2.Create a javascript application in an Object Oriented way using Classes and Modules. It should also use browser storage for persistence.

Index.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="styles.css">
    <title>Task Manager</title>
</head>
<body>
    <div class="container">
        <h1>Task Manager</h1>
        <div class="task-list">
            <ul id="task-list"></ul>
        </div>
        <div class="add-task">
            <input type="text" id="task-input" placeholder="Add a new task">
            <button id="add-button">Add</button>
        </div>
    </div>
    <script type="module" src="app.js"></script>
</body>
</html>
```

// task.js

```javascript
export class Task {
    constructor(id, text) {
        this.id = id;
        this.text = text;
    }
}
```

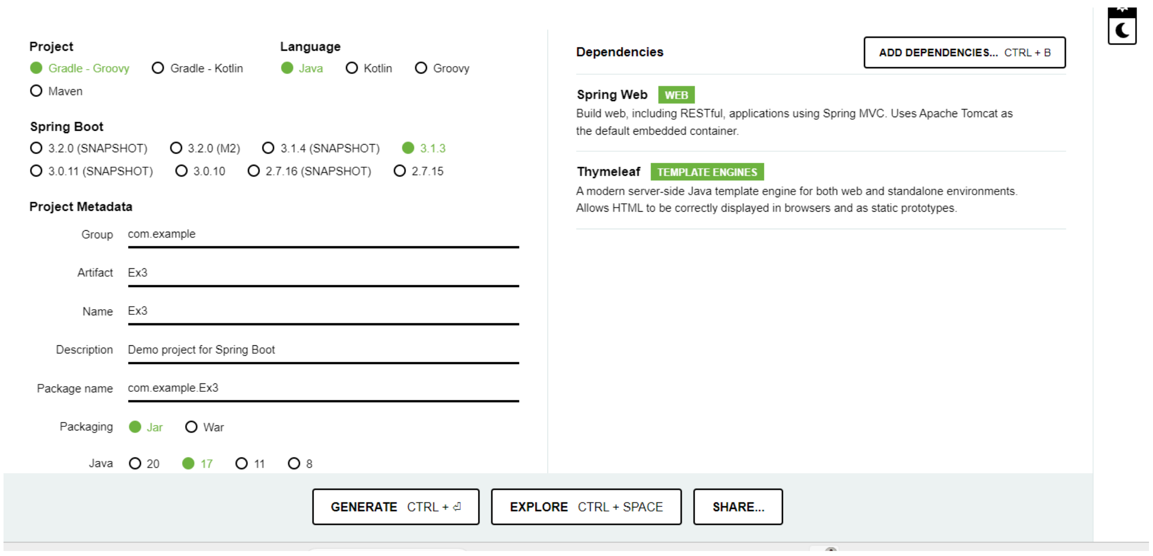
// app.js

```
import { Task } from './task.js';

class TaskManager {
   constructor() {
      this.tasks = JSON.parse(localStorage.getItem('tasks')) || [];
      this.taskList = document.getElementById('task-list');
      this.taskInput = document.getElementById('task-input');
      this.addButton = document.getElementById('add-button');

      this.addButton.addEventListener('click', this.addTask.bind(this));
      this.renderTasks();
   }

   addTask() {
      const taskText = this.taskInput.value.trim();
      if (taskText === '') return;

      const taskId = new Date().getTime();
      const task = new Task(taskId, taskText);

      this.tasks.push(task);
      this.saveTasks();
      this.renderTasks();

      this.taskInput.value = '';
   }

   saveTasks() {
      localStorage.setItem('tasks', JSON.stringify(this.tasks));
   }

   renderTasks() {
      this.taskList.innerHTML = '';
      this.tasks.forEach(task => {
         const li = document.createElement('li');
         li.innerHTML = `<span>${task.text}</span><button
data-id="${task.id}">Delete</button>`;
         this.taskList.appendChild(li);
         li.querySelector('button').addEventListener('click', this.deleteTask.bind(this));
      });
   }
```

```
    deleteTask(event) {
        const taskId = parseInt(event.target.getAttribute('data-id'));
        this.tasks = this.tasks.filter(task => task.id !== taskId);
        this.saveTasks();
        this.renderTasks();
    }
}

const taskManager = new TaskManager();
```

EX-3

1. Generate spring project with required dependencies



Project
[✓ Gradle - Groovy] Gradle - Kotlin
Maven

Language
[✓ Java] Kotlin Groovy

Spring Boot
3.2.0 (SNAPSHOT) 3.2.0 (M2) 3.1.4 (SNAPSHOT) [✓ 3.1.3]
3.0.11 (SNAPSHOT) 3.0.10 2.7.16 (SNAPSHOT) 2.7.15

Project Metadata

Group	com.example
Artifact	Ex3
Name	Ex3
Description	Demo project for Spring Boot
Package name	com.example.Ex3
Packaging	[✓ Jar] War
Java	20 [✓ 17] 11 8

Dependencies

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Thymeleaf TEMPLATE ENGINES
A modern server-side Java template engine for both web and standalone environments. Allows HTML to be correctly displayed in browsers and as static prototypes.

2. Open the genetated project in IDE.

3. Create a index.html file in **“src > main > resources > templates”**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>AJAX</title>

    <style>

        *{
            font-family: arial;
            text-align:center;
        }

    </style>

</head>
<body>
<div id="main">
    <h1>Current time: <span id="time"></span></h1>
    <button onclick="getTime()">Update time</button>

</div>

<script>

    var t = document.querySelector("#time");

    const getTime = ()=>{

        fetch("/time").then(async(res)=>{

            console.log()
            t.innerHTML = await res.text();

        })
    }

</script>

</body>
</html>
```

4. Create a new package “controllers” inside the main package
5. Create a java class “WebController” inside controllers package.

```
package com.example.Ex3.controllers;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class WebController {

    @GetMapping
    public String index(){

        return "index";
    }


}
```

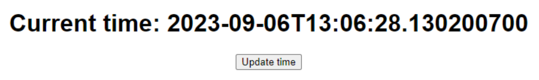
6. Create a java class “ApiController” inside controllers package

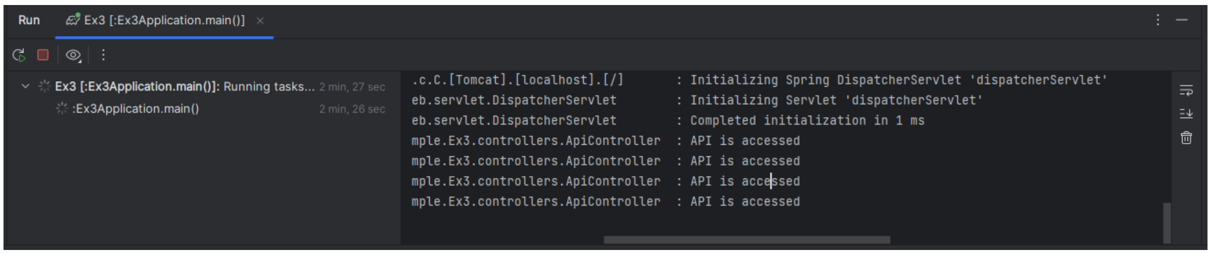
```
package com.example.Ex3.controllers;
```

