1 2  9 0

UNIVERSIDADE D
COIMBRA

Sistemas Distribuídos 2024/2025
Faculdade de Ciências e Tecnologia
Universidade de Coimbra

Choose Your Path for MVC

[MVC in Python with FastAPI] [MVC in Java with Spring Boot]

MVC in Java with Spring Boot

Web Programming with Spring Boot and Thymeleaf

In this assignment you will learn the basics of web programming using Spring Boot and Thymeleaf. The provided source code demonstrates how to build a web application that:

- Responds to specific URLs.
- Passes variables to the view (model).
- Renders HTML tables.
- Uses objects with different scopes (request, session, application) to share data among users or persist state during an interactive session.

1. Creating and Running a Project

Students can either download the provided source code or create a new project via Spring Initializer. In the initializer, select the following dependencies:

- **Spring Web**
- **Thymeleaf**
- **Spring Boot DevTools**

This configuration will create a `pom.xml` with these dependencies:

- `spring-boot-starter-thymeleaf`

- `spring-boot-starter-web`
- `spring-boot-devtools`
- `spring-boot-starter-test`

To run the project, execute the following command in the project root:

```
./mvnw spring-boot:run
```

The main class is annotated with `@SpringBootApplication` so that Spring automatically scans for controllers (e.g. in `GreetingController.java`).

2. Setting and Using Variables for the Model

Examine the `greeting()` method in the controller which is mapped to `/greeting` using the `@GetMapping` annotation. Notice how it:

- Accepts a query parameter (`name`).
- Passes variables to the model via `model.addAttribute(...)`.
- Returns the logical view name (e.g. `greeting`) so that Thymeleaf finds the matching HTML template.

Also review the `redirect()` method (which redirects to `/greeting`) and `atable()`, which demonstrates populating a table view with a list of `Employee` objects.

3. Forms

The example includes two methods:

- **createProjectForm()** – renders a form (template: `create-project.html`) to enter project details.
- **saveProjectSubmission()** – handles the submitted form data (via POST), saves the project to an in-memory list, and displays the result (template: `result.html`).

Students should explore how the form fields are bound to the `Project` object and how data is mapped between the front-end and back-end.

4. Different Scopes for Variables

The method `counters()` illustrates the use of three scopes:

- **Request Scope:** A new instance is created on every request.
- **Session Scope:** Data survives across multiple requests from the same user.
- **Application Scope:** A single instance is shared across the entire application.

By accessing `/counters` in different browsers, students can observe how each scope behaves.

5. Servlets

Although servlets mix business logic with presentation, they are used in this example to show low-level handling. Two servlets are provided:

- **Example:** Builds a simple HTML "Hello World!" programmatically.
- **ThymeleafServlet:** Uses Thymeleaf to process a template and pass variables.

These are registered in the main class with specific URL mappings. (Note that the project requires an additional dependency for OGNL.)

6. Available URLs

Once the application is running, the following URLs are available:

- `http://localhost:8080/exampleServlet/AnnotationExample`
- `http://localhost:8080/thymeleafServlet/hellofromservlet.html`
- `http://localhost:8080/thymeleafServlet/urls.html`
- `http://localhost:8080/greeting` (optionally with `?name=Student`)
- `http://localhost:8080/givemeatable`
- `http://localhost:8080/create-project`
- `http://localhost:8080/counters`
- `http://localhost:8080/` (redirects to `/greeting`)

7. Exercises

1. **Project Submission:**
   Complete the project submission application so that each submitted project is added to a list and displayed in a new view.
2. **Session Variable Type:**
   Modify the session variable (originally a `Number` object) so that it can store any type of object (e.g. to track if a user is logged in).
3. **User Login Form:**
   Create a form for user login and implement a mechanism (e.g. via servlet filters) to restrict access to certain resources for unauthenticated users.

MVC in Python with FastAPI

Web Programming with FastAPI and Jinja2

This alternative solution uses Python's FastAPI framework along with Jinja2 for templating. While the overall ideas are similar to the Java version, the implementation details differ.

