

# Java - User Story 1 - PDF

- Come Sara
- vorrei registrarmi in piattaforma per inserire un articolo,
- in modo tale da lavorare per The Aulab Chronicle.

## ACCEPTANCE CRITERIA:

- Utente deve potersi registrare/loggare in piattaforma
- Bottone “inserisci articolo” in home solo a utenti loggati
- Articolo composto da:
  - Titolo
  - Sottotitolo
  - Corpo del Testo
- Categorie dell’annuncio pre-compilate
- La relazione tra Categoria e Annuncio è 1-a-N
- La relazione tra Utente e Annuncio è 1-a-N
- Ad annuncio inserito visualizzare un messaggio di conferma

## Svolgimento ↴

### PRELIMINARI

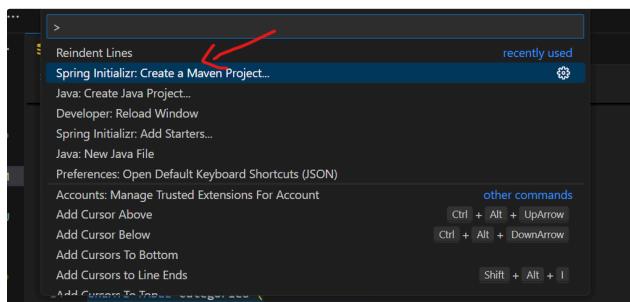
Per poter portare a svolgimento questa user story dobbiamo seguire dei passaggi preliminari per poi configurare ed impostare Spring Security.

### Creazione del progetto spring ↴

Primo fra tutti creiamo il nostro progetto.

Iniziamo creando il progetto tramite Spring Initializr presente in VsCode

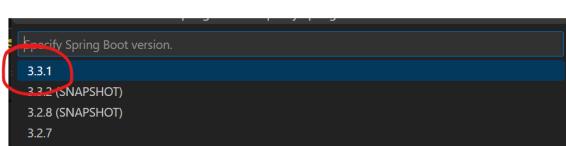
con lo shortcut **ctrl+shift+p**



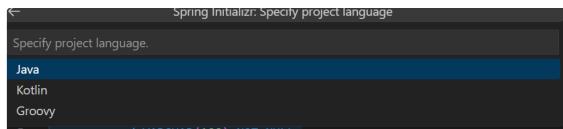
Nel passo successivo scegliamo la versione di springboot.

**⚠ ATENZIONE:** Al momento della stesura della documentazione la versione scelta è la **3.3.1** (come nella immagine). **CONSIGLIAMO QUINDI** di utilizzare la versione stabile e testata per questo progetto **3.3.4** oppure la versione **3.4.0** per le quali il codice che svilupperete è stato testato e funziona correttamente.

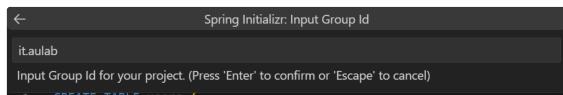
**✖** Per chi non dovesse riuscire a trovare nessuna di queste versioni **NON PREOCCUPATEVI** scegliete la versione più aggiornata (nella lista come nell’immagine) non SNAPSHOT per poi cambiarla a mano, utilizzando le versioni consigliate, a mano nel pom.



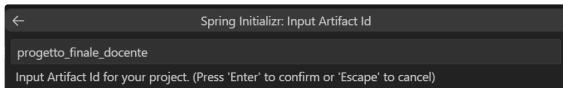
Scegliamo il linguaggio Java



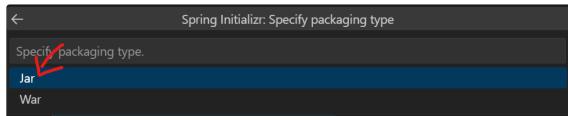
Inseriamo adesso il group id che ricordiamo segue il *reverse-domain-packages*



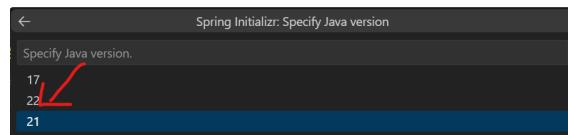
Inseriamo ora il nome del nostro progetto che in spring prende il nome di Artifact Id



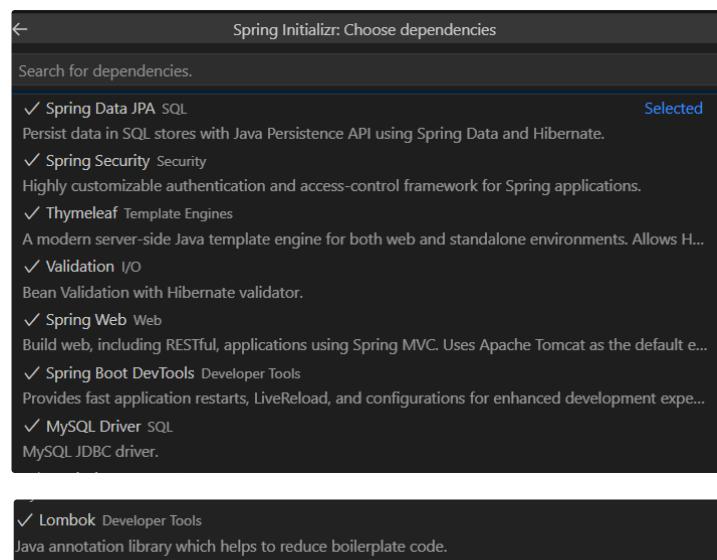
Dopodichè sceglieremo la tipologia di pacchetto finale che nel nostro caso sarà un "jar"



Fatto questo andiamo a scegliere la versione di java che sarà la 21 (versione compatibile con tutti i pacchetti utilizzati al momento della scrittura della guida)

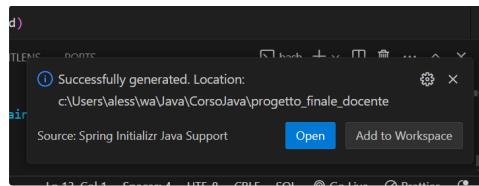


Ed è arrivato il momento di inserire tutte le dipendenza che utilizzeremo nel progetto



- Spring Data Jpa
- Spring Security
- Thymeleaf
- Validation
- Spring Web
- Spring Boot DevTools
- MySQL Driver
- Lombok

Diamo invio e ci verrà chiesto in quale cartella procedere col salvataggio del progetto, successivamente VS Code ci chiederà se vogliamo aprire il progetto con una finestra in basso a destra come questa



Procediamo quindi con l'aprire il progetto, cliccando sul pulsante Open.

### Aggiunta manuale delle dipendenze ↗

Tra le dipendenze dovremo aggiungerne altre due manualmente poiché non raggiungibili tramite Initializr.

Apriamo il file `pom.xml` e all'interno della sezione `dependencies` aggiungiamo queste due dipendenze

```
<dependency>
    <groupId>org.modelmapper</groupId>
    <artifactId>modelmapper</artifactId>
    <version>3.2.0</version>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-mail</artifactId>
</dependency>
```

Queste dipendenze ci saranno utili nelle prossime user stories.

Il pom.xml risultante nelle dipendenze dovrebbe essere simile a questo

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-validation</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.thymeleaf.extras</groupId>
        <artifactId>thymeleaf-extras-springsecurity6</artifactId>
    </dependency>
</dependencies>
```

```

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
    <optional>true</optional>
</dependency>
<dependency>
    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-test</artifactId>
    <scope>test</scope>
</dependency>

```

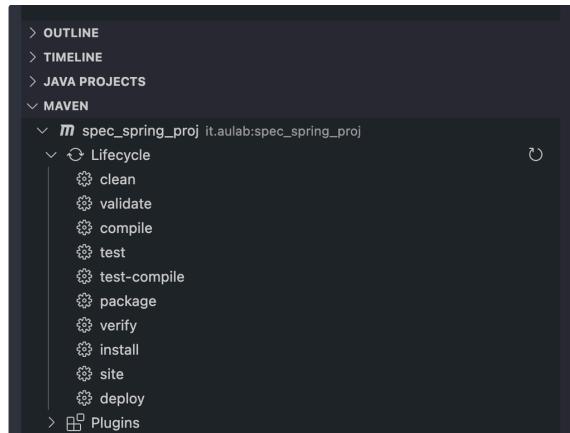
```

<dependency>
    <groupId>org.modelmapper</groupId>
    <artifactId>modelmapper</artifactId>
    <version>3.2.0</version>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-mail</artifactId>
</dependency>
</dependencies>

```

### ⚠️ Tips:

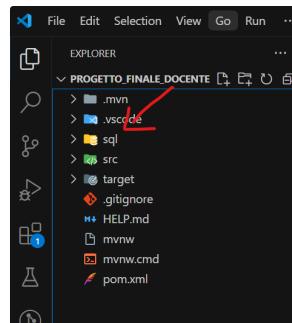
Se non vengono scaricate e installate automaticamente le dipende inserita a mano nel *pom*, procediamo con *clear* e *compile* dal pannello Maven in VsCode



## STRUTTURARE IL DATABASE ⚡

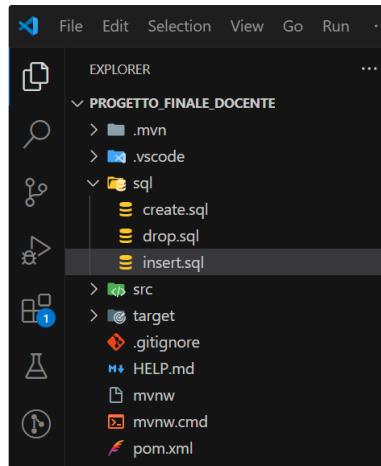
Step fondamentale sarà costruire la struttura iniziale del nostro database tramite script SQL per la creazione delle tabelle.

Creiamo all'interno della root principale del nostro progetto un nuovo folder chiamato “sql”



Ed al suo interno abbiamo bisogno di creare tre file che chiameremo:

- **create.sql**
- **drop.sql**
- **insert.sql**



### Creazione delle tabelle ↴

In "create.sql"

```

create.sql • drop.sql insert.sql
sql > create.sql
    ▶ Run on active connection | ⌂ Select block
  1 CREATE TABLE users (
  2     id BIGINT AUTO_INCREMENT PRIMARY KEY,
  3     username VARCHAR(100),
  4     email VARCHAR(100) NOT NULL UNIQUE,
  5     password VARCHAR(100) NOT NULL,
  6     created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
  7 );
  8
  9 CREATE TABLE roles (
 10     id BIGINT AUTO_INCREMENT PRIMARY KEY,
 11     name VARCHAR(100) NOT NULL
 12 );
 13
 14 CREATE TABLE categories (
 15     id BIGINT AUTO_INCREMENT PRIMARY KEY,
 16     name VARCHAR(50)
 17 );

```

```

18
19   CREATE TABLE articles (
20     id BIGINT AUTO_INCREMENT PRIMARY KEY,
21     title VARCHAR(100),
22     subtitle VARCHAR(100),
23     body TEXT,
24     publish_date DATE,
25     created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
26     user_id BIGINT,
27     category_id BIGINT,
28     FOREIGN KEY (user_id) REFERENCES users(id),
29     FOREIGN KEY (category_id) REFERENCES categories(id)
30   );
31
32   CREATE TABLE users_roles (
33     id BIGINT auto_increment PRIMARY KEY,
34     user_id BIGINT,
35     role_id BIGINT,
36     FOREIGN KEY (user_id) REFERENCES users(id),
37     FOREIGN KEY (role_id) REFERENCES roles(id)
38 );

```

In "drop.sql"

```

create.sql • drop.sql • insert.sql
sql > drop.sql
  ▶ Run on active connection | ⌂ Select block
1  DROP TABLE users_roles;
2  DROP TABLE articles;
3  DROP TABLE categories;
4  DROP TABLE roles;
5  DROP TABLE users;

```

da notare che è stato seguito un ordine ben preciso, poichè bisogna correttamente eliminare per prime le tabelle primarie poi quelle secondarie con chiavi esterne

In "insert.sql"

```

create.sql • drop.sql • insert.sql
sql > insert.sql
  ▶ Run on active connection | ⌂ Select block
1  insert into users (username, email, password, created_at) values ('admin', 'admin@aulab.it',
'${2a$10$oMiUOq5ToRfUI/Zprg5nE.qt8nT9KKjZoDBu1SIWuj.UGx8aRHwxS', '20240607');
2
3  insert into roles (name) values ('ROLE_ADMIN');
4  insert into roles (name) values ('ROLE_REVISOR');
5  insert into roles (name) values ('ROLE_WRITER');
6  insert into roles (name) values ('ROLE_USER');
7
8  insert into users_roles (user_id, role_id) values (1,1);
9
10 insert into categories (name) values ('politica'), ('economia'), ('food&drink') , ('sport'),
('intrattenimento'), ('tech');

```

dove "\${2a\$10\$oMiUOq5ToRfUI/Zprg5nE.qt8nT9KKjZoDBu1SIWuj.UGx8aRHwxS}" è l'hash criptato della password "12345678"

⚠ NB: Questa soluzione non sempre funziona

i Tips:

Possiamo anche scrivere le query di inserimento in maniera compatta se stiamo andando ad inserire più record nella stessa tabella e con gli stessi campi.

```
insert into roles (name) values ('ROLE_ADMIN'), ('ROLE_REVISOR'), ('ROLE_WRITER'), ('ROLE_USER');
```

## CREAZIONE COLLEGAMENTO AL DATABASE

Una volta creati i file script abbiamo bisogno di lanciarli sul nostro database, procediamo allora con la creazione e collegamento del nostro progetto

Usando GitBash su **Windows** lanciamo questo comando per collegarci al server di MySQL

```
1 winpty mysql -u root -p
```

Se usiamo come sistema operativo MacOS usiamo questo comando sull'app Terminale

```
1 mysql -u root -p
```

Fatto questo inseriamo la nostra password creata in fase di installazione di mysql ed una volta entrati lanciamo il comando, per la creazione del database (il nome del DB è un esempio)

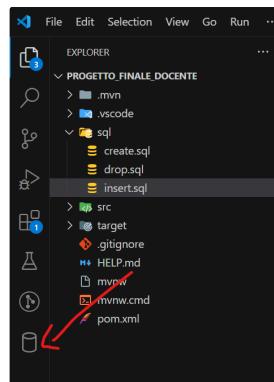
```
1 create database progettoFinaleDocente;
```

diamo invio e poi lanciamo il comando