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

import java.time.LocalDateTime;

@RestController()
public class ApiController {

    Logger logger = LoggerFactory.getLogger(ApiController.class);

    @GetMapping("/time")
    public String time(){
        logger.info("API is accessed : "+ LocalDateTime.now().toString());
        return LocalDateTime.now().toString();
    }

}
```

7. Run the application and access http://localhost:8080

Output

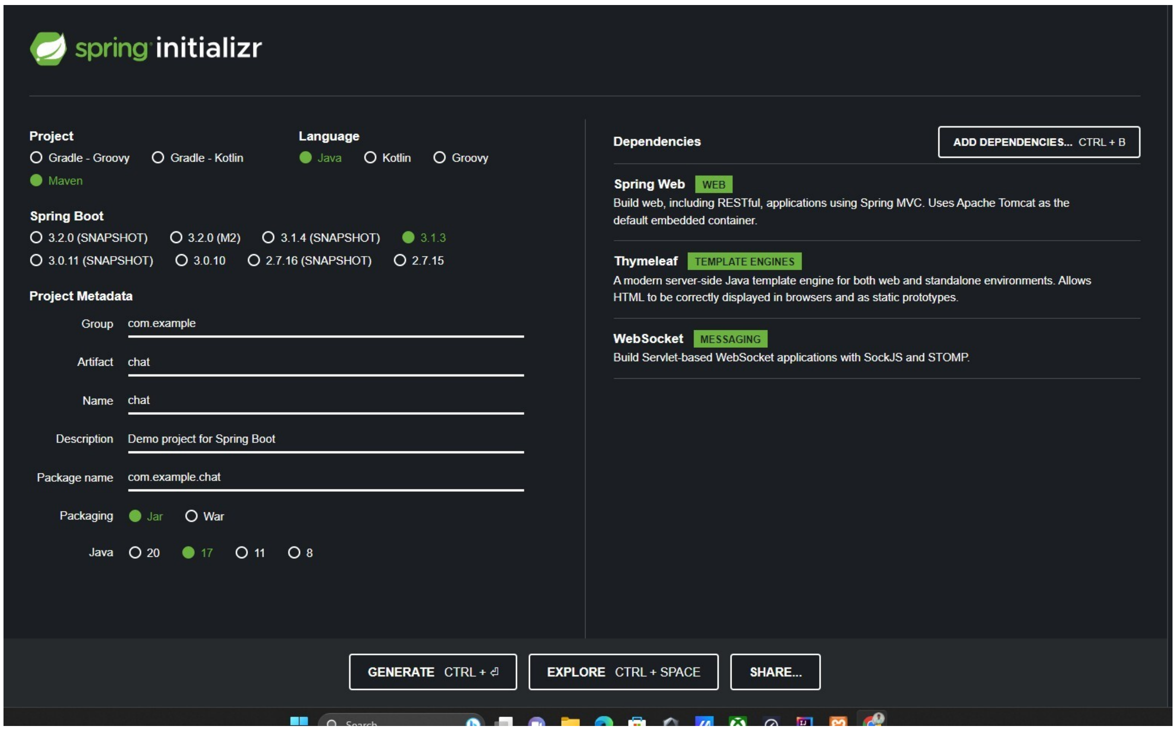

Current time: 2023-09-06T13:06:28.130200700


Run
Ex3 [:Ex3Application.main()]
Ex3 [:Ex3Application.main()]: Running tasks... 2 min, 27 sec
:Ex3Application.main() 2 min, 26 sec

```
.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
eb.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
eb.servlet.DispatcherServlet        : Completed initialization in 1 ms
mple.Ex3.controllers.ApiController  : API is accessed
mple.Ex3.controllers.ApiController  : API is accessed
mple.Ex3.controllers.ApiController  : API is accessed
mple.Ex3.controllers.ApiController  : API is accessed
```

EX-4 : Chat app with WebSocket

1. Generate spring project with required dependencies



Project
○ Gradle - Groovy ○ Gradle - Kotlin
● Maven

Language
● Java ○ Kotlin ○ Groovy

Spring Boot
○ 3.2.0 (SNAPSHOT) ○ 3.2.0 (M2) ○ 3.1.4 (SNAPSHOT) ● 3.1.3
○ 3.0.11 (SNAPSHOT) ○ 3.0.10 ○ 2.7.16 (SNAPSHOT) ○ 2.7.15

Project Metadata

Group	com.example
Artifact	chat
Name	chat
Description	Demo project for Spring Boot
Package name	com.example.chat
Packaging	● Jar ○ War
Java	○ 20 ● 17 ○ 11 ○ 8

Dependencies

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Thymeleaf TEMPLATE ENGINES
A modern server-side Java template engine for both web and standalone environments. Allows HTML to be correctly displayed in browsers and as static prototypes.

WebSocket MESSAGING
Build Servlet-based WebSocket applications with SockJS and STOMP.

2. Open the genetated project in IDE.

3. Create a index.html file in **“src > main > resources > templates”**

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>WebSocket Chat</title>
</head>
<body>
<div id="chat">
    <ul id="messages"></ul>
    <input id="messageInput" type="text" placeholder="Type your message..."
/>
    <button onclick="sendMessage()" id="sendButton">Send</button>
</div>

<script
src="https://cdn.jsdelivr.net/npm/sockjs-client@1/dist/sockjs.min.js"></
script>

<script
src="https://cdnjs.cloudflare.com/ajax/libs/stomp.js/2.3.3/stomp.min.js"
integrity="sha512-
iKDtgDyTHjAitUDdLljGhenhPwrbBfqTKWO1mkhSFH3A7blITC9MhYon6SjnMhp4o0rADGw9yAC6E
W4t5a4K3g==" crossorigin="anonymous" referrerpolicy="no-referrer"></script>
<script >

    const stompClient = Stomp.over(new SockJS('/chat'));
    stompClient.connect({}, function (frame) {
        console.log('Connected: ' + frame);
        stompClient.subscribe('/topic/public', function (message) {
        alert(0);
            showMessage(JSON.parse(message.body));
        });
    });

    function sendMessage() {
        const messageContent = document.getElementById('messageInput').value;
        const messageSender = 'User'; // You can customize the sender logic
        stompClient.send("/app/chat.sendMessage", {},
JSON.stringify({ content: messageContent, sender: messageSender }));
        document.getElementById('messageInput').value = '';
         alert(0);
    }

    function showMessage(message) {
        const messageArea = document.getElementById('messages');
        const messageElement = document.createElement('li');
        messageElement.innerHTML = '<b>' + message.sender + '</b>: ' +
message.content;
        messageArea.appendChild(messageElement);
    }


</script>
</body>
</html>
```

