

# Documentazione "Bacheca UniCollege"

---

## Prova finale Ingegneria del Software

Autore: Calandra Vincenzo Maria

Matricola: 914651

## Indice

---

- [Contesto](#)
- [Soluzione](#)
- [Risultati](#)
  - [Login Page](#)
  - [Sign Up Page](#)
  - [Home Bacheca](#)
  - [Dettaglio Attivita](#)
  - [Crea Attivita](#)
  - [Creditometro](#)
  - [Home Direttore](#)
  - [Gestione Attivita](#)
  - [Home Tutor](#)
  - [Home Segreteria](#)
- [Tecnologie](#)
- [Base di dati](#)
- [Use Case](#)
- [Dettaglio Implementazioni Tecnologiche](#)
  - [Gestione Autenticazione Autorizzazione](#)
    - [Autorizzazione](#)
    - [Autenticazione](#)
    - [Registrazione](#)
    - [MailService](#)
  - [Web](#)
    - [Spring MVC](#)
    - [Templating](#)
    - [Salvataggio Immagini](#)
  - [Testing](#)
  - [Installazione e Deploy](#)
    - [Maven](#)
    - [Docker](#)
  - [Logging E Monitoring](#)
    - [Logging](#)
    - [Monitoring](#)
  - [Class Diagram](#)

## Contesto

---

La residenza universitaria "UniCollege" permette la permanenza nella struttura subordinata al conseguimento di determinati obblighi formativi. Tali obblighi formativi consistono nell'esecuzione di determinate mansioni all'interno della residenza, nello svolgimento di attività culturali e sociali e nell'approfondimento delle proprie conoscenze attraverso la lettura di libri e nella preparazione di tertulie a tema.

Le attività vengono valutate attraverso un sistema decimale che tiene conto del numero di ore richiesto per lo svolgimento di una determinata attività e dell'impegno richiesto. Ogni studente che svolge un attività matura dunque dei "crediti" che dovranno essere segnati sul "creditometro", un foglio excel pre-formattato e condiviso, e infine validati dal tutor dello studente residente.

Ogni studente deve conseguire un tot di crediti entro il primo semestre e un altro tot entro il secondo semestre per poter proseguire il percorso interno al college rispettivamente nel secondo semestre e l'anno seguente.

Relativamente alle attività sociali e culturali gli studenti devono creare una locandina che contiene data, ora, luogo, max n. di partecipanti e descrizione. Tale locandina dovrà essere approvata dal direttore della residenza, stampata presso la segreteria e successivamente affissa in bacheca e infine dovrà essere comunicata l'affissione a tutti gli studenti tramite un messaggio sul gruppo degli studenti residenti. Ad attività conclusa lo studente organizzatore dell'attività dovrà comunicare in segreteria gli studenti che hanno partecipato all'attività per la validazione dei crediti conseguiti.

Per quanto riguarda la lettura di libri, lo studente dovrà scegliere un libro che intende leggere, richiedere al tutor di residenza se è possibile ricevere crediti dalla lettura del suddetto libro, leggere il libro, effettuare la discussione di valutazione con il tutor e infine caricare il titolo del libro letto e il numero di crediti ricevuto sul creditometro.

Infine le tertulie a tema richiedono la preparazione di una discussione su una determinata tematica, la scelta di un giorno della settimana in cui effettuare tale discussione, l'approvazione da parte della segreteria della data scelta e infine ad attività eseguita la validazione e il carimento dei crediti conseguiti sul creditometro.

A fine primo semestre il tutor dovrà assicurarsi che gli studenti a suo carico abbiano conseguito i crediti necessari per proseguire la permanenza in residenza.

## Soluzione

---

Una possibile alternativa a tale processo può essere un sistema informatico interattivo che dia la possibilità ai tutor di validare i crediti degli studenti senza dover ogni volta scaricare un file excel e aggiornarlo manualmente; che dia la possibilità agli studenti residenti di pubblicare le attività su una bacheca digitale e informare i conviventi automaticamente per email dell'avvenuta affissione e al tempo stesso permetta al direttore della residenza di poter accettare/rifiutare le attività; che mostri una lista di libri già precedentemente approvati dai tutor della residenza per il conseguimento dei crediti e che permetta di proporne dei nuovi; che permetta al segretario la supervisione del calendario delle tertulie a tema e che visualizzi un report su tutti gli studenti e varie statistiche utili e infine che permetta una gestione facilitata di inserimento di tutte le altre attività che maturano crediti su una versione innovata del "creditometro".

## Risultati

---

La soluzione proposta così si presenta:

## Login Page

### Benvenuto su Bacheca UniCollege

Effetua il login per accedere al portale

Email

Password

[Log in](#)

[Prima volta? Registrati!](#)

Powered by Vector Code

Così si presenta la pagina di accesso alla web app. Essa permette di accedere alla piattaforma tramite email e password o di passare ad una apposita pagina di registrazione. Di default vi è un account con utenza di tipo direttore attraverso cui accedere al portale.

## Sign Up Page

### Registrazione Bacheca UniCollege

Nome

Cognome

Email

Non comunicheremo la tua mail a nessuno!

Password

La password deve contenere almeno 8 caratteri, lettere maiuscole e minuscole, numeri e caratteri speciali

Conferma Password

[Registrati](#)

Powered by Vector Code

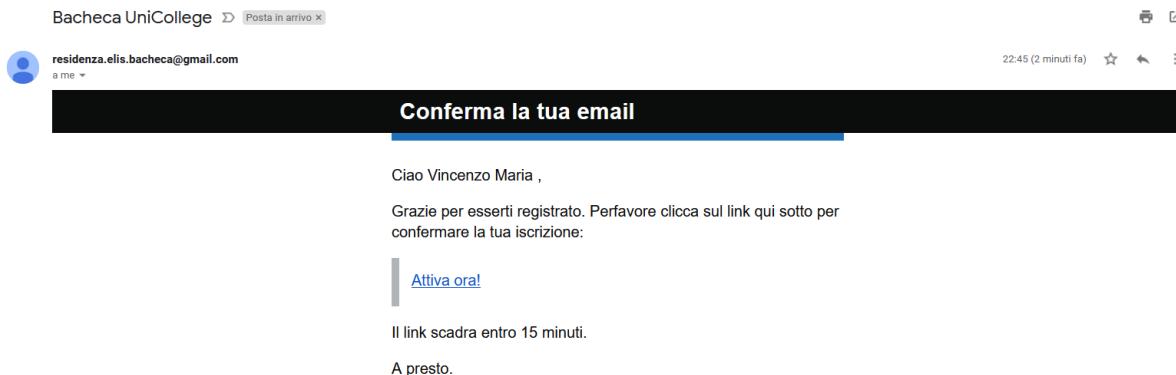
Tale form permette la registrazione sul portale. Esso richiede nome, cognome email e password del nuovo utente da registrare. Ogni campo è obbligatorio e inoltre la email inserita non deve già essere stata utilizzata. Per quanto riguarda la password essa deve contenere almeno una lettera, in caps e lower case, almeno un numero, un simbolo speciale e deve avere un minimo di 8 caratteri. Una volta generata una password che specifica la richiesta e necessario reintrodurla nell'apposito input text sottostante, per procedere è necessario che le 2 password coincidano. Una volta

cliccato sul pulsante "Registrati" si verrà reindirizzati su una pagina di conferma e verrà una mail di convalida account alla email specificata.

Registrazione avvenuta con successo, ti abbiamo spedito una mail! Per continuare conferma la tua mail per favore.

[Torna alla home](#)

La mail generata avrà l'aspetto seguente:



Nella mail possiamo trovare un messaggio di benvenuto personalizzato e un link attraverso cui confermare l'iscrizione alla web app. Il link avrà una validità di 15 minuti, dopodichè l'account creato verrà bloccato.

Una volta cliccato sul link, se non è ancora scaduto si approderà sulla seguente successful page:

Grazie di aver confermato la tua mail, ora puoi accedere al portale.

[Torna alla home](#)

## Home Bacheca

Una volta loggati con succeduto la seguente landing page ci accoglierà sul portale mostrandoci tutte le attività che sono state programmate per i giorni successivi.

In alto troviamo una navigation bar attraverso cui moversi all'interno dell'web app e un button attraverso cui chiudere la sessione sulla web app.



## Gita in montagna

Tipo: GITA

Crediti: 0.3

Giorno: 2021-12-25 - 2021-12-25

Ora: 09:00-18:00

[Dettaglio Attività](#)

## Volontariato Caritas

Tipo: VOLONTARIATO

Crediti: 0.3

Giorno: 2021-12-26 - 2021-12-26

Ora: 11:00-15:00

[Dettaglio Attività](#)

## Visita ai Musei Vaticani

Tipo: VISITA CULTURALE

Crediti: 0.3

Giorno: 2021-12-27 - 2021-12-27

Ora: 09:00-12:00

[Dettaglio Attività](#)

Ogni card rappresenta un'attività distinta dalle altre e sono caratterizzate dagli attributi principali che descrivono un'attività. Su ogni card troviamo un apposito pulsante attraverso cui è possibile accedere al dettaglio dell'attività.

## Dettaglio Attività

La pagina dettaglio attività così si presenta:

### Dettaglio Attività



#### Descrizione

Questo sentiero è davvero magico! Passeggiando nella foresta della Valle Intelvi incontrerete personaggi e animali che sembrano appena usciti da una fiaba: gufi, scoiattoli, mostri, gnomi, maschere scolpiti nei tronchi degli alberi che vi accompagneranno passo passo fino alla magnifica terrazza del Monte Comana dove potrete riposarvi godendo di un incredibile panorama a picco sul ramo comasco del Lago di Como. Il sentiero è ombreggiato e non presenta difficoltà o pericoli, circa a metà del sentiero inoltre troverete un agriturismo e una zona pic-nic vicino a un laghetto, un buon punto d'appoggio per il pranzo. Siamo certi che questa gita entusiasmerà proprio tutta la famiglia!

[Iscriviti](#)

Posti disponibili: 4

#	Nome e Cognome	Organizzatore
#	Mario Rossi	organizer

Lista Iscritti

Come è possibile facilmente constatare, essa contiene l'elenco dei partecipanti e degli organizzatori dell'attività, il numero di posti disponibili, l'immagine di locandina, la descrizione dell'attività e una coppia di button esclusivi tra loro che permettono di iscriversi o cancellarsi dall'attività.

