sure you want to delete this postit?')) return false"
182       >DELETE</a>
183     >
184   </div>
185 </div>
186 </div>
187 </div>
188 </div>
189 <!-- card end -->
190 </div>
191 </div>
192
193 <!-- Bootstrap -->
194 <script
195   src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.bundle.min.js"
196   integrity="sha384-C6RzsynM9kWDrmNeT87bh95OGNyZPhcTNXj1NW7RuBCsyN/o0jlpcV8Qyq46cDfL"
197   crossorigin="anonymous"
198 ></script>
199
200 <!-- javascript -->
201 <script type="text/javascript" th:src="@{/js/home.js}"></script>
202 </body>
203 </html>

```

Listing 5.16: postit-home.html

5.2.3 Access denied page

Se l'utente non possiede il ruolo "EMPLOYEE", viene mandato alla pagina di accesso negato.

```

1 <!DOCTYPE html>
2 <html lang="en" xmlns:th="http://www.thymeleaf.org">
3   <head>
4     <meta charset="UTF-8" />
5     <title>Access Denied</title>
6   </head>
7   <body>
8     <h2>Access Denied - You are not authorized to access this resource.</h2>
9
10    <hr />
11  </body>
12 </html>

```

Listing 5.17: access-denied.html

5.3 Demo Postit App

L'utente esegue le seguenti operazioni:

Effettua il login

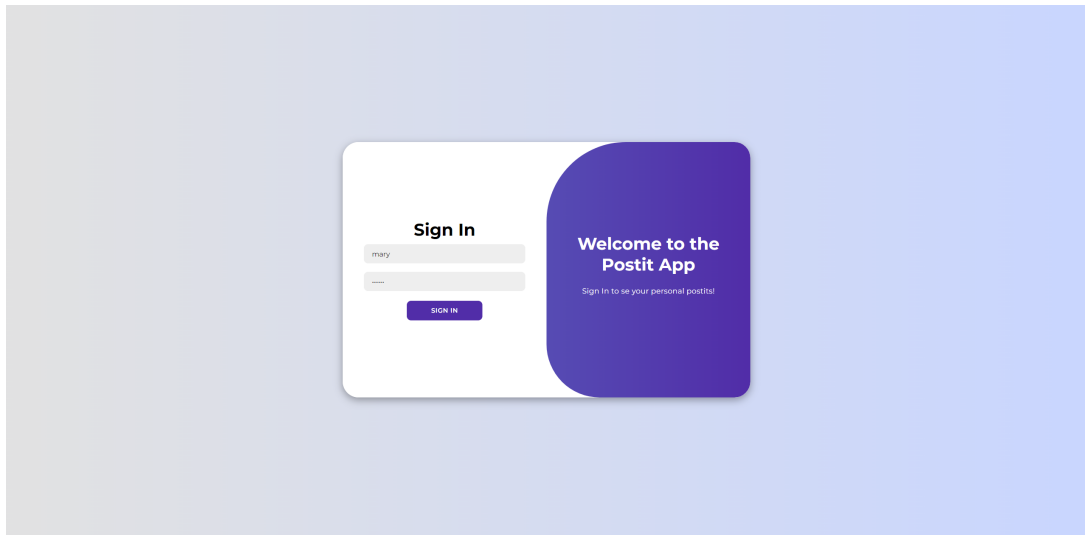


Figura 5.1: Pagina di login

Visualizza la homepage

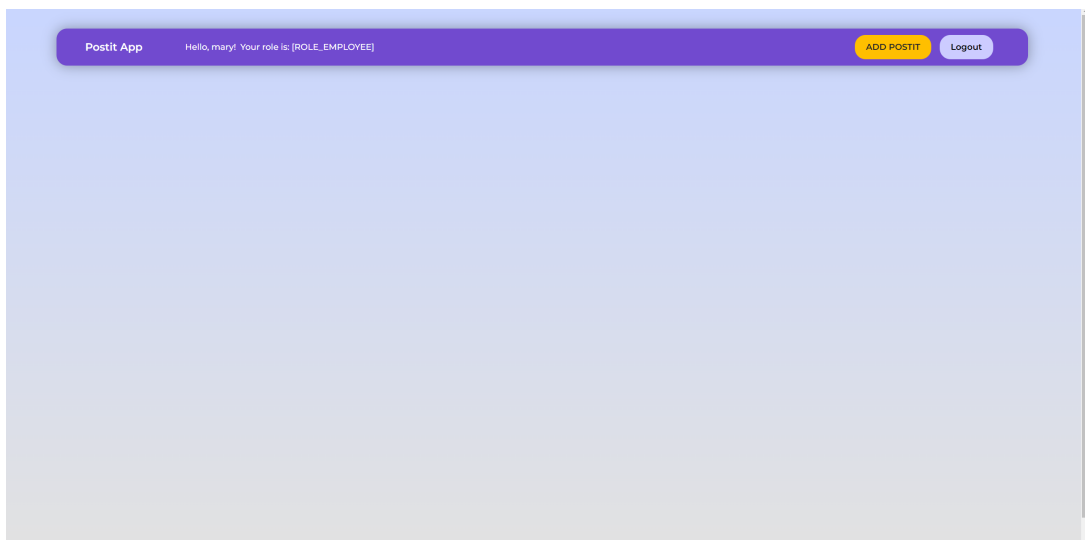


Figura 5.2: Home page

Aggiunge 3 postit

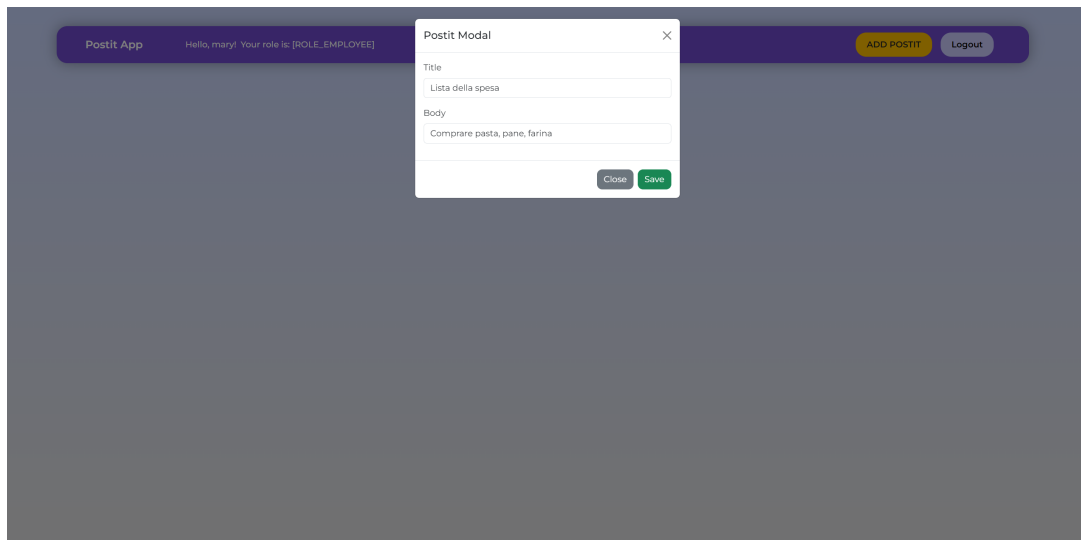


Figura 5.3: Modal creazione postit

Visualizza nella homepage i postit creati

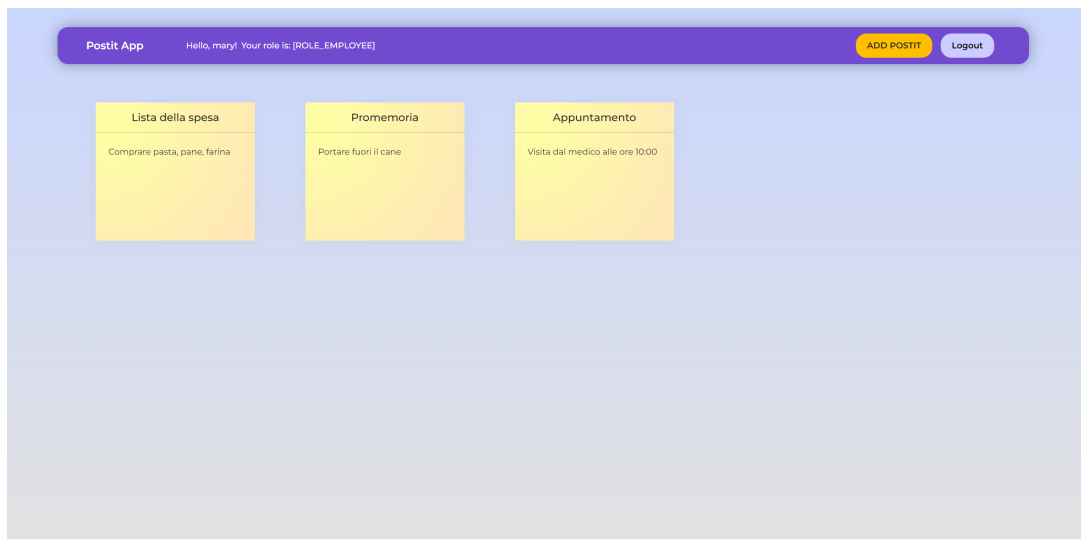


Figura 5.4: Homepage che visualizza i 3 postit creati

Modifica il primo postit

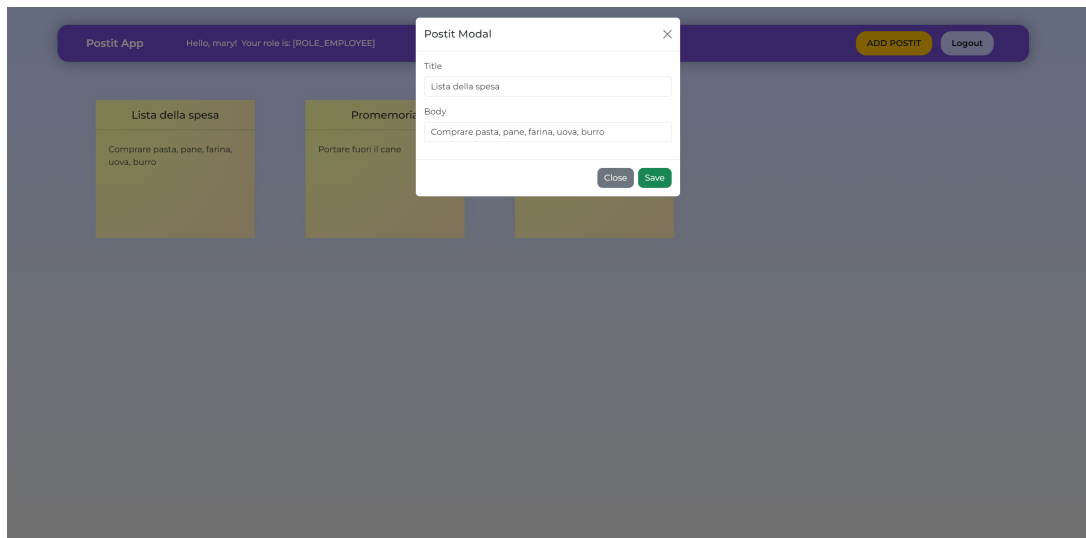


Figura 5.5: Modal di modifica postit

Elimina l'ultimo postit

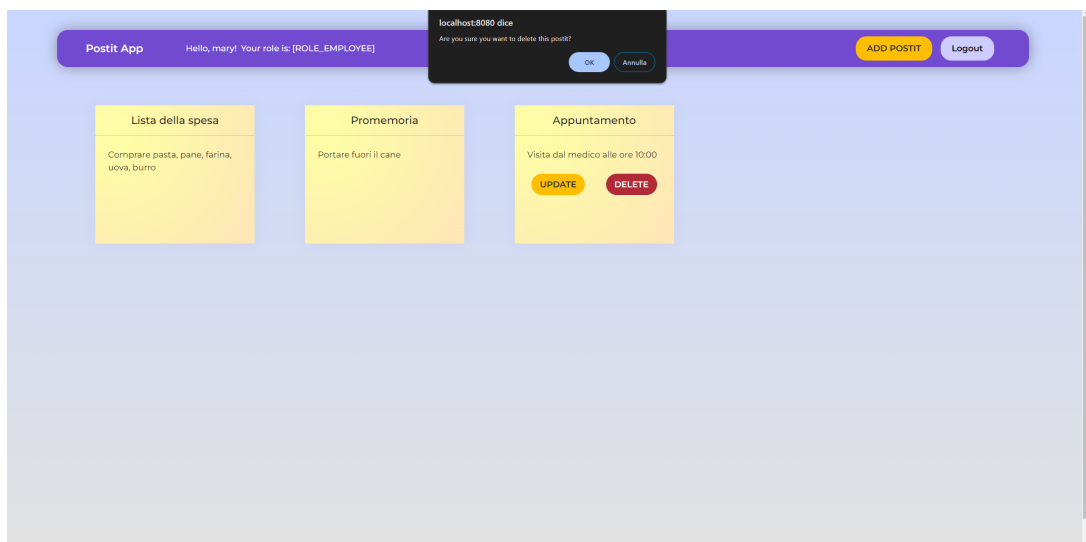