4. Create a new package “controllers” inside the main package
5. Create a java class “WebController” inside controllers package.

```java
package com.example.chat.controllers;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class WebController {

    @GetMapping
    public String index(){
        return "index";
    }

}
```

6. Create a java class “ChatController” inside controllers package

```java
package com.example.chat.controllers;

import com.example.chat.model.ChatMessage;
import org.springframework.messaging.handler.annotation.MessageMapping;
import org.springframework.messaging.handler.annotation.SendTo;
import org.springframework.stereotype.Controller;
```

```java
@Controller
public class ChatController {

    @MessageMapping("/chat.sendMessage")
    @SendTo("/topic/public")
    public ChatMessage sendMessage(ChatMessage chatMessage) {
        return chatMessage;
    }

}
```

7. Create a new package “configs” inside the main package
8. Create a java class “`WebSocketConfig`” inside configs package

```java
package com.example.chat.configs;

import org.springframework.context.annotation.Configuration;
import org.springframework.messaging.simp.config.MessageBrokerRegistry;
import
org.springframework.web.socket.config.annotation.EnableWebSocketMessageBroker
;
import
org.springframework.web.socket.config.annotation.StompEndpointRegistry;
import
org.springframework.web.socket.config.annotation.WebSocketMessageBrokerConfig
urer;


@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig implements WebSocketMessageBrokerConfigurer {
    @Override
    public void registerStompEndpoints(StompEndpointRegistry registry) {
        registry.addEndpoint("/chat").withSockJS();

    }

    @Override
    public void configureMessageBroker(MessageBrokerRegistry registry) {
        registry.enableSimpleBroker("/topic");
        registry.setApplicationDestinationPrefixes("/app");
    }
}
```

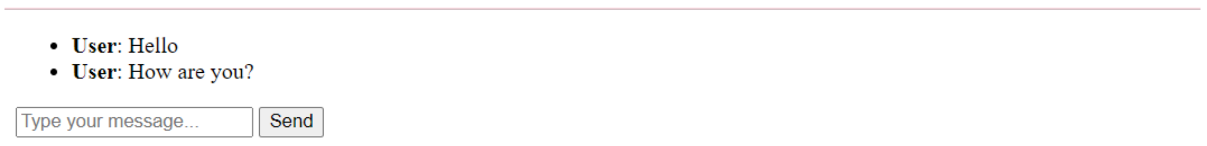
9. Create a new package “models” inside the main package

10. Create a java class “`ChatMessage`” inside models package

```java
package com.example.chat.model;

public class ChatMessage {

    private String content;
    private String sender;


    public String getContent() {
        return content;
    }

    public void setContent(String content) {
        this.content = content;
    }

    public String getSender() {
        return sender;
    }

    public void setSender(String sender) {
        this.sender = sender;
    }
}
```

11. Run the application and access http://localhost:8080

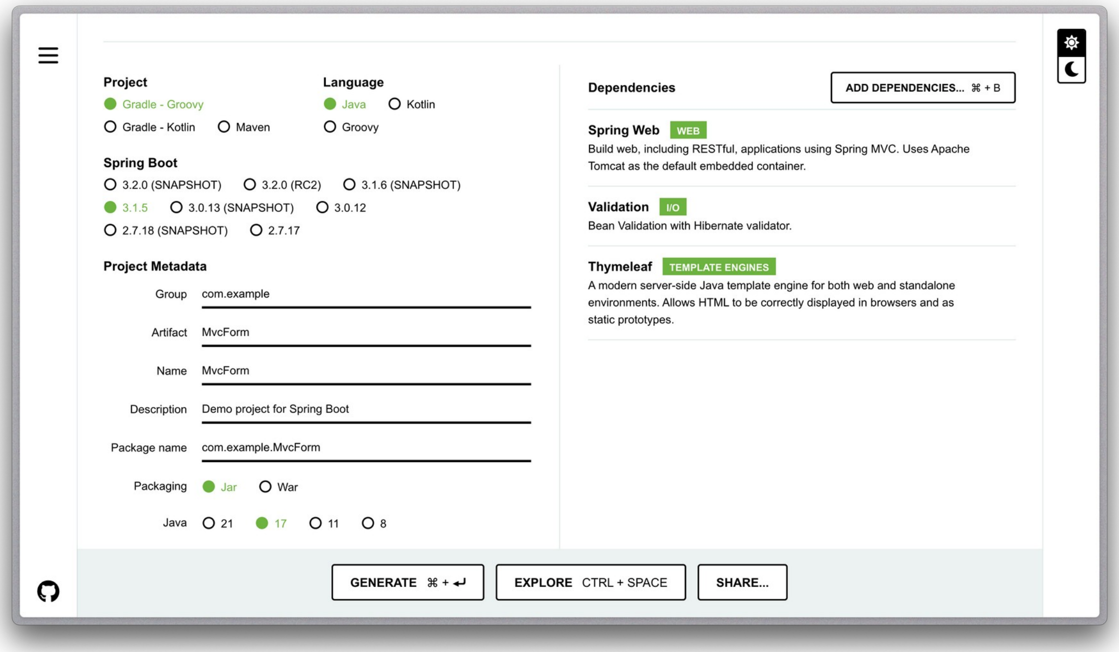
Output



- **User**: Hello
- **User**: How are you?

EX-5 : File upload and session tracking

1. Generate spring project with required dependencies



Project
[✓ Gradle - Groovy] Gradle - Kotlin Maven

Language
[✓ Java] Kotlin Groovy

Spring Boot
3.2.0 (SNAPSHOT) 3.2.0 (RC2) 3.1.6 (SNAPSHOT) [✓ 3.1.5] 3.0.13 (SNAPSHOT) 3.0.12 2.7.18 (SNAPSHOT) 2.7.17

Project Metadata

Group	com.example
Artifact	MvcForm
Name	MvcForm
Description	Demo project for Spring Boot
Package name	com.example.MvcForm
Packaging	[✓ Jar] War
Java	21 [✓ 17] 11 8

Dependencies

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Validation I/O
Bean Validation with Hibernate validator.

Thymeleaf TEMPLATE ENGINES
A modern server-side Java template engine for both web and standalone environments. Allows HTML to be correctly displayed in browsers and as static prototypes.