## Dettaglio Attività



### Descrizione

Questo sentiero è davvero magico! Passeggiando nella foresta della Valle Intelvi incontrerete personaggi e animali che sembrano appena usciti da una fiaba: gufi, scoiattoli, mostri, gnomi, maschere scolpiti nei tronchi degli alberi che vi accompagneranno passo passo fino alla magnifica terrazza del Monte Comana dove potrete riposarvi godendo di un incredibile panorama a picco sul ramo comasco del Lago di Como. Il sentiero è ombreggiato e non presenta difficoltà o pericoli, circa a metà del sentiero inoltre troverete un agriturismo e una zona pic-nic vicino a un laghetto, un buon punto d'appoggio per il pranzo. Siamo certi che questa gita entusiasmerà proprio tutta la famiglia!

[Cancellati](#)

Sei iscritto a questa attività!

Posti disponibili: 3

#	Nome e cognome	Organizzatore
#	Mario rossi	organizzatore
#	Vincenzo Maria Calandra	

Lista Iscritti

## Crea Attività

Attraverso la barra di navigazione in alto è possibile raggiungere la pagina dedicata alla creazione delle attività. E' possibile creare tre tipologie differenti di attività, suddivise per informazioni necessarie per la loro creazione.

### Nuova Attività

[Attività Generica](#) [Tertulia a tema](#) [Libro](#)

Cliccando sulle rispettive categorie si apriranno i seguenti tabs:

## Nuova Attività

[Attività Generica](#) [Tertulia a tema](#) [Libro](#)

### Attività Generica

Titolo Attività

Descrizione Attività

Data inizio

Data fine

Ora inizio

Ora fine




N. max di partecipanti

Foto per bacheca

Nessun file selezionato.

## Tertulia a tema:

Bacheca

Crea Attività

Gestione Attività

Home Direttore

UniCollege Bacheca

Logout

## Nuova Attività

[Attività Generica](#) [Tertulia a tema](#) [Libro](#)

### Tertulia a tema

Titolo Attività

Descrizione Tema

Attenzione, data e ora sono suscettibili a cambiamento!

Data inizio

Ora fine

Foto per bacheca

Nessun file selezionato.

## Libro:

## Nuova Attività

Attività Generica Tertulia a tema Libro

[Proponine uno](#) [Lasciati consigliare](#)

Titolo Libro

Titolo Attività

Descrizione Tema

Breve Descrizione

0,5

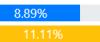
[Invia](#)

I campi sono tutti obbligatori. Una volta cliccato sul pulsante invia l'utente sarà inviato alla pagina principale e sarà mostrato un messaggio di operazione completata con successo.

Per quanto riguarda il tab "Libro" esso permette di scegliere se proporre un nuovo libro oppure se sceglierne uno già approvato.

## Creditometro

Ogni studente avrà accesso al proprio report personale di attività svolte nella seguente pagina:



#	Titolo	Crediti	Tipo	Data	Organizzatore
#	Gita in montagna	0.3	GITA	2021-12-25	organizzatore
#	Volontariato Caritas	0.3	VOLONTARIATO	2021-12-26	organizzatore
#	Visita ai Musei Vaticani	0.3	VISITA CULTURALE	2021-12-27	
#	Gli Scacchi	0.2	TERTULIA A TEMA	2021-12-18	organizzatore

[Lista Attività](#)

Come possiamo vedere troviamo 2 sezioni sostanziali. Una sezione contenente le barre di stato indicanti il numero dei crediti già approvati, in blu, e non approvati, in giallo, e una seconda sezione contenente il dettaglio delle attività svolte.

## Home Direttore

Il direttore ha accesso ad una pagina di gestione degli utenti della web app:

Nome	Cognome	Tipo	Tutor	Azione
UniCollege	Bacheca	DIRETTORE	*	<a href="#">Rimuovi Utente</a> <input type="button" value="Assegna Tutor"/> <input type="button" value="Assegna"/> <input type="button" value="Assegna Ruolo"/> <input type="button" value="Assegna"/>
Vincenzo	Maria Calandra	STUDENTE	*	<a href="#">Rimuovi Utente</a> <input type="button" value="Assegna Tutor"/> <input type="button" value="Assegna"/> <input type="button" value="Assegna Ruolo"/> <input type="button" value="Assegna"/>

Lista Utenti

Da tale pagina è possibile cambiare il ruolo all'interno della web app dei singoli account tra i seguenti tipi: STUDENTE, TUTOR, SEGRETERIA E DIRETTORE. Inoltre è anche possibile settare il tutor dello user selezionato oppure eliminare un account.

## Gestione Attività

Il direttore ha anche accesso ad una pagina di gestione attività:

#	Titolo	Crediti	Tipo	Data	Azione
#	Gita in montagna	0.3	GITA	2021-12-25	<input type="button" value="Accetta"/> <input type="button" value="Rifiuta"/>
#	Volontariato Caritas	0.3	VOLONTARIATO	2021-12-26	<input type="button" value="Accetta"/> <input type="button" value="Rifiuta"/>

Lista di attività da approvare

All'interno vengono visualizzate tutte le attività da approvare tranne i libri che vengono gestiti direttamente dai singoli tutor. Il direttore può scegliere se approvare o rifiutare un'attività oppure vederne il dettaglio cliccando sul titolo. Quando un'attività viene approvata vengono notificati tutti gli utenti dell'app.

## Home Tutor

Per quanto riguarda i tutor, essi avranno a disposizione una pagina dedicata attraverso cui approvare i crediti dei propri mentee.

Mentee	Crediti	Attività	Giorno	Crediti	Organizzatore	Azione
UniCollege Bacheca	<div style="width: 17.7%;"><div style="width: 100%;">17.7%</div></div> 0.0%	Gita in montagna	2021-12-25	0.3	organizzatore	<button>Approva</button>
Mario Rossi	<div style="width: 8.5%;"><div style="width: 100%;">8.5%</div></div> 0.1%	Visita ai Musei Vaticani	2021-12-27	0.3		<button>Approva</button>
Lista attività mentee						
Lista Mente						

Sulla destra troviamo la lista dei mentee con relativa percentuale di crediti approvati, in blu, e crediti non approvati, in giallo. Cliccando sui rispettivi nomi è possibile aprire una seconda table attraverso cui approvare le attività svolte dal mentee selezionato.

## Home Segreteria

La segreteria ha accesso ad un area riservata alla gestione degli studenti, in cui può visualizzarne le informazioni essenziali, e una pagina dedicata alla gestione delle tertulie a tema, vediamola nel dettaglio.

### Home Page Segreteria Residenza

Registro Residenti Programmazione Tertulia a tema

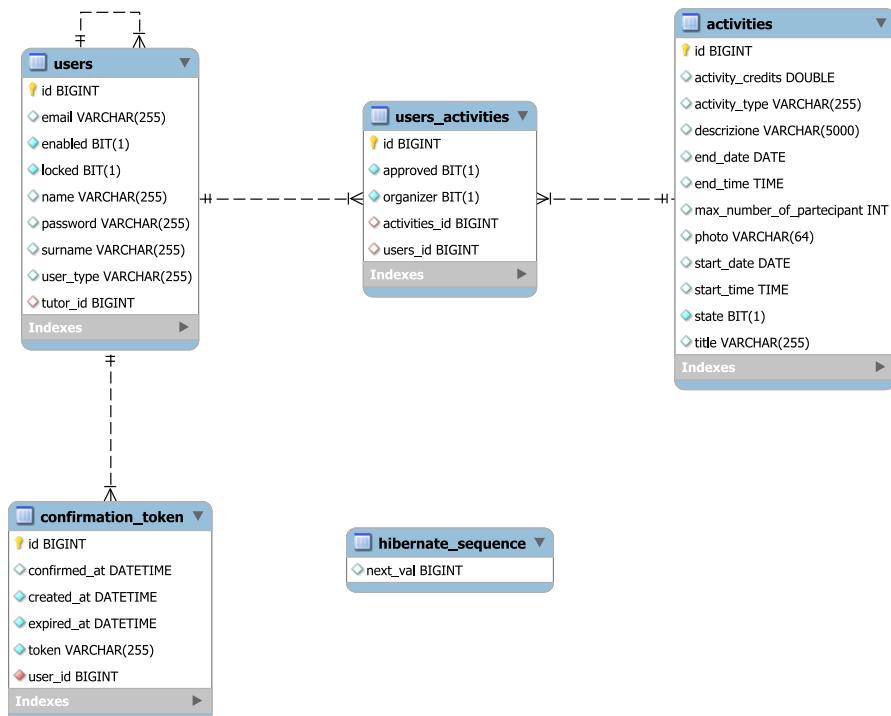
Data	Ora	Titolo	Autore	Azione	Giorno Attività
2021-12-24	13:30:00	Il galateo	Vincenzo Maria	<button>Accetta</button> <button>Rifiuta</button> <button>Aggiorna</button>	<div style="display: flex; align-items: center;"> <span>31,</span> <span></span> </div> <div style="margin-top: 10px;"> <span>Ora inizio</span> <input type="text" value="20:30"/> <span>Ora fine</span> <input type="text" value="21:00"/> </div> <div style="margin-top: 10px;"> <span><button>Invia</button></span> </div>

Lista Tertulie a tema da approvare

Cliccando su aggiorna verrà mostrato un secondo menù di modifica attività oppure è possibile accettare o rifiutare l'attività. Quando una tertulia viene accettata vengono notificati tutti gli utenti dell'app.