```
1 exit;
```

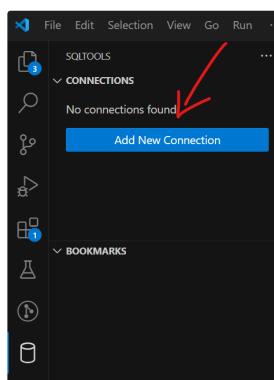
per uscire da mysql dato che non ne avremo più bisogno da ora in poi.

Utilizziamo adesso il tool di VSCode SQLTools

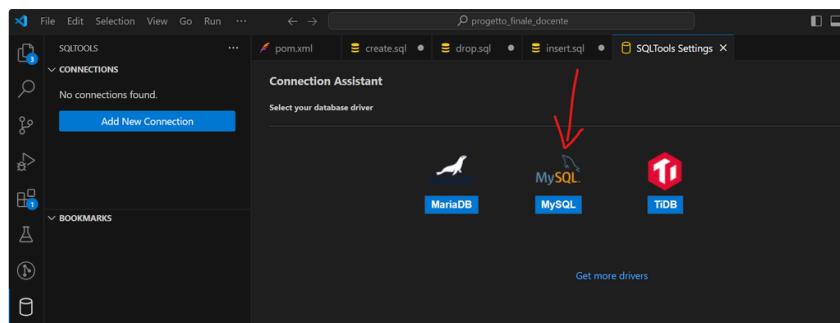


Se non vedete l'icone va installato.

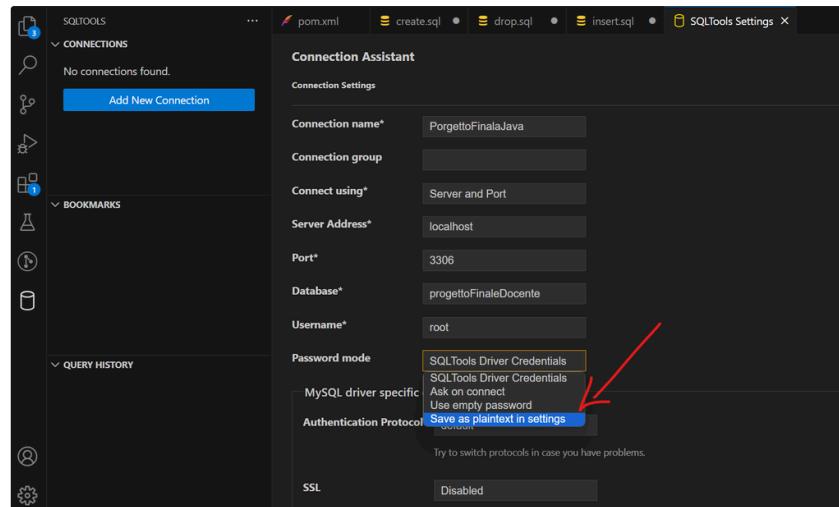
Una volta cliccata andiamo su



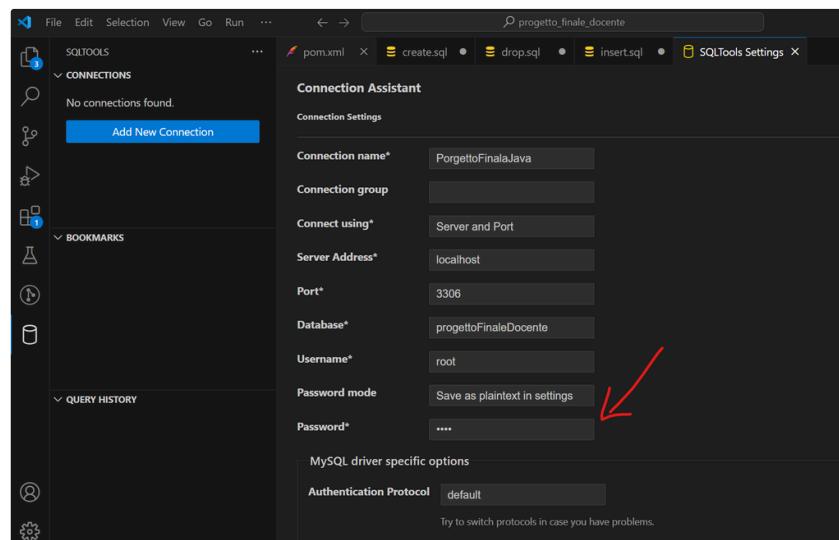
E scegliamo



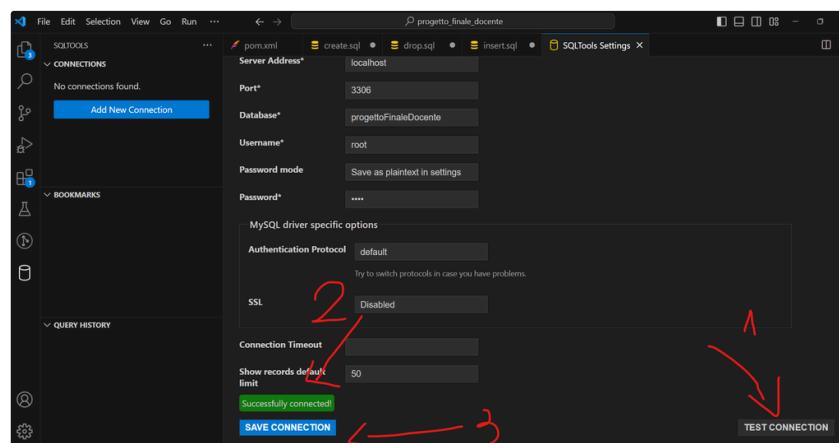
Inseriamo adesso i dati che ci servono



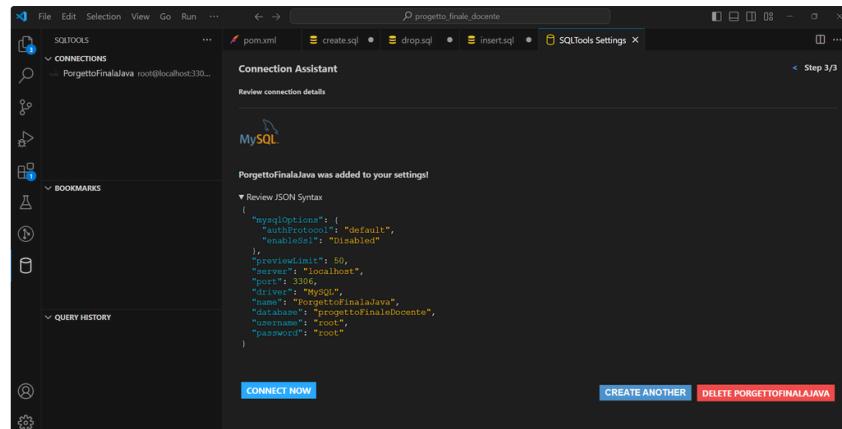
Il **“Connection name”** è di pura fantasia ma diamo un nome sensato, il **“Database”** dovrà corrispondere come nome a quello creato, come **“Username”** utilizziamo il nostro di Mysql e per la **password** clicchiamo nello spazio di **“Password mode”** cercando e selezionando **“Save as plaintext in settings”**, fatto questo avremo



il **nuovo campo di input** dove inseriremo la nostra password di mysql impostata in fase di installazione



Scorriamo verso il basso e facciamo un **test sulla connessione**, se va a buon fine ed abbiamo un **messaggio di successo** possiamo **salvare** la connessione



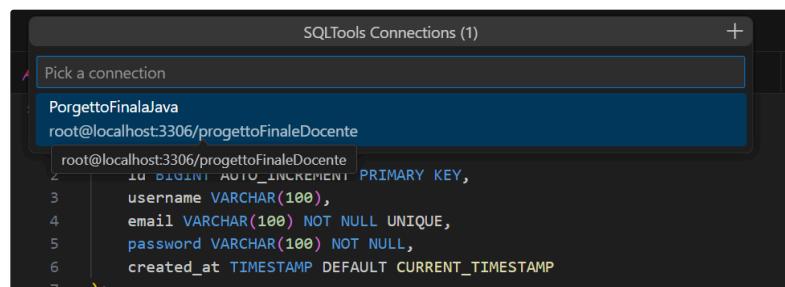
A questo punto abbiamo una connessione da poter utilizzare, torniamo ai nostri script e clicchiamo su **“Run on active connection”**

```

1 CREATE TABLE users (
2     id BIGINT AUTO_INCREMENT PRIMARY KEY,
3     username VARCHAR(100),
4     email VARCHAR(100) NOT NULL UNIQUE,
5     password VARCHAR(100) NOT NULL,
6     created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
7 );
8
9 CREATE TABLE roles (
10    id BIGINT AUTO_INCREMENT PRIMARY KEY,
11    name VARCHAR(100) NOT NULL
12 );
13
14 CREATE TABLE categories (
15    id BIGINT AUTO_INCREMENT PRIMARY KEY,
16    name VARCHAR(50)
17 );
18
19 CREATE TABLE articles (
20    id BIGINT AUTO_INCREMENT PRIMARY KEY,
21    title VARCHAR(100),
22    subtitle VARCHAR(100),
23    body TEXT,

```

E selezioniamo la connessione appena creata



Se tutto va a buon fine vedremo eseguire lo script senza alcun errore

```

pom.xml          create.sql   drop.sql   insert   ...   PorgettoFinalJava: multiple query results ...
1 CREATE TABLE users (
2     id BIGINT AUTO_INCREMENT PRIMARY KEY,
3     username VARCHAR(100),
4     email VARCHAR(100) NOT NULL UNIQUE,
5     password VARCHAR(100) NOT NULL,
6     created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
7 );
8
9 CREATE TABLE roles (
10    id BIGINT AUTO_INCREMENT PRIMARY KEY,
11    name VARCHAR(100) NOT NULL
12 );
13
14 CREATE TABLE categories (
15    id BIGINT AUTO_INCREMENT PRIMARY KEY,
16    name VARCHAR(50)
17 );
18

```

Eseguiamo poi in sequenza il **drop** per accertarci che vada a buon fine la cancellazione delle tabelle, nuovamente il **create** e poi l'**insert**.

Ricordiamoci sempre di **salvare tutto** e possiamo chiudere tutte le tab fin' ora aperte.

### APPLICATION PROPERTIES

Prima di poter andare avanti però dobbiamo inserire nel nostro *"application.properties"* la configurazione verso il database appena creato in "src\main\resources\application.properties"

```

src > main > resources > application.properties
1 spring.application.name=progetto_finale
2 spring.datasource.url=jdbc:mysql://localhost:3306/progettofinale
3 spring.datasource.username=root
4 spring.datasource.password=root
5 spring.datasource.driver-class-name=com.mysql.jdbc.Driver

```

Fine dello scaffolding del progetto.

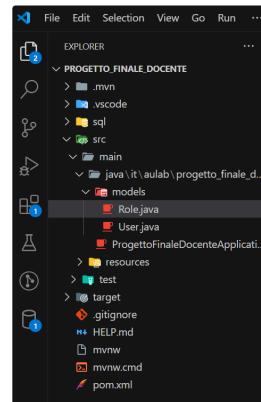
## IMPOSTAZIONE DI SPRING SECURITY ☀

È giunto il momento di configurare la funzionalità di registrazione per gli utenti. Per poterlo fare abbiamo bisogno di utilizzare la libreria **Spring Security** inserita alla creazione del progetto.

### Models ☀

Iniziamo col creare i modelli relativi alle entità strutturate all'interno delle nostre tabelle nel db (per il momento ci dedicheremo a User e Role). Creiamo un nuovo package che chiameremo **models** in "src\main\java\it\aulab\progetto\_finale\_docente" al suo interno creiamo 2 file:

- *User.java*
- *Role.java*



### User ☀

Lo user per poter corrispondere esattamente alla nostra tabella deve avere questi campi :

- id

- username
- email
- password
- roles (che descriverà una relazione many to many tra utenti e ruoli)

in "src\main\java\it\aulab\progetto\_finale\_docente\models\User.java"

```

User.java
Role.java

src > main > java > it > aulab > progetto_finale_demo_doc > models > User.java > User
1 package it.aulab.progetto_finale_demo_doc.models;
2
3 import jakarta.persistence.*;
4 import lombok.AllArgsConstructor;
5 import lombok.Getter;
6 import lombok.NoArgsConstructor;
7 import lombok.Setter;
8 import java.util.ArrayList;
9 import java.util.List;
10

11 @Getter
12 @Setter
13 @NoArgsConstructor
14 @AllArgsConstructor
15 @Entity
16 @Table(name = "users")
17 public class User {
18
19     @Id
20     @GeneratedValue(strategy = GenerationType.IDENTITY)
21     private Long id;
22     @Column(nullable = false)
23     private String username;
24     @Column(nullable = false, unique = true)
25     private String email;
26     @Column(nullable = false)
27     private String password;
28
29     @ManyToMany(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
30     @JoinTable(
31         name = "users_roles",
32         joinColumns = {@JoinColumn(name = "USER_ID", referencedColumnName = "ID")},
33         inverseJoinColumns = {@JoinColumn(name = "ROLE_ID", referencedColumnName = "ID")})
34     private List<Role> roles = new ArrayList<>();
35 }

```

i Grazie alla libreria Lombok non avremo bisogno di esplicitare costruttori e getter e setter poiché saranno lui a costruirli per noi, basterà utilizzare le annotation della libreria.

## Role

Il modello role deve avere:

- id
- name
- una lista di User a cui è collegato che descrive la relazione many to many con gli utenti

in "src\main\java\it\aulab\progetto\_finale\_docente\models\Role.java"

```

User.java
Role.java

src > main > java > it > aulab > progetto_finale_demo_doc > models > Role.java > ...
1 package it.aulab.progetto_finale_demo_doc.models;
2
3 import jakarta.persistence.*;
4 import lombok.AllArgsConstructor;
5 import lombok.Getter;
6 import lombok.NoArgsConstructor;
7 import lombok.Setter;
8 import java.util.List;
9
10 @Setter
11 @Getter
12 @NoArgsConstructor
13 @AllArgsConstructor
14 @Entity
15 @Table(name = "roles")
16 public class Role {
17     @Id
18     @GeneratedValue(strategy = GenerationType.IDENTITY)
19     private Long id;
20     @Column(nullable = false, unique = true)
21     private String name;
22
23     @ManyToMany(mappedBy = "roles")
24     private List<User> users;
25 }

```

## Repository ↗

Una volta creati i modelli dobbiamo procedere col creare i repository che avranno il ruolo fondamentale di interagire con le tabelle nel database.

Quindi procediamo col creare un nuovo package che chiameremo "repositories" in "src\main\java\it\aulab\progetto\_finale\_docente"

Al suo interno creeremo i repository relativi alle entità Role e User

### UserRepository ↗

Creiamo quindi il file "UserRepository.java"

Questo file conterrà un'interfaccia che estenderà l'interfaccia *JpaRepository*.

Al suo interno dichiareremo un metodo **findByEmail** che il repository implementerà internamente grazie alle derived query.

Inoltre, decoreremo questa classe con l'annotation **@Repository** in modo che diventi un bean e venga istanziato automaticamente dal framework.

in "src\main\java\it\aulab\progetto\_finale\_docente\repositories\UserRepository.java"

```
 UserRepository.java X
src > main > java > it > aulab > progetto_finale_demo_doc > repositories > UserRepository.java > +o UserRepository
1 package it.aulab.progetto_finale_demo_doc.repositories;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4 import org.springframework.stereotype.Repository;
5
6 import it.aulab.progetto_finale_demo_doc.models.User;
7
8 @Repository
9 public interface UserRepository extends JpaRepository<User, Long> {
10     User findByEmail(String email);
11 }
```

Ed **importiamo la classe User** secondo il nostro progetto.

## DTO ↗

Prima di poter andare avanti dobbiamo creare un DTO per la classe User.

Il DTO sarà effettivamente l'oggetto che manipoleremo per le nostre logiche.

Creiamo un package "dtos" in "src\main\java\it\aulab\progetto\_finale\_docente"

Al suo interno creiamo un file "**UserDto.java**"

Utilizzeremo questa classe come struttura per i dati che riceveremo dal form di register e login

in "src\main\java\it\aulab\progetto\_finale\_docente\dtos\UserDto.java"

```
 UserDto.java X
src > main > java > it > aulab > progetto_finale_demo_doc > dtos > UserDto.java > ...
1 package it.aulab.progetto_finale_demo_doc.dtos;
2
3 import jakarta.validation.constraints.Email;
4 import jakarta.validation.constraints.NotEmpty;
5 import lombok.AllArgsConstructor;
6 import lombok.Getter;
7 import lombok.NoArgsConstructor;
8 import lombok.Setter;
9
10 @Getter
11 @Setter
12 @NoArgsConstructor
13 @AllArgsConstructor
14 public class UserDto {
15     private Long id;
16     @NotEmpty
17     private String firstName;
18     @NotEmpty
19     private String lastName;
20     @NotEmpty(message = "Email should not be empty")
21     @Email
22     private String email;
23     @NotEmpty(message = "Password should not be empty")
24     private String password;
25 }
```

- Con l'annotation **@NotEmpty** stiamo effettuando anche una validazione sul campo affinché non venga mai memorizzato un campo vuoto all'interno del database.
- Con la parte del "message" stiamo inserendo anche dei messaggi da visualizzare in caso di validazione non rispettata.
- Con **@Email** ci stiamo assicurando tramite validazione che venga inserita una mail valida

Utilizzeremo in seguito queste *annotation* per la validazione

## Service ↗

Ora dobbiamo creare il service, il quale gestirà tutte le operazioni logiche prima di poter manipolare il nostro database attraverso il repository.

Creiamo un nuovo package "services" in "src\main\java\it\aulab\progetto\_finale\_doc" e al suo interno andiamo a creare l'interfaccia *UserService.java* che implementeremo.

I metodi che dichiareremo al suo interno saranno:

- *saveUser*: Salverà un nuovo utente tramite DTO
- *findUserByEmail*: Riceverà in input una email e ne cercherà l'utente

```

UserService.java
src > main > java > it > aulab > progetto_finale_demo_doc > services > UserService.java > UserService > findUserByEmail(String)
1 package it.aulab.progetto_finale_demo_doc.services;
2
3 import org.springframework.web.servlet.mvc.support.RedirectAttributes;
4
5 import it.aulab.progetto_finale_demo_doc.dtos.UserDto;
6 import it.aulab.progetto_finale_demo_doc.models.User;
7 import jakarta.servlet.http.HttpServletRequest;
8 import jakarta.servlet.http.HttpServletResponse;
9
10 public interface UserService {
11     void saveUser(UserDto userDto, RedirectAttributes redirectAttributes, HttpServletRequest request,
12     HttpServletResponse response);
13     User findUserByEmail(String email);
14 }

```

Ed importiamo UserDto e User secondo il nostro progetto.

## Implementazione del UserService → UserServiceImpl ↗

Passiamo però all'implementazione del service.

Creiamo allora una nuova classe con il nome ***UserServiceImpl.java*** sempre all'interno di "src\main\java\it\aulab\progetto\_finale\_doc\services"

Questa classe deve implementare l'interfaccia UserService **ergo implementare i metodi che sono presenti al suo interno**

in "src\main\java\it\aulab\progetto\_finale\_doc\services\UserServiceImpl.java"