2. Open the generated project in IDE.

3. Create a session.html file in **“src > main > resources > templates”**

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">

<head>
<title>Session</title>
</head>
<body>
<p th:text="${page_count}"></p>
</body>

</html>
```

4. Create a file.html file in **“src > main > resources > templates”**

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<body>
<form action="#" th:action="@{/file/process}" th:object="${fileForm}" method="post" enctype="multipart/form-data">
<table>
<td>Image:</td>
<td>
<input type="file" th:field="*{file}" />
</td>
</tr>
<tr>
<td><button type="submit">Submit</button></td>
                                                          </tr>
                                                        </table>
                                                 <p th:text="${message}"></p>

</form>
</body>
</html>
```

5. Create a new package “models”

6. Create a model class FileForm inside models package

```
import org.springframework.web.multipart.MultipartFile; public class FileForm {
private MultipartFile file;


public MultipartFile getFile() { return file;
}


public void setFile(MultipartFile file) { this.file = file;
}
```

7. Create a new package “controllers”

8. Create a new class FileController inside controllers package

```
import java.io.File; import java.io.IOException;

import org.springframework.stereotype.Controller; import
org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;

import com.example.Mvc.models.FileForm;

@Controller
public class FileController {

@GetMapping("/file")
public String index(FileForm fileForm) {

return "file";
}



  @PostMapping("/file/process")
  public String uploadFile(FileForm fileForm,Model model) throws IllegalStateException, IOException {


    fileForm.getFile().transferTo(new File("/Users/oswinjerome/Projects/MCA/test.jpg"));


    model.addAttribute("message","File uploaded successfully");


    return "file";

  }

}
```

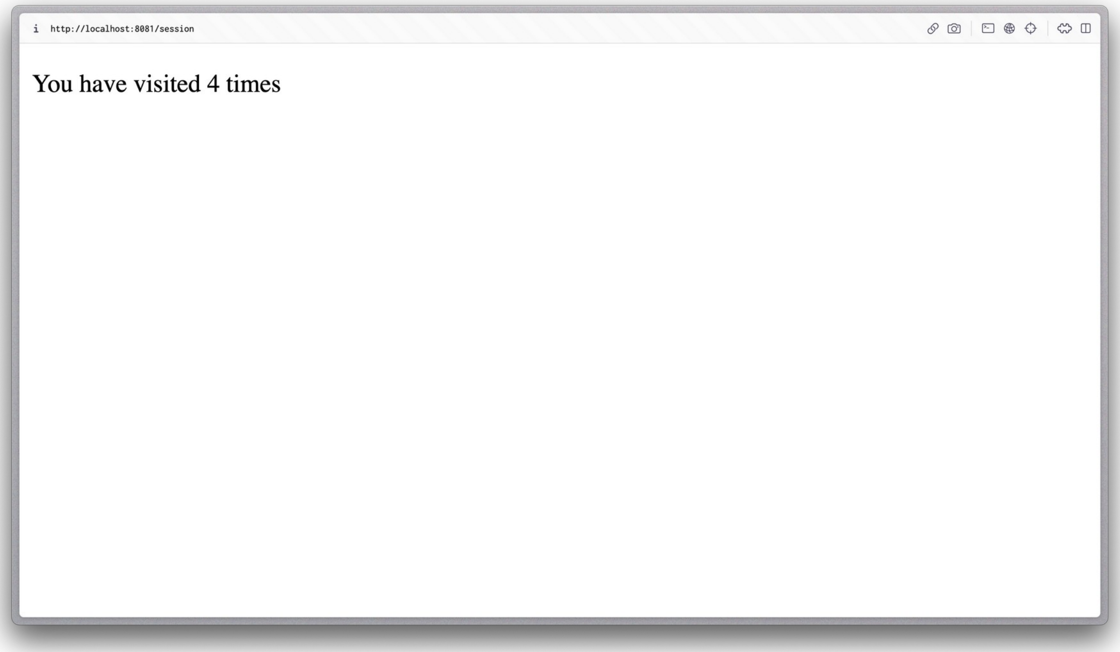
9. Create a new class SessionController inside controllers package

```
import org.springframework.stereotype.Controller; import org.springframework.ui.Model;


import org.springframework.web.bind.annotation.GetMapping; import jakarta.servlet.http.HttpSession;
@Controller
public class SessionController {
  @GetMapping("/session")
  public String index(HttpSession session,Model model) {


  int pageCount = Integer.valueOf(session.getAttribute("page_count")==null ? "0" :session.getAttribute("page_count").toString())

  session.setAttribute("page_count", pageCount + 1);
  model.addAttribute("page_count", "You have visited "+ (pageCount+1+" times"));

  return "session";
  }

}
```