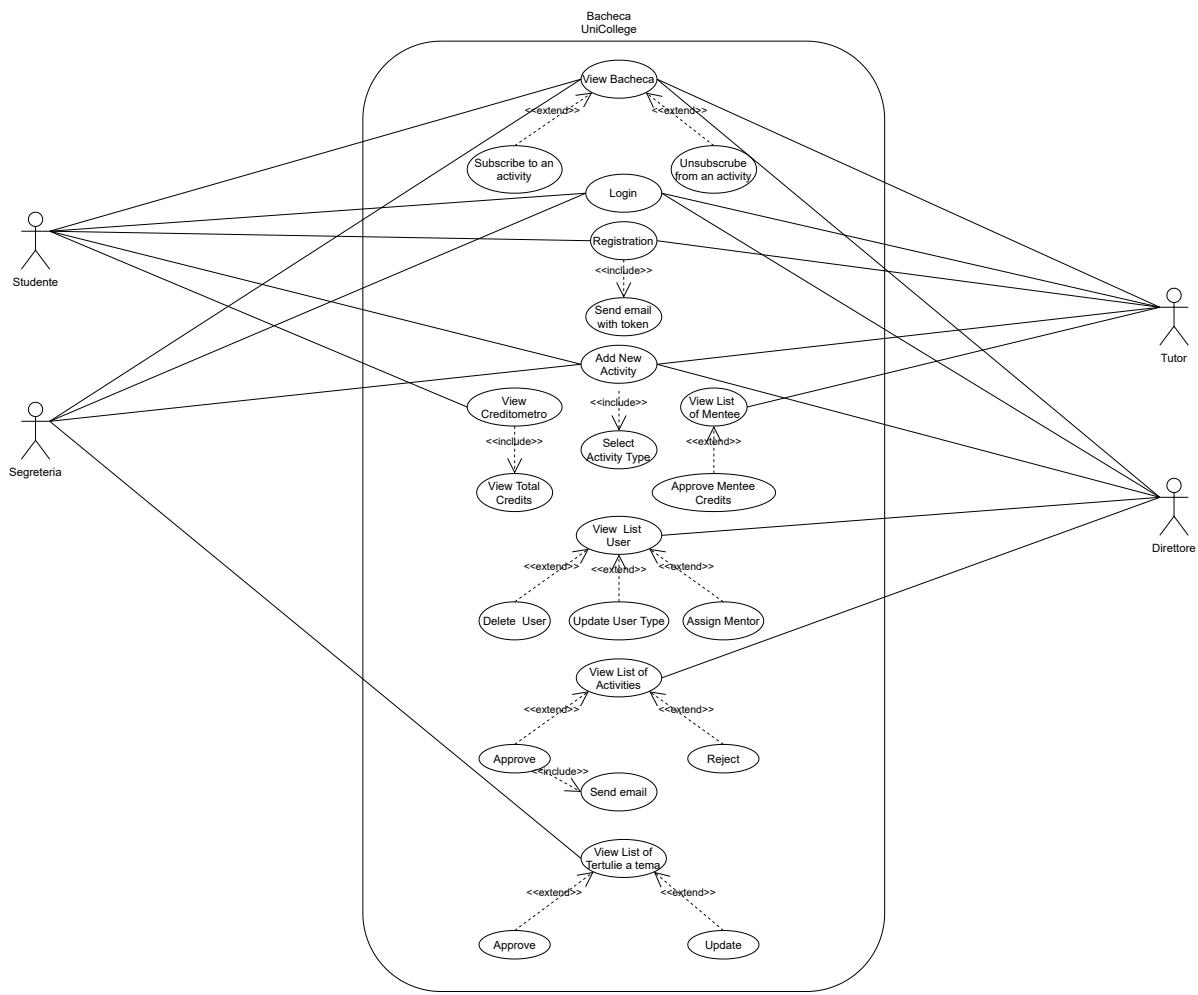
## Base di dati

Di seguito il diagramma ER della base di dati.



## Use Case

---



## Tecnologie

Per il conseguimento di tali obiettivi è stato scelto come linguaggio di programmazione JAVA 11, ad oggi esiste un numero notevole di applicazioni e siti web che fanno uso di questo linguaggio.

Infatti JAVA è un linguaggio ad alto livello orientato agli oggetti con una forte tipizzazione statica che si appoggia sulla omonima piattaforma, JVM, per questo motivo JAVA, che è linguaggio sia compilato che interpretato, riesce a girare su un numero considerevole di dispositivi differenti con prestazioni mediocri in termini di tempo di esecuzione e prestazioni.

Grazie a queste sue caratteristiche JAVA, nelle sue varie versioni, ha creato intorno a se nel corso degli anni un forte interesse da parte delle community di sviluppatori che hanno creato innumerevoli librerie e framework per questo linguaggio.

A tal proposito si è scelto di adottare Spring Boot come framework per lo sviluppo web della applicativo. Spring Boot è un evoluzione del già noto Spring, il framework di JAVA più utilizzato. Spring permette di programmare in maniera facile, veloce e sicura grazie ad una serie di librerie native che implementano le best practice per il soddisfacimento di vari use-case in termini di sicurezza e performance in grado di collaborare tra loro permettendo così allo sviluppatore di poter concentrare i propri sforzi non sulla tecnologia ma sulla logica di business che dovrà animare l'applicativo sviluppato.

Spring Boot introduce un livello di astrazione ancora superiore, esso infatti implementa una gestione facilitata delle configurazioni delle singole librerie del framework grazie ad un file di configurazione globale e una gestione automatica delle compatibilità delle versioni tra le varie librerie del framework. L'uso commerciale di Spring è regolato dalla licenza Apache 2.0.

Per quanto riguarda la gestione del processo di build del progetto è stato scelto Maven. Maven è un framework dichiarativo di gestione del progetto che segue specifiche convenzioni per quanto riguarda la struttura dello stesso. Maven fa il build di un progetto usando il suo Project Object Model (POM) file e un insieme di plugin. Una volta che si familiarizza con un progetto Maven si è in grado di conoscere qualsiasi progetto Maven e ciò fa risparmiare molto tempo allo sviluppatore. In questo progetto sono state utilizzate principalmente le funzioni di gestione delle dependencies usate nel progetto, di build, di unit testing report e coverage.

Per quanto riguarda la persistenza dei dati la scelta è ricaduta su una database SQL di largo utilizzo, MySQL. Tale tecnologia permette alte prestazioni e scalabilità su Online Transaction Processing (OLTP) application ed è transaction safe, in quanto supporta pienamente operazioni ACID. Inoltre è molto semplice da utilizzare ed è distribuito in una versione Community con licenza GPL, General Public License.

I software che vengono distribuiti con tale licenza possono essere utilizzati per tutti gli scopi, inclusi scopi commerciali e persino come strumento per la creazione di software proprietario. Infine si è scelto di effettuare il deploy dell'applicativo utilizzando la containerizzazione piuttosto che la virtualizzazione. La conteinerizzazione permette maggiore scalabilità e gestione delle risorse rispetto alla virtualizzazione, in questo modo ogni applicativo viene eseguito in processi che consumano il minimo delle risorse e che sono isolati tra loro. Inoltre la conteinerizzazione permette di impacchettare il software con tutte le sue dependencies e distribuirlo da un computing env ad un altro senza alcun problema di compatibilità. L'implementazione scelta, nonchè la più famosa e quella su cui si è poi basato lo standard di mercato,"containerd" della CNCF, è Docker. Docker Engine ha un piano di utilizzo gratuito ad uso personale o commerciale regolato dalla licenza Apache 2.0.

## Dettaglio implementazioni tecnologiche

### Gestione Autenticazione e Autorizzazione

Le esigenze di progetto hanno rivelato la necessità dell'implementazione di un layer di security nella web application, in particolar modo è venuto fuori dalle analisi che:

- Occorre permettere l'accesso al portale solo ad utenti registrati, per tenere traccia dei fruitori del portale e per permettere future implementazioni correlate.
- Evitare che gli studenti abbiano accesso ad aree del portale riservate alla segreteria, alla direzione e ai tutor.

Per tale motivo si è deciso di utilizzare le librerie di Spring Security per l'implementazione di quanto citato sopra.

Quando si lavora con Spring Boot, lo *spring-boot-starter-security* ingloba tutte le dependecies necessarie per l'implementazione di un layer di security all'interno dell'applicativo, come ad esempio *spring-security-core*, *spring-security-web*, e *spring-security-config*. Nel file *pom.xml* troviamo:

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

La classe di configurazione di Spring Security estende la classe astratta *WebSecurityConfigurerAdapter*.

Aggiungendo l'annotation `@EnableWebSecurity` attiviamo l'integrazione tra Spring MVC e Spring Security:

```
@Configuration  
@EnableWebSecurity  
public class SecurityConfig extends WebSecurityConfigurerAdapter {  
  
    private UserService userService;  
    private BCryptPasswordEncoder bCryptPasswordEncoder;  
  
    @Autowired  
    public SecurityConfig(UserService userService, BCryptPasswordEncoder  
bCryptPasswordEncoder) {  
        super();  
        this.userService = userService;  
        this.bCryptPasswordEncoder = bCryptPasswordEncoder;  
    }  
}
```

A partire da Spring 5, occorre definire un password encoder. Nel nostro caso la scelta è ricaduta su **BCryptPasswordEncoder** definito nella classe `PasswordEncoder` all'interno del package security:

```
@Configuration  
public class PasswordEncoder {  
  
    @Bean  
    public BCryptPasswordEncoder bCryptPasswordEncoder() {  
        return new BCryptPasswordEncoder();  
    }  
  
}
```

Tale password encoder è poi inglobato nella configurazione globale di Spring Security grazie all'annotation `@Bean`.

BCrypt negli ultimi anni è diventato uno standard nell'hashing delle password, è basato sull'algoritmo blowfish ed aggiunge un salt ed un fattore di costo all'hashing, in questo modo è resistente ad attacchi di tipo *rainbow table*. Inoltre è una funzione adattiva, cioè la sua complessità aumenta con il numero di iterazione, per tale motivo è anche resistente ad attacchi di tipo *brute force*. Da una serie di crypto-analisi è venuto fuori che BCrypt è molto più complesso da "crackare" rispetto a SHA-512, in quanto quest'ultimo è messo a dura prova dal calcolo parallelizzato e dall'utilizzo di GPU sempre più potenti a differenza di BCrypt che non subisce tali fattori.