```

UserServiceImpl.java
src > progetto_finale_demo_doc > services > UserServiceImpl.java > UserServiceImpl > saveUser(UserDto, RedirectAttributes, HttpServletRequest, HttpServletResponse)
1 package it.aulab.progetto_finale_demo_doc.services;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
5 import org.springframework.security.crypto.password.PasswordEncoder;
6 import org.springframework.stereotype.Service;
7 import org.springframework.web.servlet.mvc.support.RedirectAttributes;
8
9 import it.aulab.progetto_finale_demo_doc.dtos.UserDto;
10 import it.aulab.progetto_finale_demo_doc.models.User;
11 import it.aulab.progetto_finale_demo_doc.repositories.UserRepository;
12 import jakarta.servlet.http.HttpServletRequest;
13 import jakarta.servlet.http.HttpServletResponse;
14
15 @Service
16 public class UserServiceImpl implements UserService {
17
18     @Autowired
19     private UserRepository userRepository;
20
21     public PasswordEncoder passwordEncoder() {
22         return new BCryptPasswordEncoder();
23     }
24
25     @Override
26     public User findUserByEmail(String email) {
27         return userRepository.findByEmail(email);
28     }
29 }

```

```

29
30     @Override
31     public void saveUser(UserDto userDto, RedirectAttributes redirectAttributes, HttpServletRequest request,
32     HttpServletResponse response){
33         User user = new User();
34         user.setUsername(userDto.getFirstName() + " " + userDto.getLastName());
35         user.setEmail(userDto.getEmail());
36         user.setPassword(passwordEncoder().encode(userDto.getPassword()));
37     }
38 }
39

```

Ed importiamo UserDto, User e UserRepository secondo il nostro progetto.

Il metodo **findUserByEmail** è abbastanza semplice da implementare. Deve semplicemente ritornare il risultato del metodo **findByEmail** analogo del repository.

Il metodo **saveUser** è un pochino più complesso.

In input riceviamo un oggetto di classe *UserDTO* che non è una entità e quindi non è possibile lanciare una save sulla tabella user passando per il dto, quindi creiamo prima un oggetto di classe User tramite le informazioni di userDto.

Per poter criptare la password abbiamo utilizzato un metodo che ritorna un oggetto di classe PasswordEncoder. La funzione di un passwordEncoder è quella di codificare le password degli utenti in modo che non siano memorizzate in chiaro nel database, migliorando la sicurezza dell'applicazione utilizzando l'algoritmo di BCrypt.

Il metodo migliore è quello di creare un bean all'interno di una classe di configurazione (ad esempio annotata con **@Configuration**) che ci restituirà un oggetto di questa classe che possiamo iniettare all'interno della classe ma lo faremo più avanti.

Questo metodo riceverà in input anche un **HttpServletRequest request** ed un **HttpServletResponse response** che utilizzeremo più avanti per gestire la redirect verso la home dopo la register

Non dimentichiamoci di decorare questa classe con l'annotation **@Service**.

## Controller ↗

A questo punto la logica applicativa è settata, ora la dobbiamo esporre tramite un controller.

Quindi dobbiamo creare un nuovo package “*controllers*” in “src/main/java/it/aulab/progetto\_finale\_doc”

e creare al suo interno una classe **UserController.java** che annoteremo con l'annotation **@Controller**. Inoltre importeremo lo UserService in modo da poter usare la logica creata

in “src/main/java/it/aulab/progetto\_finale\_doc/controllers/UserController.java”

```

UserController.java 1 ●
src > main > java > it > aulab > progetto_finale_demo_doc > controllers > UserController.java > ...
1  package it.aulab.progetto_finale_demo_doc.controllers;
2
3  import org.springframework.beans.factory.annotation.Autowired;
4  import org.springframework.stereotype.Controller;
5  import org.springframework.web.bind.annotation.GetMapping;
6  import it.aulab.progetto_finale_demo_doc.services.UserService;
7
8  @Controller
9  public class UserController {
10
11     @Autowired
12     private UserService userService;
13
14     //Rotta di home
15     @GetMapping("/")
16     public String home() {
17         return "home";
18     }
19
20 }

```

Ed importiamo *UserService*, *GetMapping*

In questo controller creeremo tutti i nostri handler

Il primo endpoint utile è quello che ci restituisce la homepage settata sull' uri /

## TEMPLATE HOME CON THYMELEAF ↗

Attualmente non abbiamo alcun template per la home. Andiamo quindi a creare all'interno di “src/main/resources/templates” un nuovo file di template che chiameremo “*home.html*” e all'interno del quale utilizzeremo Thymeleaf

Sappiamo che la nostra home page sarà composta sicuramente da una navbar e un footer, elementi che utilizzeremo in tutte le nostre nuove pagine. Per questo motivo, è opportuno creare un template di index che contenga i fragment necessari, inclusi i riferimenti alle CDN necessarie. In questo modo, potremo riutilizzare facilmente questi componenti in tutto il nostro progetto.

in "src\main\resources\templates\index.html"

```

index.html M
src > main > resources > templates > index.html > html > body > nav.navbar.navbar-expand-lg.navbar-light.bg-light > div.container-fluid > div.navbarSupportedContent.navbar-collapse > div.navbar.nav-item-auto-re-2mb-lg > 6
  1 <!DOCTYPE html>
  2 <html xmlns="http://www.thymeleaf.org">
  3 
  4 <!-- fragment head -->
  5 <head th:fragment="head">
  6   meta charset="UTF-8"
  7   meta http-equiv="Content-Security-Policy" content="width=device-width, initial-scale=1.0"
  8   link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-QMkC2yj9PEJISv5aR0U0DfEpokYctnmb5pNlyT2d8jXb3RhjY6hneALEwH" crossorigin="anonymous"
  9 
 10  <link rel="stylesheet" href="https://cloudflare.com/gfx/1/b/font-awesome/6.5.2/css/all_min.css" integrity="sha512-5mH5Wv+bZgPiHs44dIX+LL3A29/2PKI5QIAjGt8B9e+fb3tkMfrRyxxjp#cv7noD91l0qev5CwE3m6d+-" crossorigin="anonymous" refererPolicy="no-referrer" />
 11 
 12  <title th:text="${title}" title>
 13 </head>
 14 <!-- fine fragment head -->
 15 <body>
 16   <!-- inicio navbar -->
 17   <nav th:fragment="navbar" class="navbar navbar-expand-lg navbar-light bg-light">
 18     <div class="container-fluid">
 19       <a class="navbar-brand" th:href="@{/}">Aulab Chronicle</a>
 20       <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false">
 21         <span class="navbar-toggler-icon"></span>
 22       </button>
 23       <div class="collapse navbar-collapse" id="navbarSupportedContent">
 24         <ul class="navbar-nav me-auto mb-2 mb-lg-0">
 25           <li class="nav-item">
 26             <a class="nav-link" href="#">crea articolo</a>
 27           </li>
 28           <li class="nav-item">
 29             <a class="nav-link" href="#">link</a>
 30           </li>
 31           <li class="nav-item dropdown">
 32             <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button" data-bs-toggle="dropdown" aria-expanded="false">
 33               Accesso
 34             </a>
 35             <ul class="dropdown-menu" aria-labelledby="navbarDropdown">
 36               <li><a href="#">Register</a></li>
 37               <li><a href="#">Login</a></li>
 38             </ul>
 39             <div>Benvenuto</div>
 40           </li>
 41         </ul>
 42       </div>
 43       <form class="d-flex">
 44         <input class="form-control me-2" type="search" placeholder="Search" name="keyword" aria-label="Search">
 45         <button class="btn btn-outline-success" type="submit">Search</button>
 46       </form>
 47     </nav>
 48     <!-- fine navbar -->
 49 
 50     <!-- Footer -->
 51     <footer th:fragment="footer" class="bg-dark text-white pt-4">
 52       <div class="container">
 53         <div class="row justify-content-between">
 54           <!-- Company Info -->
 55           <div class="col-md-4 mb-3">
 56             <a href="#">

##### Aulab IT

</a>
 57           </div>
 58           <!-- Contact Info -->
 59           <div class="col-md-4 mb-3 text-end">
 60             <h5>Lavora con noi</h5>
 61             <a href="#" class="btn btn-primary">Richiedi</a>
 62           </div>
 63         </div>
 64       <div class="bg-secondary text-center py-2">
 65         <p class="mb-0">© 2024 Aulab Jost. All rights reserved.</p>
 66       </div>
 67     </footer>
 68     <!-- fine del footer -->
 69 
 70     <!-- fragment script bootstrap -->
 71     <script th:fragment="bootstrapScript" src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-Ypcryf0tY3lHB0NkmXc5s9FDVZLESAA55NDz0xhy9Gkc1ds1K1eN6j1eHz" crossorigin="anonymous"></script>
 72   </div>
 73 </body>
 74 </html>
```

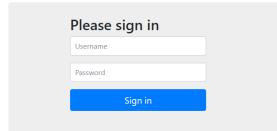
in "src\main\resources\templates\home.html"

```

home.html M
src > main > resources > templates > home.html > html
  1 <!DOCTYPE html>
  2 <html lang="en"
  3   xmlns="http://www.thymeleaf.org"
  4   > Alessandro Leo, last month * US01
  5   <head th:insert="~{index :: head}">
  6     <body>
  7 
  8       <head th:insert="~{index :: navbar}"></head>
  9 
 10       <div class="container-fluid">
 11         <div class="row ms-5 vh-100 align-items-center">
 12           <div class="col-10">
 13             <h1 class="mb-5" style="font-weight: bold;">Welcome to <span>New Aulab Chronicle</span>
 14           </div>
 15         </div>
 16       </div>
 17 
 18       <head th:insert="~{index :: footer}"></head>
 19       <script th:replace="~{index :: bootstrapScript}"></script>
 20 
 21   </body>
 22 </html>
```

Arrivati a questo punto possiamo già provare il nostro applicativo ed avviarlo.

Vedremo che una volta avviato andando sul nostro browser ed inserendo “<http://localhost:8080/>” o “<http://127.0.0.1:8080/>” verremo immediatamente reindirizzati ad una pagina di login.



Non abbiamo creato nulla effettivamente, ma questa pagina è la login di default di Spring Security. Per poter accedere e vedere la nostra pagina home dobbiamo utilizzare la password auto-generata da spring in fase di avvio, la troviamo nel nostro terminale

```
2024-05-21T17:24:55+07:00 [main] INFO org.springframework.security.core.context.SecurityContextHolder - Using generated security password: 04d0eb2e-4691-46ac-b792-8701e04fcfed4
```

E usare come Username: "user" sempre impostato di default

Una volta entrati vedremo la home appena creata

## REGISTER ↗

Fatti questi passi preliminari andiamo quindi a costruire tutta l'infrastruttura di registrazione, login e logout.

Iniziamo con la register partendo proprio dall'handler da aggiungere nel controller

in “src\main\java\it\aulab\progetto\_finale\_doc\controllers\UserController.java”

```
UserController.java 1

src > main > java > it > aulab > progetto_finale_doc > controllers > UserController.java > UserController > register(Model)
12  public class UserController {
13      private UserService userService;
14
15      //Rotta di home
16      @GetMapping("/")
17      public String home() {
18          return "home";
19      }
20
21
22      //Rotta per la registrazione
23      @GetMapping("/register")
24      public String register(Model model) {
25          model.addAttribute("user", new UserDto());
26          return "auth/register";
27      }
28
29
30 }
```

Ed importiamo “org.springframework.ui.Model” e UserDto

Fatto questo procediamo col creare il template di register che , come vediamo nella return del metodo , dovrà essere collocato all'interno del folder auth.

Creiamo quindi il nuovo folder “auth” all'interno di “src\main\resources\templates” ed il nuovo file “register.html”

in “src\main\resources\templates\auth\register.html”

```

register.html •
src > main > resources > templates > auth > register.html > html > body > head
1   <!DOCTYPE html>
2   <html lang="en"
3     xmlns:th="http://www.thymeleaf.org"
4   >
5   <head th:insert="{index :: navbar}"></head>
6   <body>
7       <head th:insert="{index :: navbar}"></head>
8
9       <div class="vh-100">
10         <br /><br />
11         <div class="container">
12             <div class="row col-md-8 offset-md-2">
13                 <div class="card">
14                     <div class="card-header">
15                         <h2 class="text-center">Registration</h2>
16                     </div>
17                     <div th:if="${param.success}">
18                         <div class="alert alert-info">
19                             You successfully registered our app!
20                         </div>
21                     </div>
22                     <div class="card-body">
23                         <form method="post" role="form" th:action="@{/register/save}" th:object="${user}">
24                             <div class="form-group mb-3">
25                                 <label class="form-label">First Name:</label>
26                                 <input class="form-control" id="firstName" name="firstName" placeholder="Enter first name" th:field="#{firstName}" type="text" />
27                                 <p th:errors = "${#fields.hasErrors('firstName')}" class="text-danger" th:if="${#fields.hasErrors('firstName')}"/>
28                             </div>
29                             <div class="form-group mb-3">
30                                 <label class="form-label">Last Name:</label>
31                                 <input class="form-control" id="lastName" name="lastName" placeholder="Enter last name" th:field="#{lastName}" type="text" />
32                                 <p th:errors = "${#fields.hasErrors('lastName')}" class="text-danger" th:if="${#fields.hasErrors('lastName')}"/>
33                             </div>
34                         </div>
35                     </div>
36
37                     <div class="form-group mb-3">
38                         <label class="form-label">Email:</label>
39                         <input class="form-control" id="email" name="email" placeholder="Enter email address" th:field="#{email}" type="email" />
40                         <p th:errors = "${#fields.hasErrors('email')}" class="text-danger" th:if="${#fields.hasErrors('email')}"/>
41                     </div>
42
43                     <div class="form-group mb-3">
44                         <label class="form-label">Password:</label>
45                         <input class="form-control" id="password" name="password" placeholder="Enter password" th:field="#{password}" type="password" />
46                         <p th:errors = "${#fields.hasErrors('password')}" class="text-danger" th:if="${#fields.hasErrors('password')}"/>
47                     </div>
48
49                     <div class="form-group">
50                         <button class="btn btn-primary" type="submit">Register</button>
51                         <span>Already registered? <a href="@{/login}">Login here</a></span>
52                     </div>
53
54                 </div>
55             </div>
56         </div>
57     </div>
58
59     <head th:insert="{index :: footer}"></head>
60     <script th:replace="{index :: bootstrapScript}"></script>
61
62 </body>
63 </html>

```

Nel form abbiamo collegato l'oggetto del model user al form tramite l'attributo **th:object** di thymeleaf

Con questo binding tra form e oggetto possiamo collegare ad ogni input del form un attributo del model tramite attributo thymeleaf **th:field**

Inoltreabbiamo fatto anche il binding degli errori

## LOGIN ↵

Altro handler da inserire è quello per la vista con il form di login

in "src\main\java\it\aulab\progetto\_finale\_docente\controllers\UserController.java"

```

UserController.java 1 •
src > main > java > it > aulab > progetto_finale_demo_doc > controllers > UserController.java > UserController > login()
12  public class UserController {
13
14      //Rotta per la registrazione
15      @GetMapping("/register")
16      public String register(Model model) {
17          model.addAttribute("user", new UserDto());
18          return "auth/register";
19      }
20
21      //Rotta per la login
22      @GetMapping(["/login"])
23      public String login() {
24          return "auth/login";
25      }
26
27
28
29
30
31
32
33
34
35
36  }

```

Andiamo allora a creare il template di login sempre all'interno del folder auth

in "src\main\resources\templates\auth\login.html"

```

login.html x
src > main > resources > templates > auth > login.html > html > body > head
1   <!DOCTYPE html>
2   <html lang="en"
3     xmlns:th="http://www.thymeleaf.org"
4   >
5   <head th:insert="~{index :: head}"></head>
6   <body>
7     <head th:insert="~{index :: navbar}"></head>
8
9     <div class="vh-100">
10    <br /><br />
11    <div class="container">
12      <div class="row">
13        <div class="col-md-6 offset-md-3">
14          <div th:if="${param.error}">
15            <div class="alert alert-danger">Invalid Email or Password</div>
16          </div>
17          <div th:if="${param.logout}">
18            <div class="alert alert-success"> You have been logged out.</div>
19          </div>
20          <div class="card">
21            <div class="card-header">
22              <h2 class="text-center">Login Form</h2>
23            </div>
24            <div class="card-body">
25              <form method="post" role="form" th:action="@{/login}" class="form-horizontal">
26                <div class="form-group mb-3">
27                  <label class="control-label"> Email </label>
28                  <input type="text" id="username" name="username" class="form-control" placeholder="Enter email address"/>
29                </div>
30                <div class="form-group mb-3">
31                  <label class="control-label"> Password </label>
32                  <input type="password" id="password" name="password" class="form-control" placeholder="Enter password"/>
33                </div>
34                <div class="form-group mb-3">
35                  <button type="submit" class="btn btn-primary">Submit</button>
36                  <span> Not registered ? </span>
37                  <a th:href="@{/register}">Register/Signup here</a>
38                </div>
39              </div>

```

