Coding Cheat Sheet: Understanding Spring MVC and Security

This reading provides a comprehensive introduction to building web applications using Spring MVC and securing them with Spring Security. Understanding these concepts will help you set up a Spring MVC project, integrate templates, manipulate forms, and secure applications through authentication and authorization. Let's explore the following Java coding concepts:

- Introduction to Spring MVC
- Templating using Thymeleaf
- Form manipulation using Spring MVC
- Overview of Spring Security
- Authentication and Authorization
- Form-based security using Spring Security

Keep this summary reading available as a reference as you progress through your course, and refer to this reading as you begin coding with Java after this course!

Introduction to Spring MVC

Spring Boot simplifies setting up and configuring a Spring MVC project.

Description	Example
Sample code for a Model	`public class Book {` `  private String title;` `  private String author;` `  private double price;` `  // Getters and setters` `}`
Sample code for a View	`<!-- bookstore.html -->` `<div class="book-details">` `  <h2>${book.title}</h2>` `  <p>Author: ${book.author}</p>` `  <p>Price: $${book.price}</p>` `</div>`
Sample code for a Controller	`@Controller` `public class BookController {` `  @GetMapping("/books/{id}")` `  public String getBook(@PathVariable Long id, Model model) {` `  Book book = bookService.findById(id);` `  model.addAttribute("book", book);` `  return "bookstore";` `  }` `}`
Creating a Search Controller	`@Controller` `public class BookSearchController {` `  @GetMapping("/books/search")` `  public String searchBooks(@RequestParam String title, Model model) {` `  List<Book> books = bookService.searchByTitle(title);` `  model.addAttribute("searchResults", books);` `  return "searchResults";` `  }` `}`

Description	Example

Getting started with Spring MVC using Spring Boot

A Spring Boot web application can be set up by implementing the Model, Controller, and View layers.

Description	Example
Configuring dependencies to be included in the pom.xml file	`<dependencies>` `  <!-- Spring Boot Web Starter - Includes Spring MVC and embedded Tomcat -->` `  <dependency>` `  <groupId>org.springframework.boot</groupId>` `  <artifactId>spring-boot-starter-web</artifactId>` `  </dependency>` `  <!-- Tomcat Jasper - Required for JSP processing -->` `  <dependency>` `  <groupId>org.apache.tomcat.embed</groupId>` `  <artifactId>tomcat-embed-jasper</artifactId>` `  <scope>provided</scope>` `  </dependency>` `  <!-- JSTL - JavaServer Pages Standard Tag Library -->` `  <dependency>` `  <groupId>javax.servlet</groupId>` `  <artifactId>jstl</artifactId>` `  </dependency>` `  <!-- Spring Boot DevTools - Enables automatic restart -->` `  <dependency>` `  <groupId>org.springframework.boot</groupId>` `  <artifactId>spring-boot-devtools</artifactId>` `  <scope>runtime</scope>` `  <optional>true</optional>` `  </dependency>` `</dependencies>`
Creating application.properties in src/main/resources	`# View resolver configuration for JSP` `spring.mvc.view.prefix=/WEB-INF/views/` `spring.mvc.view.suffix=.jsp` `# Server configuration` `server.port=8080` `# Development tools` `spring.devtools.restart.enabled=true`
Creating the Model layer	`package com.example.model;` `public class Employee {` `  // Private fields for encapsulation` `  private int id;` `  private String name;` `  private String department;` `  private double salary;` `  private String email;` `  // Default constructor - required for form binding` `  public Employee() {` `  }` `  // Parameterized constructor for creating employee objects` `  public Employee(int id, String name, String department, double salary, String email) {` `  this.id = id;` `  this.name = name;` `  this.department = department;` `  this.salary = salary;` `  this.email = email;` `  }` `  // Getters and Setters with validation` `  public int getId() {` `  return id;` `  }` `  public void setId(int id) {` `  if (id > 0) {`

Description	Example
	`` this.id = id;
        } else {
            throw new IllegalArgumentException("ID must be positive");
        }
    }

    // Additional getters and setters with similar validation
    // ... (include all getters and setters)

