

# Code Once, Use Everywhere: Building Shared Libraries for Multiple Projects



Vadzim Prudnikau



HUMAN MADE  
NO AI USED

# What you won't learn today

- How to implement a common library in YOUR team
- However, you will get some useful insights if you need to do that

# Shortly about me

I run training in Spring Boot, Architecture, DDD, TDD, etc.



**Vadzim Prudnikau**

 [vadim-prudnikov](https://www.linkedin.com/in/vadim-prudnikov)

Co-owner and instructor at  trainitek

Hands-on architect at  APOTEK 1



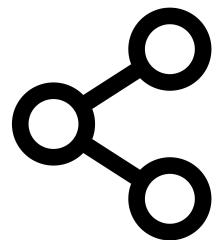
# My motivation

- I developed a common library for Apotek 1 that several projects have used for 3 years 💪
- I saw different approaches at work and conferences, and I felt I could do it a bit better💡
- I would like to share my own experience ❤️

# What will we address today?

- **When** we need **to share** our code
- **What not** to put into the shared code
- **Structure** of the solution **and** its **integration** with **Spring Boot** ★
- Documentation
- Making changes
- Forcing vs advertising

# When we need to share our code



# When we need to share our code

Multiple projects solve the same  
technical problems

&&

You are tired of copy-pasting the  
same solutions across projects

People agree to use standardised  
approaches

&&

The shared code costs less than  
repeating it

# It's not that easy

- I made mistakes myself regarding these points. It's quite hard to follow it.

Multiple projects solve the same technical problems

&&

You are tired of copy-pasting the same solutions across projects

&&

People agree to use standardised approaches

&&

The shared code costs less than repeating it

# What **not** to put into the shared code





# What **not** to put into the shared code

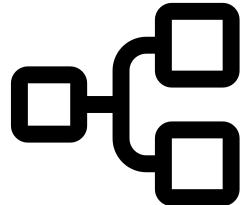
**Business logic**

**Unstable features**

**Expectations**

**Ego and/or desire  
to help**

# Structure of the library and its integration with Spring Boot



# Rationale 1/5

- The library should be easy to include, ideally by adding a Maven dependency

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```



# Rationale 2/5

- Developers should **not** be bothered by the configuration of the library



```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:task="http://www.springframework.org/schema/task"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans.xsd
                           http://www.springframework.org/schema/context
                           http://www.springframework.org/schema/context/spring-context.xsd
                           http://www.springframework.org/schema/tx
                           http://www.springframework.org/schema/tx/spring-tx.xsd
                           http://www.springframework.org/schema/task
                           http://www.springframework.org/schema/task/spring-task.xsd">

    <tx:annotation-driven transaction-manager="transactionManager"/>

    <context:component-scan base-package="com.terminal.domain, com.terminal.dao, com.terminal.utils"/>

    <bean id="transactionManager"
          class="org.springframework.orm.hibernate3.HibernateTransactionManager">
        <property name="sessionFactory" ref="sessionFactory"/>
    </bean>

    <bean id="messageSource"
          class="org.springframework.context.support.ResourceBundleMessageSource">
        <property name="basenames">
            <list>
                <value>mymessages</value>
            </list>
        </property>
    </bean>

    <task:scheduler id="jobScheduler" pool-size="10"/>

    <beans profile="test">

        <bean class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close" id="dataSource">
            <property name="driverClassName" value="org.h2.Driver" />
            <property name="url" value="jdbc:h2:~/test;MODE=PostgreSQL" />
            <property name="username" value="sa" />
            <property name="password" value="" />
        </bean>
    
```

# Rationale 3/5

- It should be possible to use only specific modules

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

## Rationale 4/5

- The library structure should follow and provide some guidelines

# Rationale 5/5

- Our code doesn't always depend on the Spring Framework



# Root level



# Root pom.xml

```
<groupId>no.apotek1.javalibs</groupId>
<artifactId>javalibs-root</artifactId>
<version>3.7-SNAPSHOT</version>
<packaging>pom</packaging>
```

```
<properties...>
<modules...>
```

```
<build>
  <pluginManagement...>
    <plugins...>
  </build>
  <dependencyManagement...>
  <dependencies...>
```