10. Run the application and access
    a. http://localhost:8080/file
    b. http://localhost:8080/session

Output

1. http://localhost:8080/session


http://localhost:8081/session

You have visited 4 times

2. http://localhost:8080/file

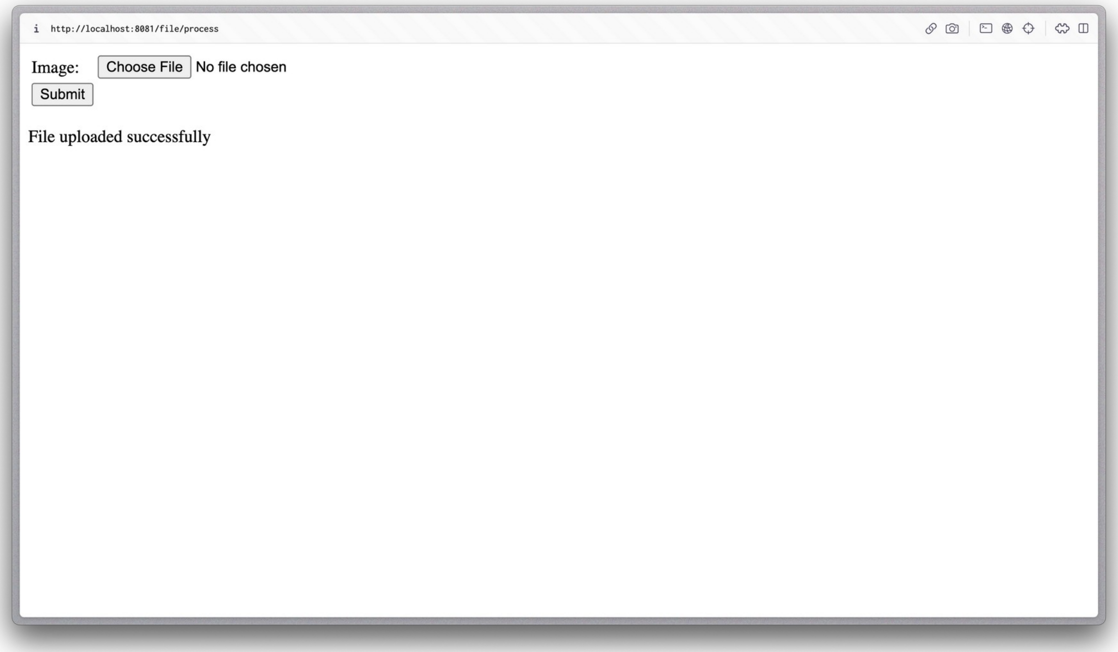

http://localhost:8081/file/process

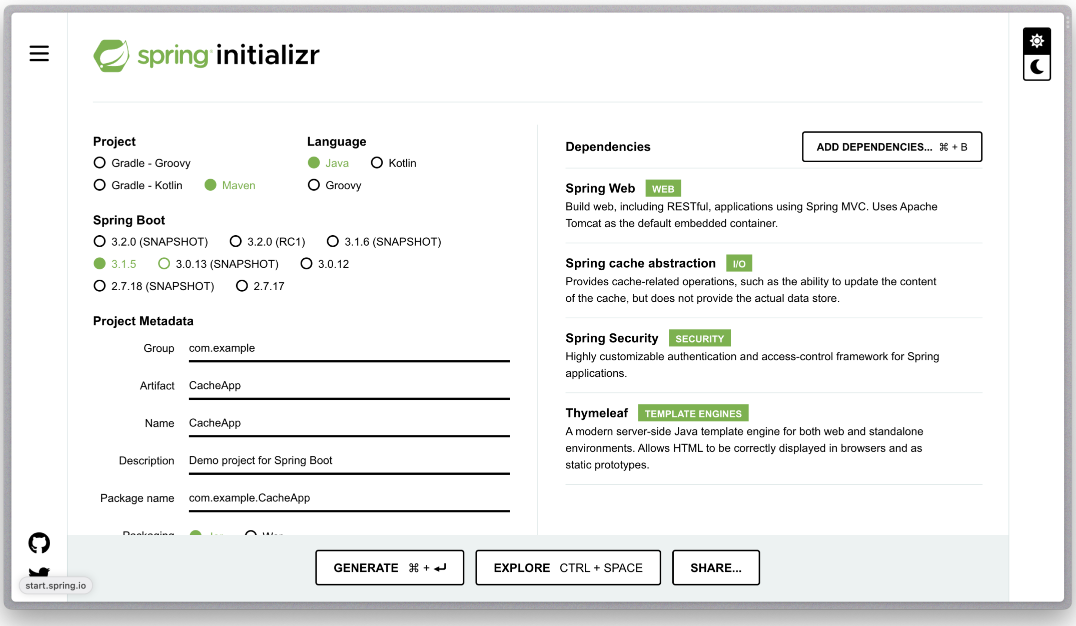
Image: Choose File No file chosen

Submit

File uploaded successfully

EX-6 : Spring Security & Cache

1. Generate spring project with required dependencies



Project
Gradle - Groovy
Gradle - Kotlin
[✓ Maven]

Language
[✓ Java]
Kotlin
Groovy

Spring Boot
3.2.0 (SNAPSHOT)
3.2.0 (RC1)
3.1.6 (SNAPSHOT)
[✓ 3.1.5]
3.0.13 (SNAPSHOT)
3.0.12
2.7.18 (SNAPSHOT)
2.7.17

Project Metadata

Group	com.example
Artifact	CacheApp
Name	CacheApp
Description	Demo project for Spring Boot
Package name	com.example.CacheApp

Dependencies

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Spring cache abstraction I/O
Provides cache-related operations, such as the ability to update the content of the cache, but does not provide the actual data store.

Spring Security SECURITY
Highly customizable authentication and access-control framework for Spring applications.

Thymeleaf TEMPLATE ENGINES
A modern server-side Java template engine for both web and standalone environments. Allows HTML to be correctly displayed in browsers and as static prototypes.

2. Open the generated project in IDE.
3. Create a login.html file in **“src > main > resources > templates”**

```
<html xmlns:th="http://www.thymeleaf.org">
  <head>
    <title>Login Page</title>
    <style>
      @import url("https://fonts.googleapis.com/css?family=Raleway:400,700");

      body {
        background: #c0c0c0;
        font-family: Raleway, sans-serif;
        color: #666;
      }

      .login {
        margin: 20px auto;
        padding: 40px 50px;
        max-width: 300px;
        border-radius: 5px;
        background: #fff;
        box-shadow: 1px 1px 1px #666;
      }
      .login input {
        width: 100%;
        display: block;
        box-sizing: border-box;
        margin: 10px 0;
        padding: 14px 12px;
        font-size: 16px;
        border-radius: 2px;
        font-family: Raleway, sans-serif;
      }

      .login input[type="text"],
      .login input[type="password"] {
        border: 1px solid #c0c0c0;
        transition: 0.2s;
      }

      .login input[type="text"]:hover {
        border-color: #f44336;
        outline: none;
        transition: all 0.2s ease-in-out;
      }

      .login input[type="submit"] {
        border: none;
        background: #ef5350;
        color: white;
        font-weight: bold;
        transition: 0.2s;
        margin: 20px 0px;
      }

      .login input[type="submit"]:hover {
        background: #f44336;
      }

      .login h2 {
        margin: 20px 0 0;
        color: #ef5350;
        font-size: 28px;
      }

      .login p {
        margin-bottom: 40px;
      }

      .links {
        display: table;
        width: 100%;
        box-sizing: border-box;
        border-top: 1px solid #c0c0c0;
        margin-bottom: 10px;
      }

      .links a {
        display: table-cell;
        padding-top: 10px;
      }
```

```
      .links a:first-child {
        text-align: left;
      }

      .links a:last-child {
        text-align: right;
      }

      .login h2,
      .login p,
      .login a {
        text-align: center;
      }

      .login a {
        text-decoration: none;
        font-size: 0.8em;
      }

      .login a:visited {
        color: inherit;
      }

      .login a:hover {
        text-decoration: underline;
      }
    </style>
  </head>
  <body>
    <form action="/" th:action="@{/login}" method="POST" class="login">
      <h2>Welcome</h2>
      <p>Please log in</p>
      <input type="text" name="username" placeholder="User Name" />
      <input type="password" name="password" placeholder="Password" />
      <input type="submit" value="Log In" />
    </form>
  </body>
</html>
```

4. Create a open.html file in **“src > main > resources > templates”**

```
<html>
  <head>
    <title>Open page</title>
  </head>
  <body>
    <h1>This is a open page</h1>
    <h2>Current time is: <span th:text="${time}"></span></h2>
  </body>
</html>
```

5. Create a protected.html file in **“src > main > resources > templates”**

```
<html>
  <head>
    <title>Protected page</title>
  </head>
  <body>
    <h1>This is a protected page</h1>
  </body>
</html>
```

6. Create a new package “configs” inside the main package
7. Create a java class “CacheConfig” inside configs package.

```
package com.example.CacheApp.config;

import org.springframework.cache.CacheManager;
import org.springframework.cache.annotation.EnableCaching;
import org.springframework.cache.concurrent.ConcurrentMapCacheManager;
import org.springframework.context.annotation.Configuration;

@Configuration
@EnableCaching
public class CacheConfig {

  public CacheManager cacheManager() {
    return new ConcurrentMapCacheManager("data");
  }

}
```

8. Create a java class “SecurityConfig” inside configs package.

```
package com.example.CacheApp.config;

import java.util.Collection;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.annotation.Order;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.User.UserBuilder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.provisioning.InMemoryUserDetailsManager;
import org.springframework.security.web.SecurityFilterChain;
import static org.springframework.security.config.Customizer.withDefaults;


@Configuration
@EnableWebSecurity
public class SecurityConfig {

  @Bean
  public UserDetailsService userDetailsService() {
    UserBuilder users = User.withDefaultPasswordEncoder();
    UserDetails user1 = User.withUsername("user1")
              .password(passwordEncoder().encode("user1"))
              .roles("USER")
              .build();


    return new InMemoryUserDetailsManager(user1);

  }

  @Order(1)
  @Bean
  public SecurityFilterChain openFilter(HttpSecurity http) throws Exception {

    http.authorizeHttpRequests(auth->auth
        .requestMatchers("/login","/open").permitAll())

    .authorizeHttpRequests(auth->auth
        .requestMatchers("/**").authenticated());

    http.formLogin(form->form.loginPage("/login").loginProcessingUrl("/login")
        .defaultSuccessUrl("/protected",true)
            .failureUrl("/login?error").permitAll());

    return http.build();
  }


  @Bean
  public PasswordEncoder passwordEncoder() {
      return new BCryptPasswordEncoder();
  }
}
```

9. Create a new package “services” inside the main package
10. Create a java class “DataService” inside services package.