Figura 5.6: Alert di eliminazione postit

Effettua il logout

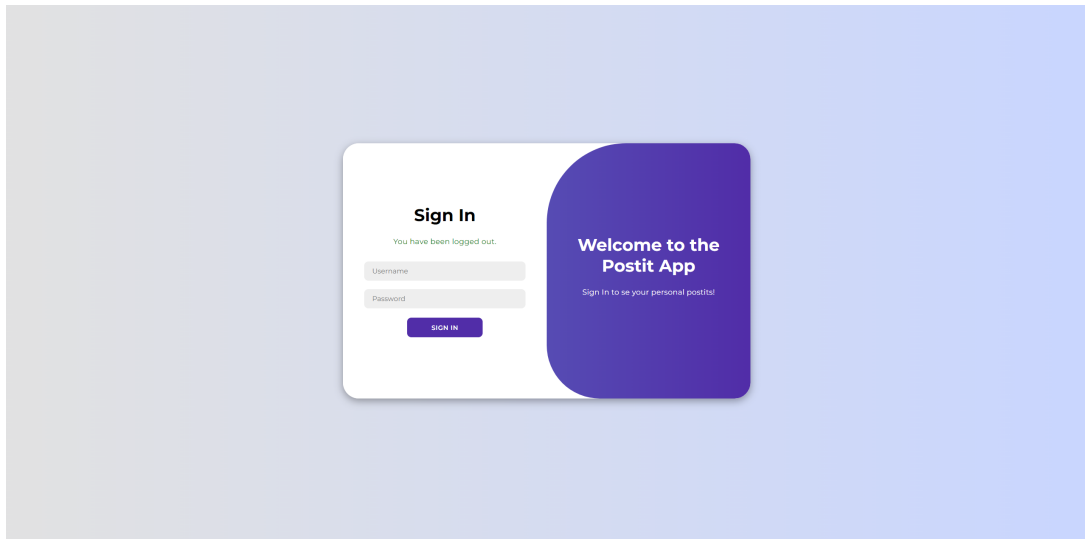


Figura 5.7: Pagina di login dopo aver effettuato il logout

Postfazione

Lavorare a questa tesi mi ha permesso di esplorare il mondo dello sviluppo web affrontando temi come la progettazione di una web app con la sua relativa struttura, composta da back-end e front-end, approfondendo anche l'ambito della gestione dei dati attraverso l'utilizzo di un database. L'impegno nello sviluppo di questo progetto si è rivelato fondamentale per comprendere le tecnologie utilizzate nello sviluppo di web app al giorno d'oggi, espandendo le mie conoscenze e capacità nel sapermi approcciare a tecnologie diverse da quelle che ho visto fino ad ora.

Ci tengo a ringraziare il mio relatore per tutte le linee guida e consigli che mi ha fornito, al fine di rendere la scrittura di questa tesi possibile.

Bibliografia

- [1] CERN. A short history of web, . URL <https://www.home.cern/science/computing/birth-web/short-history-web#:~:text=Where%20the%20Web%20was%20born,and%20institutes%20around%20the%20world>.
- [2] World Wide Web Foundation. World wide web foundation - founded by tim berners-lee, inventor of the web, the world wide web foundation empowers people to bring about positive change. URL <https://webfoundation.org/about/vision/history-of-the-web/>.
- [3] Tim Berners-Lee. Information management: A proposal. <https://cds.cern.ch/record/369245/files/dd-89-001.pdf>.
- [4] Robert Cailliau Tim Berners-Lee. Worldwideweb: Proposal for a hypertext project. https://cds.cern.ch/record/2639699/files/Proposal_Nov-1990.pdf.
- [5] CERN. Tim berners-lee's original worldwideweb browser in 1993, . URL <https://info.cern.ch/NextBrowser1.html>.