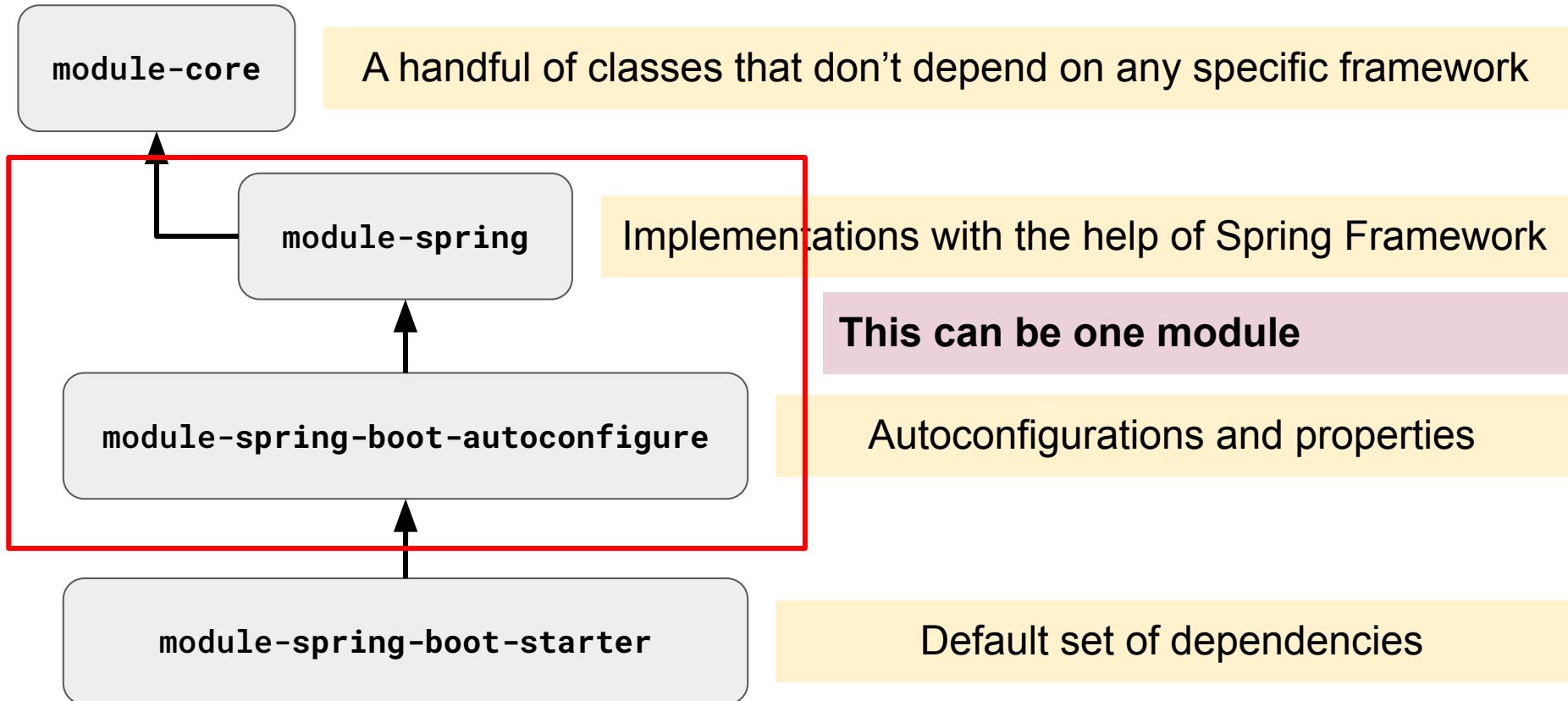
Shared properties: JDK version, dependency versions, etc.

```
<properties>
  <!-- default environment properties -->
```

