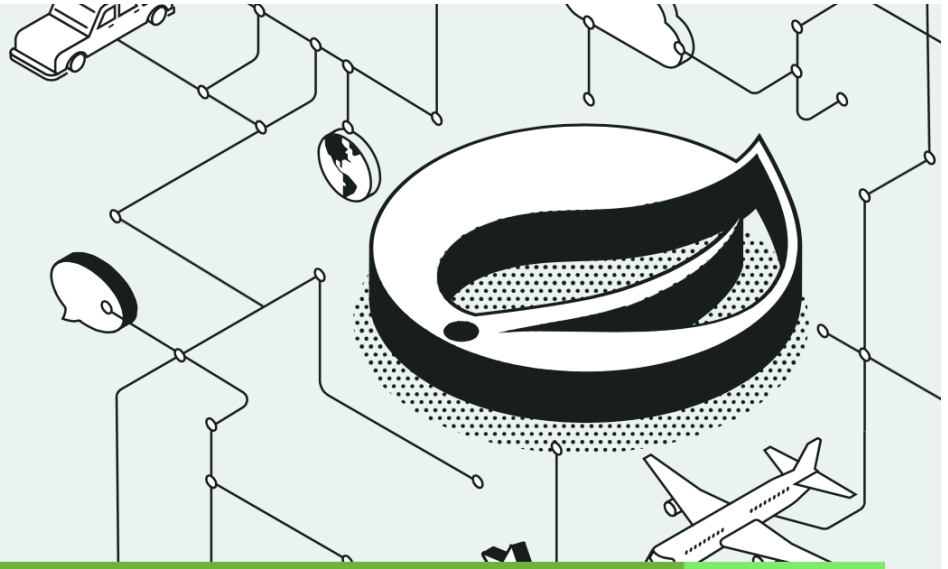
Starting Guide
Spring Boot – Spring MVC

Software Engineering II

Why Spring?

Spring makes programming Java quicker, easier, and safer for everybody. Spring's focus on speed, simplicity, and productivity has made it the world's most popular Java framework.



"We use a lot of the tools that come with the Spring framework and reap the benefits of having a lot of the out of the box solutions, and not having to worry about writing a ton of additional code—so that really saves us some time and energy."

SEAN GRAHAM, APPLICATION TRANSFORMATION LEAD, DICK'S SPORTING GOODS

Spring is everywhere

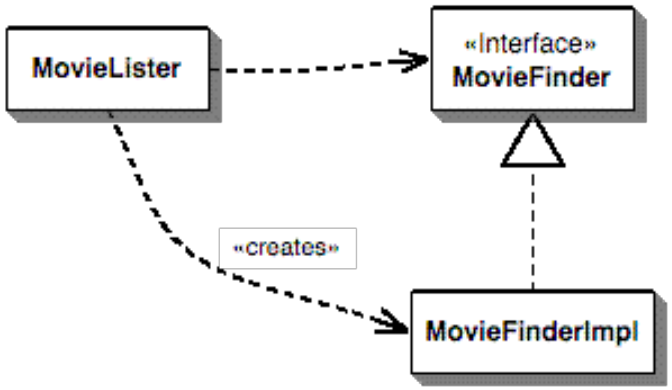
Spring's flexible libraries are trusted by developers all over the world. Spring delivers delightful experiences to millions of end-users every day—whether that's streaming TV, connected cars, online shopping, or countless other innovative solutions. Spring also has contributions from all the big names in tech, including Alibaba, Amazon, Google, Microsoft, and more.

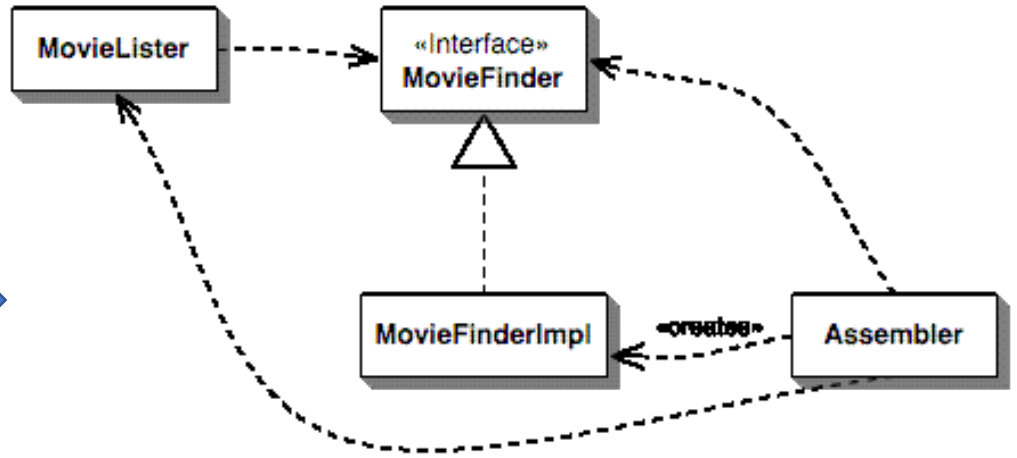
Spring is flexible

Spring's flexible and comprehensive set of extensions and third-party libraries let developers build almost any application imaginable. At its core, Spring Framework's Inversion of Control (IoC) and Dependency Injection (DI) features provide the foundation for a wide-ranging set of features and functionality. Whether you're building secure, reactive, cloud-based microservices for the web, or complex streaming data flows for the enterprise, Spring has the tools to help.

