Task -1

Login Authentication

1. Dependencies (pom.xml)

```
<dependencies>

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


  <dependency>
    <groupId>mysql</groupId>
```

```
        <artifactId>mysql-connector-java</artifactId>

        <scope>runtime</scope>

    </dependency>



    <dependency>

        <groupId>org.springframework.boot</groupId>

        <artifactId>spring-boot-starter-validation</artifactId>

    </dependency>



    <dependency>

        <groupId>org.springframework.security</groupId>

        <artifactId>spring-security-crypto</artifactId>

    </dependency>

</dependencies>
```

----

2. Database Configuration (application.properties)

properties

```
spring.datasource.url=jdbc:mysql://localhost:3306/auth_db

spring.datasource.username=root
spring.datasource.password=yourpassword

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver


spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect

spring.jpa.hibernate.ddl-auto=update
```

----

3. User Entity (User.java)

```
package com.example.auth.entity;


import jakarta.persistence.*;

import lombok.*;


@Entity

@Table(name = "users")

@Getter

@Setter

@NoArgsConstructor

@AllArgsConstructor

public class User {

    @Id

    @GeneratedValue(strategy = GenerationType.IDENTITY)

    private Long id;
    @Column(nullable = false, unique = true)
    private String username;

    @Column(nullable = false)
    private String password;

    @Column(nullable = false)
    private String role;
}
```

----

4. User Repository (UserRepository.java)

java

```
package com.example.auth.repository;

import com.example.auth.entity.User;
import org.springframework.data.jpa.repository.JpaRepository;
import java.util.Optional;

public interface UserRepository extends JpaRepository<User, Long> {
    Optional<User> findByUsername(String username);
}
```

----

5. DTOs (Request Objects)

Auth Request (AuthRequest.java)

```
package com.example.auth.dto;

import lombok.Getter;
import lombok.Setter;

@Getter
@Setter
public class AuthRequest {
    private String username;
    private String password;
}
```

Register Request (RegisterRequest.java)

```
package com.example.auth.dto;

import lombok.Getter;
import lombok.Setter;
```

```
@Getter

@Setter

public class RegisterRequest {

    private String username;

    private String password;

}
```

----

6. User Service (UserService.java)

```
package com.example.auth.service;


import com.example.auth.dto.RegisterRequest;

import com.example.auth.entity.User;

import com.example.auth.repository.UserRepository;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.stereotype.Service;

import java.util.Optional;


@Service

public class UserService {

    private final UserRepository userRepository;

    private final BCryptPasswordEncoder passwordEncoder;
    public UserService(UserRepository userRepository) {
        this.userRepository = userRepository;
        this.passwordEncoder = new BCryptPasswordEncoder();
    }

    public void registerUser(RegisterRequest request) {
        User user = new User();
        user.setUsername(request.getUsername());
        user.setPassword(passwordEncoder.encode(request.getPassword()));
        user.setRole("USER");
        userRepository.save(user);
    }

    public Optional<User> findByUsername(String username) {
        return userRepository.findByUsername(username);
    }
}
```

----

7. Security Configuration (SecurityConfig.java)

```
package com.example.auth.config;
```

```
import com.example.auth.service.UserService;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.authentication.dao.DaoAuthenticationProvider;

import
org.springframework.security.config.annotation.authentication.configuration.AuthenticationConfigurati
on;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.security.crypto.password.PasswordEncoder;

import org.springframework.security.provisioning.UserDetailsManager;

import org.springframework.security.core.userdetails.UserDetailsService;

import org.springframework.security.core.userdetails.User;


@Configuration

public class SecurityConfig {


  @Bean

  public PasswordEncoder passwordEncoder() {

    return new BCryptPasswordEncoder();

  }


  @Bean

  public AuthenticationManager authenticationManager(AuthenticationConfiguration config) throws
Exception {
        return config.getAuthenticationManager();
    }

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http.authorizeHttpRequests(auth -> auth
            .requestMatchers("/api/auth/**").permitAll()
            .anyRequest().authenticated()
        )
        .csrf().disable()
        .formLogin().disable();

        return http.build();
    }

}
```

----

8. Authentication Controller (AuthController.java)

```
package com.example.auth.controller;
```

```
}


import com.example.auth.dto.AuthRequest;

import com.example.auth.dto.RegisterRequest;

import com.example.auth.entity.User;

import com.example.auth.service.UserService;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

import org.springframework.security.core.Authentication;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.web.bind.annotation.*;


import java.util.Optional;


@RestController

@RequestMapping("/api/auth")

public class AuthController {

  private final UserService userService;

  private final AuthenticationManager authenticationManager;

  private final BCryptPasswordEncoder passwordEncoder;


  public AuthController(UserService userService, AuthenticationManager authenticationManager) {

    this.userService = userService;

    this.authenticationManager = authenticationManager;

    this.passwordEncoder = new BCryptPasswordEncoder();

  }
```

```
    @PostMapping("/register")
    public String register(@RequestBody RegisterRequest request) {
        userService.registerUser(request);
        return "User registered successfully!";
    }

    @PostMapping("/login")
    public String login(@RequestBody AuthRequest request) {
        Optional<User> userOptional = userService.findByUsername(request.getUsername());
        if (userOptional.isPresent()) {
            User user = userOptional.get();
            if (passwordEncoder.matches(request.getPassword(), user.getPassword())) {
                return "Login successful!";
            }
        }
        return "Invalid username or password!";
    }
}
```