    // toString method for debugging and logging
    @Override
    public String toString() {
        return "Employee{" +
                "id=" + id +
                ", name='" + name + ''' +
                ", department='" + department + ''' +
                ", salary=" + salary +
                ", email='" + email + ''' +
                '}';
    }
}
``` |
| Creating the Controller layer | ```package com.example.controller;

import com.example.model.Employee;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;
import java.util.ArrayList;
import java.util.List;

@Controller
@RequestMapping("/employees") // Base URL for all employee operations
public class EmployeeController {

    // In-memory list to store employees (replace with database in production)
    private List<Employee> employees = new ArrayList<>();
    private int nextId = 1; // Simple ID generator

    // Constructor to initialize sample data
    public EmployeeController() {
        // Add sample employees for demonstration
        addSampleEmployees();
    }

    // Handler for displaying all employees
    @GetMapping
    public String listEmployees(Model model) {
        // Add employees list to the model for view rendering
        model.addAttribute("employees", employees);
        // Return view name - will be resolved to /WEB-INF/views/employees/list.jsp
        return "employees/list";
    }

    // Handler for displaying the add employee form
    @GetMapping("/add")
    public String showAddForm(Model model) {
        // Add empty employee object to model for form binding
        model.addAttribute("employee", new Employee());
        return "employees/addForm";
    }

    // Handler for processing the add employee form submission
    @PostMapping("/add")
    public String addEmployee(@ModelAttribute Employee employee) {
        // Set the next available ID
        employee.setId(nextId++);
        // Add to our list
        employees.add(employee);
        // Redirect to prevent form resubmission
        return "redirect:/employees";
    }

    // Handler for displaying employee details
    @GetMapping("/{id}")
    public String viewEmployee(@PathVariable int id, Model model) {
        // Find employee by ID
        Employee employee = findEmployeeById(id);
        if (employee != null) {
            model.addAttribute("employee", employee);
            return "employees/view";
        }
        return "redirect:/employees";
    }

    // Helper method to find employee by ID
    private Employee findEmployeeById(int id) {
        return employees.stream()
                .filter(emp -> emp.getId() == id)
                .findFirst()
                .orElse(null);
    }

    // Helper method to add sample data
    private void addSampleEmployees() {
``` |

Description	Example
	`` employees.add(new Employee(nextId++, "John Doe", "IT", 75000.0, "john@example.com")); employees.add(new Employee(nextId++, "Jane Smith", "HR", 65000.0, "jane@example.com")); } } ```
Creating the list View	``` <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%> <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %> <!DOCTYPE html> <html> <head> <title>Employee Management - List</title> <style> /* Add some basic styling */ table { width: 100%; border-collapse: collapse; margin: 20px 0; } th, td { padding: 10px; border: 1px solid #ddd; text-align: left; } th { background-color: #f5f5f5; } .action-links { margin: 20px 0; } </style> </head> <body> <h1>Employee Directory</h1> <div class="action-links"> Add New Employee </div> <table> <thead> <tr> <th>ID</th> <th>Name</th> <th>Department</th> <th>Salary</th> <th>Email</th> <th>Actions</th> </tr> </thead> <tbody> <c:forEach items="${employees}" var="emp"> <tr> <td>${emp.id}</td> <td>${emp.name}</td> <td>${emp.department}</td> <td>$${emp.salary}</td> <td>${emp.email}</td> <td> View </td> </tr> </c:forEach> </tbody> </table> </body> </html> ```
Creating an Add form	``` <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%> <%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %> <!DOCTYPE html> <html> <head> <title>Add New Employee</title> <style> .form-group { margin-bottom: 15px; } label { display: block; margin-bottom: 5px; } input[type="text"], input[type="email"], input[type="number"] { ```

Description	Example
	width: 300px; padding: 8px; border: 1px solid #ddd; border-radius: 4px; } button { padding: 10px 20px; background-color: #007bff; color: white; border: none; border-radius: 4px; cursor: pointer; } </style> </head> <body> <h1>Add New Employee</h1> <form:form action="${pageContext.request.contextPath}/employees/add" method="post" modelAttribute="employee"> <div class="form-group"> <label for="name">Name:</label> <form:input path="name" required="true" /> </div> <div class="form-group"> <label for="department">Department:</label> <form:input path="department" required="true" /> </div> <div class="form-group"> <label for="salary">Salary:</label> <form:input path="salary" type="number" step="0.01" required="true" /> </div> <div class="form-group"> <label for="email">Email:</label> <form:input path="email" type="email" required="true" /> </div> <button type="submit">Add Employee</button> </form:form> <div style="margin-top: 20px"> Back to List </body> </html>

Templating using Thymeleaf

For dynamic web content generation, you can set up Thymeleaf in a Spring Boot project.