1. Overview

In this exercise you will build a web application using FastAPI. The project demonstrates:

- Defining routes that respond to HTTP GET and POST methods.
- Passing variables to Jinja2 templates.
- Creating HTML forms and processing submissions.
- Simulating different scopes:
  - **Request Scope:** Variables created per request.
  - **Session Scope:** Using Starlette's session middleware.
  - **Application Scope:** A globally shared counter stored in the FastAPI app state.
- Mimicking servlet-like endpoints for demonstration purposes.

2. Creating and Running a Project

A sample project (named, for instance, `p4-fastapi-solution`) has been provided with the following structure:

```
p4-fastapi-solution
|-- app
|   |-- __init__.py
|   |-- main.py
|   |-- models.py
|   |-- routes.py
|   |-- templates
|   |   |-- greeting.html
|   |   |-- table.html
|   |   |-- create-project.html
|   |   |-- result.html
|   |   |-- projects.html
|   |   `-- counter.html
|   `-- servlet_templates
|       |-- hellofromservlet.html
|       `-- urls.html
|-- tests
|   `-- test_app.py
`-- requirements.txt
```

To set up the project:

1. **Install dependencies:**
   Create a virtual environment and install packages using the provided `requirements.txt`.

   ```
   pip install -r requirements.txt
   ```

2. **Run the application:**
   Use uvicorn to start the FastAPI server:

   ```
   uvicorn app.main:app --host 127.0.0.1 --port 8080 --reload
   ```

   or

   ```
   python -m app.main
   ```

3. Setting and Using Variables for the Model

In FastAPI, route functions pass variables to the Jinja2 templates via the `TemplateResponse` method. For example, the `/greeting` endpoint sets variables such as `name` and `othername` that are then rendered by `greeting.html`.

4. Forms

The FastAPI version handles form submissions using Python's native support for form data. The `/create-project` endpoint renders a form (using `create-project.html`), and the `/save-project` endpoint processes the POSTed form data to create a `Project` object. The submitted data is then rendered in the `result.html` template.

5. Different Scopes for Variables

FastAPI does not provide session or request beans as in Spring. Instead:

- **Request Scope:** A new counter is created in each route call.
- **Session Scope:** Starlette's `SessionMiddleware` is used to store session data (e.g. counters).
- **Application Scope:** A globally shared counter is stored in `app.state` (accessible by all users).

Students can test these scopes by visiting `/counters` in different browsers and observing how the counters behave.

6. "Servlets" in FastAPI

While the concept of servlets does not exist in Python, similar functionality is demonstrated by endpoints such as:

- `/exampleServlet/AnnotationExample`: Returns a simple "Hello World!" HTML response.
- `/thymeleafServlet/hellofromservlet.html` and `/thymeleafServlet/urls.html`: Use Jinja2 templates to simulate variable injection as seen in the Java Thymeleaf servlet.

7. Available URLs

The FastAPI solution exposes endpoints equivalent to the Java version:

- `http://localhost:8080/exampleServlet/AnnotationExample`
- `http://localhost:8080/thymeleafServlet/hellofromservlet.html`
- `http://localhost:8080/thymeleafServlet/urls.html`
- `http://localhost:8080/greeting` (optionally with `?name=Student`)
- `http://localhost:8080/givemeatable`
- `http://localhost:8080/create-project`
- `http://localhost:8080/counters`
- `http://localhost:8080/` (redirects to `/greeting`)

8. Exercises

1. **Project Submission:**
   Complete the FastAPI project submission application so that each new project is added to a list and displayed on a separate page.
2. **Session Variable Storage:**
   Modify the session data storage to support saving different types of objects (for instance, to record whether a user is logged in).
3. **User Login Form:**
   Create a login form and design a mechanism (using middleware or dependency injection) to restrict access to certain endpoints for unauthenticated users.

Note: While the Java and Python solutions share similar concepts (MVC, forms, template rendering, variable scopes), the implementation details differ significantly. Students are encouraged to compare the approaches to deepen their understanding of web application design in both ecosystems.