```

40              </form>
41            </div>
42          </div>
43        </div>
44      </div>
45    </div>
46  </div>
47
48
49  <head th:insert="~{index :: footer}"></head>
50  <script th:replace="~{index :: bootstrapScript}"></script>
51 </body>
52 </html>

```

## Post mapping register ↴

Ora possiamo passare all'handler post della richiesta register().

Questa funzione avrà accesso ai dati compilati nel form che, ricordiamo, sono sottoforma di oggetto di classe UserDTO.

Per ricevere questo oggetto nella funzione, dobbiamo aggiungere un parametro formale decorato con l'annotazione **@ModelAttribute** e specificare il nome dell'attributo. Nel nostro caso, alla vista del form abbiamo passato un attributo con il nome "user" quindi anche nell'annotation *ModelAttribute* utilizzeremo lo stesso nome

in "src/main/java/it/aulab/progetto\_finale\_demo/controllers/UserController.java"

```

UserController.java ●
src > it > aulab > progetto_finale_demo.doc > controllers > UserController.java > UserController > registration(UserDto, BindingResult, Model, RedirectAttributes, HttpServletRequest)
20  public class UserController {
21
22    //Rotta per il salvataggio della registrazione
23    @PostMapping("/register/save")
24    public String registration(@Valid @ModelAttribute("user") UserDto userDto,
25                               BindingResult result,
26                               Model model,
27                               RedirectAttributes redirectAttributes,
28                               HttpServletRequest request, HttpServletResponse response){
29
30      User existingUser = userService.findUserByEmail(userDto.getEmail());
31
32      if (existingUser != null && existingUser.getEmail() != null && existingUser.getEmail().isEmpty()) {
33        result.rejectValue("email", null,
34                           "There is already an account registered with the same email");
35      }
36
37      if (result.hasErrors()) {
38        model.addAttribute("user", userDto);
39        return "auth/register";
40      }
41
42      userService.saveUser(userDto, redirectAttributes, request, response);
43
44      redirectAttributes.addFlashAttribute(attributeName:"successMessage", attributeValue:"Registrazione avvenuta!");
45
46      return "redirect:/register?success";
47    }
48  }

```

Ed importiamo *ModelAttribute* e *user* se non fatto in automatico

```

import it.aulab.progetto_finale_demo_doc.models.User;
import it.aulab.progetto_finale_demo_doc.dtos.UserDto;

```

In questo metodo abbiamo attivato dei check:

- Validazioni
- Impossibilità di inserire una mail già esistente

Tra i parametri notiamo `@Valid` utilizzato per abilitare la validazione dei dati in base alle regole definite nelle classi `UserDto`, `BindingResult result` che contiene il risultato della validazione dei dati. Viene utilizzato per raccogliere eventuali errori di validazione che si verificano durante il processo di binding dei dati.

Abbiamo quindi inserito dei controlli sulla esistenza dell'utente all'interno della nostra base di dati e la cattura degli errori di validazione.

Questo metodo riceve anche `HttpServletRequest request` e `HttpServletResponse response` che utilizzeremo sempre per la redirect dopo la register che non possiamo ancora attivare.

In più abbiamo inserito una redirect con parametro "success" che attiverà nel form di register la parte di codice

```
17 |     |     |     |     <div th:if="${param.success}">
18 |     |     |     |     <div class="alert alert-info">
19 |     |     |     |     |     You have successfully registered our app!
20 |     |     |     |     </div>
21 |     </div>
```

## Spring security configuration ↗

È arrivato il momento di personalizzare la UI della nostra login, non più con quella di default ma con il form di login che abbiamo creato noi.

Il login di default è un comportamento dovuto alla configurazione standard di Spring Security.

Abbiamo importato la libreria ma non l'abbiamo configurata e **Spring security di default protegge tutte le rotte** rendendole disponibili solo agli utenti autenticati.

Possiamo cambiare questo comportamento configurando la libreria a nostro piacimento.

Per personalizzare Spring dobbiamo creare un nuovo file di configurazione e lo faremo nel package "*config*" che creeremo in "`src\main\java\it\aulab\progetto_finale_docente`"

All'interno di questo package dobbiamo creare una nuova classe chiamata **SecurityConfig** ( il nome è a vostra discrezione ).

Al suo interno configureremo un oggetto con il comportamento che ci aspettiamo dalla libreria.

Questa classe deve essere decorata con due annotation :

- *Configuration* → ci serve per poter settare dei bean da caricare in memoria
- *EnableWebSecurity* → ci serve per aggiungere queste configurazioni all'istanza di security che stiamo allocando

in "`src\main\java\it\aulab\progetto_finale_docente\config\SecurityConfig.java`"

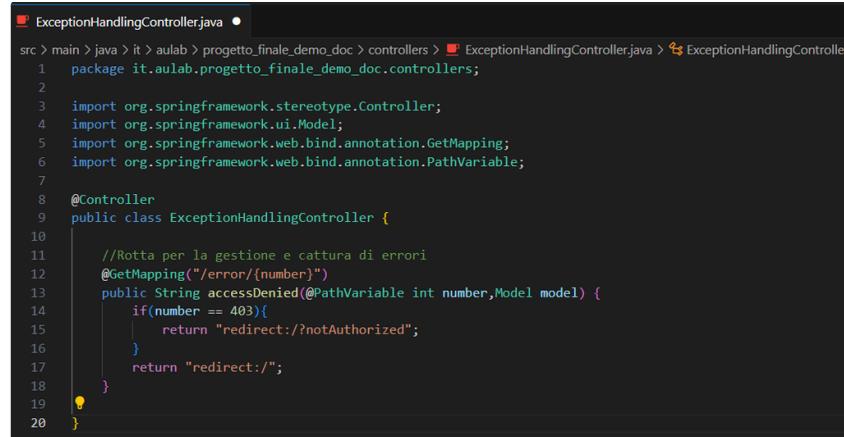
```
SecurityConfig.java
src > main > java > it > aulab > progetto_finale_demo_doc > config > SecurityConfig.java > SecurityConfig > filterChain(HttpSecurity)
1  package it.aulab.progetto_finale_demo.config;
2
3  import org.springframework.context.annotation.Bean;
4  import org.springframework.context.annotation.Configuration;
5  import org.springframework.security.config.annotation.web.builders.HttpSecurity;
6  import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
7  import org.springframework.security.config.http.SessionCreationPolicy;
8  import org.springframework.security.web.SecurityFilterChain;
9  import org.springframework.security.web.util.matcher.AntPathRequestMatcher;
10
11 @Configuration
12 @EnableWebSecurity
13 public class SecurityConfig{
14
15     @Bean
16     public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
17         http
18             .csrf(csrf -> csrf.disable())
19             .authorizeHttpRequests((authorize) ->
20                 authorize.requestMatchers(...patterns:"/register/**").permitAll()
21                 .requestMatchers(...patterns:"/register").permitAll()
22                 .anyRequest().authenticated()
23             ).formLogin(form ->
24                 form.loginPage("/login")
25                 .loginProcessingUrl(loginProcessingUrl:"/login")
26                 .defaultSuccessUrl(defaultSuccessUrl:"/")
27                 .permitAll()
28             ).logout(logout -> logout
29                 .logoutRequestMatcher(new AntPathRequestMatcher("/logout"))
30                 .permitAll()
31             ).exceptionHandling(exception -> exception.accessDeniedPage(accessDeniedUrl:"/error/403"))
32             .sessionManagement(session -> session
33                 .sessionCreationPolicy(SessionCreationPolicy.IF_REQUIRED)
34                 .maximumSessions(maximumSessions:1)
35                 .expiredUrl(expiredUrl:"/login?session-expired=true")
36             );
37         return http.build();
38     }
39 }
```

Oltre alle configurazioni di sicurezza abbiamo anche inserito un sistema di gestione delle richieste non autorizzate e di gestione delle sessioni.

## Gestione errore 403 ↗

Per completare la gestione di operazioni non autorizzate dobbiamo costruire un nuovo controller che cattura l'uri di reindirizzamento inserito nelle configurazioni “/error/403”

Creiamo un nuovo controller che chiameremo “ExceptionHandlingController” all'interno di “src\main\java\it\aulab\spec\_prog\_finale\controllers” in “src\main\java\it\aulab\progetto\_finale\_docente\controllers\ExceptionHandlingController.java”

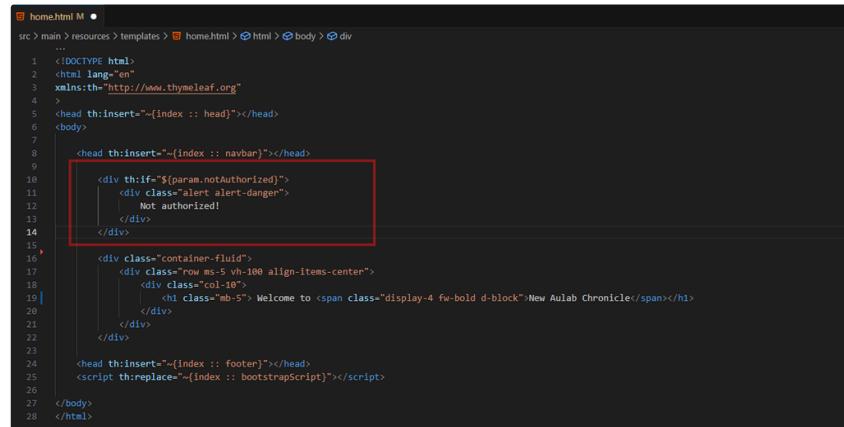


```
src > main > java > it > aulab > progetto_finale_demo_doc > controllers > ExceptionHandlingController.java > ExceptionHandlingController.java
1 package it.aulab.progetto_finale_demo_doc.controllers;
2
3 import org.springframework.stereotype.Controller;
4 import org.springframework.ui.Model;
5 import org.springframework.web.bind.annotation.GetMapping;
6 import org.springframework.web.bind.annotation.PathVariable;
7
8 @Controller
9 public class ExceptionHandlingController {
10
11     //Rotta per la gestione e cattura di errori
12     @GetMapping("/error/{number}")
13     public String accessDenied(@PathVariable int number, Model model) {
14         if(number == 403){
15             return "redirect:/notAuthorized";
16         }
17         return "redirect:/";
18     }
19 }
20 }
```

il metodo `accessDenied(@PathVariable int number,Model model)` è un metodo dinamico che riceve il codice dell'errore ed attiva delle logiche, l'unica attualmente presente se il codice è “403” reindirizza l'utente verso la pagina home con un messaggio flash che specifica di non essere autorizzato.

Il messaggio flash va aggiunto proprio alla pagina `home` quindi aggiungiamo

in “src\main\resources\templates\home.html”



```
src > main > resources > templates > home.html > home.html > body > div
...
1 <!DOCTYPE html>
2 <html lang="en">
3     <head th:insert=~{index :: head}></head>
4     <body>
5
6         <head th:insert=~{index :: navbar}></head>
7
8             <div th:if="${param.notAuthorized}">
9                 <div class="alert alert-danger">
10                     Not authorized!
11                 </div>
12             </div>
13
14
15         <div class="container-fluid">
16             <div class="row ms-5 vh-100 align-items-center">
17                 <div class="col-10">
18                     <h1 class="mb-5">Welcome to <span class="display-4 fw-bold d-block">New Aulab Chronicle</span></h1>
19                 </div>
20             </div>
21         </div>
22
23
24         <head th:insert=~{index :: footer}></head>
25         <script th:replace=~{index :: bootstrapScript}></script>
26
27     </body>
28 </html>
```

immediatamente sotto la navbar

Arrivati a questo punto possiamo già fare un test della sola login, poiché per far funzionare il tutto correttamente abbiamo bisogno degli sviluppi successivi.

Se non avviato facciamolo e andiamo su “<http://localhost:8080/>” e vedremo che ci verrà mostrato il nostro form di login. Come username e password inseriamo sempre quelli costruiti di default da spring security.

## CustomUserDetails ↗

Nel nostro progetto gestiremo dati personalizzati , di conseguenza abbiamo bisogno della costruzione di un UserDetails e UserDetailsService personalizzato.

Creiamo allora la classe “CustomUserDetails” all'interno di “src\main\java\it\aulab\progetto\_finale\_docente\services”

in “src\main\java\it\aulab\progetto\_finale\_docente\services\CustomUserDetails.java”

```

1  CustomUserDetails.java •
src > main > java > it > aulab > progetto_finale_demo_doc > services > CustomUserDetails.java > CustomUserDetails
1   package it.aulab.progetto_finale_demo_doc.services;
2
3   import java.util.Collection;
4
5   import org.springframework.security.core.GrantedAuthority;
6   import org.springframework.security.core.userdetails.UserDetails;
7
8   import lombok.AllArgsConstructor;
9   import lombok.NoArgsConstructor;
10
11  @AllArgsConstructor
12  @NoArgsConstructor
13  public class CustomUserDetails implements UserDetails{
14
15      private Long id;
16      private String username;
17      private String email;
18      private String password;
19      private Collection< GrantedAuthority> authorities;
20
21      @Override
22      public boolean isAccountNonExpired() {
23          return true;
24      }
25
26      @Override
27      public boolean isAccountNonLocked() {
28          return true;
29      }
30
31      @Override
32      public boolean isCredentialsNonExpired() {
33          return true;
34      }
35
36      @Override
37      public boolean isEnabled() {
38          return true;
39      }
40
41      @Override
42      public Collection< GrantedAuthority> getAuthorities() {
43          return authorities;
44      }
45
46      @Override
47      public String getPassword() {
48          return password;
49      }
50
51      @Override
52      public String getUsername() {
53          return email;
54      }
55
56      public Long getId() {
57          return id;
58      }
59
60      public void setId(Long id) {
61          this.id = id;
62      }
63
64      public String getFullscreen() {
65          return username;
66      }
67
68  }

```

Questa classe contiene tutti i metodi che indicano se un utente è attivo in piattaforma poiché indicano se l'account non è scaduto, se le credenziali non sono scadute, se l'account non è bloccato e se è abilitato.

Contiene anche tutti i getter necessari al recupero delle informazioni dell'utente.

Utilizziamo anche i decoratori di **lombok** per avere i costruttori che ci servono ed importiamoli.

## CustomUserDetailsService ↗

Ora dobbiamo creare un service per gestire queste informazioni.

**CustomUserDetailsService** è una classe personalizzata ed implementa l'interfaccia **UserDetailsService**.

Quando si utilizza **CustomUserDetailsService**, è necessario configurarlo per sostituire l'implementazione predefinita. Ciò può essere fatto nella configurazione di Spring Security tramite il metodo **configureGlobal** in una classe che estende **WebSecurityConfigurerAdapter**. Più avanti faremo questa configurazione.

Andiamo quindi a creare in “src\main\java\it\aulab\progetto\_finale\_docente\services” la classe “CustomUserDetailsService.java”.

in “src\main\java\it\aulab\progetto\_finale\_docente\services\CustomUserDetailsService.java”

```

1 package it.aulab.progetto_finale_demo_doc.services;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.security.core.GrantedAuthority;
5 import org.springframework.security.core.authority.SimpleGrantedAuthority;
6 import org.springframework.security.core.userdetails.UserDetailsService;
7 import org.springframework.security.core.userdetails.UsernameNotFoundException;
8 import org.springframework.stereotype.Service;
9
10 import it.aulab.progetto_finale_demo_doc.models.User;
11 import it.aulab.progetto_finale_demo_doc.repositories.UserRepository;
12
13 import java.util.Arrays;
14 import java.util.Collection;
15
16 @Service
17 public class CustomUserDetailsService implements UserDetailsService {
18
19     @Autowired
20     private UserRepository userRepository;
21
22     @Override
23     public CustomUserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
24         User user = userRepository.findByEmail(username);
25         if(user == null) {
26             throw new UsernameNotFoundException(msg:"Invalid credentials");
27         }
28         return new CustomUserDetails(
29             user.getId(),
30             user.getUsername(),
31             user.getEmail(),
32             user.getPassword(),
33             getAuthorities()
34         );
35     }
36
37     public Collection<? extends GrantedAuthority> getAuthorities() {
38         return Arrays.asList(new SimpleGrantedAuthority(role:"user"));
39     }
40 }
41

```

Ed importiamo tutto ciò di cui abbiamo bisogno.

Da notare che abbiamo utilizzato anche una funzione **getAuthorities()** dove viene restituita una collezione di autorizzazioni (**GrantedAuthority**). Queste autorizzazioni rappresentano i permessi assegnati a un utente autenticato.

Per il momento assegniamo solo i permessi ad un utente di utente semplice "user".

## Security Config

Avendo creato la nuova gestione dell'utente aggiungiamo CustomUserDetails nel nostro security config.

in "src\main\java\it\aulab\progetto\_finale\_docente\config\SecurityConfig.java"