Inversion of Control (IoC) and Dependency Injection

- **Inversion of control** is a common characteristic of frameworks
  - A too generic term, and thus people find it is confusing
- **Dependency Injection** is a design Pattern
  - Named by various IoC advocates including **Martin Fowler**
- There are 3 types of Dependency Injection
  1. **interface injection**
  2. **setter injection <– be used in Spring framework**
  3. **constructor injection**
- https://martinfowler.com/articles/injection.html [23 January 2004]






```
public interface MovieFinder {
    List findAll();
}
```

class MovieLister...

```
  private MovieFinder finder;
  public MovieLister() {
    finder = new ColonDelimitedMovieFinder("movies1.txt");
  }
```

```
public interface MovieFinder {
    List findAll();
}
```

class MovieLister...

```
  private MovieFinder finder;
public void setFinder(MovieFinder finder) {
  this.finder = finder;
}
```

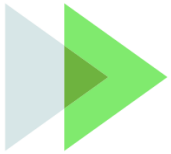
class ColonMovieFinder...

```
  public void setFilename(String filename) {
      this.filename = filename;
  }
```

Spring is productive

Spring Boot transforms how you approach Java programming tasks, radically streamlining your experience. Spring Boot combines necessities such as an application context and an auto-configured, embedded web server to make microservice development a cinch. To go even faster, you can combine Spring Boot with Spring Cloud's rich set of supporting libraries, servers, patterns, and templates, to safely deploy entire microservices-based architectures into the cloud, in record time.

Spring is fast



Our engineers care deeply about performance. With Spring, you'll notice fast startup, fast shutdown, and optimized execution, by default. Increasingly, Spring projects also support the reactive (nonblocking) programming model for even greater efficiency. Developer productivity is Spring's superpower. Spring Boot helps developers build applications with ease and with far less toil than other competing paradigms. Embedded web servers, auto-configuration, and "fat jars" help you get started quickly, and innovations like LiveReload in Spring DevTools mean developers can iterate faster than ever before. You can even start a new Spring project in seconds, with the Spring Initializr at start.spring.io.

Spring is secure

Spring has a proven track record of dealing with security issues quickly and responsibly. The Spring committers work with security professionals to patch and test any reported vulnerabilities. Third-party dependencies are also monitored closely, and regular updates are issued to help keep your data and applications as safe as possible. In addition, Spring Security makes it easier for you to integrate with industry-standard security schemes and deliver trustworthy solutions that are secure by default.

Spring is supportive

The Spring community is enormous, global, diverse, and spans folks of all ages and capabilities, from complete beginners to seasoned pros. No matter where you are on your journey, you can find the support and resources you need to get you to the next level: quickstarts, guides & tutorials, videos, meetups, support, or even formal training and certification.

What can Spring do?

Microservices

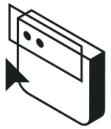
Quickly deliver production-grade features with independently evolvable microservices.

Reactive

Spring's asynchronous, nonblocking architecture means you can get more from your computing resources.

Cloud

Your code, any cloud—we've got you covered. Connect and scale your services, whatever your platform.



Web apps

Frameworks for fast, secure, and responsive web applications connected to any data store.

Serverless

The ultimate flexibility. Scale up on demand and scale to zero when there's no demand.

Event Driven

Integrate with your enterprise. React to business events. Act on your streaming data in realtime.

Batch

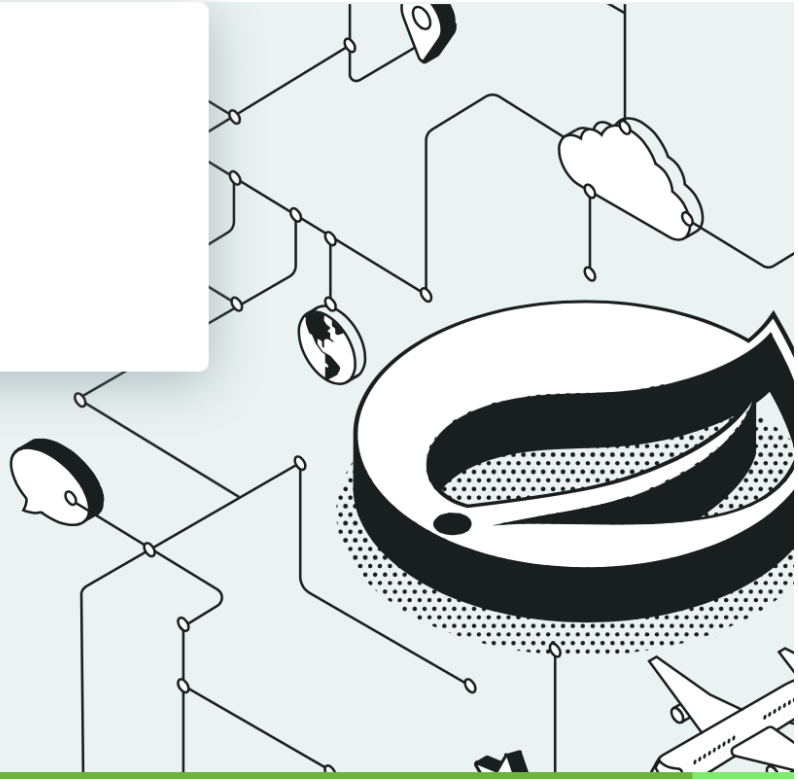
Automated tasks. Offline processing of data at a time to suit you.

spring.io

spring

Why Spring | Learn | Projects | Training | Support | Community

- Overview
- Quickstart
- Guides
- Blog



Why Spring?

Spring makes programming Java quicker, easier, and safer for everybody. Spring's focus on speed, simplicity, and productivity has made it the world's most popular Java framework.

Spring Quickstart Guide

What you'll build

You will build a classic “Hello World!” endpoint which any browser can connect to. You can even tell it your name, and it will respond in a more friendly way.