## Autorizzazione

A questo punto possiamo procedere con il settaggio delle autorizzazioni sulle richieste ai vari endpoint. La configurazione base prevede l'accesso anonimo agli endpoint `/`, `/login`, `/registration` e `/error`. In questo modo gli utenti possono registrarsi e autenticarsi.

```
@Override  
protected void configure(HttpSecurity http) throws Exception {  
  
    http.csrf().disable().authorizeRequests()
```

```

        .antMatchers("/registration/**", "/login/**", "/error/**", "/", "/confirmationPage/**").permitAll()
            .antMatchers("/homeDirettore/**").hasAuthority("DIRETTORE")
            .antMatchers("/gestioneAttivita/**").hasAuthority("DIRETTORE")
            .antMatchers("/homeTutor/**").hasAuthority("TUTOR")
            .antMatchers("/homeSegreteria/**").hasAuthority("SEGRETERIA")
            .anyRequest().authenticated()

        .and().formLogin().loginPage("/login").defaultSuccessUrl("/homePage",
        true)
            .failureUrl("/login?error=true")
        .and().logout()
            .logoutSuccessUrl("/")
            .logoutUrl("/perform_logout")
            .invalidateHttpSession(true);

    }

```

Inoltre si è ristretto l'accesso agli endpoint /homeDirettore e /gestioneAttivita ai soli utenti etichettati come DIRETTORE. In maniera analoga si è proceduto con gli endpoint /homeTutor e /homeSegreteria rispettivamente per gli utenti etichettati come TUTOR e SEGRETERIA.

```

        .antMatchers("/homeDirettore/**").hasAuthority("DIRETTORE")
        .antMatchers("/gestioneAttivita/**").hasAuthority("DIRETTORE")
        .antMatchers("/homeTutor/**").hasAuthority("TUTOR")
        .antMatchers("/homeSegreteria/**").hasAuthority("SEGRETERIA")

```

## Autenticazione

Tutti gli altri accessi al portale sono vincolati alla sola autenticazione tramite /login.

```

        .anyRequest().authenticated()
        .and().formLogin().loginPage("/login").defaultSuccessUrl("/homeBacheca",true)

```

Da notare che l'ordine degli elementi contrassegnati da *antMatchers()* è significativo, le regole più specifiche sono poste prima di quelle generali.

Di seguito sono riportate le configurazioni di logout:

```

        .and().logout()
            .logoutSuccessUrl("/")
            .logoutUrl("/perform_logout")
            .invalidateHttpSession(true);

```

Da notare che sono stati cambiati tutti i valori di default degli endpoint, questa buona pressi evita che il framework sia facilmente identificabile dall'esterno, evitando così attacchi mirati.

Il login form di Spring deve contenere i seguenti attributi:

- *login* – lo URL dove effettuare la chiamata POST.
- *username* – lo username.
- *password* – la password.

```

<form action="login" method="post">
    <div class="form-floating">
```

```

<label for="username"></label>
<input type="text" class="form-control" name="username"
id="floatingInput" aria-describedby="emailHelpId"
placeholder="your.email@example.com">
</div>

<div class="form-floating">
<label for="password"></label>
<input type="password" class="form-control" name="password"
id="floatingPassword" placeholder="password"> <br>
</div>

<div class="form-floating d-flex flex-column">
<input class="w-100 btn btn-lg btn-primary mr-3" type="submit"
value="Log in" />
<a th:href="@{/registration}" href="#" class="mt-2 font-weight-light
font-size">First time? Click to Sign up!</a>
</div>
<p class="mt-5 mb-3 text-muted">Powered by Vector Code</p>
</form>

```

Tale form è contenuto all'interno della file login.html nella cartella *src/main/resources/templates*. Tale path è il percorso che viene utilizzato da Spring per servire i contenuti dinamici della web app.

## Benvenuto su Bacheca UniCollege

Effettua il login per accedere al portale

Email

mail@example.com

Password

password

Log in

[Prima volta? Registrati!](#)

Powered by Vector Code

A questo punto occorre chiedersi: "Dove avviene tutta la magia dell'autenticazione?". La risposta è semplice, occorre dichiarare un provider che si occupi di tutto ciò:

```

@Bean
public DaoAuthenticationProvider daoAuthenticationProvider() {

    DaoAuthenticationProvider provider = new DaoAuthenticationProvider();

    provider.setPasswordEncoder(bCryptPasswordEncoder);
    provider.setUserDetailsService(userService);

    return provider;
}

```

In questo caso viene detto al provider che l'autenticazione va effettuata usando il `bCryptPasswordEncoder` dichiarato precedentemente e lo user `userService`, una classe che si occupa esclusivamente di interagire con la entity `AppUser` e che implementa l'interfaccia `UserDetailsService` che richiede di implementare il seguente metodo:

```

@Service
public class UserService implements UserDetailsService {
    ...

    // Login a user by username (email)
    @Override
    public UserDetails loadUserByUsername(String email) throws
        UsernameNotFoundException {
        ...
        if (email == null || email.isEmpty() || email.isBlank()) {
            throw new UsernameNotFoundException("Email could not be null!");
        }

        Optional<AppUser> userOptional = userRepository.findByEmail(email);

        if (userOptional.isPresent()) {
            return userOptional.get();
        } else {
            throw new UsernameNotFoundException("Unable to find user with e: " +
                email);
        }
        ...
    }
}

```

Grazie a tale `override` è possibile effettuare il login alla web app usando come username la mail dell'utente, che è univoca.

## Registrazione

Per quanto riguardo la registrazione si è adottato un meccanismo di verifica a 2 passaggi, di seguito sono elencati i punti principali:

- Registrazione su apposita pagina raggiungibile tramite endpoint "/registration" o tramite opportuna page di login.

## Registrazione Bacheca UniCollege

Nome

Vincenzo Maria

Cognome

Calandra

Email

vincenzomaria.calandra@gmail.com

Non comunicheremo la tua mail a nessuno!

Password

La password deve contenere almeno 8 caratteri, lettere maiuscole e minuscole, numeri e caratteri speciali

Conferma Password

**Registrati**

Powered by Vector Code

- Conferma dell'avvenuta registrazione correttamente o incorrettamente.

## Registrazione Bacheca UniCollege

Password must contains letters, numbers and special characters!

Nome

Vincenzo

Cognome

Calandra

Email

example@mail.com

We'll never share your email with anyone else.

Password

Confirm Password

**Registrati**

Or:

Registrazione avvenuta con successo, ti abbiamo spedito una mail! Per continuare conferma la tua mail perfavore.

[Torna alla home](#)

- Ricevimento della mail contenente il token nella url di conferma.

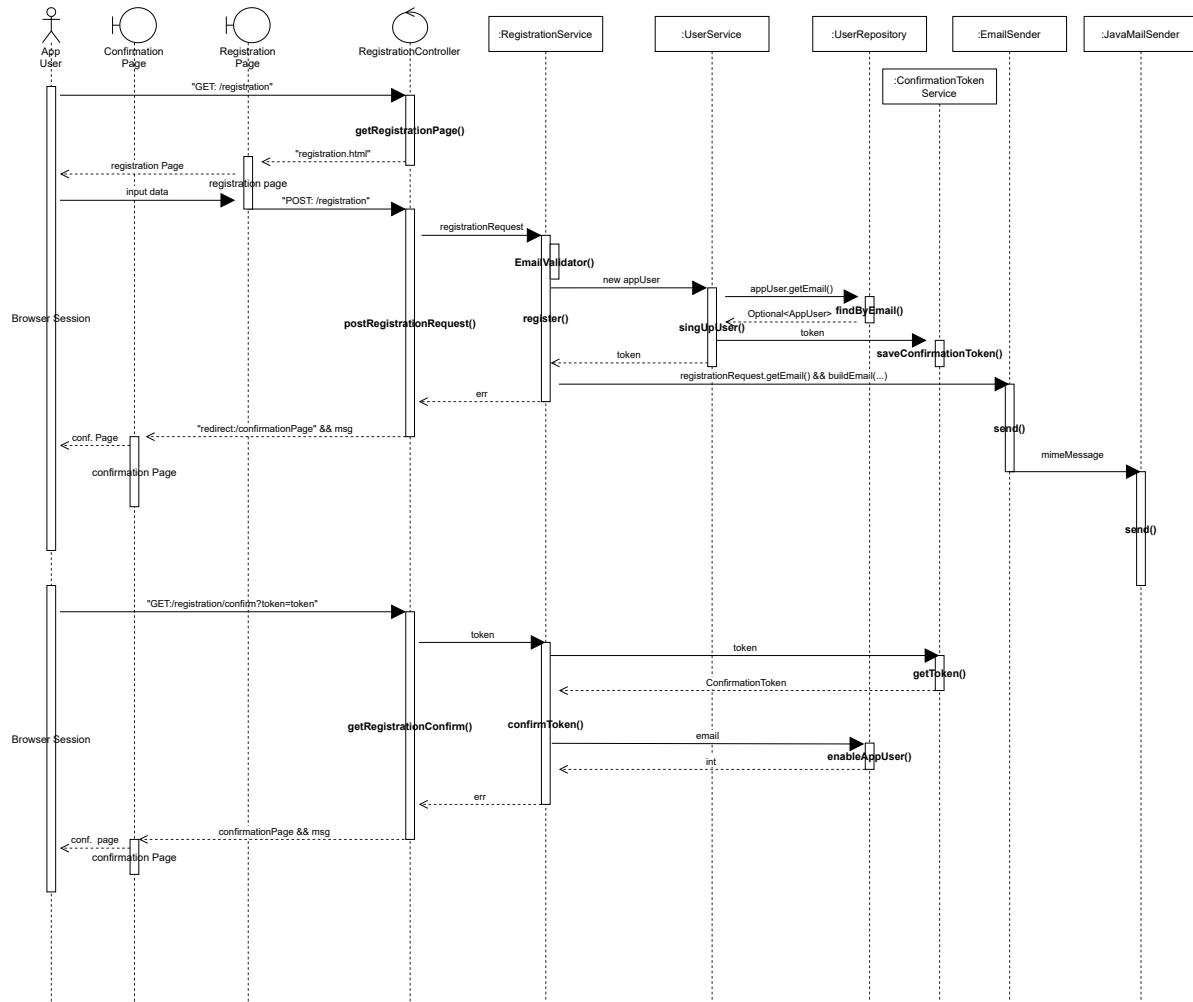


- Messaggio di success:

Grazie di aver confermato la tua mail, ora puoi accedere al portale.

[Torna alla home](#)

Il processo di registrazione segue dunque un percorso un po' più articolato rispetto all'autenticazione, per tale motivo si è provvisto a creare una rappresentazione grafica opportuna tramite sequenza diagram.



In questo modo è possibile avere una visione di insieme del processo di registrazione. Come si può notare i nomi dei processi sono stati scelti in modo da poter essere auto esplicativi nella comprensione di tale meccanismo. Unica nota che occorre fare è relativa agli ultimi step del processo.