```

1 package it.aulab.progetto_finale_demo_doc.config;
2
3 import org.springframework.context.annotation.Configuration;
4 import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
5 import org.springframework.security.config.annotation.web.builders.HttpSecurity;
6 import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
7 import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
8
9 @Configuration
10 @EnableWebSecurity
11 public class SecurityConfig extends WebSecurityConfigurerAdapter {
12
13     @Autowired
14     private CustomUserDetailsService customUserDetailsService;
15
16     @Bean
17     public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
18         http
19             .csrf(csrf -> csrf.disable())
20             .authorizeHttpRequests(authorize ->
21                 authorize.requestMatchers(...,patterns:"/register/**").permitAll()
22                 .requestMatchers(...,patterns:"/register").permitAll()
23                 .anyRequest().authenticated()
24             )
25             .formLogin(form ->
26                 form.loginPage(loginPage:"/login")
27                 .loginProcessingUrl(loginProcessingUrl:"/login")
28                 .defaultSuccessUrl(defaultSuccessUrl:"/")
29                 .permitAll()
30             )
31             .logout(logout -> logout
32                 .logoutRequestMatcher(new AntPathRequestMatcher(pattern:"/logout"))
33                 .permitAll()
34             )
35             .exceptionHandling(exception -> exception.accessDeniedPage(accessDeniedUrl:"/error/403"))
36             .sessionManagement(session -> session
37                 .sessionCreationPolicy(SessionCreationPolicy.IF_REQUIRED)
38                 .maximumSessions(maximumSessions:1)
39                 .expiredUrl(expiredUrl:"/login?session-expired=true")
40             );
41         return http.build();
42     }
43
44     @Autowired
45     public void configureGlobal(AuthenticationManagerBuilder auth)
46     throws Exception {
47         auth.userDetailsService(customUserDetailsService)
48             .passwordEncoder(passwordEncoder);
49     }
50 }
51

```

Ed importiamo quello che ci serve.

Se ci viene mostrato un errore su passwordEncoder, verrà risolto tra poco.

Il metodo configureGlobal come anticipato configura lo userDetailService nel nostro progetto ed utilizza un passwordEncoder che però vediamo andare in errore.

E' arrivato il momento di configurare diversamente il nostro encoder delle password

La funzione passwordEncoder che abbiamo creato all'interno di **UserServiceImpl** possiamo spostarla all'interno della classe **"ProgettoFinaleDocenteApplication"**, la nostra classe principale e trasformarla in un bean.

Questa classe essendo decorata con **@SpringBootApplication** comprende anche l'annotation **@Configuration**

Di conseguenza possiamo inserire la funzione di encoding trasformata in un @bean così da poterla utilizzare e richiamare anche in altre classi tramite injection.

in "src\main\java\it\aulab\progetto\_finale\_docente\ProgettoFinaleDocenteApplication.java"

```
ProgettoFinaleDocApplication.java
src > main > java > it > aulab > progetto_finale_demo_doc > ProgettoFinaleDocApplication.java > ProgettoFinaleDocApplication
1 package it.aulab.progetto_finale_demo_doc;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.context.annotation.Bean;
6 import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
7 import org.springframework.security.crypto.password.PasswordEncoder;
8
9 @SpringBootApplication
10 public class ProgettoFinaleDocApplication {
11
12     Run | Debug
13     public static void main(String[] args) {
14         SpringApplication.run(ProgettoFinaleDocApplication.class, args);
15     }
16
17     @Bean
18     public PasswordEncoder passwordEncoder() {
19         return new BCryptPasswordEncoder();
20     }
21 }
22
```

Ed importiamo quello che ci serve.

Quindi adesso possiamo fare le modifiche che ci servono

in "src\main\java\it\aulab\progetto\_finale\_docente\config\SecurityConfig.java"

```
SecurityConfig.java
src > main > java > it > aulab > progetto_finale_demo_doc > config > SecurityConfig.java > SecurityConfig > filterChain(HttpSecurity)
1 import org.springframework.security.web.SecurityFilterChain;
2 import org.springframework.security.web.util.matcher.AntPathRequestMatcher;
3
4 import it.aulab.progetto_finale_demo_doc.services.CustomUserDetailsService;
5
6 @Configuration
7 @EnableWebSecurity
8 public class SecurityConfig{
9
10     @Autowired
11     private CustomUserDetailsService customUserDetailsService;
12
13     @Autowired
14     private PasswordEncoder passwordEncoder;
15
16     @Bean
17     public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
18         http
19             .csrf(csrf -> csrf.disable())
20             .authorizeHttpRequests(authorize) ->
21                 authorize.requestMatchers(..., patterns:"/register/**").permitAll()
22                 .requestMatchers(..., patterns:"/register").permitAll()
23                 .anyRequest().authenticated()
24             .formLogin(form ->
25                 form.loginPage("/login")
26                 .loginProcessingUrl(loginProcessingUrl:"/login")
27                 .defaultSuccessUrl(defaultSuccessUrl:"/")
28                 .permitAll()
29             ).logout(logout -> logout
30                 .logoutRequestMatcher(new AntPathRequestMatcher(pattern:"/logout"))
31                 .permitAll()
32             ).exceptionHandling(exception -> exception.accessDeniedPage(accessDeniedUrl:"/error/403"))
33             .sessionManagement(session -> session
34                 .sessionCreationPolicy(SessionCreationPolicy.IF_REQUIRED)
35                 .maximumSessions(maximumSessions:1)
36                 .expiredUrl(expiredUrl:"/login?session-expired=true")
37             );
38         return http.build();
39     }
40
41     @Autowired
42     public void configureGlobal(AuthenticationManagerBuilder auth)
43         throws Exception {
44         auth.userDetailsService(customUserDetailsService)
45             .passwordEncoder(passwordEncoder);
46     }
47 }
48
49
50
51
52
53
54
55
56
57 }
```

E vedremo che l'errore non ci sarà più

in "src\main\java\it\aulab\progetto\_finale\_docente\services\UserServiceImpl.java", modifichiamo la funzione saveUser()

```

UserServiceImpl.java 1 x
src > main > java > it > aulab > progetto_finale_demo_doc > services > UserServiceImpl.java > saveUser(UserDto, RedirectAttributes, HttpServletRequest, HttpServletResponse)
8
9 import it.aulab.progetto_finale_demo_doc.dto.UserDto;
10 import it.aulab.progetto_finale_demo_doc.models.User;
11 import it.aulab.progetto_finale_demo_doc.repositories.UserRepository;
12 import jakarta.servlet.http.HttpServletRequest;
13 import jakarta.servlet.http.HttpServletResponse;
14
15 @Service
16 public class UserServiceImpl implements UserService {
17
18     @Autowired
19     private UserRepository userRepository;
20
21     @Autowired
22     private PasswordEncoder passwordEncoder;
23
24     @Override
25     public User findUserByEmail(String email) {
26         return userRepository.findByEmail(email);
27     }
28
29     @Override
30     public void saveUser(UserDto userDto, RedirectAttributes redirectAttributes, HttpServletRequest request, HttpServletResponse response){
31         User user = new User();
32         user.setUsername(userDto.getFirstName() + " " + userDto.getLastName());
33         user.setEmail(userDto.getEmail());
34         user.setPassword(passwordEncoder.encode(userDto.getPassword()));
35
36         userRepository.save(user);
37     }
38 }

```

Dove abbiamo migliorato il nostro passwordEncoder tramite injection.

E' arrivato finalmente il momento di provare tutta la Login e la Register e vediamo che tutto funziona correttamente.

■ Non avendo ancora configurato nulla per il logout, ci basterà inserire "<http://localhost:8080/logout>" nella barra degli url

■ Per il test non utilizziamo i link nella navbar poichè non ancora collegati

⚠ Se dovete avere problemi nella memorizzazione degli utenti controllate bene che il datasource sia correttamente costruito

## GESTIONE DEI RUOLI

Dal momento in cui abbiamo creato il nostro progetto abbiamo aggiunto l'entità **Role** che fino ad ora non abbiamo in alcun modo gestito.

Partiamo col creare il repository che ci consentirà di comunicare con il database.

Nel path "`src\main\java\it\aulab\progetto_finale_docente\repositories`" creiamo la classe "`RoleRepository.java`"

in "`src\main\java\it\aulab\progetto_finale_docente\repositories\RoleRepository.java`"

```

RoleRepository.java X
src > main > java > it > aulab > progetto_finale_demo_doc > repositories > RoleRepository.java > RoleRepository
1 package it.aulab.progetto_finale_demo_doc.repositories;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4
5 import it.aulab.progetto_finale_demo_doc.models.Role;
6
7 public interface RoleRepository extends JpaRepository<Role, Long> {
8     Role findByName(String name);
9 }

```

Dove abbiamo inserito la derived query per la ricerca tramite il nome del ruolo.

Ed importiamo ciò che ci serve.

Una volta creato il nostro repository andiamo ad utilizzarlo per la funzione di salvataggio dell' utente,

in "`src\main\java\it\aulab\progetto_finale_docente\services\UserServiceImpl.java`"

```

1  import java.util.List;
2
3  @Service
4  public class UserServiceImpl implements UserService {
5
6      @Autowired
7      private UserRepository userRepository;
8
9      @Autowired
10     private RoleRepository roleRepository;
11
12     @Autowired
13     private PasswordEncoder passwordEncoder;
14
15     @Override
16     public User findUserByEmail(String email) {
17         return userRepository.findByEmail(email);
18     }
19
20     @Override
21     public void saveUser(UserDTO userDTO, RedirectAttributes redirectAttributes, HttpServletRequest request, HttpServletResponse response){
22         User user = new User();
23         user.setUsername(userDTO.getFirstName() + " " + userDTO.getLastName());
24         user.setEmail(userDTO.getEmail());
25         user.setPassword(passwordEncoder.encode(userDTO.getPassword()));
26
27         Role role = roleRepository.findByName(name: "ROLE_USER");
28         user.setRoles(List.of(role));
29
30         userRepository.save(user);
31     }
32 }

```

Ed importiamo quello che serve, **occhio ad importare per list quello di java.utils**

Iniziamo prima di tutto con l'injection del nostro repository.

- Tramite il metodo **findByName** troviamo il ruolo per poi assegnarlo allo user.
- Il metodo **setRoles** di user assegna un elenco di ruoli all'utente ma in prima battuta un utente entra con ruolo User

Se procediamo con una registrazione, osserveremo che verrà creato un nuovo record nella tabella users. Inoltre, verrà aggiunta una nuova coppia di "id" nella tabella users\_roles.

user_id	role_id
17	2

Questa tabella osserva quella che è la relazione impostata all'interno del modello User

Collega quindi l'id di uno user all'id di un ruolo specifico creando così una relazione tra i due.

## MODIFICA DEL SERVICE ↗

Dobbiamo fare un piccolo step. Avendo aggiunto i ruoli dobbiamo modificare la classe UserDetailsService aggiungendo la possibilità di mappare e caricare tutti i ruoli in base alle Authorities.

In "src/main/java/it/aulab/progetto\_finale\_docente/services/CustomUserDetailsService.java"

```

1  import java.util.Arrays;
2  import java.util.Collection;
3  import java.util.stream.Collectors;
4
5  @Service
6  public class CustomUserDetailsService implements UserDetailsService {
7
8      @Autowired
9      private UserRepository userRepository;
10
11     @Override
12     public CustomUserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
13         User user = userRepository.findByEmail(username);
14         if(user == null) {
15             throw new UsernameNotFoundException("Invalid credentials");
16         }
17         return new CustomUserDetails(
18             user.getId(),
19             user.getUsername(),
20             user.getEmail(),
21             user.getPassword(),
22             mapRolesToAuthorities(user.getRoles())
23         );
24     }
25
26     private Collection<? extends GrantedAuthority> mapRolesToAuthorities(Collection<Role> roles) {
27         Collection<? extends GrantedAuthority> mapRoles = null;
28         if(roles.size() != 0){
29             mapRoles = roles.stream()
30                 .map(role -> new SimpleGrantedAuthority(role.getName()))
31                 .collect(Collectors.toList());
32         }
33         else{
34             mapRoles = Arrays.asList(new SimpleGrantedAuthority("ROLE_USER"));
35         }
36         return mapRoles;
37     }
38 }

```

Abbiamo praticamente sostituito la funzione "getAuthorities()" con "mapRolesToAuthorities(user.getRoles())", questa funzione come dice la parola stessa mapperà i ruoli collegati al nostro utente.

Ed importiamo quello che serve.

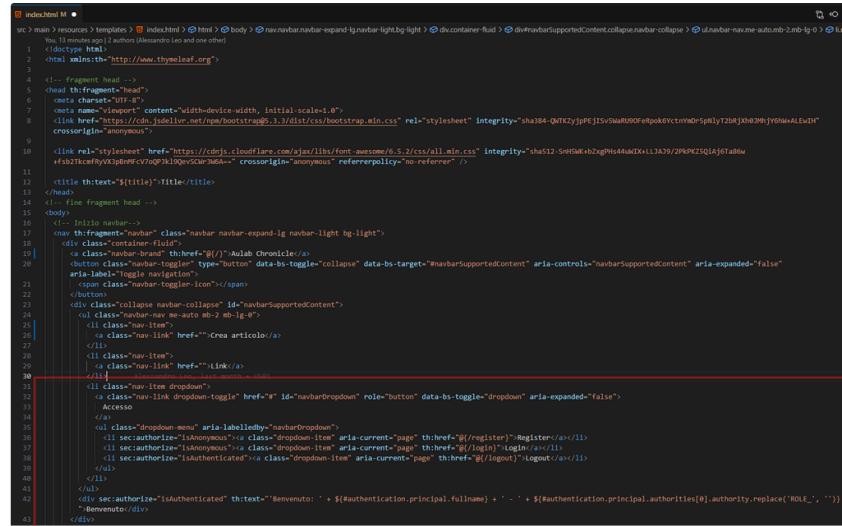
La funzione include anche un controllo per assegnare un ruolo predefinito agli utenti che non sono associati a nessun ruolo specifico. A questi utenti viene quindi assegnata l'autorizzazione pari a quella di uno "user".

## PERSONALIZZIAMO LA NAVBAR ☺

Attualmente anche se provassimo a registrarci o loggarci non riusciremmo a vedere un vero e proprio feedback nel browser dobbiamo quindi migliorare la nostra UX del progetto, possiamo notare che sulla navbar i collegamenti alla login/register/logout non funzionano.

Andiamo quindi a personalizzare la nostra navbar in tal modo che ci dia tutti i feedback necessari e funzionino i collegamenti.

in "src/main/resources/templates/index.html"



```
<html>
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-QMkZyjPFISvMaR9QFqekYcrnDzSpklyT2hXJX03WjYvWnAExH" crossorigin="anonymous">
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.5.2/css/all.min.css" integrity="sha512-SmSmhbZxgpn4ukIXLLJA/29PMK2SQIAjGtaBfue21c2tKfYXtpdWevzNog3k1DQy5Cw3HdA" crossorigin="anonymous" referrerpolicy="no-referrer">
    <title th:text="${title}">Title</title>
  </head>
  <body>
    <nav th:fragment="navbar" class="navbar navbar-expand-lg navbar-light bg-light">
      <div class="container-fluid">
        <a class="navbar-brand" href="#">AutLab Chronicle</a>
        <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
          <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarSupportedContent">
          <ul class="navbar-nav me-auto mb-2 mb-lg-0">
            <li class="nav-item">
              <a class="nav-link" href="#">Crea articolo</a>
            </li>
            <li class="nav-item">
              <a class="nav-link" href="#">Link</a>
            </li>
            <li class="nav-item dropdown">
              <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button" data-bs-toggle="dropdown" aria-expanded="false">
                Accesso
              </a>
              <ul class="dropdown-menu" aria-labelledby="navbarDropdown">
                <li sec:authorize="isAnonymous">
                  <a class="dropdown-item" aria-current="page" href="#">Register</a>
                </li>
                <li sec:authorize="isAnonymous">
                  <a class="dropdown-item" href="#">Logout</a>
                </li>
                <li sec:authenticated="isAuthenticated">
                  <a class="dropdown-item" aria-current="page" href="#">Logout</a>
                </li>
              </ul>
            </li>
          </ul>
        </div>
        <div sec:authorize="isAuthenticated" th:text="${#authentication.principal.firstname} + ' - ' + ${#authentication.principal.authorities[0].authority.replace('ROLE_', '')}>Benvenuto</div>
      </div>
    </nav>
  </body>
</html>
```

Dove tramite la direttiva "**sec:authorize**" riusciamo a gestire la visualizzazione dei tasti in base alla caratteristica dell'utente di essere "isAnonymous" o "isAuthenticated".

Recuperiamo il nome dell'utente autenticato ed il suo ruolo tramite l'oggetto *principal* che rappresenta l'utente attivo in sessione in quel momento con

```
<div sec:authorize="isAuthenticated" th:text="${#authentication.principal.firstname} + ' - ' + ${#authentication.principal.authorities[0].authority.replace('ROLE_', '')}>Benvenuto</div>
```

Il risultato sulla navbar sarà simile a questo

Benvenuto: Robbie Russian - USER

## REDIRECT IN HOME DOPO LA REGISTER ☺

Occupiamoci adesso di configurare la redirect in home dopo aver effettuato la registrazione ma rimanendo sempre loggati.

Attualmente la registrazione è molto confusionaria e possiamo incappare in errori di utilizzo, quindi miglioriamo la nostra UX.

Gran parte della struttura è stata già impostata quello che dobbiamo aggiungere però è un modo per effettuare una login "manuale" subito dopo la register

Andiamo ad aggiungere un nuovo metodo a "UserServiceImpl"

in "src/main/java/it/aulab/progetto\_finale\_docente/services/UserServiceImpl.java"