Default plugins settings: checkstyle, surefire, etc.

```
<plugin>nexus-plugin.version>4.1.1</nexus-plugin.version>
  <artifactId>maven-surefire-plugin</artifactId>
  <version>${maven-surefire-plugin.version}</version>
  <configuration>lon>10.21.4</configuration>
    <includes>
      <include>**/*Spec*.*</include>
      <include>**/Test*.java</include>
    <dependency>include>**/*Test.java</dependency>
      <groupId>org.springdoc</groupId>
      <artifactId>springdoc-openapi-common</artifactId>
      <version>${springdoc-openapi.version}</version>
    </dependency>S</skip> surefire tests</skip>
```

# Module level: concept





# Module level: example

The screenshot shows a file tree structure in a IDE. The root folder is 'cqs [javalibs-cqs]'. It contains several subfolders and files:

- 'core [javalibs-cqs-core]' (indicated by a blue folder icon)
- 'spring [javalibs-cqs-spring]' (indicated by a blue folder icon)
- 'spring-boot-autoconfigure [javalibs-cqs-spring-boot-autoconfigure]' (indicated by a blue folder icon)
- 'spring-boot-starter [javalibs-cqs-spring-boot-starter]' (indicated by a blue folder icon)
- 'spring-test [javalibs-cqs-spring-test]' (indicated by a blue folder icon)
- 'javalibs-cqs.iml' (indicated by an orange folder icon)
- 'pom.xml' (indicated by a blue file icon)
- 'README.md' (indicated by a blue file icon)



# -core and -spring

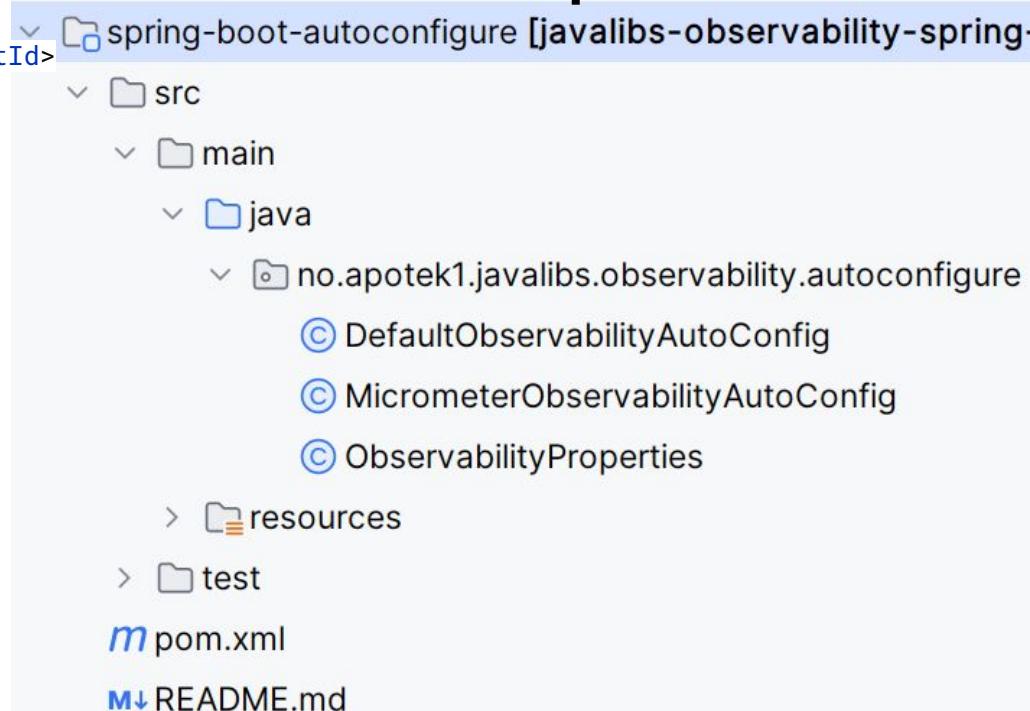
```
observability [javalibs-observability]
  core [javalibs-observability-core]
    src
      main
        java
          no.apotek1.javalibs.observability
            CurrentSpan
            CurrentTrace
            NopObservabilityContext
            ObservabilityContext
    test
    pom.xml
    README.md
```

```
spring [javalibs-observability-spring]
  src
    main
      java
        no.apotek1.javalibs.observability.spring
        test
    pom.xml
    README.md
```