## Mail Service

Come si può ben notare negli ultimi passi del processo di registrazione viene fatto riferimento a due classi particolari, *EmailSender* e *JavaMailSender*, vediamole nel dettaglio. Tali classi forniscono l'implementazione Java di servizi di invio messaggi via mail, nel file *pom.xml* troviamo:

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

Le interfacce e le classi per tale supporto sono organizzate in questo modo:

- **EmailSender interface:** è l'interfaccia di primo livello che integra le funzionalità base per l'invio di emails.
- **JavaMailSender interface:** è la subinterface di *MailSender*. Supporta MIME messages ed è usata in congiunzione con la classe *MimeMessageHelper* per la creazione di un *MimeMessage*.
- **MailSender class** è l'implementazione dell'interfaccia *JavaMailSender*.
- **SimpleMailMessage class:** viene utilizzata per creare email usuali, con campi di to, from, cc, oggetto e testo.
- **MimeMessageHelper class:** è una classe di supporto per la creazione di MIME messages, in particolar modo per l'inserimento di immagini come allegato e per l'inserimento di testo in formato html.

Come accennato precedentemente, Spring Boot permette la configurazione di tali dependencies in un apposito file di properties chiamato *application.properties*, in esso troviamo:

```
# For Google Mail Server
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=residenza.elis.bacheca@gmail.com
spring.mail.password=elis1928
spring.mail.properties.mail.smtp.ssl.trust=*
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
spring.mail.smtp.ssl.protocols=TLSv1.2
```

Quelle riportate sono le properties per configurare il server SMTP Gmail. Quello che resta da fare è dunque fornire l'implementazione dell'interfaccia *EmailSender*:

```
@EnableAsync
@Service
public class EmailService implements EmailSender {

    // Setup Logger for logging email sending operation
    private final static Logger LOGGER =
        LoggerFactory.getLogger(EmailService.class);

    // List all service to use
    private final JavaMailSender javaMailsender;
```

```

public EmailService(JavaMailSender javaMailsender) {
    super();
    this.javaMailsender = javaMailsender;
}

```

E effettuare l'override del metodo che conterrà la composizione del messaggio a nostro piacimento:

```

// Send Email
@Override
@Async
public void send(String to, String email) {

    if (to == null || email == null || to.isBlank() || email.isBlank()) {
        throw new IllegalArgumentException();
    }

    try {
        // Util class to setup mail sender
        MimeMessage mimeMessage = javaMailsender.createMimeMessage();
        MimeMessageHelper helper = new MimeMessageHelper(mimeMessage, "utf-8");

        // Fill the mail
        helper.setText(email, true);
        helper.setTo(to);
        helper.setSubject("Bacheca Unicollage");
        helper.setFrom("residenza.elis.bacheca@gmail.com");

        // Send it
        javaMailsender.send(mimeMessage);

    } catch (Exception e) {

        // Else logged the err and then thrown a new IllegalStateException
        LOGGER.error("Failed to send the email", e);
        throw new IllegalStateException("Failed to send the email");
    }
}

```

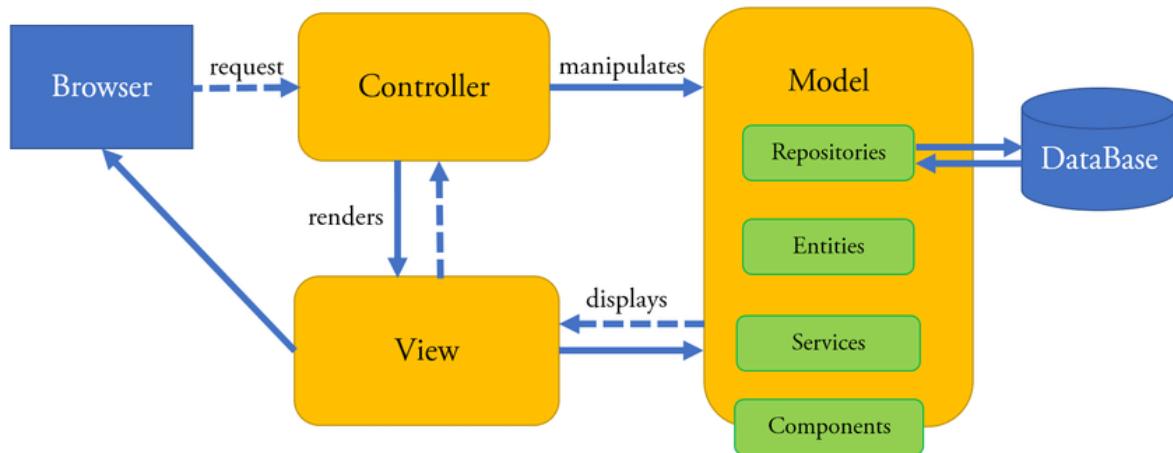
Da notare che il metodo è stato etichettato con l'annotation @Async, in questo modo l'invio delle mail non rallenterà la web app e dunque la user experience.

## Web

### Spring MVC

Spring MVC è il framework di Spring per lo sviluppo di web app. Proprio come suggerisce il suo nome segue il design pattern MVC e fa uso del *DispatcherServlet*. Il *DispatcherServlet* è una classe che agisce da front controller nell'architettura web, essa riceve le richieste in entrata, converte i payload delle richieste con le rappresentazioni dei dati interne alla web app, manda le

rappresentazioni dei dati ai model per successivi riprocessamenti e infine effettua il rendering delle view con i dati processati.



Più precisamente:

- Il *model* layer si occupa delle strutture dati e della business logic della web app implementata nelle entities, nei repository e nei services.
- Il *controller* layer si occupa di legare dati e template in modo che possano essere poi elaborati dal *view resolver*.
- Le *view* forniscono la rappresentazione grafica finale dei dati attraverso cui lo user può interagire.

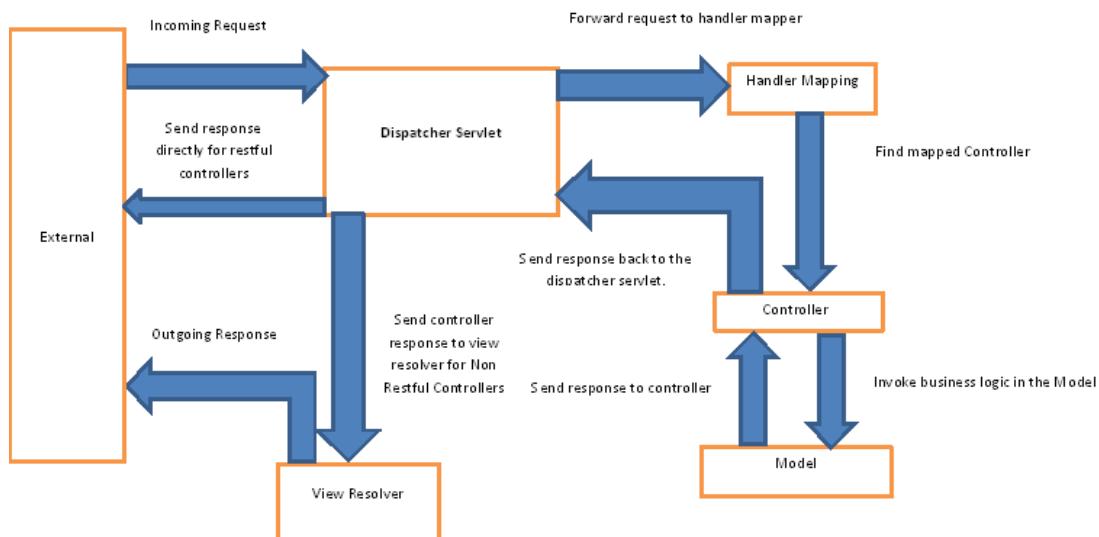


Fig 1 MVC Architecture flow

Ancora una volta, tale integrazione è possibile grazie alla seguente Spring Boot dependencies, esplicitata all'interno del file *pom.xml*:

```

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

Le uniche configurazioni effettuate all'interno del file *application.properties* sono le seguenti:

```
#Configuration for conversion of date and time
spring.mvc.format.date=yyyy-MM-dd
spring.mvc.format.time=HH:mm:ss
```

In questo modo la web app utilizza una rappresentazione univoca globale di conversione delle date e delle ore, evitando così problemi di compatibilità.

Un altro settaggio fine effettuato è il seguente:

```
@Configuration
public class MvcConfig implements WebMvcConfigurer {

    // Configuration to expose filesystem directory where to store the imgs of
    the
    // activity
    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry) {

        // reference dir name
        registry.addResourceHandler("/activity-photos/**")
            // real path dir reference
            .addResourceLocations("file:/tmp/activity-photos/");

    }

}
```

E' stata inserita una directory nel file system dove effettuare lo store delle immagini della bacheca ed è stata registrata come custom registry resource location. In questo modo Spring MVC può caricare da questa location le risorse e comporre correttamente le views. Notasi che la classe è stata etichettata con l'annotation `@Configuration`, in questo modo Spring Boot è in grado di riconoscere la configurazione ed effettuarne il bind con l'env. globale. Tale configurazione poteva essere fatta anche tramite file `application.properties` ma per maggiore chiarezza si è preferito procedere in maniera classica.

A questo punto possiamo passare al layer Controller, esso viene sviluppato all'interno del package `com.vincenzomariacalandra.provaFinale.BachecaUniCollege.controller`. Ogni classe al suo interno è una classe controller che mappa un apposito template contenuto all'interno della cartella `/BachecaUniCollege/src/main/resources/templates`. Per maggior rigore si è scelto chiamare di chiamare ogni template con lo stesso nome della corrispettiva classe controller.

Ogni classe controller ha dunque la seguente struttura:

```
@Controller
@RequestMapping("/homeBacheca")
public class HomeBachecaController {

    // All Services required
    private final ActivityService activityService;

    @Autowired
    public HomeBachecaController(ActivityService activityService) {
        this.activityService = activityService;
    }