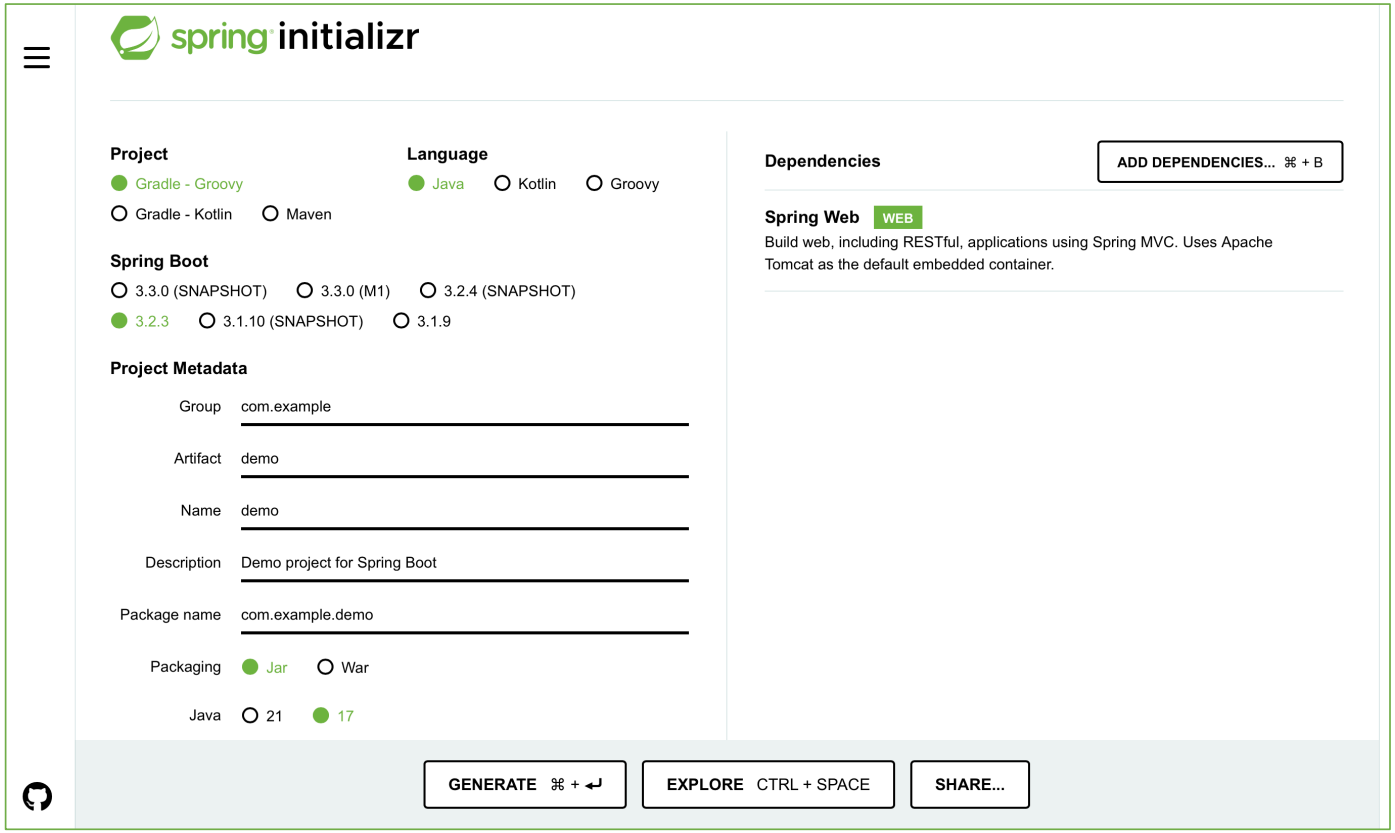
What you’ll need

An Integrated Developer Environment (IDE)
Popular choices include IntelliJ IDEA, Spring Tools, Visual Studio Code, or Eclipse, and many more.

A Java™ Development Kit (JDK)
We recommend BellSoft Liberica JDK version 17.

Step 1: Start a new Spring Boot project

Use start.spring.io to create a “web” project. In the “Dependencies” dialog search for and add the “web” dependency as shown in the screenshot. Hit the “Generate” button, download the zip, and unpack it into a folder on your computer.



Step 2: Add your code

Open up the project in your IDE and locate the `DemoApplication.java` file in the `src/main/java/com/example/demo` folder. Now change the contents of the file by adding the extra method and annotations shown in the code below. You can copy and paste the code or just type it.

COPY

```
package com.example.demo;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication
@RestController
public class DemoApplication {
    public static void main(String[] args) {
      SpringApplication.run(DemoApplication.class, args);
    }
    @GetMapping("/hello")
    public String hello(@RequestParam(value = "name", defaultValue = "World") String name) {
      return String.format("Hello %s!", name);
    }
}
```

Step 3: Try it

Let's build and run the program. Open a command line (or terminal) and navigate to the folder where you have the project files. We can build and run the application by issuing the following command:

MacOS/Linux:

```
./gradlew bootRun
```

COPY

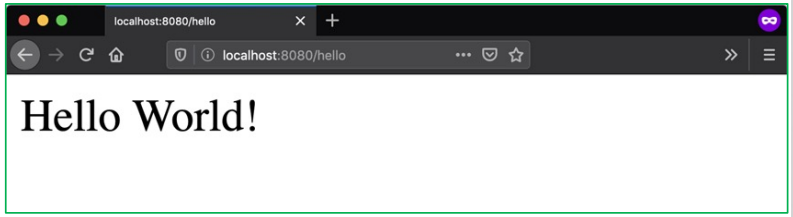
Windows:

```
.\gradlew.bat bootRun
```

COPY

```
boot:run --quiet

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.2.4.RELEASE)

2020-02-14 16:16:47.746  INFO 4838 --- [           main] com.example.demo.DemoApplication         : Starting DemoApplication
on Brians-MacBook-Pro.local with PID 4838 (/Users/bclozel/workspace/tmp/demo/target/classes started by bclozel in /Users/bclo
zel/workspace/tmp/demo)
2020-02-14 16:16:47.748  INFO 4838 --- [           main] com.example.demo.DemoApplication         : No active profile set, fa
lling back to default profiles: default
2020-02-14 16:16:48.272  INFO 4838 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with p
ort(s): 8080 (http)
2020-02-14 16:16:48.279  INFO 4838 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2020-02-14 16:16:48.279  INFO 4838 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine:
[Apache Tomcat/9.0.30]
2020-02-14 16:16:48.323  INFO 4838 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embed
ded WebApplicationContext
2020-02-14 16:16:48.324  INFO 4838 --- [           main] o.s.web.context.ContextLoader            : Root WebApplicationContex
t: initialization completed in 532 ms
2020-02-14 16:16:48.438  INFO 4838 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorServ
ice 'applicationTaskExecutor'
2020-02-14 16:16:48.533  INFO 4838 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s)
: 8080 (http) with context path ''
2020-02-14 16:16:48.535  INFO 4838 --- [           main] com.example.demo.DemoApplication         : Started DemoApplication i
n 1.006 seconds (JVM running for 1.248)
```

http://localhost:8080/hello



Model/View/Controller (MVC)

