# Documentazione "Bacheca UniCollege"

## Prova finale Ingegneria del Software

**Autore: Calandra Vincenzo Maria**

# 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 determianta 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 assicurasi 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 resisdenti 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 precedentemenenti approvati dai tutor della residenza per il conseguitmento 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”.

# 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 ogetti 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 comunity 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 collabborare tra loro permettendo così allo sviluppatore di poter concentrare i propri sforzi non sulla tecnologia ma sulla logica di busisness che dovrà animare l'appliccativo 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.

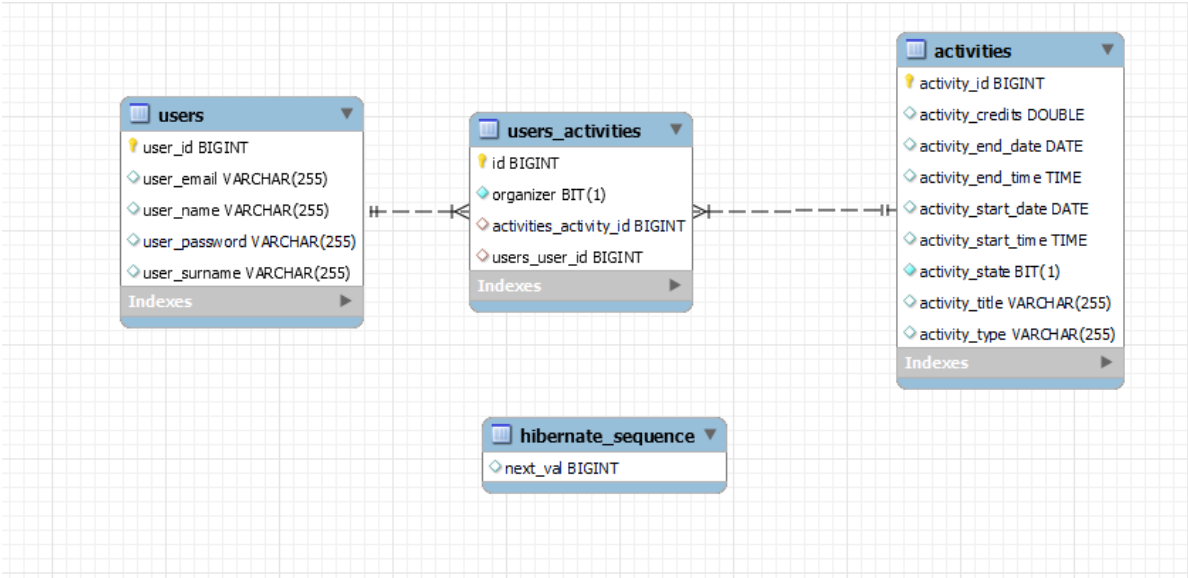
Per quanto riguarda la gestione del processo di build del progetto è stato scelto Maven. Maven è un framework dichiarativo di gestione del progetto che segue specifiche convenzioni per quanto riguarda la struttura dello stesso. Maven fa il build di un progetto usando il suo Project Object Model (POM) 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,