```

UserServiceImpl.java
src > main > java > it > aulab > progetto_finale_demo_doc > services > UserServiceImpl.java > ...
1 package it.aulab.progetto_finale_demo_doc.services;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.security.authentication.AuthenticationManager;
5 import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
6 import org.springframework.security.core.Authentication;
7 import org.springframework.security.core.AuthenticationException;
8 import org.springframework.security.core.context.SecurityContextHolder;
9 import org.springframework.security.crypto.password.PasswordEncoder;
10 import org.springframework.stereotype.Service;
11 import org.springframework.web.servlet.mvc.support.RedirectAttributes;
12
13 import it.aulab.progetto_finale_demo_doc.dtos.UserDto;
14 import it.aulab.progetto_finale_demo_doc.models.Role;
15 import it.aulab.progetto_finale_demo_doc.models.User;
16 import it.aulab.progetto_finale_demo_doc.repositories.RoleRepository;
17 import it.aulab.progetto_finale_demo_doc.repositories.UserRepository;
18 import jakarta.servlet.http.HttpServletRequest;
19 import jakarta.servlet.http.HttpServletResponse;
20 import jakarta.servlet.http.HttpSession;

36     @Autowired
37     private CustomUserDetailsService customUserDetailsService;
38
39     @Autowired
40     private AuthenticationManager authenticationManager;

```

```

UserServiceImpl.java
src > main > java > it > aulab > progetto_finale_demo_doc > services > UserServiceImpl.java > authenticateUserAndSetSession(User, UserDto, HttpServletRequest)
25 public class UserServiceImpl implements UserService {
26     @Override
27     public void saveUser(UserDto userDto, RedirectAttributes redirectAttributes, HttpServletRequest request, HttpServletResponse response) {
28         User user = new User();
29         user.setUsername(userDto.getFirstName() + " " + userDto.getLastName());
30         user.setEmail(userDto.getEmail());
31         user.setPassword(passwordEncoder.encode(userDto.getPassword()));
32
33         Role role = roleRepository.findByName(name:"ROLE_USER");
34         user.setRoles(List.of(role));
35
36         userRepository.save(user);
37
38         authenticateUserAndSetSession(user, userDto, request);
39     }
40 }

```

```

UserServiceImpl.java
src > main > java > it > aulab > progetto_finale_demo_doc > services > UserServiceImpl.java > authenticateUserAndSetSession(User, UserDto, HttpServletRequest)
25 public class UserServiceImpl implements UserService {
26     @Override
27     public void authenticateUserAndSetSession(User user, UserDto userDto, HttpServletRequest request) {
28         try {
29             CustomUserDetails userDetails = customUserDetailsService.loadUserByUsername(user.getEmail());
30
31             UsernamePasswordAuthenticationToken authToken = new UsernamePasswordAuthenticationToken(userDetails.getUsername(), userDto.getPassword());
32
33             Authentication authentication = authenticationManager.authenticate(authToken);
34
35             SecurityContextHolder.getContext().setAuthentication(authentication);
36
37             HttpSession session = request.getSession(create:true);
38             session.setAttribute(name:"SPRING_SECURITY_CONTEXT", SecurityContextHolder.getContext());
39         } catch (AuthenticationException e) {
40             e.printStackTrace();
41         }
42     }
43 }

```

Il metodo “`authenticateUserAndSetSession`” effettua una login tramite token all’interno della sessione mantenendola attiva.

Abbiamo anche fatto una injection del bean “`AuthenticationManager`”, se provassimo a lanciare l’applicativo andrà sicuramente in errore poiché non riuscirebbe a trovare il bean iniettato.

Quindi prima di procedere andiamo all’interno del nostro “`SecurityConfig`” ed aggiungiamo il nuovo bean

in “`src\main\java\it\aulab\progetto_finale_docente\config\SecurityConfig.java`”

```

SecurityConfig.java
src > main > java > it > aulab > progetto_finale_demo_doc > config > SecurityConfig.java > authenticationManager(AuthenticationConfiguration)
1 package it.aulab.progetto_finale_demo_doc.config;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.context.annotation.Bean;
5 import org.springframework.context.annotation.Configuration;
6 import org.springframework.security.AuthenticationManager;
7 import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
8 import org.springframework.security.config.annotation.authentication.configuration.AuthenticationConfiguration;
9 import org.springframework.security.config.annotation.web.builders.HttpSecurity;
10 import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
11 import org.springframework.security.config.http.SessionCreationPolicy;
12 import org.springframework.security.crypto.password.PasswordEncoder;
13 import org.springframework.security.web.SecurityFilterChain;
14 import org.springframework.security.web.util.matcher.AntPathRequestMatcher;
15
16 import it.aulab.progetto_finale_demo_doc.services.CustomUserDetailsService;

```

```

59
60     @Bean
61     public AuthenticationManager authenticationManager(AuthenticationConfiguration authenticationConfiguration)
62         throws Exception {
63     }
64 }
```

Ultimo step è modificare la redirect della funzione di register verso la home

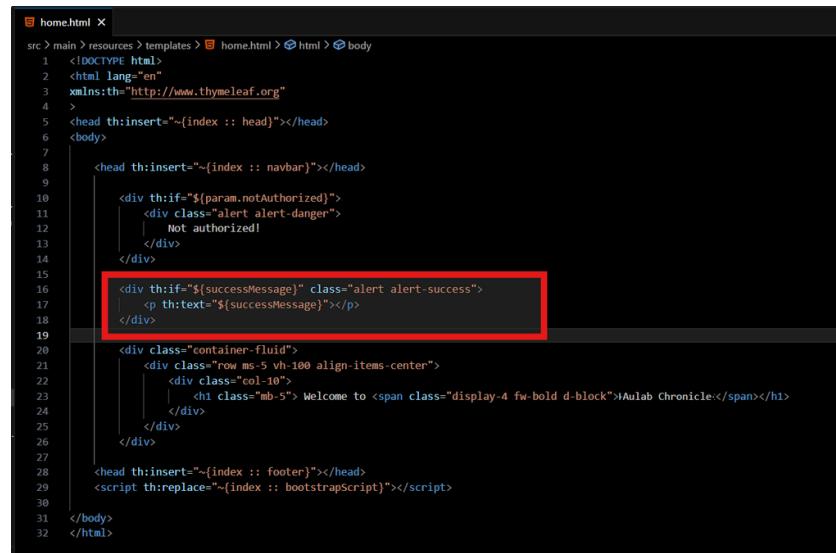
in "src\main\java\it\aulab\progetto\_finale\_docente\controllers\UserController.java"

```

45 //Rotta per il salvataggio della registrazione
46 @PostMapping("/register/save")
47 public String registration(@Valid @ModelAttribute("user") UserDto userDto,
48                             BindingResult result,
49                             Model model,
50                             RedirectAttributes redirectAttributes,
51                             HttpServletRequest request, HttpServletResponse response){
52
53     User existingUser = userService.findUserByEmail(userDto.getEmail());
54
55     if (existingUser != null & existingUser.getEmail() != null & !existingUser.getEmail().isEmpty()) {
56         result.rejectValue("email", null,
57                             "There is already an account registered with the same email");
58     }
59
60     if (result.hasErrors()) {
61         model.addAttribute("user", userDto);
62         return "auth/register";
63     }
64
65     userService.saveUser(userDto, redirectAttributes, request, response);
66
67     redirectAttributes.addFlashAttribute(attributeName:"successMessage", attributeValue:"Registrazione avvenuta!");
68
69     return "redirect:/";
70 }
71
72 }
```

Ma nel nostro "UserController" all'interno del metodo "registration" abbiamo sempre un "redirectAttributes" con un messaggio che ci conferma l'avvenuta registrazione, questo viene attualmente visualizzato nel template di register, avendo però effettuato una redirect verso la home " redirect:/" dobbiamo aggiungere il codice che mostrerà il messaggio di successo all'interno della home.

in "src\main\resources\templates\home.html"



```

1 <!DOCTYPE html>
2 <html lang="en"
3 xmlns:th="http://www.thymeleaf.org"
4 >
5 <head th:insert="~{index :: head}"></head>
6 <body>
7
8     <head th:insert="~{index :: navbar}"></head>
9
10    <div th:if="${param.notAuthorized}">
11        <div class="alert alert-danger">
12            | Not authorized!
13        </div>
14    </div>
15
16    <div th:if="${successMessage}" class="alert alert-success">
17        <p th:text="${successMessage}"></p>
18    </div>
19
20    <div class="container-fluid">
21        <div class="row ms-5 vh-100 align-items-center">
22            <div class="col-10">
23                <h1 class="mb-5 fw-bold d-block">Aulab Chronicle</h1>
24            </div>
25        </div>
26    </div>
27
28    <head th:insert="~{index :: footer}"></head>
29    <script th:replace="~{index :: bootstrapScript}"></script>
30
31 </body>
32 </html>
```

Ultimo passo sarà effettuare un test di queste funzionalità e verificare se tutto viene eseguito correttamente.

## ARTICLE E CATEGORY ☰

Impostata tutta la logica di registrazione possiamo dedicarci alla creazione e gestione degli articoli.

La tabella nel database è stata già creata in precedenza attraverso il file script sql quindi dedichiamoci a mappare ed utilizzar il tutto all'interno del nostro progetto.

Iniziamo col creare all'interno di "src\main\java\it\aulab\spec\_prog\_finale\models" la nuova classe "Article.java"

in "src\main\java\it\aulab\progetto\_finale\_docente\models\Article.java"

```

Article.java X
src > main > java > it > aulab > progetto_finale_demo_doc > models > Article.java > ...
1 package it.aulab.progetto_finale_demo_doc.models;
2
3 import java.time.LocalDate;
4
5 import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
6
7 import jakarta.persistence.Column;
8 import jakarta.persistence.Entity;
9 import jakarta.persistence.GeneratedValue;
10 import jakarta.persistence.GenerationType;
11 import jakarta.persistence.Id;
12 import jakarta.persistence.JoinColumn;
13 import jakarta.persistence.ManyToOne;
14 import jakarta.persistence.Table;
15 import jakarta.validation.constraints.NotEmpty;
16 import jakarta.validation.constraints.Size;
17 import lombok.Getter;
18 import lombok.Setter;
19 import lombok.NoArgsConstructor;
20
21 @Setter
22 @Getter
23 @NoArgsConstructor
24 @Entity
25 @Table(name = "articles")
26 public class Article {
27     @Id
28     @GeneratedValue(strategy = GenerationType.IDENTITY)
29     private Long id;
30
31     @Column(nullable = false, length = 100)
32     @NotEmpty
33     @Size(max = 100)
34     private String title;
35
36     @Column(nullable = false, length = 100)
37     @NotEmpty
38     @Size(max = 100)
39     private String subtitle;
40
41     @Column(nullable = false, length = 1000)
42     @NotEmpty
43     @Size(max = 1000)
44     private String body;
45
46     @Column(nullable = true, length = 8)
47     @NotNull
48     private LocalDate publishDate;
49
50     @ManyToOne
51     @JoinColumn(name = "user_id")
52     @JsonIgnoreProperties({"articles"})
53     private User user;
54
55     @ManyToOne
56     @JsonIgnoreProperties({"articles"})
57     private Category category;
58 }

```

Come impostato anche nella tabella in questa classe dobbiamo descrivere le relazioni che un articolo ha con gli utenti e con le categorie. Nel nostro caso un utente è collegato a più articoli ed anche una categoria.

⚠️ Ma il modello category non esiste e ci va in errore, quindi procediamo col creare la nuova classe “Category.java” in  
“src\main\java\it\aulab\progetto\_finale\_docente\models”

in “src\main\java\it\aulab\progetto\_finale\_docente\models\Category.java”

```

Category.java X
src > main > java > it > aulab > progetto_finale_demo_doc > models > Category.java > ...
1 package it.aulab.progetto_finale_demo_doc.models;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 import jakarta.persistence.Column;
7 import jakarta.persistence.Entity;
8 import jakarta.persistence.GeneratedValue;
9 import jakarta.persistence.GenerationType;
10 import jakarta.persistence.Id;
11 import jakarta.persistence.OneToMany;
12 import jakarta.persistence.Table;
13 import jakarta.validation.constraints.NotEmpty;
14 import jakarta.validation.constraints.Size;
15 import lombok.Getter;
16 import lombok.NoArgsConstructor;
17 import lombok.Setter;
18
19 @Setter
20 @Getter
21 @NoArgsConstructor
22 @Entity
23 @Table(name = "categories")
24 public class Category {
25     @Id
26     @GeneratedValue(strategy = GenerationType.IDENTITY)
27     private Long id;
28     @Column(nullable = false, length = 100)
29     @NotEmpty(message = "Il nome non deve essere vuoto")
30     @Size(max = 50)
31     private String name;
32
33     @OneToMany(mappedBy = "category")
34     private List<Article> articles = new ArrayList<Article>();
35 }

```

## ARTICLE CONTROLLER ↴

Step immediatamente successivo è quello di realizzare tutta l'infrastruttura che ci mostrerà il form per la creazione del nostro articolo.

Andiamo quindi a creare il controller per gli articoli all'interno di "src\main\java\it\aulab\progetto\_finale\_docente\controllers" con all'interno il primo handler

in "src\main\java\it\aulab\progetto\_finale\_docente\controllers\ArticleController.java"

```
ArticleController.java 1 ●
src > main > java > it > aulab > progetto_finale_demo_doc > controllers > ArticleController.java > ArticleController

1 package it.aulab.progetto_finale_demo_doc.controllers;
2
3 import org.springframework.stereotype.Controller;
4 import org.springframework.ui.Model;
5 import org.springframework.web.bind.annotation.GetMapping;
6 import org.springframework.web.bind.annotation.RequestMapping;
7
8 import it.aulab.progetto_finale_demo_doc.models.Article;
9
10 @Controller
11 @RequestMapping("/articles")
12 public class ArticleController {
13
14     //Rotta per la creazione di un articolo
15     @GetMapping("create")
16     public String articleCreate(Model viewModel) {
17         viewModel.addAttribute("title", "crea un articolo");
18         viewModel.addAttribute("article", new Article());
19         viewModel.addAttribute("categories", categoryService.readAll());
20         return "article/create";
21     }
22 }
```

Come vediamo per instaurare la relazione tra articoli e categorie abbiamo bisogno di passare tutte le categorie alla vista e ci sarà possibile farlo solo attraverso un service.

⚠ categoryService andrà in errore, non vi preoccupate sarà risolto con i passaggi successivi

## Category Service & CrudService ↴

Andiamo all'interno del folder service per la creazione della classe "CategoryService".

Prima di procedere però possiamo introdurre qualcosa di molto interessante, **la costruzione di una interfaccia generica** che verrà implementata dalle nostre interfacce specifiche e che racchiude al suo interno tutti i metodi basilari del crud.

Creiamo all'interno del folder service un nuovo file chiamato ""CrudService"

in "src\main\java\it\aulab\spec\_prog\_finale\services\CrudService.java"

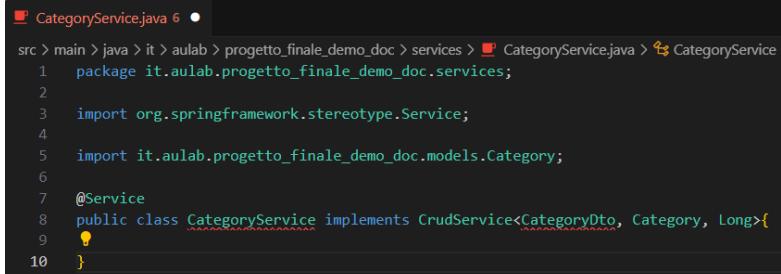
```
CrudService.java ×
src > main > java > it > aulab > progetto_finale_demo_doc > services > CrudService.java > ...
1 package it.aulab.progetto_finale_demo_doc.services;
2
3 import java.security.Principal;
4 import java.util.List;
5
6 import org.springframework.web.multipart.MultipartFile;
7
8 public interface CrudService<ReadDto, Model, Key> {
9     List<ReadDto> readAll();
10    ReadDto read(Key key);
11    ReadDto create(Model model, Principal principal, MultipartFile file);
12    ReadDto update(Key key, Model model, MultipartFile file);
13    void delete(Key key);
14 }
```

Alcuni parametri dei metodi potrebbero non essere utili per le categorie ma vedremo che li utilizzeremo tutti per gli articoli.

## CategoryService ↴

Creata quindi uno stampo generale che verrà poi di volta in volta implementato da delle classi specifiche possiamo procedere con la creazione del service "CategoryService.java" all'interno di "src\main\java\it\aulab\progetto\_finale\_docente\services"

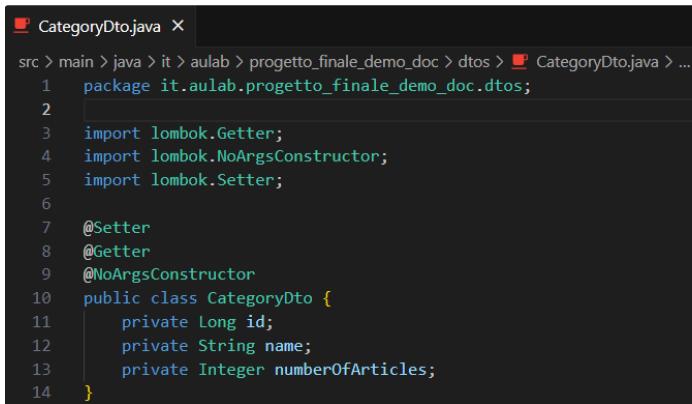
in "src\main\java\it\aulab\progetto\_finale\_docente\services\CategoryService.java"