- Subsystems are classified into 3 different types
  - **Model** subsystem: Responsible for application domain knowledge
  - **View** subsystem: Responsible for displaying application domain objects to the user
  - **Controller** subsystem: Responsible for sequence of interactions with the user and notifying views of changes in the model

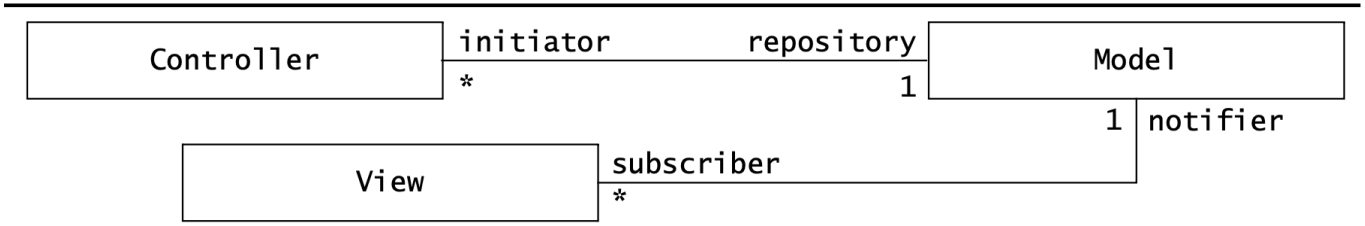
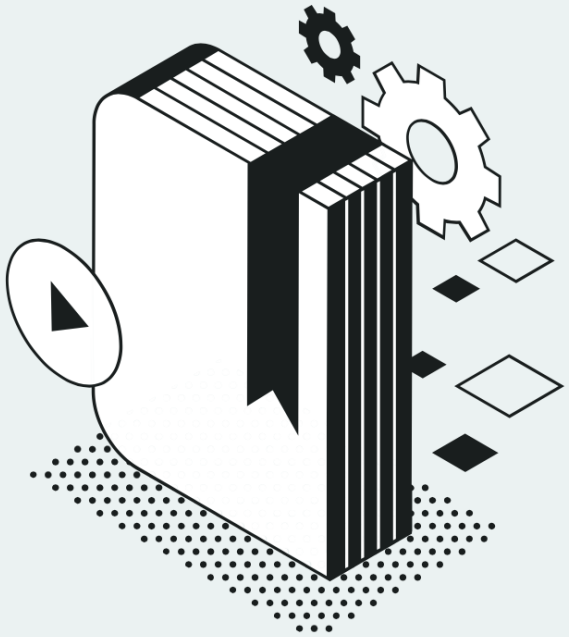


Figure 6-15 Model/View/Controller architectural style (UML class diagram). The `Controller` gathers input from the user and sends messages to the `Model`. The `Model` maintains the central data structure. The `Views` display the `Model` and are notified (via a subscribe/notify protocol) whenever the `Model` is changed.

spring

Why Spring | Learn | Projects | Training | Support | Community

- Overview
- Quickstart
- Guides
- Blog



Guide

Whatever you're building, these guides are designed to get you productive as quickly as possible – using the latest Spring project releases and techniques as recommended by the Spring team.

Building a Hypermedia-Driven RESTful Web Service
Learn how to create a hypermedia-driven RESTful Web service with Spring.

Integrating Data
Learn how to build an application that uses Spring Integration to fetch data, process it, and write it to a file.

Managing Transactions
Learn how to wrap key parts of code with transactions.

Accessing Data with MongoDB
Learn how to persist data in MongoDB.

Converting a Spring Boot JAR Application to a WAR
Learn how to convert your Spring Boot JAR-based application to a WAR file.

Handling Form Submission
Learn how to create and submit a web form with Spring.

Accessing Data in Pivotal GemFire
Learn how to build an application using Gemfire's data fabric.

Caching Data with Pivotal GemFire
Learn how to cache data in GemFire.

Accessing Data with JPA
Learn how to work with JPA data persistence using Spring Data JPA.

Serving Web Content with Spring MVC
Learn how to create a web page with Spring MVC and Thymeleaf.

Creating Asynchronous Methods
Learn how to create asynchronous service methods.

Building an Application with Spring Boot
Learn how to build an application with minimal configuration.

Serving Web Content with Spring MVC

This guide walks you through the process of creating a “Hello, World” web site with Spring.

What You Will Build

You will build an application that has a static home page and that will also accept HTTP GET requests at: `http://localhost:8080/greeting`.

It will respond with a web page that displays HTML. The body of the HTML will contain a greeting: “Hello, World!”

You can customize the greeting with an optional `name` parameter in the query string. The URL might then be `http://localhost:8080/greeting?name=User`.

The `name` parameter value overrides the default value of `World` and is reflected in the response by the content changing to “Hello, User!”

Starting with Spring Initializr

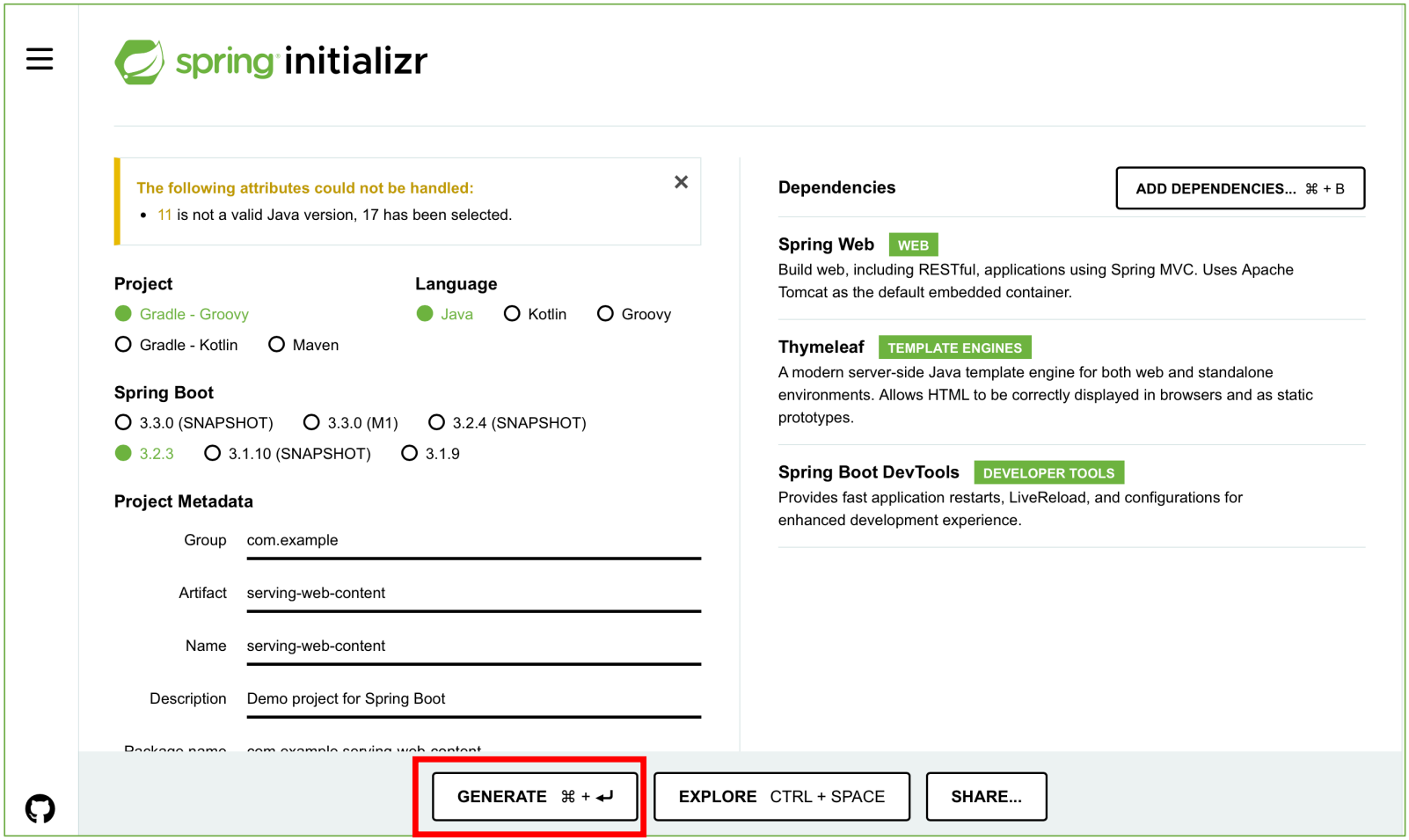
You can use this pre-initialized project and click Generate to download a ZIP file. This project is configured to fit the examples in this tutorial.

To manually initialize the project:

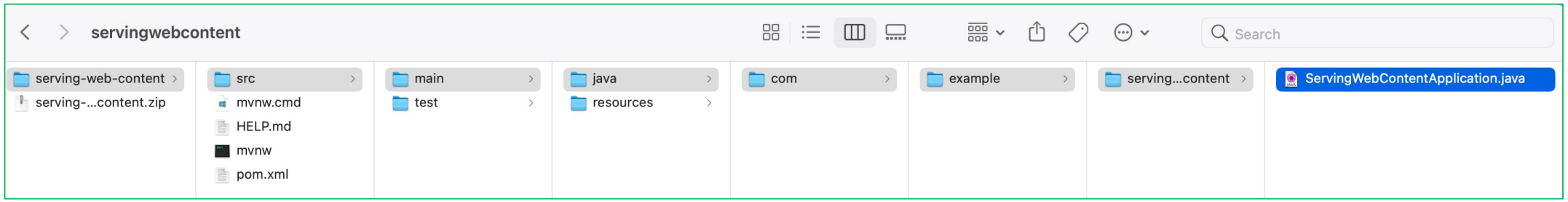
1. Navigate to https://start.spring.io. This service pulls in all the dependencies you need for an application and does most of the setup for you.
2. Choose either Gradle or Maven and the language you want to use. This guide assumes that you chose Java.
3. Click **Dependencies** and select **Spring Web**, **Thymeleaf**, and **Spring Boot DevTools**.
4. Click **Generate**.
5. Download the resulting ZIP file, which is an archive of a web application that is configured with your choices.

Starting with Spring Initializr

If you use Maven, visit the Spring Initializr to generate a new project with the required dependencies (Spring Web, Thymeleaf, and Spring Boot DevTools).



- Unzip the file and place in your working directory.



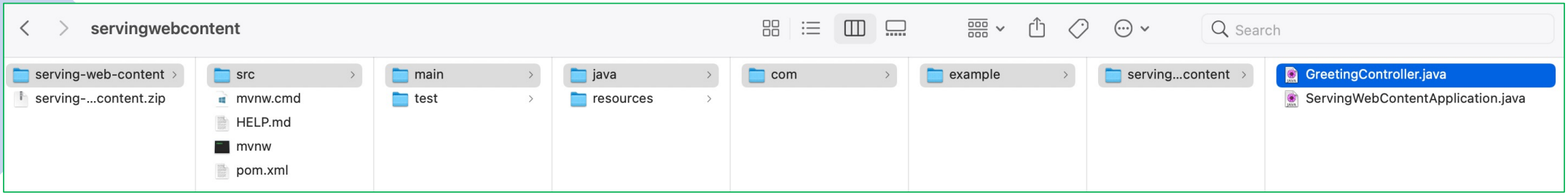
ServingWebContentApplication.java ●

serving-web-content > src > main > java > com > example > servingwebcontent > ServingWebContentApplication.java > ...

```java
package com.example.servingwebcontent;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class ServingWebContentApplication {

    Run | Debug
    public static void main(String[] args) {
        SpringApplication.run(ServingWebContentApplication.class, args);
    }

}
```

Create a Web Controller

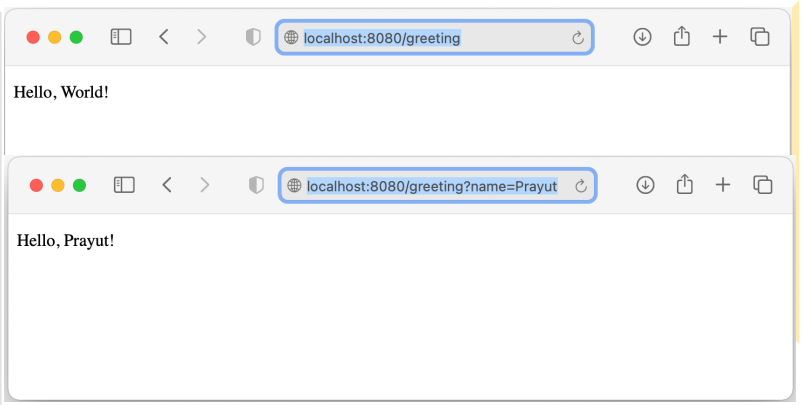
- In Spring's approach to building web sites, HTTP requests are handled by a controller.
- You can easily identify the controller by the @Controller annotation
- In the following example, GreetingController handles GET requests for /greeting by returning the name of a View (in this case, greeting).
- A View is responsible for rendering the HTML content.
- The following listing (from src/main/java/com/example/servingwebcontent/GreetingController.java) shows the controller:



- The implementation of the method body relies on a view technology (in this case, Thymeleaf) to perform server-side rendering of the HTML.
- Thymeleaf parses the greeting.html template and evaluates the th:text expression to render the value of the ${name} parameter that was set in the controller.
- The following listing (from src/main/resources/templates/greeting.html) shows the greeting.html template:

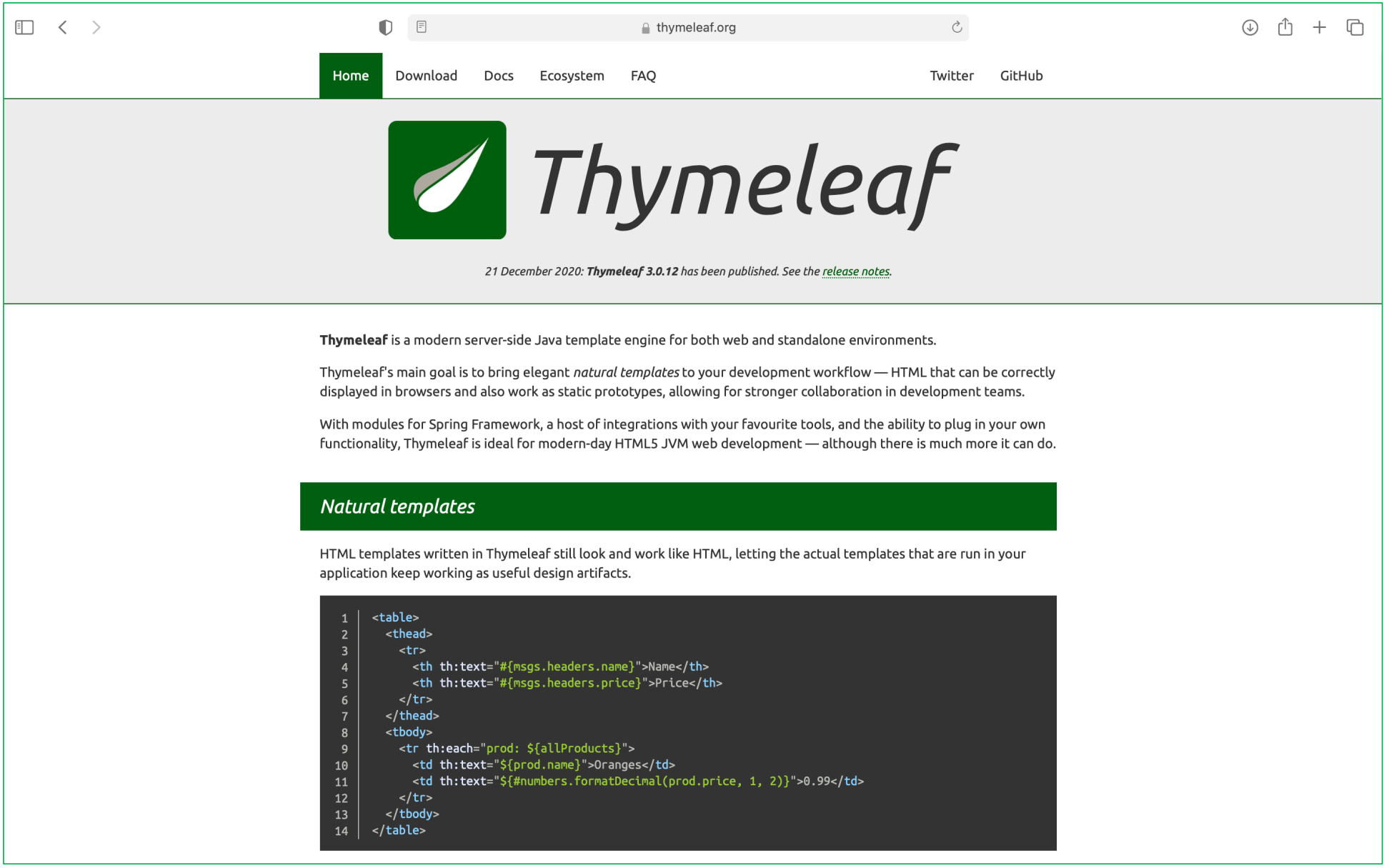
```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Getting Started: Serving Web Content</title>
    <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />
</head>
<body>
    <p th:text="'Hello, ' + ${name} + '!'" />
</body>
</html>
```

```java
package com.example.servingwebcontent;

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

@Controller
public class GreetingController {

        @GetMapping("/greeting")
        public String greeting(@RequestParam(name="name",
required=false, defaultValue="World") String name, Model model) {
                model.addAttribute("name", name);
                return "greeting";
        }

}
```



- This query string parameter is not required.
- If it is absent in the request, the defaultValue of World is used.

- The @GetMapping annotation ensures that HTTP GET requests to /greeting are mapped to the greeting() method.
- @RequestParam binds the value of the query string parameter name into the name parameter of the greeting() method.
- The value of the name parameter is added to a Model object, ultimately making it accessible to the view template.


thymeleaf.org
Home
Download
Docs
Ecosystem
FAQ
Twitter
GitHub
Thymeleaf
21 December 2020: Thymeleaf 3.0.12 has been published. See the release notes.
Thymeleaf is a modern server-side Java template engine for both web and standalone environments.
Thymeleaf's main goal is to bring elegant natural templates to your development workflow — HTML that can be correctly displayed in browsers and also work as static prototypes, allowing for stronger collaboration in development teams.
With modules for Spring Framework, a host of integrations with your favourite tools, and the ability to plug in your own functionality, Thymeleaf is ideal for modern-day HTML5 JVM web development — although there is much more it can do.
Natural templates
HTML templates written in Thymeleaf still look and work like HTML, letting the actual templates that are run in your application keep working as useful design artifacts.
<table>
<thead>
<tr>
<th th:text="#{msgs.headers.name}">Name</th>
<th th:text="#{msgs.headers.price}">Price</th>
</tr>
</thead>
<tbody>
<tr th:each="prod: ${allProducts}">
<td th:text="${prod.name}">Oranges</td>
<td th:text="${#numbers.formatDecimal(prod.price, 1, 2)}">0.99</td>
</tr>
</tbody>
</table>

Three-tier architecture

- Subsystems are organized into 3 layers:
  - Interface layer: includes boundary objects that deal with the user, windows, forms, web pages.
  - Application logic layer: includes control and entity objects.
  - Storage layer: realizes the storage, retrieval, and query of persistent objects.