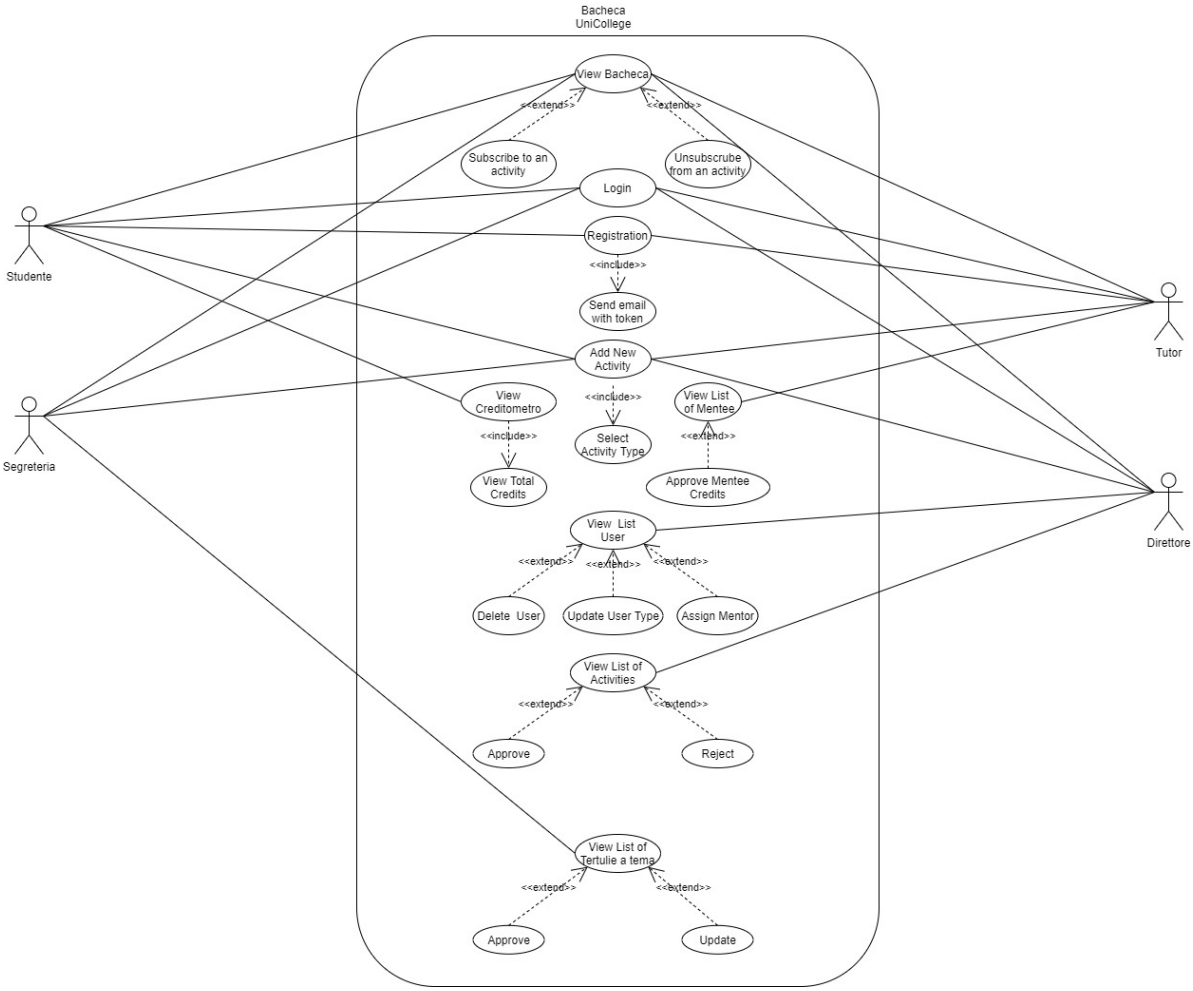
nonchè la più famosa e quella su cui si è poi basato lo standard di mercato,"containerd" della CNCF, è Docker. Docker Engine ha un piano di utilizzo gratuito ad uso personale o commerciale regolato dalla licenza Apache 2.0.

# Base di dati

Di seguito il database schema e il diagramma ER.