# -spring-boot-autoconfigure

```
<dependency>
    <groupId>no.apotek1.javalibs</groupId>
    <artifactId>javalibs-observability-spring</artifactId>
    <version>${project.version}</version>
</dependency>
<dependency>
    <groupId>no.apotek1.javalibs</groupId>
    <artifactId>javalibs-test-core</artifactId>
    <version>${project.version}</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>no.apotek1.javalibs</groupId>
    <artifactId>javalibs-test-spring</artifactId>
    <version>${project.version}</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>io.micrometer</groupId>
    <artifactId>micrometer-tracing</artifactId>
    <optional>true</optional>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-autoconfigure</artifactId>
</dependency>
```





# -spring-boot-starter



spring-boot-starter [javalibs-observability-spring-boot-starter]

*m* pom.xml

*M* README.md

```
<dependency>
    <groupId>no.apotek1.javalibs</groupId>
    <artifactId>javalibs-observability-spring-boot-autoconfigure</artifactId>
    <version>${project.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<dependency>
    <groupId>io.micrometer</groupId>
    <artifactId>micrometer-tracing</artifactId>
</dependency>
<dependency>
    <groupId>io.micrometer</groupId>
    <artifactId>micrometer-tracing-bridge-otel</artifactId>
</dependency>
```

# Auto Configurations

```
@AutoConfiguration  
@ConditionalOnClass(Tracer.class)  
@EnableConfigurationProperties(ObservabilityProperties.class)  
public class MicrometerObservabilityAutoConfig {  
  
    @Bean  
    @ConditionalOnMissingBean  
    ObservabilityContext micrometerObservabilityContext(Tracer tracer) {  
        return new MicrometerObservabilityContext(tracer);  
    }  
}
```

# @Configuration and @AutoConfiguration

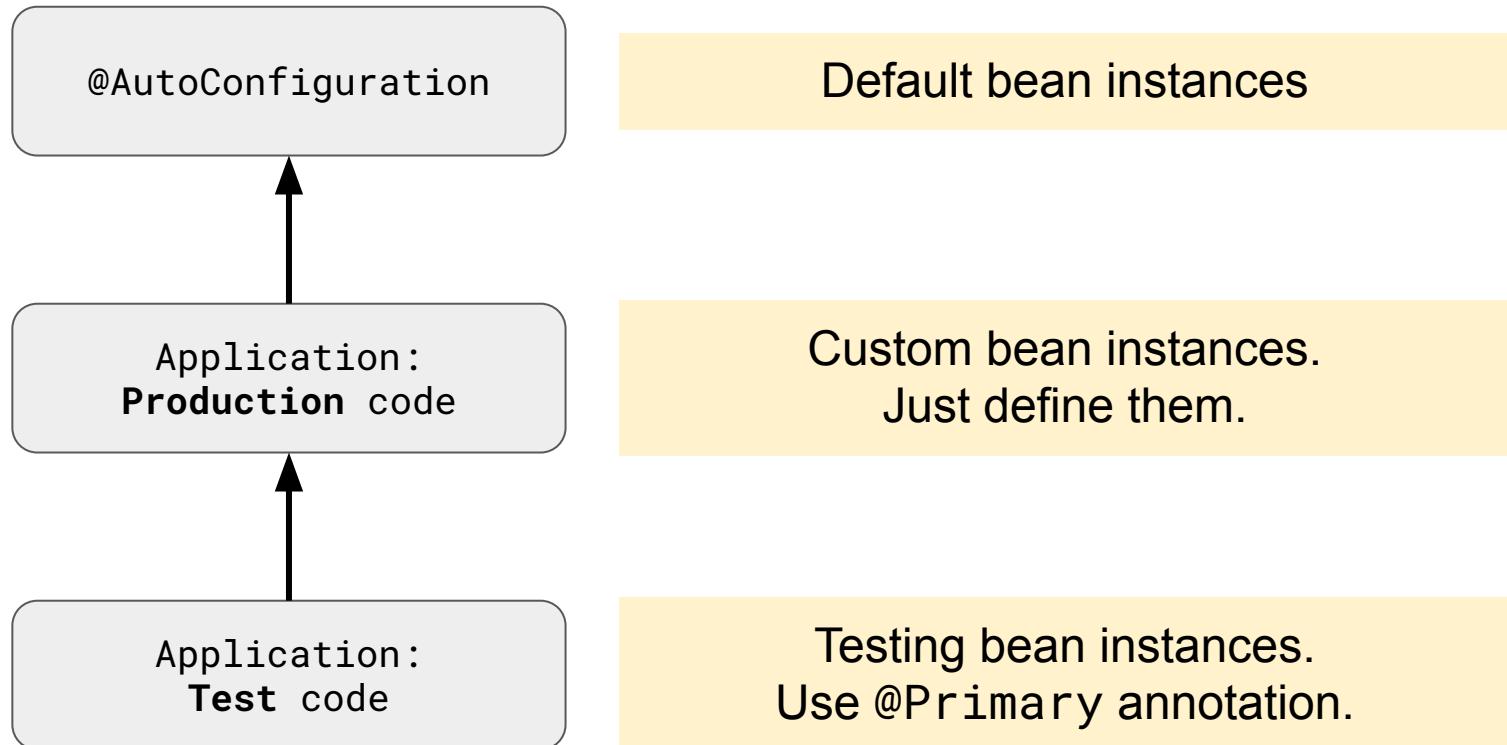
```
@Configuration  
public class MyConfig {  
  
    @Bean  
    ServiceA serviceA() {  
        return new ServiceA(serviceB());  
    }  
  
    @Bean  
    ServiceB serviceB() {  
        return new ServiceB();  
    }  
}
```

```
@AutoConfiguration  
public class MyAutoConfig {  
  
    @Bean  
    ServiceA serviceA(ServiceB serviceB) {  
        return new ServiceA(serviceB);  
    }  
  
    @Bean  
    ServiceB serviceB() {  
        return new ServiceB();  
    }  
}
```