```
CategoryService.java 6 ●
src > main > java > it > aulab > progetto_finale_demo_doc > services > CategoryService.java > CategoryService
1 package it.aulab.progetto_finale_demo_doc.services;
2
3 import org.springframework.stereotype.Service;
4
5 import it.aulab.progetto_finale_demo_doc.models.Category;
6
7 @Service
8 public class CategoryService implements CrudService<CategoryDto, Category, Long>{
9
10 }
```

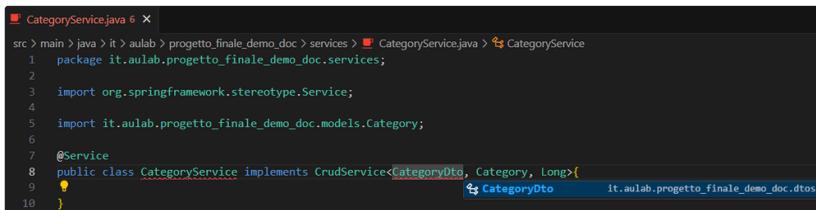
⚠ Il CategoryDto ci va in errore poiché non abbiamo un dto dedicato al dao category procediamo col crearlo all'interno di "src\main\java\it\aulab\progetto\_finale\_docente\dtos"

in "src\main\java\it\aulab\progetto\_finale\_docente\dtos\CategoryDto.java"



```
CategoryDto.java X
src > main > java > it > aulab > progetto_finale_demo_doc > dtos > CategoryDto.java > ...
1 package it.aulab.progetto_finale_demo_doc.dtos;
2
3 import lombok.Getter;
4 import lombok.NoArgsConstructor;
5 import lombok.Setter;
6
7 @Setter
8 @Getter
9 @NoArgsConstructor
10 public class CategoryDto {
11     private Long id;
12     private String name;
13     private Integer numberofarticles;
14 }
```

Fatto questo possiamo importare anche il **dto** appena creato all'interno del nostro service.



```
CategoryService.java 6 ●
src > main > java > it > aulab > progetto_finale_demo_doc > services > CategoryService.java > CategoryService
1 package it.aulab.progetto_finale_demo_doc.services;
2
3 import org.springframework.stereotype.Service;
4
5 import it.aulab.progetto_finale_demo_doc.models.Category;
6
7 @Service
8 public class CategoryService implements CrudService<CategoryDto, Category, Long>{ CategoryDto it.aulab.progetto_finale_demo_doc.dtos
9
10 }
```

⚠ Il service continua a darci errore poiché ha bisogno che vengano implementati tutti i metodi del **CrudService**, quindi procediamo col completare il service come richiesto.

in "src\main\java\it\aulab\progetto\_finale\_docente\services\CategoryService.java"

```

CategoryService.java ●
src > main > java > it > aulab > progetto_finale_demo_doc > services > CategoryService.java > CategoryService
1 package it.aulab.progetto_finale_demo_doc.services;
2
3 import java.security.Principal;
4 import java.util.List;
5
6 import org.springframework.stereotype.Service;
7 import org.springframework.web.multipart.MultipartFile;
8
9 import it.aulab.progetto_finale_demo_doc.dtos.CategoryDto;
10 import it.aulab.progetto_finale_demo_doc.models.Category;
11
12 @Service
13 public class CategoryService implements CrudService<CategoryDto, Category, Long>{
14
15     @Override
16     public List<CategoryDto> readAll() {
17         // TODO Auto-generated method stub
18         throw new UnsupportedOperationException(message:"Unimplemented method 'readAll'");
19     }
20
21     @Override
22     public CategoryDto read(Long key) {
23         // TODO Auto-generated method stub
24         throw new UnsupportedOperationException(message:"Unimplemented method 'read'");
25     }
26
27     @Override
28     public CategoryDto create(Category model, Principal principal, MultipartFile file) {
29         // TODO Auto-generated method stub
30         throw new UnsupportedOperationException(message:"Unimplemented method 'create'");
31     }
32
33     @Override
34     public CategoryDto update(Long key, Category model, MultipartFile file) {
35         // TODO Auto-generated method stub
36         throw new UnsupportedOperationException(message:"Unimplemented method 'update'");
37     }
38
39     @Override
40     public void delete(Long key) {
41         // TODO Auto-generated method stub
42         throw new UnsupportedOperationException(message:"Unimplemented method 'delete'");
43     }
44
45 }

```

Fatto questo andiamo quindi ad implementare il metodo **readAll()**

in "src\main\java\it\aulab\progetto\_finale\_docente\services\CategoryService.java"

```

CategoryService.java 3 ●
src > main > java > it > aulab > progetto_finale_demo_doc > services > CategoryService.java > CategoryService > readAll()
1 package it.aulab.progetto_finale_demo_doc.services;
2
3 import java.security.Principal;
4 import java.util.List;
5
6 import org.springframework.stereotype.Service;
7 import org.springframework.web.multipart.MultipartFile;
8
9 import it.aulab.progetto_finale_demo_doc.dtos.CategoryDto;
10 import it.aulab.progetto_finale_demo_doc.models.Category;
11
12 @Service
13 public class CategoryService implements CrudService<CategoryDto, Category, Long>{
14
15     @Override
16     public List<CategoryDto> readAll() {
17         List<CategoryDto> dtos = new ArrayList<CategoryDto>();
18         for(Category category: categoryRepository.findAll()){
19             dtos.add(modelMapper.map(category, CategoryDto.class));
20         }
21
22     }
23

```

Ma vediamo che questo metodo richiede l'utilizzo di un repository e di un model mapper che attualmente non abbiamo, procediamo quindi con la creazione ed implementazione.

Andiamo quindi all'interno di "src\main\java\it\aulab\progetto\_finale\_docente\repositories" e creiamo la nostra classe "CategoryRepository.java"

in "src\main\java\it\aulab\progetto\_finale\_docente\repositories\CategoryRepository.java"

```

CategoryRepository.java ●
src > main > java > it > aulab > progetto_finale_demo_doc > repositories > CategoryRepository.java > ...
1 package it.aulab.progetto_finale_demo_doc.repositories;
2
3 import org.springframework.data.repository.ListCrudRepository;
4
5 import it.aulab.progetto_finale_demo_doc.models.Category;
6
7 public interface CategoryRepository extends ListCrudRepository<Category, Long>{
8
9 }

```

Ed importiamo quello che ci serve.

Altra cosa da implementare è proprio il **model mapper** che sappiamo andrà messo all'interno di una classe di configurazione, noi inseriremo il bean all'interno di "src\main\java\it\aulab\progetto\_finale\_docente\ProgettoFinaleDocenteApplication.java"

in "src\main\java\it\aulab\progetto\_finale\_docente\ProgettoFinaleDocenteApplication.java"

```

ProgettoFinaleDemoDocApplication.java ×
src > main > java > it > aulab > progetto_finale_demo_doc > ProgettoFinaleDemoDocApplication.java > ProgettoFinaleDemoDocApplication
1 package it.aulab.progetto_finale_demo_doc;
2
3 import org.modelmapper.ModelMapper;
4 import org.springframework.boot.SpringApplication;
5 import org.springframework.boot.autoconfigure.SpringBootApplication;
6 import org.springframework.context.annotation.Bean;
7 import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
8 import org.springframework.security.crypto.password.PasswordEncoder;
9
10 @SpringBootApplication
11 public class ProgettoFinaleDemoDocApplication {
12
13     Run | Debug
14     public static void main(String[] args) {
15         SpringApplication.run(ProgettoFinaleDemoDocApplication.class, args);
16     }
17
18     @Bean
19     public PasswordEncoder passwordEncoder() {
20         return new BCryptPasswordEncoder();
21     }
22
23     @Bean
24     public ModelMapper instanceModelMapper(){
25         ModelMapper mapper = new ModelMapper();
26         return mapper;
27     }
28 }
29

```

Ed importiamo quello che ci serve.

Possiamo procedere ora con l'injection all'interno del nostro service delle ultime cose che ci servono

in "src\main\java\it\aulab\progetto\_finale\_docente\services\CategoryService.java"

```

CategoryService.java ×
src > main > java > it > aulab > progetto_finale_demo_doc > services > CategoryService.java > CategoryService > modelMapper
1 package it.aulab.progetto_finale_demo_doc.services;
2
3 import java.security.Principal;
4 import javax.servlet.http.HttpServletRequest;
5 import java.util.ArrayList;
6
7 import org.modelmapper.ModelMapper;
8 import org.springframework.beans.factory.annotation.Autowired;
9 import org.springframework.stereotype.Service;
10 import org.springframework.web.multipart.MultipartFile;
11
12 import it.aulab.progetto_finale_demo_doc.dtos.CategoryDto;
13 import it.aulab.progetto_finale_demo_doc.models.Category;
14 import it.aulab.progetto_finale_demo_doc.repositories.CategoryRepository;
15
16 @Service
17 public class CategoryService implements CrudService<CategoryDto, Category, Long>{
18
19     @Autowired
20     private CategoryRepository categoryRepository;
21
22     @Autowired
23     private ModelMapper modelMapper;
24
25     @Override
26     public List<CategoryDto> readAll() {
27         List<CategoryDto> dtos = new ArrayList<CategoryDto>();
28         for(Category category: categoryRepository.findAll()){
29             dtos.add(modelMapper.map(category, destinationType:CategoryDto.class));
30         }
31     }
32     return dtos;
33 }
34

```

Ed importiamo anche "java.util.ArrayList"

Ed infine torniamo dove eravamo partiti

in "src\main\java\it\aulab\progetto\_finale\_docente\controllers\ArticleController.java"

```

ArticleController.java X
src > main > java > it > aulab > progetto_finale_demo.doc > controllers > ArticleController.java > ...
1 package it.autlab.progetto_finale_demo.controllers;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.beans.factory.annotation.Qualifier;
5 import org.springframework.stereotype.Controller;
6 import org.springframework.ui.Model;
7 import org.springframework.web.bind.annotation.GetMapping;
8 import org.springframework.web.bind.annotation.RequestMapping;
9
10 import it.autlab.progetto_finale_demo.dtos.CategoryDto;
11 import it.autlab.progetto_finale_demo.models.Article;
12 import it.autlab.progetto_finale_demo.models.Category;
13 import it.autlab.progetto_finale_demo.services.CrudService;
14
15
16 @Controller
17 @RequestMapping("/articles")
18 public class ArticleController {
19
20     @Autowired
21     @Qualifier("categoryService")
22     private CrudService<CategoryDto, Category, Long> categoryService;
23
24     //Rotta per la creazione di un articolo
25     @GetMapping("create")
26     public String articleCreate(Model viewModel) {
27         viewModel.addAttribute("title", "Crea un articolo");
28         viewModel.addAttribute("article", new Article());
29         viewModel.addAttribute("categories", categoryService.readAll());
30         return "article/create";
31     }
32 }

```

Ed importiamo quello che ci serve

## FORM CREAZIONE ARTICOLO E ROTTA POST ↗

Finalmente, dopo aver completato l'handler nel nostro controller possiamo procedere con la costruzione del template all'interno di un folder in "src\main\resources\templates" che chiameremo "article" ed al suo interno creeremo il template "create.html"

in "src\main\resources\templates\article\create.html"

```

create.html M X
src > main > resources > templates > article > create.html > html > body > div.vh-100 > div.container > div.row > div.col-12.my-5 > form > div.mb-3
You, yesterday | 2 authors (Alessandro Leo and one other)
1 <html xmlns="http://www.w3.org/1999/xhtml">
2     <head th:insert="~{index :: head}"></head>
3     <body>
4         <div th:insert="~{index :: navbar}"></div>
5         <div class="vh-100">
6             <div class="container">
7                 <div class="row">
8                     <div class="col-12 my-5 d-flex justify-content-between align-items-center">
9                         <div class="flex justify-content-center align-items-center flex-grow-1">
10                             <h1 th:text="${title} Inserisci un articolo:</h1>
11                         </div>
12                         <div class="col-3">
13                             <form action="#" method="POST" th:object="${article}" enctype="multipart/form-data">
14                                 <div class="row mb-3">
15                                     <div class="col">
16                                         <label for="title" class="form-label">Titolo:</label>
17                                         <input id="title" type="text" class="form-control" th:field="*{title}" placeholder="Inserisci un titolo...">
18                                         <p th:errors = "*{title}" class="text-danger" th:if="${#fields.hasErrors('title')}"/>
19                                     </div>
20                                     <div class="col">
21                                         <label for="subtitle" class="form-label">Sottotitolo:</label>
22                                         <input id="subtitle" type="text" class="form-control" th:field="*{subtitle}" placeholder="Inserisci un sottotitolo...">
23                                         <p th:errors = "*{subtitle}" class="text-danger" th:if="${#fields.hasErrors('subtitle')}"/>
24                                     </div>
25                                 </div>
26                                 <div class="mb-3">
27                                     <label for="body" class="form-label">Articolo:</label>
28                                     <input id="body" type="text" class="form-control" th:field="*{body}" placeholder="Inserisci un testo....">
29                                     <p th:errors = "*{body}" class="text-danger" th:if="${#fields.hasErrors('body')}"/>
30                                 </div>
31                                 <div class="mb-3">
32                                     <label for="date" class="form-label">Publish date (8 chars):</label>
33                                     <input id="date" type="date" class="form-control" th:field="*{publishDate}" placeholder="Inserisci una data...">
34                                     <p th:errors = "*{publishDate}" class="text-danger" th:if="${#fields.hasErrors('publishDate')}"/>
35                                 </div>
36                             </form>
37                         <div class="mb-3">
38                             <button type="submit" class="btn btn-success">Crea articolo</button>
39                         </div>
40                     </div>
41                 </div>
42             </div>
43         </div>
44     </body>
45     <head th:insert="~{index :: footer}"></head>
46     <script th:replace="~{index :: bootstrapScript}"></script>
47 </html>

```

All'interno di questo form abbiamo anche inserito i campi "error" per le validazioni.

Una volta creato il nostro template possiamo fare un test verso il nostro handler.

Entriamo con delle credenziali ed all'interno della barra degli url inseriamo “<http://localhost:8080/articles/create>” e vedremo il nostro template appena creato.

**⚠ ATTENZIONE:** Non possiamo ancora memorizzare nulla

Ora però notiamo di aver inserito

```
<form th:action="@{/articles}" method="POST" th:object="${article}" enctype="multipart/form-data">
```

Dove abbiamo inserito la gestione di un file immagine tramite “`enctype="multipart/form-data"`” nel form un campo di input per il file che per questa user story lasceremo non funzionante, sarà tutto abilitato nelle prossime.

Procediamo quindi con la creazione nel controller del nostro nuovo handler per lo store dell'articolo.

in “src\main\java\it\aulab\progetto\_finale\_docente\controllers\ArticleController.java”