# Use Case



# Dettaglio implementazioni tecnologiche

## Documentazione gestione Autenticazione e Registrazione

When working with Spring Boot, the *spring-boot-starter-security* starter will automatically include all dependencies, such as *spring-security-core*, *spring-security-web*, and *spring-security-config* among others:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
    <version>2.3.3.RELEASE</version>
</dependency>
```

The Spring Security configuration class extends *WebSecurityConfigurerAdapter.*

By adding *@EnableWebSecurity*, we get Spring Security and MVC integration support:

```
@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;
    }
```

**Starting with Spring 5, we also have to define a password encoder**. In our case, we'll use the *BCryptPasswordEncoder* defined in the *PasswordEncoder* class:

```
@Configuration
public class PasswordEncoder {

    @Bean
    public BCryptPasswordEncoder bCryptPasswordEncoder() {
        return new BCryptPasswordEncoder();
    }

}
```

The necessary configurations to Authorize Requests it allowing anonymous access on */login* so that users can authenticate. We'll restrict */admin* to *ADMIN* roles and securing everything else:

```
@Override
protected void configure(HttpSecurity http) throws Exception {

        http
        .csrf().disable()
            .authorizeRequests()
```

```
            .antMatchers("/api/v*/registration/**", "/login/**", "/error/**")
                .permitAll()
        .anyRequest()
            .authenticated().and()
            .formLogin()
            .loginPage("/login")
            .failureUrl("/login-error")
           .and()
        .logout();
    }
```

```
@Override
protected void configure(final HttpSecurity http) throws Exception {
    http
      .csrf().disable()
      .authorizeRequests()
      .antMatchers("/admin/**").hasRole("ADMIN")
      .antMatchers("/anonymous*").anonymous()
      .antMatchers("/login*").permitAll()
      .anyRequest().authenticated()
      .and()
      // ...
}
```

Note that the order of the *antMatchers()* elements is significant; **the more specific rules need to come first, followed by the more general ones**.

Next we'll extend the above configuration for form login and logout:

```
@Override
protected void configure(final HttpSecurity http) throws Exception {
    http
      // ...
      .and()
      .formLogin()
      .loginPage("/login.html")
      .loginProcessingUrl("/perform_login")
      .defaultSuccessUrl("/homepage.html", true)
      .failureUrl("/login.html?error=true")
      .failureHandler(authenticationFailureHandler())
      .and()
      .logout()
      .logoutUrl("/perform_logout")
      .deleteCookies("JSESSIONID")
      .logoutSuccessHandler(logoutSuccessHandler());
}
```

- *loginPage()* – the custom login page
- *loginProcessingUrl()* – the URL to submit the username and password to
- *defaultSuccessUrl()* – the landing page after a successful login
- *failureUrl()* – the landing page after an unsuccessful login
- *logoutUrl()* – the custom logout

The login form page is going to be registered with Spring MVC using the straightforward mechanism to map views names to URLs. Furthermore, there is no need for an explicit controller in between:

```
<html>
<head></head>
<body>
   <h1>Login</h1>
   <form name='f' action="login" method='POST'>
      <table>
         <tr>
            <td>User:</td>
            <td><input type='text' name='username' value=''></td>
         </tr>
         <tr>
            <td>Password:</td>
            <td><input type='password' name='password' /></td>
         </tr>
         <tr>
            <td><input name="submit" type="submit" value="submit" /></td>
         </tr>
      </table>
  </form>
</body>
</html>
```

The **Spring Login form** has the following relevant artifacts:

- *login* – the URL where the form is POSTed to trigger the authentication process
- *username* – the username
- *password* – the password

The default URL where the Spring Login will POST to trigger the authentication process is */login,* which used to be */j_spring_security_check* before Spring Security 4. We can use the *loginProcessingUrl* method to override this URL:

```
http.
  ...
  .formLogin()
  .loginProcessingUrl("/perform_login")
```

After successfully logging in, we will be redirected to a page that by default is the root of the web application.

We can override this via the *defaultSuccessUrl()* method:

```
http.
  ...
  .formLogin()
  .defaultSuccessUrl("/homePage")
```

Similar to the Login Page, the Login Failure Page is autogenerated by Spring Security at */login?error* by default.

To override this, we can use the *failureUrl()* method:

```
http.formLogin()
  .failureUrl("/login.html?error=true")
```