# Auto Configurations

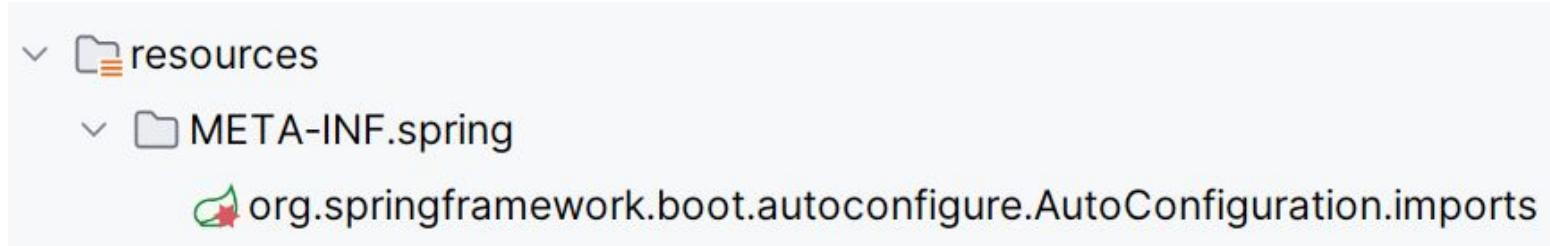
```
@AutoConfiguration  
@ConditionalOnClass(Tracer.class)  
@EnableConfigurationProperties(ObservabilityProperties.class)  
public class MicrometerObservabilityAutoConfig {  
  
    @Bean  
    @ConditionalOnMissingBean  
    ObservabilityContext micrometerObservabilityContext(Tracer tracer) {  
        return new MicrometerObservabilityContext(tracer);  
    }  
}
```

# Overriding beans



# Auto Configurations

- Make sure your auto configuration runs **after** ordinary configurations!



# @Fallback ?

```
@Configuration  
public class MovieConfiguration {  
  
    @Bean  
    public MovieCatalog firstMovieCatalog() { ... }  
  
    @Bean  
    @Fallback  
    public MovieCatalog secondMovieCatalog() { ... }
```

- `@Fallback` should be used by your application e.g. when you have a multi-module setup. It's idiomatically incorrect to use it in auto configurations.