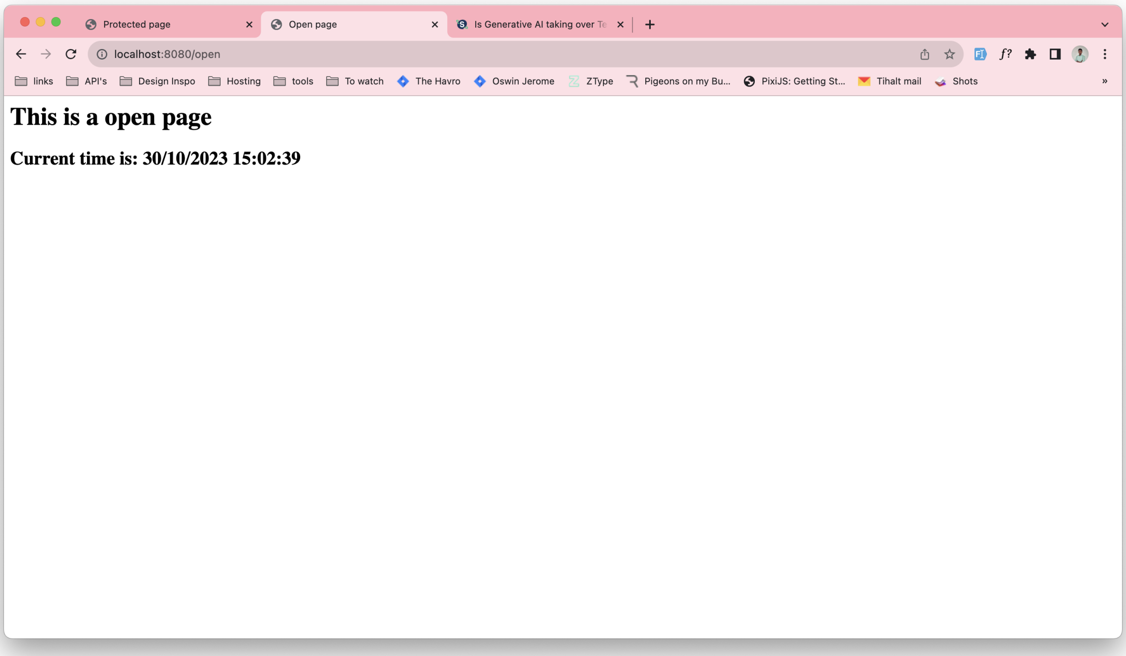
```
package com.example.CacheApp.services;

import java.text.SimpleDateFormat;
import java.util.Date;
import org.springframework.cache.annotation.Cacheable;
import org.springframework.stereotype.Service;

@Service
public class DataService {


  @Cacheable("data")
  public String getData() throws InterruptedException {

    Thread.sleep(5000);
    SimpleDateFormat formatter = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
      Date date = new Date();

    return formatter.format(date);
  }

}
```

11. Create a new package “Controllers” inside the main package
12. Create a java class “WebController” inside Controllers package.

```
package com.example.CacheApp.controllers;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.cache.CacheManager;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.view.RedirectView;

import com.example.CacheApp.services.DataService;

@Controller()
public class WebController {

  @Autowired
  DataService dataService;

  @Autowired
  CacheManager cacheManager;

  @GetMapping("/open")
  public String testPage(Model model) throws InterruptedException {
    String time = dataService.getData();
    model.addAttribute("time",time);
    return "open";
  }


  @GetMapping("/protected")
  public String protectedPage() {
    return "protected";
  }

  @GetMapping("/login")
  public String loginPage() {
    return "login";
  }

  @GetMapping("/remove_cache")
  public RedirectView removeCache() {
    cacheManager.getCache("data").clear();
    return new RedirectView("/protected");
  }

}
```

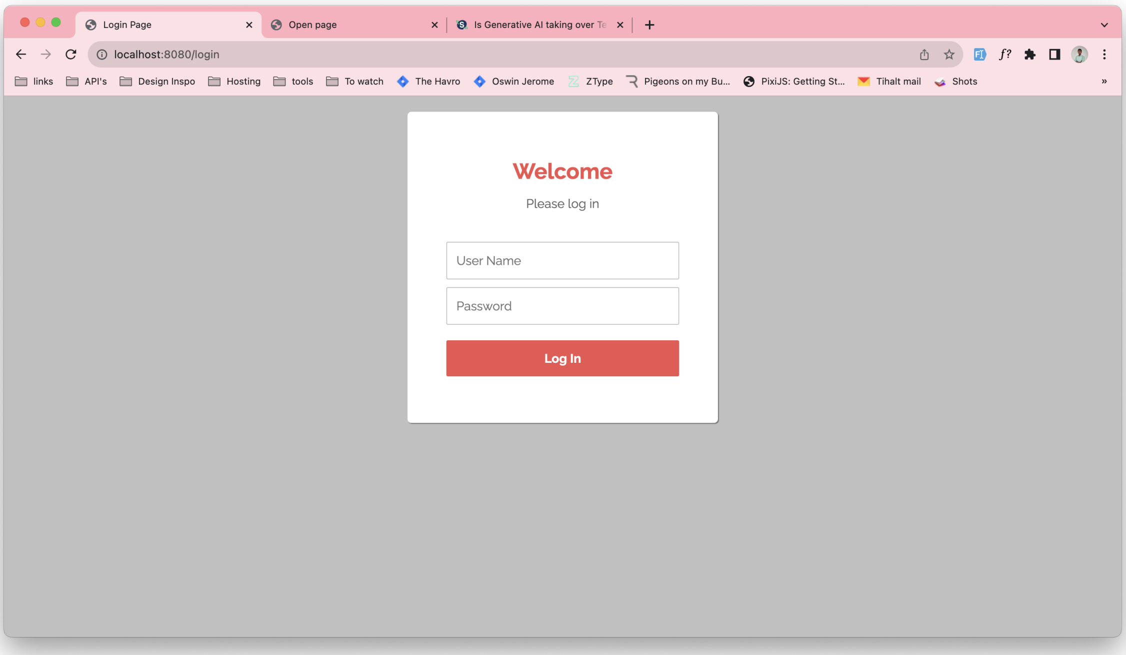
13. Run the application and access http://localhost:8080