- [6] CERN. Line mode browser 2013, . URL <https://line-mode.cern.ch/>.
- [7] Wikipedia. Interoperabilità, . URL <https://it.wikipedia.org/wiki/Interoperabilit%C3%A0>.
- [8] Wikipedia. Internazionalizzazione, . URL <https://it.wikipedia.org/wiki/Internazionalizzazione>.
- [9] W3C. Web standards. URL <https://www.w3.org/standards/>.
- [10] Wikipedia. Sistema client/server, . URL https://it.wikipedia.org/wiki/Sistema_client/server.
- [11] Automation Tomorrow. Definizione di sistemi client/server. URL <https://www.automationtomorrow.com/definizione-di-sistemi-client-server/#:~:text=Si%20definiscono%20sistemi%20Client%2FServer,hardware%2Fsoftware%20con%20altri%20client>.
- [12] Informatica e Ingegneria Online. Principali vantaggi e svantaggi dell'architettura client-server. URL <https://vitolavecchia.altervista.org/principali-vantaggi-e-svantaggi-architettura-client-server/>.

Bibliografia

- [13] Aulab. Che cos'è l'http: scopriamo il protocollo http, . URL <https://aulab.it/guide/110/che-cose-http#>.
- [14] Aulab. Cos'è una web app e come funziona, . URL <https://aulab.it/notizia/190/cose-una-web-app>.
- [15] HTML.it. Spring mvc: introduzione. URL <https://www.html.it/pag/44655/spring-mvc-introduzione-2/>.
- [16] IBM. Cos'è java spring boot?, . URL <https://www.ibm.com/it-it/topics/java-spring-boot>.
- [17] GeekandJob. Cos'è maven e a cosa serve, . URL <https://www.geekandjob.com/wiki/maven>.
- [18] IBM. Cos'è docker?, . URL <https://www.ibm.com/it-it/topics/docker>.
- [19] IBM. Cosa sono i container?, . URL <https://www.ibm.com/it-it/topics/containers>.
- [20] GeekandJob. Cos'è jpa e a cosa serve, . URL <https://www.geekandjob.com/wiki/jpa>.
- [21] Wikipedia. Persistenza (informatica), . URL [https://it.wikipedia.org/wiki/Persistenza_\(informatica\)#::~:~:text=In%20informatica%2C%20il%20concetto%20di,persi%20allo%20spegnimento%20del%20computer](https://it.wikipedia.org/wiki/Persistenza_(informatica)#::~:~:text=In%20informatica%2C%20il%20concetto%20di,persi%20allo%20spegnimento%20del%20computer).
- [22] GeekandJob. Cos'è hibernate e a cosa serve, . URL <https://www.geekandjob.com/wiki/hibernate>.
- [23] GeekandJob. Cos'è jdbc e a cosa serve, . URL <https://www.geekandjob.com/wiki/jdbc>.
- [24] Psicografici. Object-relational mapping: vantaggi e svantaggi. URL <https://psicografici.com/object-relational-mapping/>.
- [25] Wikipedia. Funzione crittografica di hash, . URL https://it.wikipedia.org/wiki/Funzione_crittografica_di_hash.
- [26] NordVPN. What is bcrypt and how it works? URL <https://nordvpn.com/it/blog/what-is-bcrypt/>.
- [27] GeekandJob. Cos'è html e a cosa serve, . URL <https://www.geekandjob.com/wiki/html>.
- [28] GeekandJob. Cos'è css e a cosa serve, . URL <https://www.geekandjob.com/wiki/css>.
- [29] Whynet. Approccio mobile first per una progettazione sempre più "customer oriented". URL <https://www.whynet.info/mobile-first-progettazione-customer-oriented#::~:~:text=Il%20termine%20Mobile%20First%20%E2%80%94%20che,integrata%20per%20i%20display%20pi%C3%B9>.

Bibliografia

- [30] Creativemotions. Cos'è bootstrap e come funziona? URL <https://www.creativemotions.it/cose-bootstrap/>.
- [31] Boolean Careers. Javascript: cos'è, come funziona e a cosa serve. URL <https://boolean.careers/blog/javascript-cose-come-funziona-e-a-cosa-serve>.
- [32] riptutorial. thymeleaf. URL <https://riptutorial.com/Download/thymeleaf-it.pdf>.