# @TestBean ?

```
class OverrideBeanTests {  
    @TestBean  
    CustomService customService;  
  
    // test case body...  
  
    static CustomService customService() {  
        return new MyFakeCustomService();  
    }  
}
```

- `@TestBean` is useful when you need to redefine a bean for a particular test class.

# Code quality

- To reduce maintenance costs, we need to delivery code of good quality.  
Ideally, automatically verified.
- At Apotek 1, we use the following tools:
  - **EditorConfig** - for code formatting
  - **Checkstyle** and **Error Prone** - to adhere to coding standards
  - **ArchUnit** - to spot barely visible issues and for knowledge sharing
  - **Snyk** - for automated vulnerability scans

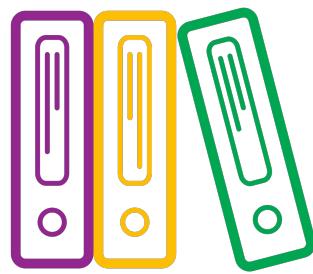
# ArchUnit example: shared library

```
@ArchTest
@ArchTest
public static final ArchRule autoconfiguredBeansAreAnnotatedWithAnyConditionalAnnotation =
    public static final ArchRule controllersAreAnnotatedWithOpenAPIAnnotations =
        methods().that().areAnnotatedWith(Bean.class)
            controllers().should().beAnnotatedWith(Tag.class);
            .and().areDeclaredInClassesThat().areMetaAnnotatedWith(AutoConfiguration.class)
            .should().beMetaAnnotatedWith(Conditional.class)
            .as("Auto-configuration beans should be annotated with anyage) {
return slice @Conditional annotation")+ ".(**)"
            .because("Spring Framework should not create those beans
                    if an application already declared beans of the same type");
            .as("Code is free of package cycles")
            .because("package cycles is an indicator of problems with
                    code design related to modularity and dependency direction.")
}
```

# ArchUnit example: client project

```
class ArchitectureTest {  
  
    @ArchTest  
    public static final ArchTests standardTests = ArchTests.in(StandardArchTests.class);  
  
    @ArchTest  
    public static final ArchRule codeIsFreeOfCycles =  
        codeIsFreeOfPackageCyclesRule(ArchitectureTest.class.getPackageName());  
  
    @ArchTest  
    public static final ArchTests springTests = ArchTests.in(SpringFrameworkArchTests.class);  
  
    @ArchTest  
    public static final ArchTests dddTests = ArchTests.in(DDDArchTests.class);  
  
    @ArchTest  
    public static final ArchTests cqsTests = ArchTests.in(CQSArchTests.class);  
  
    @ArchTest  
    public static final ArchTests webTests = ArchTests.in(WebArchTests.class);  
}
```

# Documentation



# Make sure people can find it

- README files
- Confluence
- etc.

# DRY?

- The DRY principle doesn't work that well with documentation
- Sometimes, it's worth having duplicated information, cross-links, etc. in multiple places
- This is a price to pay for your convenience

# README.md

- Make sure people can navigate through multiple README files by using **links**
- Imagine that your documentation is read by **junior** developers
- Supply your documentation with **examples** and **guidelines**

## Modules

Check `README.md` files in each submodule for more information.

### test

[Common dependencies and classes that help to write tests](#)

### security

Security related classes for authentication and authorization

### web

Components that help to write REST API and test it

## Test

This module contains code that helps to simplify writing tests.

## Usage

## Maven Dependencies

In most cases, you just need to include the `javalibs-test-spring-boot-starter` module to a list of your dependencies.