Output



This is a open page

Current time is: 30/10/2023 15:02:39


Login Page
Open page
Is Generative AI taking over Te
localhost:8080/login
links
API's
Design Inspo
Hosting
tools
To watch
The Havro
Oswin Jerome
ZType
Pigeons on my Bu...
PixiJS: Getting St...
Tihalt mail
Shots
Welcome
Please log in
User Name
Password
Log In

Ex. 7. JPADEMO (file Name: JPADEMO)

1. start.spring.io

    Choose: Maven

    Artifact: JPADEMO

    Add Dependencies :

        MySQL Driver

        Spring Data JPA

2. Open the application "JPADEMO" in IntelliJ Framework

3. Open MySQL workbench

    i) Create Schema "student_tracker"

```
CREATE DATABASE IF NOT EXISTS `STUDENT_TRACKER`;

use student_tracker;

DROP TABLE IF EXISTS STUDENT;

create table student(

    id int NOT NULL AUTO_INCREMENT,

    first_name varchar(45) DEFAULT NULL,

    last_name varchar(45) DEFAULT NULL,

    email varchar(45) DEFAULT NULL,

    PRIMARY KEY(id)

    )
```

4. Type the below in "application.properties"

```
spring.datasource.url=jdbc:mysql://localhost:3306/student_tracker
spring.datasource.username=root
spring.datasource.password=password
```

5. Add the Shaded contents into the main application

```
package com.example.JPADEMO;

import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

@SpringBootApplication
public class JpademoApplication {

    public static void main(String[] args) {
       SpringApplication.run(JpademoApplication.class, args);
    }

    @Bean
    public CommandLineRunner commandLineRunner(String[] args)
    {
       return runner -> {
          System.out.println("Hello World");
       };
    }

}
```

6. Create new package “entity”

7. Create new class “Student” and Type the below

```
      package com.example.JPADEMO.entity;
      import jakarta.persistence.*;

      @Entity
      @Table(name="student")
      public class Student {
          @Id
          @GeneratedValue(strategy=GenerationType.IDENTITY)
          @Column(name="id")
          private int id;

          @Column(name="first_name")
          private String firstName;

          @Column(name="last_name")
          private String lastName;

          @Column(name="email")
          private String email;

          public Student() {

          }
```

```java
    public Student(String firstName, String lastName, String email) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.email = email;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    @Override
    public String toString() {
        return "Student{" +
                "id=" + id +
                ", firstName='" + firstName + '\'' +
                ", lastName='" + lastName + '\'' +
                ", email='" + email + '\'' +
                '}';
    }
}
```

8. create new package “DAO”

9. Create new interface “StudentDAO” and type the below:

```
package com.example.JPADEMO.DAO;

import com.example.JPADEMO.entity.Student;

public interface StudentDAO {
   void save(Student theStudent);
}
```

10. Create new class “StudentDAOImpl” and type the below.

```
package com.example.JPADEMO.DAO;

import com.example.JPADEMO.entity.Student;
import jakarta.persistence.EntityManager;
import org.springframework.beans.factory.annotation.Autowired;

      @Repository

      public class StudentDAOimpl implements StudentDAO {


          private EntityManager entityManager;

          @Autowired
          public StudentDAOimpl(EntityManager theEntityManager)
          {
              entityManager=theEntityManager;
          }

      @Override
      @Transactional

         public void save(Student theStudent) {

         entityManager.persist(theStudent);

         }
}
```

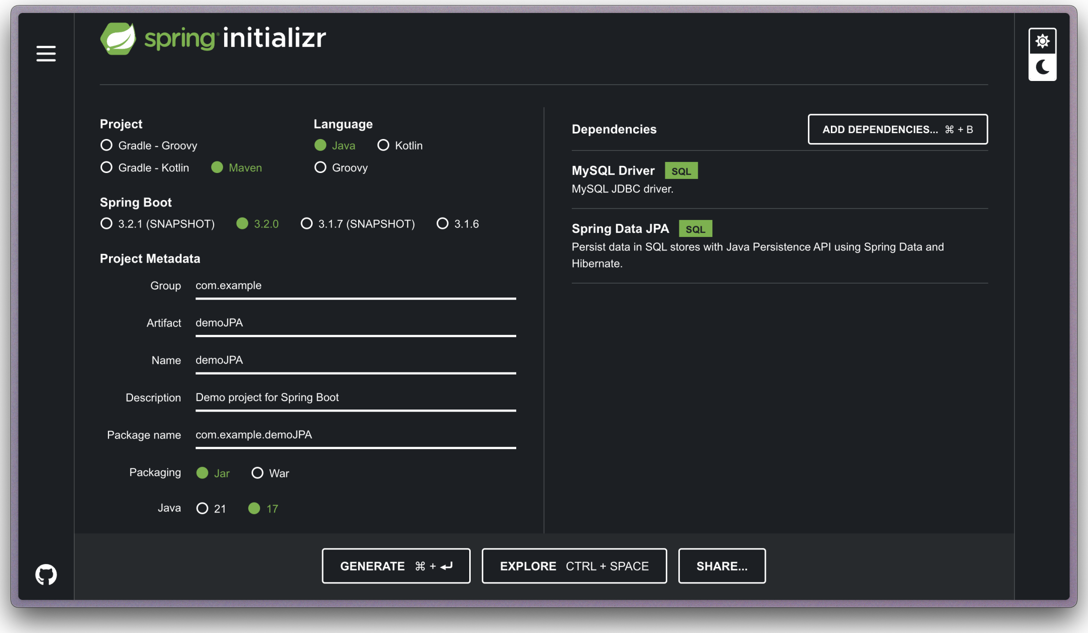
11. Update the main Java app

```
package com.example.JPADEMO;

import com.example.JPADEMO.DAO.StudentDAO;
import com.example.JPADEMO.entity.Student;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

@SpringBootApplication
public class JpademoApplication {

    public static void main(String[] args) {
       SpringApplication.run(JpademoApplication.class, args);
    }

    @Bean
    public CommandLineRunner commandLineRunner(StudentDAO studentDAO) {
       return runner -> {

          createStudent(studentDAO);
       };
    }
    private void createStudent(StudentDAO studentDAO){

       //Create the student object
       System.out.println("Creating new student object ... ");
       Student tempStudent = new Student("Jeya",
"Sutha","jeyasutha@sxcce.edu.in");

       //save the student object
       System.out.println("Saving the student");
       studentDAO.save(tempStudent);

       //display id of the saved student

       System.out.println("Saved student. Generated id:
"+tempStudent.getId());
    }

}
```

12. Run the application

EX-8 : Data JPA (Paging and Searching) (1)

1. Generate spring project with required dependencies



Project
Gradle - Groovy · Gradle - Kotlin · [✓ Maven]

Language
[✓ Java] · Kotlin · Groovy

Spring Boot
3.2.1 (SNAPSHOT) · [✓ 3.2.0] · 3.1.7 (SNAPSHOT) · 3.1.6

Project Metadata

Group	com.example
Artifact	demoJPA
Name	demoJPA
Description	Demo project for Spring Boot
Package name	com.example.demoJPA
Packaging	[✓ Jar] War
Java	21 [✓ 17]

Dependencies

MySQL Driver SQL
MySQL JDBC driver.