Description	Example
Setting up Thymeleaf in a Spring Boot project	<dependency> <groupId>org.springframework.boot</groupId> <artifactId>spring-boot-starter-thymeleaf</artifactId> </dependency>
Creating a Thymeleaf template, books.html, to display the list of books	<!DOCTYPE html> <html xmlns:th="http://www.thymeleaf.org"> <head> <title>Book List</title> </head> <body> Available Books <!-- Iterating over a list of books --> <li th:each="book : ${books}"> <!-- Displaying book title --> <span th:text="${book.title}">Book Title by <!-- Displaying author name --> Author Name - Price: $<span th:text="${book.price}">0.00 </body> </html>

Description	Example
Creating a Spring MVC Controller to handle requests and display the list	`import org.springframework.stereotype.Controller;` `import org.springframework.ui.Model;` `import org.springframework.web.bind.annotation.GetMapping;` `import java.util.List;` `import java.util.ArrayList;` `@Controller` `public class BookController {` `@GetMapping("/books")` `public String getBooks(Model model) {` `// Creating a list of books` `List<Book> books = new ArrayList<>();` `books.add(new Book("Effective Java", "Joshua Bloch", 45.00));` `books.add(new Book("Spring in Action", "Craig Walls", 40.00));` `books.add(new Book("Clean Code", "Robert C. Martin", 50.00));` `// Adding the list of books to the model` `model.addAttribute("books", books);` `// Returning the name of the template file (without the extension)` `return "books";` `}` `}`
Creating a Book Class with added getters and setters for each property	`public class Book {` `private String title;` `private String author;` `private double price;` `public Book(String title, String author, double price) {` `this.title = title;` `this.author = author;` `this.price = price;` `}` `// Getter for title` `public String getTitle() {` `return title;` `}` `// Setter for title` `public void setTitle(String title) {` `this.title = title;` `}` `// Getter for author` `public String getAuthor() {` `return author;` `}` `// Setter for author` `public void setAuthor(String author) {` `this.author = author;` `}` `// Getter for price` `public double getPrice() {` `return price;` `}` `// Setter for price` `public void setPrice(double price) {` `this.price = price;` `}` `}`
Creating an external CSS file, styles.css, in the src/main/resources/static/css directory	`body {` `font-family: Arial, sans-serif;` `background-color: #f4f4f9;` `color: #333;` `margin: 0;`

Description	Example
	padding: 20px; } h1 { color: #4a90e2; } ul { list-style-type: none; padding: 0; } li { background-color: #fff; margin-bottom: 10px; padding: 15px; border-radius: 5px; box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1); } span { display: inline-block; margin-right: 10px; } .price { font-weight: bold; color: #e94e77; }
Updating the books.html template to include the CSS file and add some inline styles using Thymeleaf	<!DOCTYPE html> <html xmlns:th="http://www.thymeleaf.org"> <head> <title>Book List</title> <!-- Link to external CSS file --> <link rel="stylesheet" th:href="@{/css/styles.css}" /> </head> <body> Available Books <!-- Iterating over a list of books --> <li th:each="book : ${books}" th:style="'background-color:' + (${book.price} > 45 ? '#eef9f9' : '#fff')"> <!-- Displaying book title --> <span th:text="${book.title}" th:style="'font-weight:' + (${book.title.length()} > 15 ? 'bold' : 'normal')"> Book Title by <!-- Displaying author name --> Author Name - Price: $<span th:text="${book.price}" class="price">0.00 </body> </html>

Form manipulation using Spring MVC

Description	Example
Creating a Model Class User	package com.example.demo.model; public class User { private String firstName; private String lastName; private String email; // Getters and Setters public String getFirstName() { return firstName; } public void setFirstName(String firstName) { this.firstName = firstName; } public String getLastName() { return lastName; }

Description	Example
	`` public void setLastName(String lastName) { this.lastName = lastName; } public String getEmail() { return email; } public void setEmail(String email) { this.email = email; } } ```
Creating a Controller to handle HTTP requests	``` package com.example.demo.controller; import com.example.demo.model.User; 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 org.springframework.web.bind.annotation.ModelAttribute; @Controller public class UserController { // Display the form to the user @GetMapping("/userForm") public String showForm(Model model) { model.addAttribute("user", new User()); return "userForm"; } // Handle form submission @PostMapping("/submitForm") public String submitForm(@ModelAttribute User user, Model model) { model.addAttribute("user", user); return "formResult"; } } ```
Creating a View template to display the form	``` userForm.html <!DOCTYPE html> <html xmlns:th="http://www.thymeleaf.org"> <head> <title>User Form</title> </head> <body> Enter User Details <form th:action="@{/submitForm}" th:object="${user}" method="post"> <label>First Name:</label> <input type="text" th:field="*{firstName}" /> <label>Last Name:</label> <input type="text" th:field="{lastName}" /> <label>Email:</label> <input type="email" th:field="*{email}" /> <button type="submit">Submit</button> </form> </body> </html> ```
Creating a View template to show the results	``` formResult.html <!DOCTYPE html> <html xmlns:th="http://www.thymeleaf.org"> <head> <title>Form Result</title> </head> <body> Submitted User Details First Name: Last Name: ```