    // Initialization of homepage
```

```

@GetMapping
public String listAllActivities(Model model) {

    //Add list attribute to the model
    ArrayList<Activity> list = new ArrayList<>();

    activityService.getActivitiesApproved().iterator().forEachRemaining(list::add);
    model.addAttribute("activities", list);

    return "homeBacheca";
}

}

```

- L'annotation `@Controller` serve a mappare la classe all'interno di Spring come custom controller.
- L'annotation `@RequestMapping` specifica l'endpoint da mappare.
- L'annotation `@GetMapping` specifica il metodo da chiamare quando viene effettuata una GET.
- L'annotation `@PostMapping` specifica il metodo da chiamare quando viene effettuata una POST.
- In alternativa alle annotation `@Get..` e `@Post` è possibile usare `@RequestMapping(path = "/path", method = RequestMethod.GET)` prima di ogni metodo.
- Ogni metodo del controller ritorna sempre una stringa che rappresenta il template html corrispondente o un altro endpoint.
- E' possibile specificare che un endpoint richiede dei parametri nella URL tramite `@RequestParam("")`

```

// Update Activity handler
@RequestMapping(path = "/updateTertulia", method = RequestMethod.GET)
public String updateShowFormTertuliaATema(@RequestParam("id") Long id, ...)

```

- E' possibile specificare degli attributi di modello tramite l'annotation `@ModelAttribute`

```

// Update activity post handler
@RequestMapping(path = "/updateTertulia", method = RequestMethod.POST)
public String updateTertuliaATema(@ModelAttribute Activity activity, ... )

```

Inoltre tutti gli endpoint sono stati documentati meglio all'interno della relative classi.

## Templating

Come motore di templating dinamico delle view è stato scelto **Thymeleaf**. Il punto di forza principale di Thymeleaf è quello di usare un linguaggio di template naturale.

Con le dependencies per SpringBoot sono possibili diverse integrazioni, Thymeleaf risulta ideale per il moderno sviluppo web HTML5 JVM e in particolar modo per essere integrato con *Spring MVC*.

```

<!-- Framework to template dinamically html files -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>

<dependency>
    <groupId>org.thymeleaf.extras</groupId>
    <artifactId>thymeleaf-extras-springsecurity5</artifactId>
</dependency>

```

Le uniche configurazioni apportate sono di utilities e non introducono sostanziali implementazioni, inoltre sono auto esplicative:

```

# Thymeleaf settings for templating HTML files
spring.thymeleaf.enabled=true
spring.thymeleaf.cache=false
spring.thymeleaf.prefix=classpath:/templates/
spring.thymeleaf.suffix=.html

```

Per attivare l'integrazione occore importare i namespace nel seguente modo:

```

<!doctype html>
<html lang="en">
    <ns:th="http://www.thymeleaf.org"
     ns:sec="https://www.thymeleaf.org/thymeleaf-extras-springsecurity5">
<head>
    ...

```

Tra le integrazioni più importanti vi è quella della security. Tali dependencies introducono gli *Spring Security Dialect* che ci permettono di mostrare i contenuti all'interno della view in base al ROLE dell'utente, ai suoi permessi o ad altre espressioni di sicurezza specifiche di Spring.

Un banale utilizzo che ne è stato fatto nel progetto è il seguente:

```

<li th:if="#{authentication.getPrincipal().isDirettore()}" class=" nav-item"><a
class="nav-link" href="/gestioneAttività">Gestione Attività</a></li>

```

- Il tag *th:if* permette l'introduzione di una condizione da valutare prima di renderizzare il <div>.
- La stringa "\${...}" comunica al tag che il contenuto della stringa è una variabile.
- L'oggetto identificato con *#authentication* viene introdotto con *thymeleaf-extras-springsecurity5* e rappresenta lo Spring Security authentication object.

Altri tag utilizzati sono stati:

- *th:text=""* permette la sostituzione del testo "Example..." con quello contenuto all'interno della variabile "\${msg}".

```

<p th:text="${msg}">Example text for confirmation page</p>

```

- *th:href=""* permette l'inserimento di hyperlink e @ {...} suggerisce che la stringa venga valutata come URL

```
<a th:href="@{/dettaglioAttività?id=${activity.id}}" class="btn btn-primary">Dettaglio Attività</a>
```

Per una totale comprensione dei tag introdotti si rimanda alla documentazione ufficiale al seguente [link](#) e alla repository di [thymeleaf-extras-springsecurity5](#).

## Salvataggio immagini

Come già visto in precedenza si è provvisto a creare un apposita configurazione per permettere alla classe *NuovaAttivitàController* di salvare i file img in un apposita directory nel file system e successivamente permettere così alle view di poterle renderizzare.

Il processo di salvataggio delle immagini avviene in 3 fasi:

- Caricamento del file tramite apposito field nel form di creazione attività.

The screenshot shows a web form titled "Nuova Attività". At the top, there are navigation links: "Bacheca" (highlighted in blue), "Crea Attività" (highlighted in red), "Gestione Attività", "Home Direttore", and "UniCollege Bacheca". On the right is a "Logout" button. Below the title, there are tabs: "Attività Generica", "Tertulia a tema", and "Libro". The main form area has several input fields:  
 - "Titolo Attività": An input field containing "Titolo Attività".  
 - "Descrizione Attività": A large text area containing "Breve Descrizione".  
 - "Data inizio": An input field with placeholder "gg / mm / aaaa".  
 - "Data fine": An input field with placeholder "gg / mm / aaaa".  
 - "Ora inizio": An input field with placeholder "-- : --".  
 - "Ora fine": An input field with placeholder "-- : --".  
 - "Durata": A dropdown menu showing "0,2" and "VOLONTARIATO".  
 - "N. max di partecipanti": An input field with a dropdown arrow.  
 - "Foto per bacheca": A file input field showing "Sfoglia..." and "Nessun file selezionato.". A preview image of a person's face is displayed next to the input field.  
 At the bottom is a blue "Invia" button.

```
<form action="#" th:action="@{/nuovaAttività}" th:object="${activity}"
method="post"
enctype="multipart/form-data">
...
<div class="form-group">
    <label for="fileImage">Foto per bacheca</label>
    <input class="form-control" id="fileImage" name="fileImage" type="file"
accept="image/png, image/jpeg" required></input>
</div>
...
```

Viene specificato che l'encoding sarà di tipo *multipart* e *form-data* e che il file caricato dovrà essere con estensione *png* o *jpg*.

- Il file viene recepito dal'apposito controller che procede alla validazione dell'activity.

```
@RequestMapping(path = "/nuovaAttivita", method = RequestMethod.POST)
public String addActivity (@Valid @ModelAttribute("activity") Activity activity,
@RequestParam(name = "fileImage", required = false) MultipartFile multipartFile,
Model model, HttpServletRequest request, RedirectAttributes redirectAttributes)
throws IOException {

if (multipartFile != null) {

    //Error checking business constraints
    String err = processing.multipartProcessing(multipartFile, activity);
    ...
}}
```

- *multipartProcessing(...)* è un metodo di utility che gestisce il salvataggio delle imgs nel file system e viene implementato nella classe *MultipartProcessingUtility*, vediamolo nel dettaglio.

```
@Component
public class MultipartProcessingUtility {

    // All services required
    private final ActivityService activityService;

    private static String ROOT_UPLOAD_DIR = "/tmp/activity-photos/";

    @Autowired
    public MultipartProcessingUtility(ActivityService activityService) {
        this.activityService = activityService;
    }

    public String multipartProcessing(MultipartFile multipartFile, Activity activity) throws IOException {

        // Generation of filename
        String fileName =
StringUtils.cleanPath(multipartFile.getOriginalFilename());

        // Insertion of filename in Activity entity
        activity.setPhoto(fileName);

        // Saving Activity in DB
        String err = activityService.addNewActivity(activity);

        if (err != null) {

            System.out.println(err);

            return err;
        }

        // Checks if ROOT_UPLOAD_DIR exist
        if (!Files.exists(Paths.get(ROOT_UPLOAD_DIR))) {
            Files.createDirectory(Paths.get(ROOT_UPLOAD_DIR));
        }
    }
}
```

```

// Generation of path to the directory where to store the photo
String uploadDir = ROOT_UPLOAD_DIR + activity.getId();

Path uploadPath = Paths.get(uploadDir);

// Checks if folder exist
if (!Files.exists(uploadPath)) {
    Files.createDirectory(uploadPath);
}

// Saving file in the correct dir
try (InputStream inputStream = multipartFile.getInputStream()) {

    Path filePath = uploadPath.resolve(fileName);

    Files.copy(inputStream, filePath,
StandardCopyOption.REPLACE_EXISTING);

} catch (IOException ioe) {
    throw new IOException("Could not save the file!");
}

return null;
}
}

```

- In primis viene generato il nome del file a partire dall'originale.
- A questo punto viene salvato il nome del file nella entity che rappresenta tale activity.
- Viene controllata se la root dir dove varranno salvate le immagini esiste già e viene creata se non esiste.
- Viene generato il path della dir dove effettuare lo store del file img e viene creata la dir se non esiste.
- A questo punto viene scritto il file img all'interno della dir aprendo uno stream dati.

Le configurazione apportate per la gestione delle richieste *HTTP multipart* sono le seguenti:

```

#MultiPartFile Configuration to store Images
spring.servlet.multipart.max-file-size=50MB
spring.servlet.multipart.max-request-size=50MB
spring.servlet.multipart.enabled=true

```

## Testing

Il testing è stato un punto chiave nello sviluppo della web app, infatti si è evoluto con essa. In origine, quando ancora l'interfaccia grafica non era stata sviluppata e l'app comunica con l'esterno tramite un RESTController, la metologia di testing principale è stata relegata all'utilizzo di software come Postman.

Postman è una piattaforma API per la costruzione e l'utilizzo di API. Postman semplifica ogni fase del ciclo di vita delle API e snellisce la collaborazione in modo da poter creare API migliori, più velocemente.

The screenshot shows the Postman interface. On the left sidebar, there's a tree view of collections, APIs, environments, mock servers, monitors, flows, and history. The main workspace shows a POST request to 'http://localhost:8080/api/v1/activity'. The request body is set to 'JSON' and contains the following JSON:

```

4     "startDate": "2021-10-07",
5     "endDate": "2021-11-11",
6     "startTime": "09:23:58",
7     "endTime": "10:10:00",
8     "userActivities": [],
9     "activityType": "VOLONTARIATO",
10    "activityCredits": "TWO"
11

```

The response area has a cartoon character and a button labeled 'Click Send to get a response'.

In questo modo si è potuto subito constatare la presenza di errori nello sviluppo della logica di business.

Congiuntamente a quanto riportato sopra si è proceduto allo sviluppo di apposite unità di testing per le singole classi che compongono la logica di business, tutte contenute all'interno del package `com.vincenzomariacalandra.provaFinale.BachecaUniCollege.service`. Tali unità di testing fanno uso di `JUnit5` e `Mockito` piuttosto che utilizzare il supporto nativo offerto da `SpringBoot starter test`. Tale scelta è motivata dal fatto che `SpringBoot` appesantisce di molto l'elaborazione degli unit test da pochi millisecondi a decine di secondi ostacolando così il flusso **test -> sviluppo -> test** promosso dalla logica ***Test Driven Development, TDD***.

Per fare ciò è stato necessario effettuare un'esclusione di particolari librerie e import esterne di altre, di seguito quanto detto:

```

<!-- For Testing purpose -->
<!-- exclude junit 4 and Mockito-->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
    <exclusions>
        <exclusion>
            <groupId>org.junit</groupId>
            <artifactId>junit</artifactId>
        </exclusion>
        <exclusion>
            <groupId>org.mockito</groupId>
            <artifactId>mockito-core</artifactId>
        </exclusion>
        <exclusion>
            <groupId>org.mockito</groupId>
            <artifactId>mockito-all</artifactId>
        </exclusion>
    </exclusions>
</dependency>

<!-- junit 5 -->
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-engine</artifactId>
    <version>${junit-jupiter.version}</version>

```

```

<scope>test</scope>
</dependency>

<!-- Mockito dependency -->
<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-core</artifactId>
    <version>2.21.0</version>
</dependency>

<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-junit-jupiter</artifactId>
    <version>2.23.0</version>
    <scope>test</scope>
</dependency>

<!-- Old vintage -->
<dependency>
    <groupId>org.junit.platform</groupId>
    <artifactId>junit-platform-runner</artifactId>
    <version>1.2.0</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.junit.vintage</groupId>
    <artifactId>junit-vintage-engine</artifactId>
    <version>5.2.0</version>
    <scope>test</scope>
</dependency>

```

Le classi di unit test hanno tutte una struttura così fatta:

```

@Extendwith(MockitoExtension.class)
public class ActivityServiceImplUnitTest {

    @Mock
    private ActivityRepository activityRepository;

    @Mock
    private UserService userService;

    @Mock
    private EmailSender emailSender;

    private ActivityService service;

    @BeforeEach
    public void setUp() {

        service = new ActivityService(activityRepository, emailSender,
userService);

    }

    @Test
    public void contextLoads() throws Exception {

```

```
    assertNotNull(service);
}
...
```

Il nome della classe fa riferimento alla classe "bersaglio" dello unit test.

La classe è etichettata con l'annotation `@ExtendWith(MockitoExtension.class)`, utilizzata per abilitare Mockito, e opzionalmente con l'annotation `@RunWith(JUnitPlatform.class)` utilizzata per runnare la classe di unit test singolarmente.

Ogni attributo della classe di test etichettato con l'annotation `@Mock` verrà fatto diventare una classe fantoccio che restituirà `null` ogni volta che verrà chiamato un suo metodo dalla classe che subisce il test salvo particolari casi in cui chi scrive il test richiede che tale comportamento venga sovrascritto da un altro.

Il metodo etichetta con l'annotation `@BeforeEach` verrà chiamato prima di ogni metodo etichettato con l'annotation `@Test`, quest'ultimi contengono appunto i test che verranno svolti sul metodo della classe bersaglio.

```
@Test
public void deleteActivityTest() {

    Random rand = new Random();
    Long id = rand.nextLong();

    Activity activity = new Activity();
    Optional<Activity> optional = optional.of(activity);

    reset(activityRepository);

    lenient().when(activityRepository.findById(id)).thenReturn(optional);

    assertNull(service.deleteActivity(id));
    verify(activityRepository, VerificationModeFactory.times(1)).findById(id);

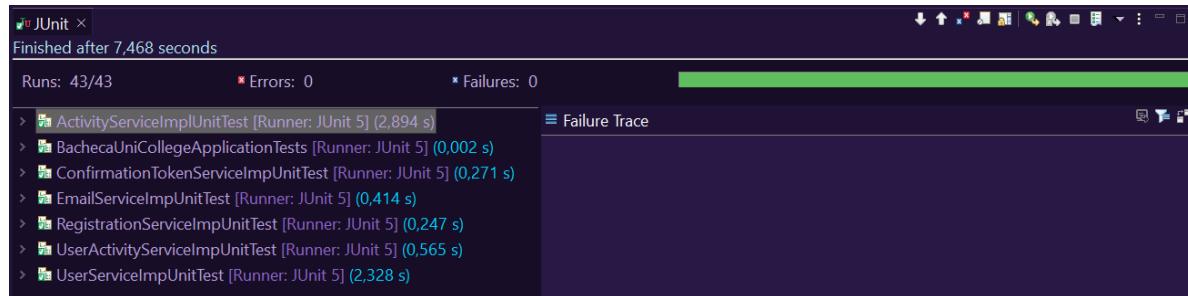
    reset(activityRepository);
    lenient().when(activityRepository.findById(null)).thenReturn(optional);
    assertEquals("Activity id could not be null!",
        service.deleteActivity(null));
}
```

I punti salienti sono i seguenti:

- Il metodo `reset(activityRepository)` si occupa di resettare il mock da eventuali override di comportamenti precedenti e di azzerarne il numero di istanze.
- Il metodo `lenient().when(activityRepository.findById(id)).thenReturn(optional);` si occupa di fare appunto l'override del comportamento del metodo `findById(...)` della classe fantoccio.
- Infine, ma non per ultimo, il metodo `assertNull(service.deleteActivity(id));` si occupa di effettuare il test sul metodo della classe bersaglio, in questo caso verifica se ritorna valore nullo.
- Il metodo `verify(activityRepository, VerificationModeFactory.times(1)).findById(id);` verifica che il metodo sia stato chiamato correttamente.

I risultati ottenuti grazie a questa strategia di testing sono valutabili in termini di velocità di verifica e destrezza, infatti hanno permesso di velocizzare lo sviluppo e la risoluzione di errori prima che l'utente finale possa averli potuti sperimentare durante l'utilizzo del prodotto causando così disagio e insoddisfazione.

Di seguito i risultati ottenuti:



The screenshot shows a JUnit test results window. At the top, it says "JUnit" and "Finished after 7,468 seconds". Below that, it displays the following statistics: "Runs: 43/43", "Errors: 0", and "Failures: 0". A green progress bar is almost fully filled. On the left side, there is a list of test classes and their execution times: ActivityServiceImplUnitTest [Runner: JUnit 5] (2,894 s), BachecaUniCollegeApplicationTests [Runner: JUnit 5] (0.002 s), ConfirmationTokenServiceImplUnitTest [Runner: JUnit 5] (0.271 s), EmailServiceImplUnitTest [Runner: JUnit 5] (0.414 s), RegistrationServiceImplUnitTest [Runner: JUnit 5] (0.247 s), UserActivityServiceImplUnitTest [Runner: JUnit 5] (0.565 s), and UserServiceImplUnitTest [Runner: JUnit 5] (2,328 s). To the right of the list is a "Failure Trace" section which is currently empty.

Proseguendo, implementando l'interfaccia grafica è stato possibile effettuare i primi test di **workflow**, ovvero:

- Testare il flusso di lavoro end-to-end che portano l'utente attraverso una serie di pagine web da completare.
- Testare gli scenari negativi, in modo che quando un utente esegue un passo inaspettato nella vostra web app venga mostrato un messaggio di errore o un aiuto appropriato.

La conclusione di tale test ha portato alla conclusione che il prodotto ottenuto rispecchia le richieste dei futuri utilizzatori.

Si è poi passati ai test di **usabilità**. I test di usabilità sono una parte vitale di qualsiasi progetto web. Può essere effettuato da testers o da un piccolo gruppo di utenti facenti parte del target finale dell'applicazione web.

- Testata la navigazione del sito: Menu, pulsanti o link a diverse pagine del tuo sito dovrebbero essere facilmente visibili e coerenti su tutte le pagine web.
- Testato il contenuto: Il contenuto dovrebbe essere leggibile senza errori di ortografia o grammatica. Le immagini, se presenti, dovrebbero contenere un testo "alt".

Il risultato di tale test, effettuato su 1/3 della popolazione finale che utilizzerà l'app, ha permesso la risoluzione di svariati problemi inosservabili all'occhio dello sviluppatore emigliorando così la qualità del prodotto finale.

Continuando, si è passati ai test di **compatibilità**. I test di compatibilità assicurano che la web app venga visualizzata correttamente su diversi dispositivi. Questo include:

- Test di compatibilità del browser: la stessa pagina in diversi browser venga visualizzata in ugual modo.
- Test di compatibilità con i browser mobili. Il rendering del layout è differente da dispositivo a dispositivo.

Di seguito i risultati ottenuti:

- Explorer

http://51.68.138.245/homePage

Home Page

Bacheca Crea Attività Home Tutor Vincenzo Maria Calandra Logout



Gita in montagna 

Tipo: VISITA CULTURALE

Crediti: 0.5

Giorno: 2021-12-12 - 2021-12-12

Ora: 09:30-18:30

[Dettaglio Attività](#)



Mensa Caritas

Tipo: VOLONTARIATO

Crediti: 0.3

Giorno: 2021-12-19 - 2021-12-19

Ora: 09:30-15:30

[Dettaglio Attività](#)

- Edge

Home Page

Not secure | 51.68.138.245/homePage

Bacheca Crea Attività Home Tutor Vincenzo Maria Calandra Logout



Gita in montagna 

Tipo: VISITA CULTURALE

Crediti: 0.5

Giorno: 2021-12-12 - 2021-12-12

Ora: 09:30-18:30

[Dettaglio Attività](#)



Mensa Caritas

Tipo: VOLONTARIATO

Crediti: 0.3

Giorno: 2021-12-19 - 2021-12-19

Ora: 09:30-15:30

[Dettaglio Attività](#)

- Firefox

Home Page

51.68.138.245/homePage

Bacheca Crea Attività Home Tutor Vincenzo Maria Calandra Logout



Gita in montagna 

Tipo: VISITA CULTURALE

Crediti: 0.5

Giorno: 2021-12-12 - 2021-12-12

Ora: 09:30-18:30

[Dettaglio Attività](#)



Mensa Caritas

Tipo: VOLONTARIATO

Crediti: 0.3

Giorno: 2021-12-19 - 2021-12-19

Ora: 09:30-15:30

[Dettaglio Attività](#)

- Android - Browser Opera

**Bacheca**[Crea Attività](#)[Gestione Attività](#)[Home Direttore](#)[UniCollege Bacheca](#)[Logout](#)**Gita in montagna** Tipo: **VISITA CULTURALE**Crediti: **0.5**Giorno: **2021-12-12 - 2021-12-12**Ora: **09:30-15:30**[Dettaglio Attività](#)

I risultati sono in linea rispetto a quanto ci si aspettava, minime differenze tra i browser sono dovute alle differenti condizioni di zoom di default.

Infine sono stati effettuati dei test sull'infrastruttura cloud per testarne la resistenza minima ad attacchi di tipo *DDoS*. Il cloud provider, *OVH*, ha immediatamente provvisto a deviare il traffico sull'infrastruttura di mitigazione notificando l'utente dell'avvenuto attacco.

[cv240305-ovh] rilevato attacco sull'IP 51.68.138.245 [Posta in arrivo](#)



**supporto@ovh.it**  
a me ▾

\*\*\*Questa è una email automatica. Se hai bisogno di maggiori informazioni apri un ticket dallo Spazio Cliente\*\*\*

Gentile Cliente,

abbiamo rilevato un attacco sull'IP 51.68.138.245.

Per proteggere la tua infrastruttura, abbiamo aspirato il tuo traffico sulla nostra infrastruttura di mitigazione

Tutto l'attacco sarà filtrato dalla nostra infrastruttura e solo il traffico valido arriverà fino ai tuoi server.

Al termine dell'attacco, la mitigazione della tua infrastruttura terminerà.

Per maggiori informazioni sull'infrastruttura di protezione anti-DDoS: <http://www.ovh.it/anti-ddos/>

Il team OVHcloud

Hai bisogno di aiuto? Accedi al Centro di assistenza: <https://help.ovhcloud.com/>  
Troverai le nostre Guide, FAQ, la Community e le informazioni sullo stato del sistema.

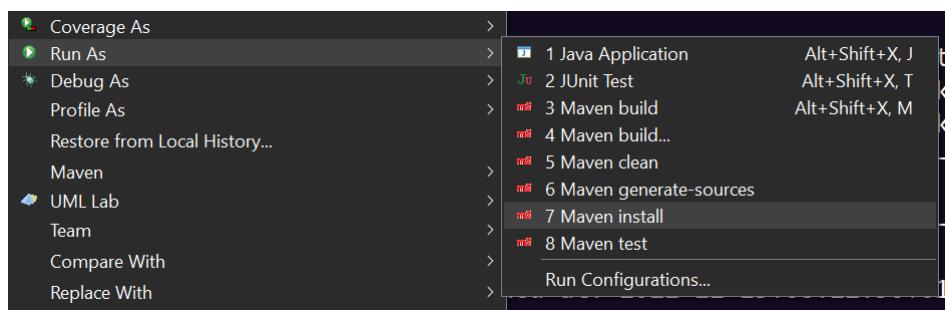
OVH Srl è una filiale della società OVH Groupe SAS, immatricolata al RCS di Lille con il numero 537 407 926 e con sede in rue Kellermann 2, a Roubaix, 59100.  
[ref=1.089a6e9e]

## Installazione e Deploy

### Maven

Per prima cosa occorre effettuare la generazione del file *.jar*, per fare ciò:

- E' possibile aprire il progetto con un IDE che abbia l'estensione per *Maven* installa, fare tasto destro sul progetto e cliccare su *maven install*.



- Eseguire direttamente il comando da terminale all'interno della root directory del progetto.