Spring Data JPA SQL
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

2. Open the genetated project in IDE.
3. Configure database details in “application.properties” file

```
spring.datasource.url=jdbc:mysql://localhost:3306/student_tracker
spring.datasource.username=root
spring.datasource.password=password
spring.jpa.hibernate.ddl-auto=create-drop
```

4. Create a new package “entities” inside the main package
5. Create a java class “Student” inside models package.

```
package com.example.demoJPA.entity;
```

```
import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Table;

@Entity
@Table(name = "students")
public class Student {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  @Column(name = "id")
  private int id;

  @Column(name = "first_name")
  private String firstName;

  @Column(name = "last_name")
  private String lastName;


  public Student() {

  }

  public Student(String firstName, String lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
  }

  public int getId() {
    return id;
  }

  public void setId(int id) {
    this.id = id;
  }

  public String getFirstName() {
    return firstName;
  }

  public void setFirstName(String firstName) {
    this.firstName = firstName;
  }

  public String getLastName() {
    return lastName;
  }

  public void setLastName(String lastName) {
    this.lastName = lastName;
  }
  @Override
  public String toString() {
    return "Student [ id = " + id + ", firstName = " + firstName + ", lastName = "
+ lastName + " ]";
  }


}
```

6. Create a new package “repos” inside the main package
7. Create a java interface “StudentRepo” inside controllers package.

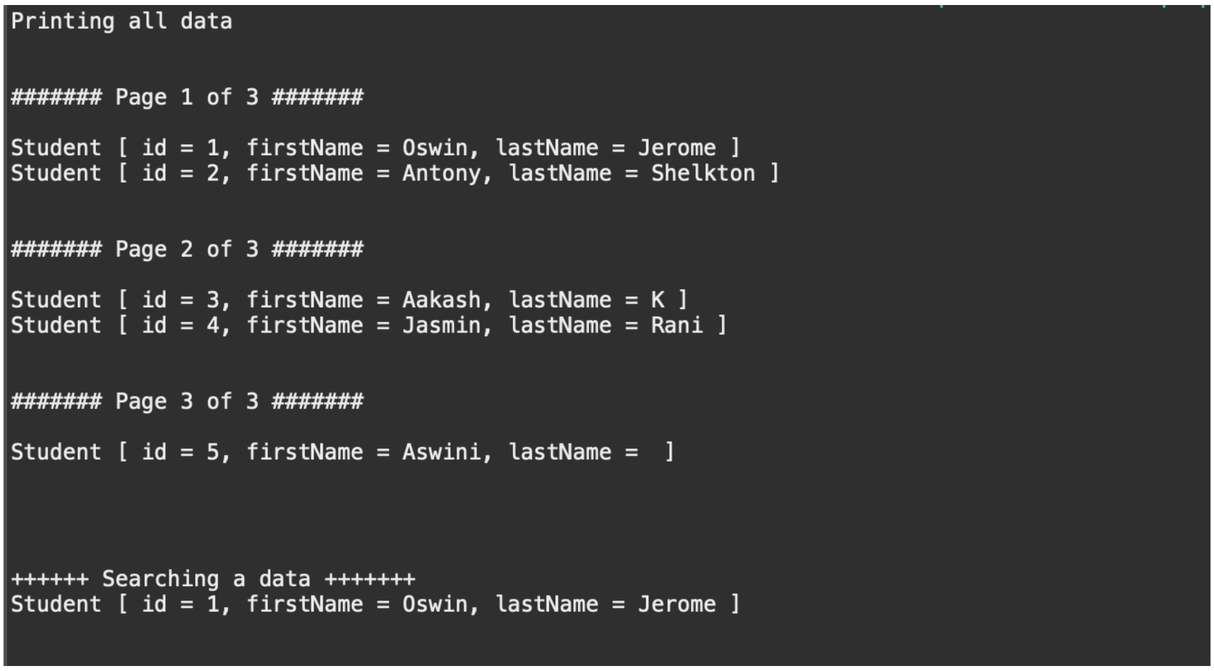
```
package com.example.demoJPA.repos;

import org.springframework.data.jpa.repository.config.EnableJpaRepositories;
import org.springframework.data.repository.CrudRepository;
import org.springframework.data.repository.PagingAndSortingRepository;

import com.example.demoJPA.entity.Student;

@EnableJpaRepositories
public interface StudentRepo extends CrudRepository<Student, Integer>, PagingAndSo
rtingRepository<Student, Integer> {

  Student findByFirstName(String firstName);

}
```

8. Add the command line runner code to the main application java class (Add after main method)

```
@Bean
  CommandLineRunner commandLineRunner(StudentRepo studentRepo) {
    return runner->{

      createStudents(studentRepo);

      System.out.println("Printing all data");

      int itemPerPage = 2;
      int totalPages = studentRepo.findAll(PageRequest.of(1, itemPerPage)).getTota
lPages();


      for(int i=0;i<totalPages;i++) {
        System.out.println("\n\n####### Page "+(i+1)+" of "+totalPages+" #######
\n");

        studentRepo.findAll(PageRequest.of(i, itemPerPage)).forEach(student->{
          System.out.println(student);
        });
      }

      System.out.println("\n\n\n");

      System.out.println("++++++ Searching a data +++++++");
      Student one = studentRepo.findByFirstName("Oswin");
      System.out.println(one);


      System.out.println("\n\n\n");

    };
  }


  private void createStudents(StudentRepo studentRepo) {

    studentRepo.save(new Student("Oswin", "Jerome"));
    studentRepo.save(new Student("Antony", "Shelkton"));
    studentRepo.save(new Student("Aakash", "K"));
    studentRepo.save(new Student("Jasmin", "Rani"));
    studentRepo.save(new Student("Aswini", ""));

  }
```

9. Run the application and view the console

Output


```
Printing all data


####### Page 1 of 3 #######

Student [ id = 1, firstName = Oswin, lastName = Jerome ]
Student [ id = 2, firstName = Antony, lastName = Shelkton ]


####### Page 2 of 3 #######

Student [ id = 3, firstName = Aakash, lastName = K ]
Student [ id = 4, firstName = Jasmin, lastName = Rani ]


####### Page 3 of 3 #######

Student [ id = 5, firstName = Aswini, lastName =  ]




++++++ Searching a data +++++++
Student [ id = 1, firstName = Oswin, lastName = Jerome ]
```