Description	Example
	Email: </body> </html>
Adding validation annotations to the Model to specify validation rules for each field	package com.example.demo.model; import jakarta.validation.constraints.Email; import jakarta.validation.constraints.NotEmpty; import jakarta.validation.constraints.Size; public class User { @NotEmpty(message = "First name is required") @Size(min = 2, max = 30, message = "First name must be between 2 and 30 characters") private String firstName; @NotEmpty(message = "Last name is required") @Size(min = 2, max = 30, message = "Last name must be between 2 and 30 characters") private String lastName; @NotEmpty(message = "Email is required") @Email(message = "Please provide a valid email address") private String email; // Getters and Setters 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; } }
Updating the Controller to handle validation errors	package com.example.demo.controller; import com.example.demo.model.User; import jakarta.validation.Valid; import org.springframework.stereotype.Controller; import org.springframework.ui.Model; import org.springframework.validation.BindingResult; import org.springframework.web.bind.annotation.GetMapping; import org.springframework.web.bind.annotation.ModelAttribute; import org.springframework.web.bind.annotation.PostMapping; @Controller public class UserController { @GetMapping("/userForm") public String showForm(Model model) { model.addAttribute("user", new User()); return "userForm"; } @PostMapping("/submitForm") public String submitForm(@Valid @ModelAttribute User user, BindingResult bindingResult, Model model) { if (bindingResult.hasErrors()) { return "userForm"; } model.addAttribute("user", user); return "formResult"; } }

Description	Example
Updating the Form View to display validation error messages	<!DOCTYPE html> <html xmlns:th="http://www.thymeleaf.org"> <head> <title>User Form</title> </head> <body> Enter User Details <form th:action="@{/submitForm}" th:object="${user}" method="post"> <label>First Name:</label> <input type="text" th:field="*{firstName}" /> <div th:if="${#fields.hasErrors('firstName')}" th:errors="*{firstName}"> <label>Last Name:</label> <input type="text" th:field="*{lastName}" /> <div th:if="${#fields.hasErrors('lastName')}" th:errors="*{lastName}"> <label>Email:</label> <input type="email" th:field="*{email}" /> <div th:if="${#fields.hasErrors('email')}" th:errors="*{email}"> <button type="submit">Submit</button> </form> </body> </html>

Authentication and authorization

The process of implementing authentication and authorization in a Spring Boot application is simple.

Description	Example
Including Spring Security in the project	<dependency> <groupId>org.springframework.boot</groupId> <artifactId>spring-boot-starter-security</artifactId> </dependency>
Creating a Security Configuration Class	import org.springframework.context.annotation.Configuration; import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder; import org.springframework.security.config.annotation.web.builders.HttpSecurity; import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity; import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter; @Configuration @EnableWebSecurity public class WebSecurityConfig extends WebSecurityConfigurerAdapter { @Override protected void configure(AuthenticationManagerBuilder auth) throws Exception { auth.inMemoryAuthentication() .withUser("user").password("{noop}password").roles("USER") .and() .withUser("admin").password("{noop}adminpass").roles("ADMIN"); } @Override protected void configure(HttpSecurity http) throws Exception { http .authorizeRequests() .antMatchers("/admin/").hasRole("ADMIN") .antMatchers("/user/").hasRole("USER") .antMatchers("/public/**").permitAll() .and() .formLogin(); } }