```
ArticleController.java 1 ×
src > main > java > it > aulab > progetto_finale_demo_doc > controllers > ArticleController.java > ...
1 package it.aulab.progetto_finale_demo_doc.controllers;
2
3 import java.security.Principal;
4
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.beans.factory.annotation.Qualifier;
7 import org.springframework.stereotype.Controller;
8 import org.springframework.ui.Model;
9 import org.springframework.validation.BindingResult;
10 import org.springframework.web.bind.annotation.GetMapping;
11 import org.springframework.web.bind.annotation.ModelAttribute;
12 import org.springframework.web.bind.annotation.PostMapping;
13 import org.springframework.web.bind.annotation.RequestMapping;
14 import org.springframework.web.multipart.MultipartFile;
15 import org.springframework.web.servlet.mvc.support.RedirectAttributes;
16
17 import it.aulab.progetto_finale_demo_doc.dtos.CategoryDto;
18 import it.aulab.progetto_finale_demo_doc.models.Article;
19 import it.aulab.progetto_finale_demo_doc.models.Category;
20 import it.aulab.progetto_finale_demo_doc.services.CrudService;
21 import jakarta.validation.Valid;
22
23
24 @Controller
25 @RequestMapping("/articles")
26 public class ArticleController {
27
28     @Autowired
29     @Qualifier("categoryService")
30     private CrudService<CategoryDto, Category, Long> categoryService;
31
32     //Rotta per la creazione di un articolo
33     @GetMapping("create")
34     public String articleCreate(Model viewModel) {
35         viewModel.addAttribute("title", "Crea un articolo");
36         viewModel.addAttribute("article", new Article());
37         viewModel.addAttribute("categories", categoryService.readAll());
38         return "article/create";
39     }
40
41     //Rotta per lo store di un articolo
42     @PostMapping("store")
43     public String articleStore(@Valid @ModelAttribute("article") Article article,
44                               BindingResult result,
45                               RedirectAttributes redirectAttributes,
46                               Principal principal,
47                               MultipartFile file,
48                               Model viewModel) {
49
50         //Controllo degli errori con validazioni
51         if (result.hasErrors()) {
52             viewModel.addAttribute("title", "Crea un articolo");
53             viewModel.addAttribute("article", article);
54             viewModel.addAttribute("categories", categoryService.readAll());
55             return "article/create";
56         }
57
58         articleService.create(article, principal, file);
59         redirectAttributes.addFlashAttribute("attributeName", attributeValue:"Articolo aggiunto con successo!");
60
61         return "redirect:/";
62     }
63 }
```

Ma notiamo immediatamente che abbiamo bisogno di un service e del metodo create che riceverà in input come parametri l'articolo, l'utente da gestire ed il file da gestire.

## ARTICLE SERVICE

Andiamo allora all'interno di “src\main\java\it\aulab\progetto\_finale\_docente\services” e creiamo il nostro service “ArticleService.java”

in “src\main\java\it\aulab\progetto\_finale\_docente\services\ArticleService.java”

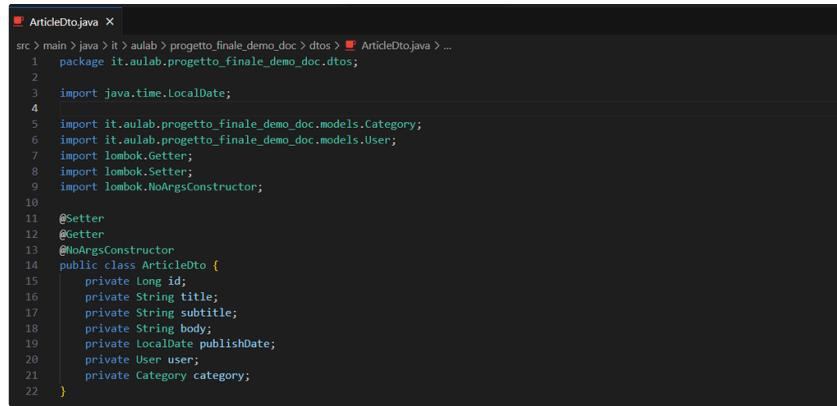
```
ArticleService.java 6 •
src > main > java > it > aulab > progetto_finale_demo_doc > services > ArticleService.java > ArticleService
1 package it.aulab.progetto_finale_demo_doc.services;
2
3 import org.springframework.stereotype.Service;
4
5 import it.aulab.progetto_finale_demo_doc.models.Article;
6
7 @Service
8 public class ArticleService implements CrudService<ArticleDto, Article, Long>{
9
10
11 }
```

Ed importiamo quello che serve

Come successo in precedenza vediamo che ArticleDto va in errore e non possiamo importarlo poichè non esiste.

Percediamo allora con la creazione all'interno di "src\main\java\it\aulab\progetto\_finale\_docente\dtos"

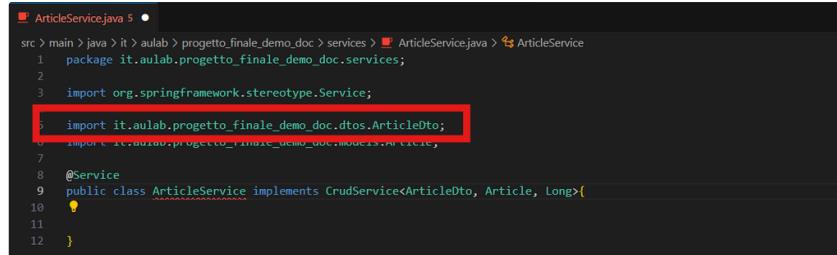
in "src\main\java\it\aulab\progetto\_finale\_docente\dtos\ArticleDto.java"



```
ArticleDto.java
src > main > java > it > aulab > progetto_finale_demo_doc > dtos > ArticleDto.java > ...
1 package it.aulab.progetto_finale_demo_doc.dtos;
2
3 import java.time.LocalDate;
4
5 import it.aulab.progetto_finale_demo_doc.models.Category;
6 import it.aulab.progetto_finale_demo_doc.models.User;
7 import lombok.Getter;
8 import lombok.Setter;
9 import lombok.NoArgsConstructor;
```

The code defines a class ArticleDto with fields id, title, subtitle, body, publishDate, user, and category. It uses Lombok annotations for getters, setters, and a no-args constructor.

Fatto questo possiamo **importare il dto nel nostro service**.



```
ArticleService.java
src > main > java > it > aulab > progetto_finale_demo_doc > services > ArticleService.java > ArticleService
1 package it.aulab.progetto_finale_demo_doc.services;
2
3 import org.springframework.stereotype.Service;
4
5 import it.aulab.progetto_finale_demo_doc.dtos.ArticleDto;
6
7
8 @Service
9 public class ArticleService implements CrudService<ArticleDto, Article, Long>{
10
11
12 }
```

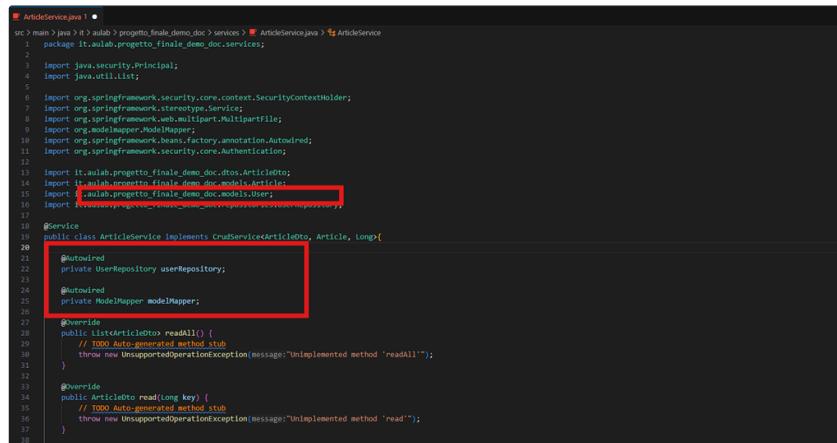
The code defines a service ArticleService that implements the CrudService interface with ArticleDto, Article, and Long as type parameters.

Completiamo poi il service inserendo tutti i metodi che andremo ad implementare

in "src\main\java\it\aulab\progetto\_finale\_docente\services\ArticleService.java"

Fatto questo implementiamo il metodo create

in "src\main\java\it\aulab\progetto\_finale\_docente\services\ArticleService.java"



```
ArticleService.java
src > main > java > it > aulab > progetto_finale_demo_doc > services > ArticleService.java > ArticleService
1 package it.aulab.progetto_finale_demo_doc.services;
2
3 import java.security.Principal;
4 import java.util.List;
5
6 import org.springframework.core.context.SecurityContextHolder;
7 import org.springframework.stereotype.Service;
8 import org.springframework.web.multipart.MultipartFile;
9 import org.modelmapper.ModelMapper;
10 import org.springframework.beans.factory.annotation.Autowired;
11 import org.springframework.security.core.Authentication;
12
13 import it.aulab.progetto_finale_demo_doc.dtos.ArticleDto;
14 import it.aulab.progetto_finale_demo_doc.models.Article;
15 import it.aulab.progetto_finale_demo_doc.models.User;
16
17
18 @Service
19 public class ArticleService implements CrudService<ArticleDto, Article, Long>{
20
21     @Autowired
22     private UserRepository userRepository;
23
24     @Autowired
25     private ModelMapper modelMapper;
26
27     @Override
28     public List<ArticleDto> readAll() {
29         // TODO Auto-generated method stub
30         throw new UnsupportedOperationException("Unimplemented method 'readAll'");
31     }
32
33     @Override
34     public ArticleDto read(long key) {
35         // TODO Auto-generated method stub
36         throw new UnsupportedOperationException("Unimplemented method 'read'");
37     }
38 }
```

The code defines a service ArticleService that implements the CrudService interface with ArticleDto, Article, and Long as type parameters. It includes imports for UserRepository and ModelMapper, and uses @Autowired annotations to inject these dependencies.

```

39     @Override
40     public ArticleDto create(Article article, Principal principal, MultipartFile file) {
41         Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
42         if (authentication != null) {
43             CustomUserDetails userDetails = (CustomUserDetails) authentication.getPrincipal();
44             User user = (userRepository.findById(userDetails.getId())).get();
45             article.setUser(user);
46         }
47
48         ArticleDto dto = modelMapper.map(articleRepository.save(article), destination:ArticleDto.class);
49
50         return dto;
51     }
52
53     @Override
54     public ArticleDto update(long key, Article model, MultipartFile file) {
55         // TODO Auto-generated method stub
56         throw new UnsupportedOperationException(message:"Unimplemented method 'update'");
57     }
58
59     @Override
60     public void delete(long key) {
61         // TODO Auto-generated method stub
62         throw new UnsupportedOperationException(message:"Unimplemented method 'delete'");
63     }
64
65 }

```

Ed importiamo quello di cui abbiamo bisogno

⚠️ NB: Per Authentication importare "org.springframework.security.core.Authentication"

## ARTICLE REPOSITORY

Questo metodo, però, come possiamo notare utilizza il metodo save() del repository "ArticleRepository" , ma il repository che erediterò il metodo non esiste.

Procediamo allora con la creazione all'interno di "src\main\java\it\aulab\progetto\_finale\_docente\repositories"

in "src\main\java\it\aulab\progetto\_finale\_docente\repositories\ArticleRepository.java"

```

ArticleRepository.java x
src > main > java > it > aulab > progetto_finale_demo_doc > repositories > ArticleRepository.java > ...
1 package it.aulab.progetto_finale_demo_doc.repositories;
2
3 import java.util.List;
4
5 import org.springframework.data.repository.ListCrudRepository;
6
7 import it.aulab.progetto_finale_demo_doc.models.Article;
8 import it.aulab.progetto_finale_demo_doc.models.Category;
9 import it.aulab.progetto_finale_demo_doc.models.User;
10
11 public interface ArticleRepository extends ListCrudRepository<Article, Long>{
12     List<Article> findByCategory(Category category);
13     List<Article> findByUser(User user);
14 }

```

Notiamo immediatamente la presenza di due metodi "derived query" che ci restituiscono una lista di articoli in base alla categoria ed una lista di articoli in base ad un utente specifico.

Li utilizzeremo all'interno del nostro progetto più avanti.

Fatto questo inseriamo l'injection nel nostro service

in "src\main\java\it\aulab\progetto\_finale\_docente\services\ArticleService.java"

```

ArticleService.java x
src > main > java > it > aulab > progetto_finale_demo_doc > services > ArticleService.java > ArticleService > userRepository
1 package it.aulab.progetto_finale_demo_doc.services;
2
3 import java.security.Principal;
4 import java.util.List;
5
6 import org.springframework.security.core.context.SecurityContextHolder;
7 import org.springframework.stereotype.Service;
8 import org.springframework.web.multipart.MultipartFile;
9 import org.modelmapper.ModelMapper;
10 import org.springframework.beans.factory.annotation.Autowired;
11 import org.springframework.security.core.Authentication;
12
13 import it.aulab.progetto_finale_demo_doc.dtos.ArticleDto;
14 import it.aulab.progetto_finale_demo_doc.models.Article;
15 import it.aulab.progetto_finale_demo_doc.models.User;
16 import it.aulab.progetto_finale_demo_doc.repositories.ArticleRepository;
17 import it.aulab.progetto_finale_demo_doc.repositories.UserRepository;
18
19 @Service
20 public class ArticleService implements CrudService<ArticleDto, Article, Long>{
21
22     @Autowired
23     private UserRepository userRepository;
24
25     @Autowired
26     private ArticleRepository articleRepository;
27
28     @Autowired
29     private ModelMapper modelMapper;
30

```

Ultimo step, importare tramite injection il service dell'articolo appena creato all'interno del controller

in "src\main\java\it\aulab\progetto\_finale\_docente\controllers\ArticleController.java"

```

ArticleController.java X
src > main > java > it.aulab > progetto_finale_demo_doc > controllers > ArticleController.java > articleCreate(Model)
1 package it.aulab.progetto_finale_demo_doc.controllers;
2
3 import java.security.Principal;
4
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.beans.factory.annotation.Qualifier;
7 import org.springframework.stereotype.Controller;
8 import org.springframework.ui.Model;
9 import org.springframework.validation.BindingResult;
10 import org.springframework.web.bind.annotation.GetMapping;
11 import org.springframework.web.bind.annotation.ModelAttribute;
12 import org.springframework.web.bind.annotation.PostMapping;
13 import org.springframework.web.bind.annotation.RequestMapping;
14 import org.springframework.web.multipart.MultipartFile;
15 import org.springframework.web.servlet.mvc.support.RedirectAttributes;
16
17 import it.aulab.progetto_finale_demo_doc.dtos.CategoryDto;
18 import it.aulab.progetto_finale_demo_doc.models.Article;
19 import it.aulab.progetto_finale_demo_doc.models.Category;
20 import it.aulab.progetto_finale_demo_doc.services.ArticleService;
21 import it.aulab.progetto_finale_demo_doc.services.CrudService;
22 import jakarta.validation.Valid;
23
24
25 @Controller
26 @RequestMapping("articles")
27 public class ArticleController {
28
29     @Autowired
30     @Qualifier("categoryService")
31     private CrudService<CategoryDto, Category, Long> categoryService;
32
33     @Autowired
34     private ArticleService articleService;

```

Possiamo fare ora finalmente un test avviando il progetto e recandoci sempre verso l'handler di creazione “<http://localhost:8080/articles/create>” e provare l'inserimento di un articolo.

Vedremo che tutto andrà a buon fine con la conferma in home tramite un messaggio, possiamo verificarlo anche all'interno del nostro database.

## LINK CREAZIONE ARTICOLO

Ora però dobbiamo far in modo che l'handler di creazione dell'articolo sia chiamato tramite un link visibile solo ad utenti che hanno effettuato l'accesso

L'unica modifica quindi è all'interno della navbar dove tramite thymeleaf controlleremo se un utente è autenticato e mostreremo di conseguenza il link stesso

in “src\main\resources\templates\index.html”

```

index.html # X
src > main > resources > templates > index.html > index.html > body > nav.navbar.navbar-expand-lg.navbar-light.bg-light > div.container-fluid > button.navbar-toggler
You, 6 minutes ago | 2 authors (Alessandro Leo and one other)
1 <html xmlns="http://www.thymeleaf.org">
2     <head th:fragment="head">
3         <meta charset="UTF-8">
4         <meta name="viewport" content="width=device-width, initial-scale=1.0">
5         <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-QWTKzyJpPEJSvSwR0OFePpokOYcnwDcSpMlyTz2RjXn0HbYV0h+ALEwIH" crossorigin="anonymous">
6
7         <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.5.2/css/all.min.css" integrity="sha512-SfQzKdXpxrXKqZyvz8vLWpqQkx0Q3kQ1Q95Hn6aU" crossorigin="anonymous" referrerpolicy="no-referrer" />
8
9     <title>${title}</title>
10    </head>
11    <!-- Fine fragment head -->
12    <body>
13        <!-- Inizio navbar -->
14        <nav class="navbar navbar-expand-lg navbar-light bg-light">
15            <div class="container-fluid">
16                <a class="navbar-brand" href="#">Aulab Chronicle</a>
17                <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
18                    <span>Alessandro Leo, last month (58)</span>
19                </button>
20                <div class="collapse navbar-collapse" id="navbarSupportedContent">
21                    <ul class="navbar-nav me-2 me-lg-0">
22                        <li class="nav-item dropdown">
23                            <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button" data-bs-toggle="dropdown" aria-expanded="false">
24                                Accesso
25                            </a>
26                            <ul class="dropdown-menu" aria-labelledby="navbarDropdown">
27                                <li sec:authorize="isAnonymous"><a class="dropdown-item" href="#">Crea articolo</a>
28                                </li>
29                                <li sec:authorize="isAuthenticated"><a class="dropdown-item" href="#">Logout</a></li>
30                            </ul>
31                        </li>
32                    </ul>
33                </div>
34            </div>
35        </nav>
36        <div sec:authorize="isAnonymous">
37            <sec:authorize><a href="#">Registrazione</a></sec:authorize>
38            <sec:authorize><a href="#">Login</a></sec:authorize>
39        </div>
40    </body>
41

```

Tramite “ sec:authorize=”isAuthenticated” ” facciamo un check sull'utente in tal modo che se ha effettuato l'accesso il link verrà visualizzato