```
mvn install
```

Verrà generato il file *.jar* all'interno della cartella *target*.

### Docker

Come già accennato in precedenza è stato scelto di utilizzare la conteinerizzazione come metologia di deploy. Per fare ciò è stato definito un Dockerfile contenente tutte le informazioni necessarie per costruire la nostra immagine.

```
FROM openjdk:11
COPY target/BachecaUnicollge-0.0.1-SNAPSHOT.jar BachecaUnicollge.jar
EXPOSE 8080
VOLUME ["/tmp/activity-photos", "/var/log"]
ENTRYPOINT ["java","-jar","/BachecaUnicollge.jar"]
```

Vediamolo nel dettaglio:

- Il primo comando invoca il download dell'immagine *openjdk:11*, tale immagine è predisposta per ospitare applicativi JAVA11.
- Il secondo comando copia il file *.jar* generato all'interno del container.
- Il terzo comando espone la porta 8080 del container.
- Il quarto comando dichiara i volumi che verranno appesi al container.
- Il quinto comando contiene la sequenza dei comandi da eseguire per avviare il file *.jar*.

A questo punto non ci resta che scrivere il *docker-compose.yaml*:

```
version: '3'
services:
  bacheaca-uni-college:
    build:
      context: .
      dockerfile: Dockerfile
    image: bacheaca-uni-college:latest
    volumes:
      - BacheacaUniCollege-data:/tmp/activity-photos
      - BacheacaUniCollege-log:/var/log
    ports:
      - "80:8080"
    networks:
      - bacheaca-uni-college-mysqlDb-network
    depends_on:
      - mysqlDb

  mysqlDb:
    image: mysql:8
    networks:
      - bacheaca-uni-college-mysqlDb-network
    environment:
      - MYSQL_ROOT_PASSWORD=root
      - MYSQL_DATABASE=database

volumes:
  BacheacaUniCollege-data:
    driver: local
    driver_opts:
      o: bind
      type: none
      device: /tmp
  BacheacaUniCollege-log:
    driver: local
    driver_opts:
      o: bind
      type: none
      device: /var/log/BacheacaUniCollege

networks:
  bacheaca-uni-college-mysqlDb-network: null
```

Come possiamo ben vedere abbiamo 3 layers di definizione:

- Il primo layer *services* contiene i servizi veri e propri della nostra web app, ovvero il nostro applicativo SpringBoot e il database MySQL.

```

o  bacheaca-uni-college:
   build:
     context: .
     dockerfile: Dockerfile
     image: bacheaca-uni-college:latest
     volumes:
       - BachecaUniCollege-data:/tmp/activity-photos
       - BachecaUniCollege-log:/var/log
     ports:
       - "80:8080"
     networks:
       - bacheaca-uni-college-mysql1db-network
   depends_on:
     - mysql1db

```

Soffermandosi brevemente su tale definizione, sottolineamo alcune parti salienti:

- *build*: . indica che l'immagine verrà creata con il Dockerfile.
- *volumes*: dichiara che le dirs elencate sarranno legate ai volumi dichiarati sotto.
- *ports*: 80:8080 mappa la porta 80 del server con la porta 8080 del container in modo che sia raggiungibile dall'esterno.
- *networks*: contiene una lista di reti attraverso cui il container potrà comunicare.
- *depends\_on*: dichiara che tale service dipende dal *mysql1db*, dunque prima di essere avviato dovrà attendere le tutte le dipendenze siano risolte.
- Il secondo layer contiene i volumi che verranno condivisi con il container che contiene la SpringBoot app, in particolare:
  - *BachecaUniCollege-data* conterrà le immagini salvate nel filesystem.
  - *BachecaUniCollege-log* conterrà le informazioni di log generate dalla web app e dal container. **Occorre che la cartella /var/log/BachecaUniCollege esista già, altrimenti l'esecuzione fallira.**
- Il terzo layer contiene la rete che conterrà i 2 services dichiarati sopra.

A questo punto non resta che effettuare il build e il run:

```
docker-compose up --build
```

Se si vuole che il processo venga eseguito come demone:

```
docker-compose up -d --build
```

Per stoppare il tutto:

```
docker-compose down
```

## Logging e Monitoring

### Logging

Per quanto riguarda il logging Spring Boot Starter integra *Apache Commons Logging* e *Logback*, inoltre implementa un interfaccia che standardizza il loro utilizzo, *slf4j* o *Simple Logging Facade for Java*.

Un esempio di utilizzo lo troviamo nell'*EmailService*:

```

@EnableAsync
@Service
public class EmailService implements EmailSender {

    // Setup Logger for logging email sending operation
    private final static Logger LOGGER =
LoggerFactory.getLogger(EmailService.class);
    ...

    // Send Email
    @Override
    @Async
    public void send(String to, String email) {

        try {
            // Util class to setup mail sender
            ...

        } catch (Exception e) {

            // Else logged the err and then thrown a new IllegalStateException
            LOGGER.error("Failed to send the email", e);
            throw new IllegalStateException("Failed to send the email");
        }
    }
    ...
}

```

- Il logger viene instanziato con riferimento alla classe da monitorare con l'aiuto del *LoggerFactory*.
- Una volta instanziato è possibile selezionare il tipo di messaggio di logging, es. *Error*, assegnare un tag e un messaggio.

## Monitoring

Per quanto riguarda il monitoring è stato integrato *Spring Actuator*:

```

<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
</dependencies>

```

Spring Boot include una serie di funzioni aggiuntive per monitorare e gestire quando la web app viene messa in produzione. E' possibile monitorare l'applicazione usando gli endpoint HTTP alberati sotto */actuator/\*\**.

La configurazione di seguito elenca espone tutti gli endpoint di Spring Actuator tranne quello per lo shutdown dell'applicativo:

```

# Configuration for Spring Actuator tool
management.endpoints.web.exposure.include=*
# if you'd like to expose shutdown:
# management.endpoint.shutdown.enabled=true

```

Per esempio un endpoint molto utile è `/actuatorlogfile` che ritorna il contenuto del file di log.

A proposito del file di log, è stato specificato un file di logging per la web app in un apposita cartella:

```
# Logging configuration
logging.file.name=/var/log/BachecaUniCollege.log
server.tomcat.accesslog.enabled=true
```

Tale cartella è un volume denominato `BachecaUniCollege-log` e montato sul container docker. Tale volume punta al filesystem del server che lo ospita, in questo modo è sempre accessibile, vedi [qui](#).

## Class Diagram

Di seguito viene riportato il class diagram del progetto per avere una visione complessiva di esso:

