Western Norway
University of
Applied Sciences

Web Frameworks: Spring

Created by Lasse Jenssen

Home

Agenda: The Spring Framwork

- History
- The IoC Container
- IoC - Inversion of Controll
- Dependency Injection
- Spring Boot
- Spring Initializer

The Spring Framework

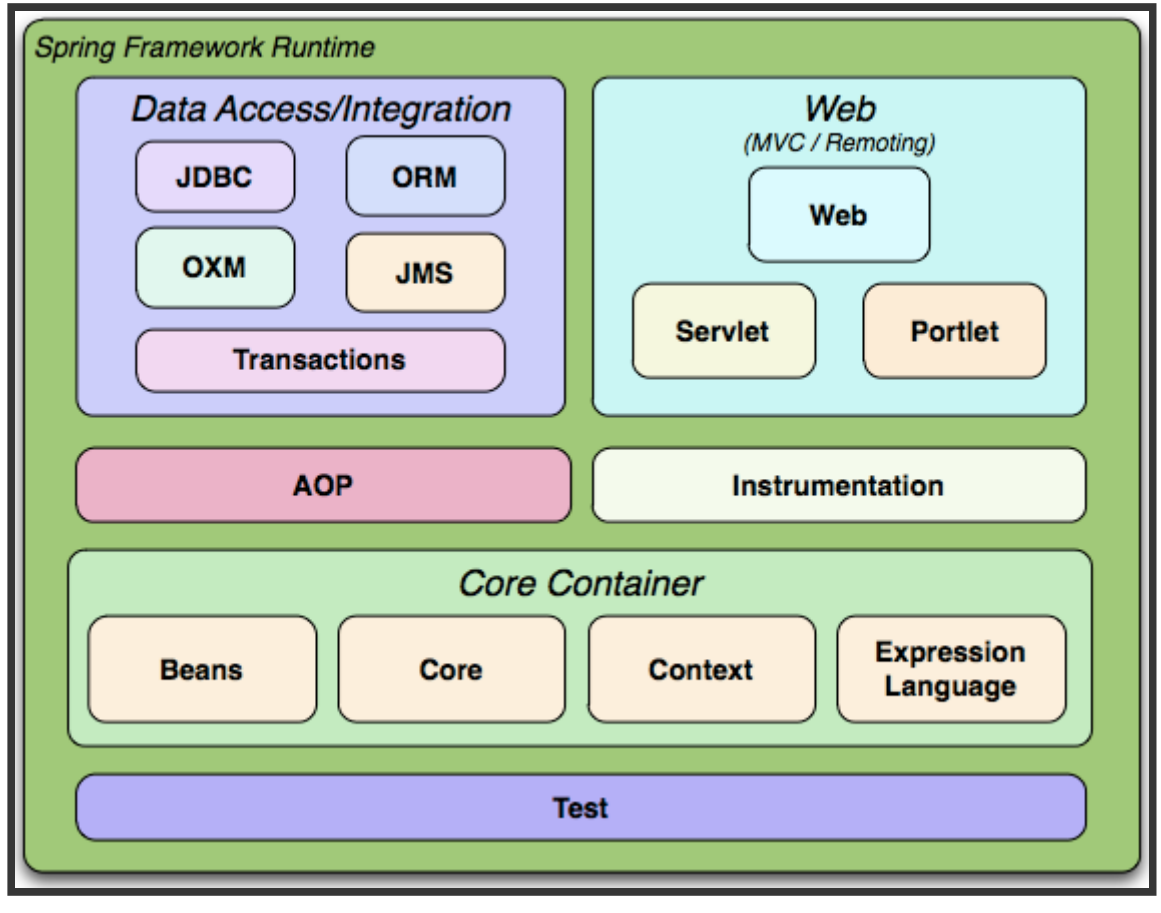
History

- 2002: Book "Expert One-on-One J2EE Design and Development", Rod Johnsen
- 2003: 0.9 version of the Framwork released
- 2004 March: First production release (1.0)
- Latest version (Per 5th of May 2022): 5.3

The Spring Framework

Introduction

- A response to the complexity of the early J2EE specifications.
- A complementary to Java EE.
- Integrates with carefully selected individual specifications from the EE umbrella: Servlet API, WebSocket API, JSON Binding API, JPA etc.
- Very much integrated with Maven (or Gradle).


Spring Framework Runtime
Data Access/Integration
JDBC
ORM
OXM
JMS
Transactions
Web
(MVC / Remoting)
Web
Servlet
Portlet
AOP
Instrumentation
Core Container
Beans
Core
Context
Expression Language
Test

The Spring Framework

Design Philosophy

- Provide choice at every level.
- Accommodate diverse perspectives.
- Maintain strong backward compatibility.
- Care about API design.
- Set high standards for code quality.

http://spring.io/projects

The Spring Framework

The IoC Container

- The core of Spring Framework.
- Responsible to instantiate, configure and assemble the objects.
- Components are called "Beans" (can be any POJOs).
- Two types of IoC Containers: BeanFactory and **ApplicationContext**
- Tasks:
  - to instantiate the application class.
  - to configure the objects.
  - to assemble the dependencies between the objects.

```
public class App {

  public static void main( String[] args ) {

    ApplicationContext context =
              new ClassPathXmlApplicationContext("spring-config.xml");

    SequenceGenerator sequence = context.getBean("sequenceGenerator");
    System.out.println("Next:" + sequence.getNext());

  }
}
```

Bean initialization **by propery**

src/main/resources/spring-config.xml

```
# File:
<beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://ww
   ...
   <bean name="sequenceGenerator" class="no.hvl.dat152.sequence.FirstSequenceGe
      <property name="prefix" value="30"></property>
      <property name="suffix" value="A"></property>
      <property name="initial" value="1000"></property>
   </bean>
</beans>
```

Bean initialization **by constructor**

src/main/resources/spring-config.xml

```
# File: spring-config.xml
<beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://ww
   ...
   <bean name="sequenceGenerator" class="no.hvl.dat152.sequence.FirstSequenceGe
      <constructor-arg value="30"></constructor-arg>
      <constructor-arg value="A"></constructor-arg>
      <constructor-arg value="1000"></constructor-arg>
   </bean>
</beans>
```

The Spring Framework

Dependency Injection (DI)

```
1  public class DateSequenceGenerator {
2
3     private PrefixGenerator prefix;
4     private String suffix;
5     private int initial;
6     private int counter;
7
8     public DateSequenceGenerator() {}
9
10    public void setPrefix(PrefixGenerator generator) {
11      this.prefix = generator;
12    }
13    ...
14 }
```

The Spring Framework

Dependency Injection (DI)

```
<bean id="datePrefixGenerator" class="no.hvl.dat152.prefix.DatePrefixGenerator
    <property name="pattern" value="yyyyMMdd"></property>
</bean>

<bean name="dateSequenceGenerator" class="no.hvl.dat152.sequence.DateSequenceG
    <property name="suffix" value="A"></property>
    <property name="initial" value="1000"></property>
    <property name="datePrefixGenerator">
        <ref bean="datePrefixGenerator"></ref>
    </property>
</bean>
```

The Spring Framework

Dependency Injection: Autowiring

```
<bean id="datePrefixGenerator" class="no.hvl.dat152.prefix.DatePrefixGenerator
    <property name="pattern" value="yyyyMMdd"></property>
</bean>

<bean name="dateSequenceGenerator" class="no.hvl.dat152.sequence.DateSequenceG
    <property name="suffix" value="A"></property>
    <property name="initial" value="1000"></property>



</bean>
```

The Spring Framework

Dependency Injection (DI)

```
1  public class DateSequenceGenerator {
2
3      private PrefixGenerator prefix;
4      private String suffix;
5      private int initial;
6      private int counter;
7
8      public DateSequenceGenerator() {}
9
10     @Autowired
11     public DateSequenceGenerator(PrefixGenerator generator) {
12         this.prefix = generator;
13     }
14     ...
15 }
```

The heart of the Spring Framework

Dependency Injection

- Programming technique .
  - Making a class independent of its dependencies.
- By decoupling the usage of an object from its creation.
- Dependent of 4 roles:
  - The service/instance you want to use.
  - The client that uses the service/instance.
  - An interface that’s used by the client and implemented by the service.
  - The injector which creates a service instance and injects it into the client.

The Spring Framework

Dependency Injection (DI)

```
1  public class DateSequenceGenerator {          # <= Client
2
3      private PrefixGenerator prefix;             # <= Interface
4      private String suffix;
5      private int initial;
6      private int counter;
7
8      public DateSequenceGenerator() {}
9
10     @Autowired                              # <= Injector (one way to inject)
11     public DateSequenceGenerator(PrefixGenerator generator) {
12         this.prefix = generator;
13     }
14     ...
15 }
16
17 public class DatePrefixGenerator {             # <= Service/Instance
18 }
```

Other ways to initialize beans

Configuration Code (Java)

```java
@Configuration
@ComponentScan("no.hvl.dat152")
public class GeneratorConfig {
   @Bean
   public PrefixGenerator() {
     return new DatePrefixGenarator("yyyyMMdd");
   }
}
```

Sources

Dependency Injection

https://www.programmergirl.com/spring-dependency-injection/

https://www.baeldung.com/constructor-injection-in-spring

Example with SequenceGenerator:

"Spring Recipes: A Problem-Solution Approach" (Second Editon),
Gary Mak, Josh Long and Daniel Rubio

Spring Boot

An easier approach to the Spring Framework

An introduction to

Spring Boot

- Spring Boot is built on the top of the Spring framework (an extension).
- Features:
  - Standalone (Embedded server: No need for a Tomcat Web Server).
  - Opinionated (Prebuild configuration of dependencies).
  - Autoconfiguration.

Spring Boot

Provides a number of starter dependencies for different Spring modules.

- spring-boot-starter-web
- spring-boot-starter-data-jpa
- spring-boot-starter-security
- spring-boot-starter-test
- And several more

Maven Dependencies

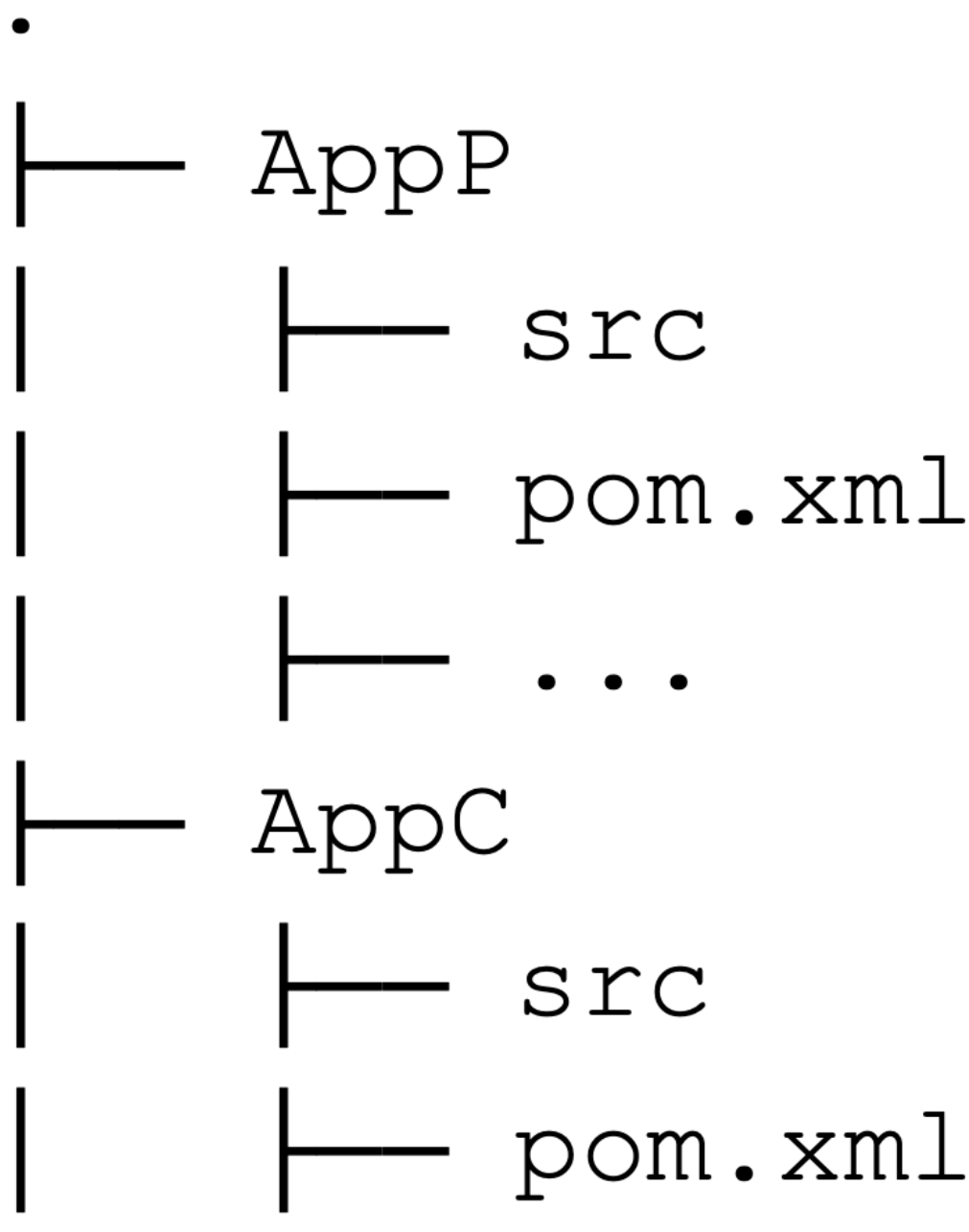
The parent dependency

```
<parent>
    <groupid>org.springframework.boot</groupid>
    <artifactid>spring-boot-starter-parent</artifactid>
    <version>2.7.0</version>
    <relativepath></relativepath>  <!-- lookup parent from repository -->
</parent>
```

Maven Dependencies

Starter Dependencies

```
<dependency>
    <groupid>org.springframework.boot</groupid>
    <artifactid>spring-boot-starter-web</artifactid>
    <version>2.4.4</version>
</dependency>
```


.
AppP
src
pom.xml
...
AppC
src
pom.xml

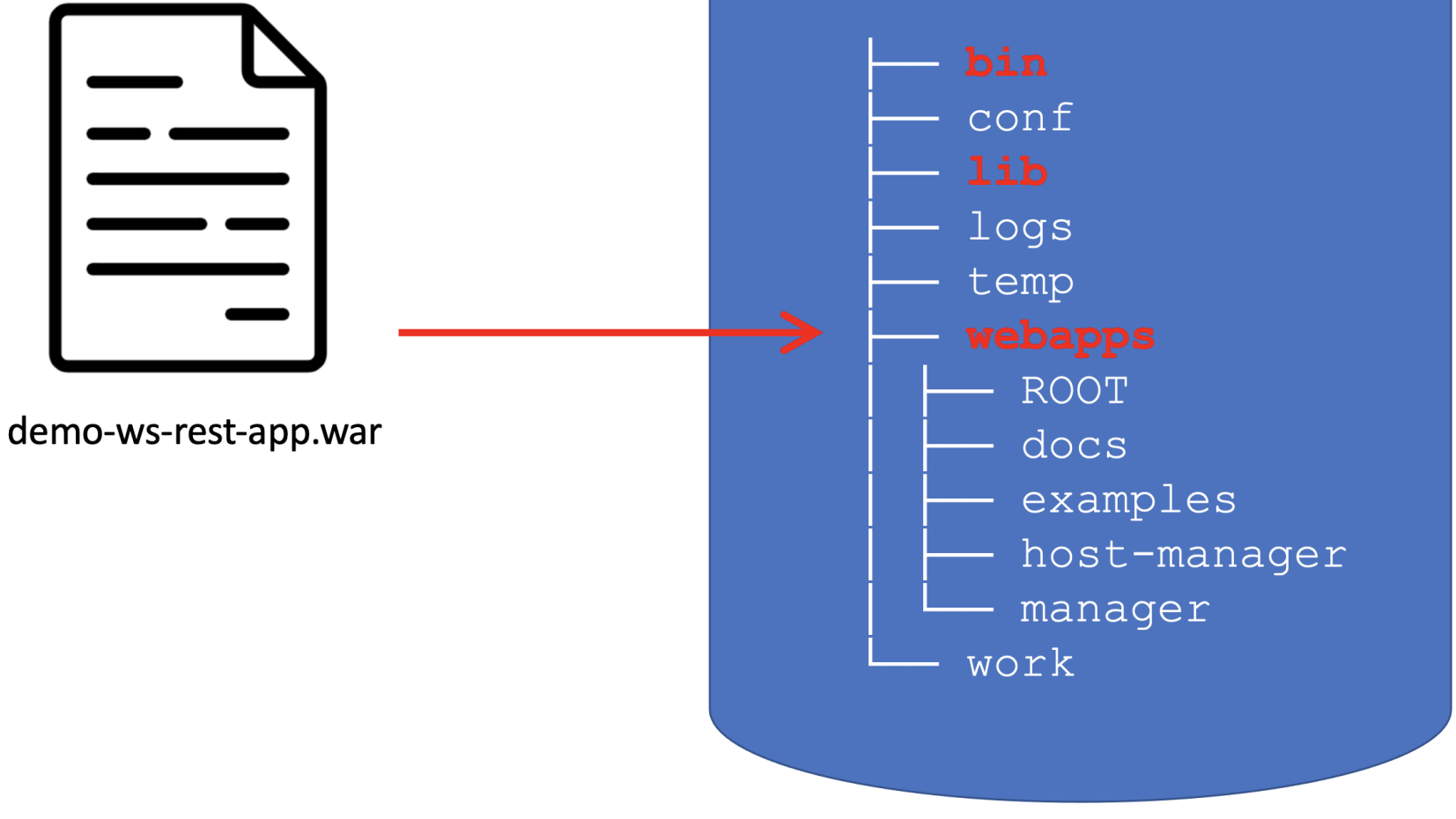
Maven Dependencies

Parent RelativePath

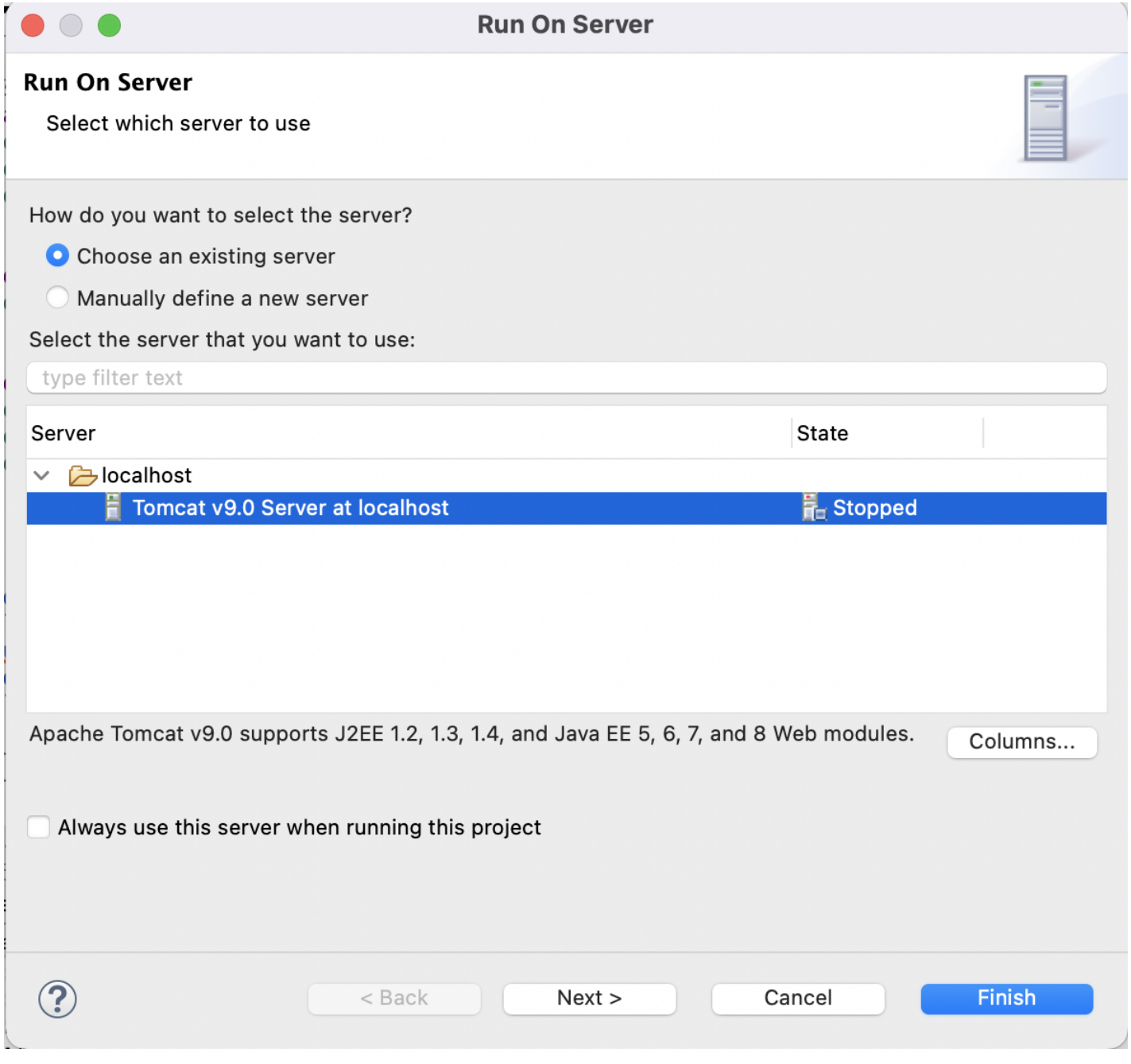
```
<parent>
    <groupid>com.example.demo</groupid>
    <artifactid>AppP</artifactid>
    <version>1.0.0</version>
    <relativepath>../AppP</relativepath>
</parent>
```

Spring Boot & Spring Web

External vs Embedded Server


demo-ws-rest-app.war
Tomcat EE
bin
conf
lib
logs
temp
webapps
ROOT
docs
examples
host-manager
manager
work

Applications					
Path	**Version**	**Display Name**	**Running**	**Sessions**	**Commands**
/	*None specified*	Welcome to Tomcat	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/demo-01-several-controllers	*None specified*	Demo 01 - Serveral Controllers	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/docs	*None specified*	Tomcat Documentation	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/examples	*None specified*	Servlet and JSP Examples	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/host-manager	*None specified*	Tomcat Host Manager Application	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/manager	*None specified*	Tomcat Manager Application	true	1	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes


Run On Server
Run On Server
Select which server to use
How do you want to select the server?
Choose an existing server
Manually define a new server
Select the server that you want to use:
type filter text
Server
State
localhost
Tomcat v9.0 Server at localhost
Stopped
Apache Tomcat v9.0 supports J2EE 1.2, 1.3, 1.4, and Java EE 5, 6, 7, and 8 Web modules.
Columns...
Always use this server when running this project
< Back
Next >
Cancel
Finish

External Web Servers

Pros

- Potentially more flexible application architecture.
- Really easy to switch servers later.
- Application errors can't harm the server.
- Easy to deploy app updates without restarting the server.
- Performance and correctness: servers like nginx are highly optimized and tested for complete HTTP correctness, which your app then gets for free.

External Web Servers

Cons

- Extra performance overhead: there could be anything from an extra layer of method abstraction up to CGI-level overhead for your app and the server to communicate.
- Deployment complexity: you have to maintain the web server and the application, deploy them individually, ad hoc version testing, etc.
- Trickier development environment.

Spring Boot Default

Embedded Web Server

```
1 > mvn spring-boot:run
```

```
1 > mvn package
2 > java -cp target/demo-01-several-controllers-0.0.1-SNAPSHOT.jar /
3         no.hvl.dat152.Demo01JettyWebApp
```

Embedded Web Servers

Pros

- More self-contained applications. This helps a lot during development.
- As a dependency of your application, you can test against server versions just like any other dependency.
- More control over how the web server behaves (custom filters, headers, caching).
- Single object to be deployed.
- Easy to integrate with Docker, Kubernetes, OpenShift etc.

Embedded Web Servers

Cons

- Your application has to be designed around the API of whatever server you are using, making it harder to change servers later.
  (Java doesn't really have this problem, as you can still use the servlet API when embedding)
- Dependency bloat, as you have to include all the dependencies of the web server.
- More effort to deploy hotfixes to security exploits in the server.
- You can't group multiple applications behind one server without a proxy.
  (Not really an issue if deploying to virtual platform)
- A single uncaught exception is enough to take down the entire application server.

Next

Web Development: Using Spring Web MVC

Home