Description	Example
Adding Controller endpoints	`import org.springframework.web.bind.annotation.GetMapping;` `import org.springframework.web.bind.annotation.RestController;` `@RestController` `public class SampleController {` `    @GetMapping("/public/welcome")` `    public String welcome() {` `        return "Welcome to the public endpoint!";` `    }` `    @GetMapping("/user/profile")` `    public String userProfile() {` `        return "User Profile: Access Granted";` `    }` `    @GetMapping("/admin/dashboard")` `    public String adminDashboard() {` `        return "Admin Dashboard: Access Granted";` `    }` `}`
Creating a publicly accessible endpoint	`@GetMapping("/public/welcome")` `public String welcome() {` `    return "Welcome to the public endpoint!";` `}`
Creating an endpoint accessible only to specific users	`@GetMapping("/user/profile")` `@PreAuthorize("hasRole('USER')")` `public String userProfile() {` `    return "User Profile: Access Granted";` `}`
Creating an admin-only endpoint	`@GetMapping("/admin/dashboard")` `@PreAuthorize("hasRole('ADMIN')")` `public String adminDashboard() {` `    return "Admin Dashboard: Access Granted";` `}`

Form-based Security using Spring Security

Spring Security configurations allow you to restrict access and enable form-based authentication.

Description	Example
Creating a class SecurityConfig in src/main/java/com/example/demosecurity/config	`package com.example.demosecurity.config;` `import org.springframework.context.annotation.Configuration;` `import org.springframework.security.config.annotation.web.builders.HttpSecurity;` `import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;` `import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;` `@Configuration` `@EnableWebSecurity` `public class SecurityConfig extends WebSecurityConfigurerAdapter {` `    @Override` `    protected void configure(HttpSecurity http) throws Exception {` `        http` `            .authorizeRequests()` `                .antMatchers("/public/**").permitAll() // Allow access to public URLs` `                .anyRequest().authenticated() // Require authentication for other URLs` `                .and()` `            .formLogin() // Enable form-based authentication` `                .loginPage("/login") // Custom login page URL` `                .permitAll()` `                .and()` `            .logout() // Enable logout support` `                .permitAll();` `    }` `}`
Creating a custom login page	`<!DOCTYPE html>` `<html>` `<head>` `    <title>Login</title>` `</head>` `<body>` `    <h2>Login</h2>` `    <form method="post" action="/login">` `        <div>` `            <label>Username:</label>` `            <input type="text" name="username">` `        </div>` `        <div>` `            <label>Password:</label>` `            <input type="password" name="password">` `        </div>` `        <div>` `            <button type="submit">Login</button>` `        </div>` `    </form>` `</body>` `</html>`
Including Spring Security dependency in pom.xml for Maven	`<dependency>` `    <groupId>org.springframework.boot</groupId>` `    <artifactId>spring-boot-starter-security</artifactId>` `</dependency>`
Configuring Password Encoder	`package com.example.demosecurity.config;` `import org.springframework.context.annotation.Bean;` `import org.springframework.context.annotation.Configuration;` `import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;` `import org.springframework.security.crypto.password.PasswordEncoder;` `@Configuration` `public class SecurityConfig {` `    @Bean` `    public PasswordEncoder passwordEncoder() {` `        return new BCryptPasswordEncoder();` `    }` `}`

Description	Example
Encrypting passwords before saving	`package com.example.demosecurity.service;` `import com.example.demosecurity.model.User;` `import com.example.demosecurity.repository.UserRepository;` `import org.springframework.beans.factory.annotation.Autowired;` `import org.springframework.security.crypto.password.PasswordEncoder;` `import org.springframework.stereotype.Service;` `@Service` `public class UserService {` `    @Autowired` `    private UserRepository userRepository;` `    @Autowired` `    private PasswordEncoder passwordEncoder;` `    public void saveUser(User user) {` `        // Encrypt the user's password before saving` `        user.setPassword(passwordEncoder.encode(user.getPassword()));` `        userRepository.save(user);` `    }` `}`
Creating a simple user entity	`package com.example.demosecurity.model;` `import javax.persistence.Entity;` `import javax.persistence.GeneratedValue;` `import javax.persistence.GenerationType;` `import javax.persistence.Id;` `@Entity` `public class User {` `    @Id` `    @GeneratedValue(strategy = GenerationType.IDENTITY)` `    private Long id;` `    private String username;` `    private String password;` `    // Getters and setters` `    public Long getId() {` `        return id;` `    }` `    public void setId(Long id) {` `        this.id = id;` `    }` `    public String getUsername() {` `        return username;` `    }` `    public void setUsername(String username) {` `        this.username = username;` `    }` `    public String getPassword() {` `        return password;` `    }` `    public void setPassword(String password) {` `        this.password = password;` `    }` `}`

Author(s)

Ramanujam Srinivasan
Lavanya Thiruvali Sunderarajan