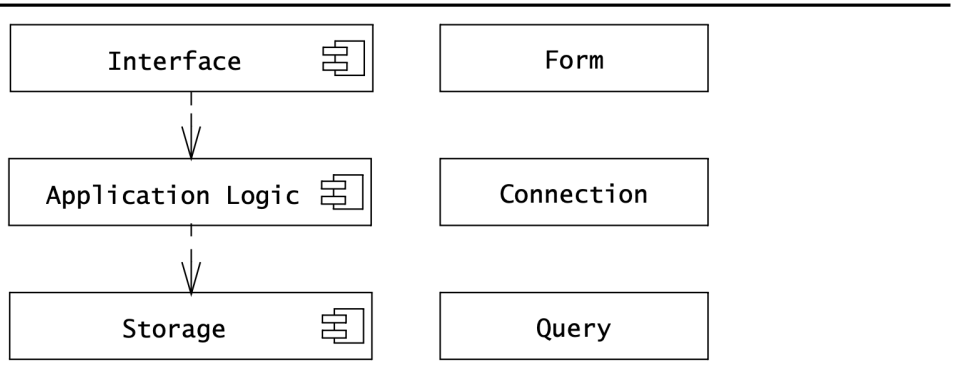
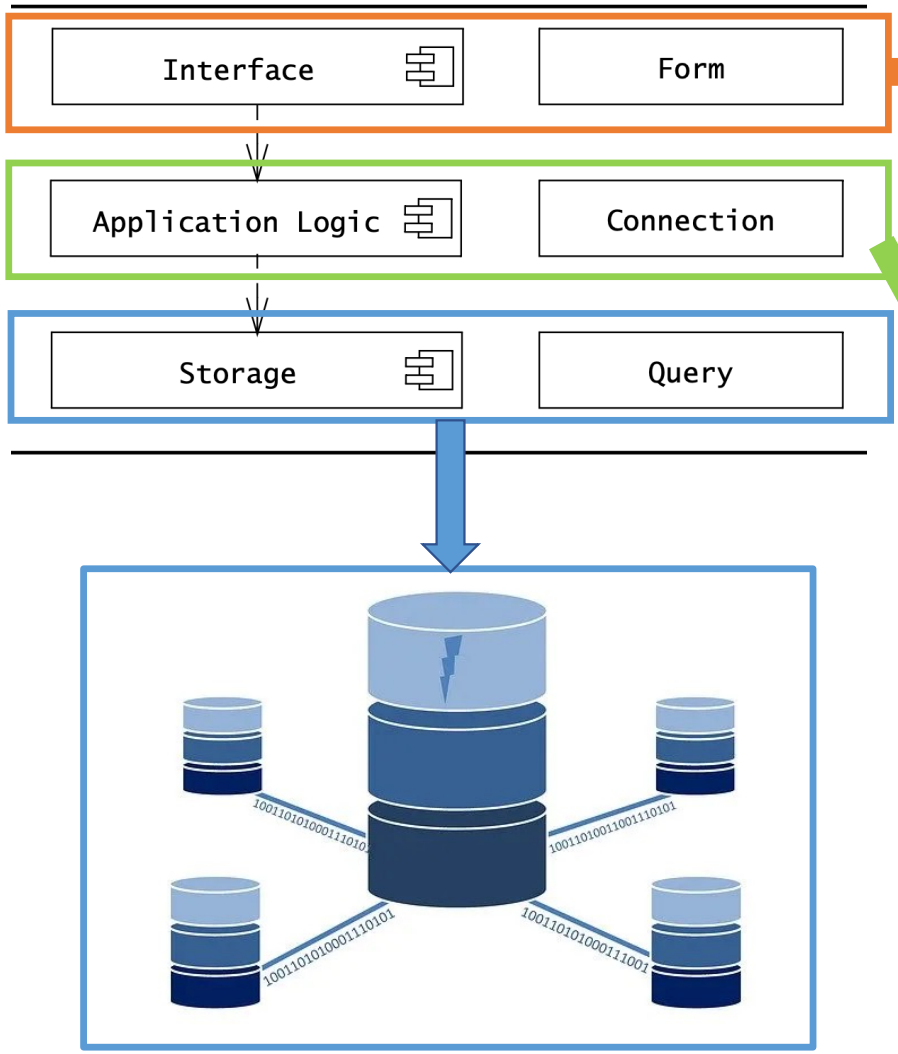
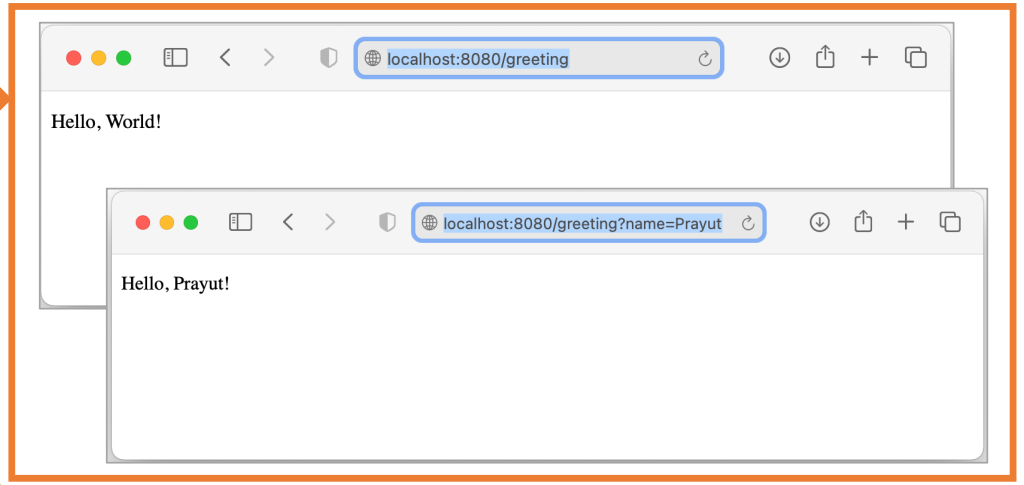
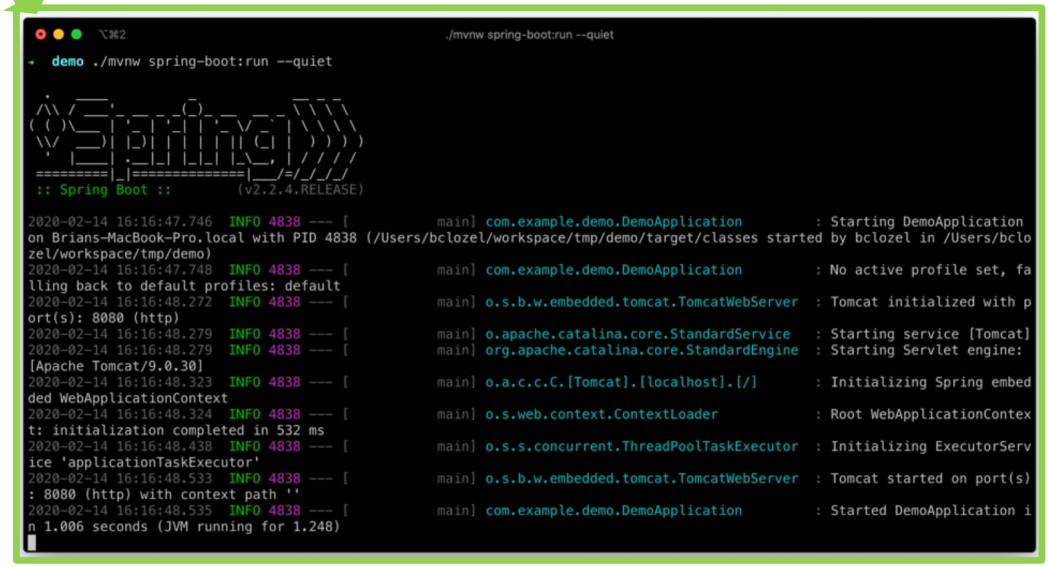


Figure 6-22 Three-tier architectural style (UML component diagram). Objects are organized into three layers realizing the user interface, the processing, and the storage.


Interface
Form
Application Logic
Connection
Storage
Query


localhost:8080/greeting
Hello, World!
localhost:8080/greeting?name=Prayut
Hello, Prayut!


./mvnw spring-boot:run --quiet
demo ./mvnw spring-boot:run --quiet
:: Spring Boot :: (v2.2.4.RELEASE)
2020-02-14 16:16:47.746 INFO 4838 --- [main] com.example.demo.DemoApplication : Starting DemoApplication on Brians-MacBook-Pro.local with PID 4838 (/Users/bclozel/workspace/tmp/demo/target/classes started by bclozel in /Users/bclozel/workspace/tmp/demo)
2020-02-14 16:16:47.748 INFO 4838 --- [main] com.example.demo.DemoApplication : No active profile set, falling back to default profiles: default
2020-02-14 16:16:48.272 INFO 4838 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2020-02-14 16:16:48.279 INFO 4838 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2020-02-14 16:16:48.279 INFO 4838 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.30]
2020-02-14 16:16:48.323 INFO 4838 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2020-02-14 16:16:48.324 INFO 4838 --- [main] o.s.web.context.ContextLoader : Root WebApplicationContext: initialization completed in 532 ms
2020-02-14 16:16:48.438 INFO 4838 --- [main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
2020-02-14 16:16:48.533 INFO 4838 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2020-02-14 16:16:48.535 INFO 4838 --- [main] com.example.demo.DemoApplication : Started DemoApplication in 1.006 seconds (JVM running for 1.248)

Congratulations!

You have just developed a web page by using Spring.