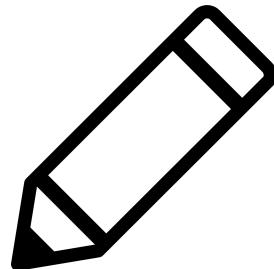
## Database Support

### Transaction in Tests

If you are writing a test that needs a transaction then you can use the `TransactionSupport` trait:

```
class MySpec extends Specification implements TransactionSupport {  
    ...  
  
    def "test"() {  
        ...  
        def result = tx { doSomethingInTransaction() }  
        ...  
    }  
}
```

# Making changes



# General tips

- Making changes should be straightforward for contributors
- Avoid long and complicated routines
- Fix the guidelines if people can't follow them
- If it happens, it's your mistake, not theirs

# Versioning (3.7.10893)

- **Major** - for incompatible changes like:
  - JDK 17 -> JDK 21
  - Upgrading major versions of Spring Framework
  - etc.
- **Minor** - ALSO for changes where some migration is required.
  - We chose this because we don't want to have too high major version (e.g. 34).
  - However, it doesn't mean that you should do the same.
- **Patch** - this version is automatically updated by the CI/CD pipeline and is set to the build ID

# Documenting the changes

- We decided to use a **CHANGELOG** file - something simple that works for us
- <https://keepachangelog.com/en/1.0.0/>

## [3.5.x] - 2025-01-15

### Changed

- <https://apotek1.atlassian.net/browse/JAAV-341>: exceptions occurred in REST controllers are now handled in a more robust way. The response body now *always* contains an `errorId` field that can help to identify the error in the logs. In most cases, this change should not affect the existing projects, however, some titles of the error messages may change.

## [3.4.x] - 2025-01-08

### Changed

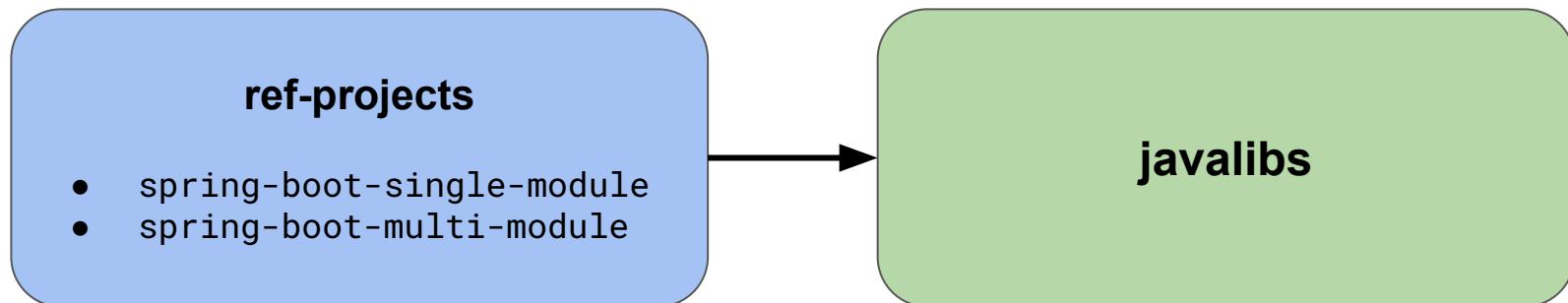
- <https://apotek1.atlassian.net/browse/JAAV-354>: all versions of dependencies have been updated to the latest stable versions.

# Supporting old versions

- Our shared library doesn't support multiple major version 🤔
- Instead, we chose to constantly upgrade our systems and depend on the latest version of the library💡
- **Result:** simplicity at the given capacity🚀

# The change is done! Just ship it?

- A successful build is not enough
- It's smart to create an example of a client project that can help verify the changes
- In our case, we call it “reference projects”

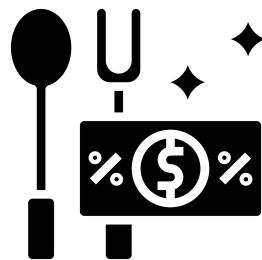
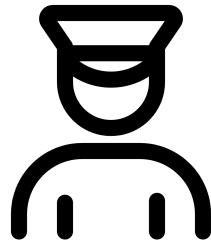


# Reproducible builds?

```
<properties>
    <javalibs.version>[3.7,3.8)</javalibs.version>
</properties>

...
<dependency>
    <groupId>no.apotek1.javalibs</groupId>
    <artifactId>javalibs-cqs-spring-boot-starter</artifactId>
    <version>${javalibs.version}</version>
</dependency>
```

# Forcing vs Advertising



# Forcing vs Advertising

- Prefer to advertise common libraries rather than forcing people to use them 
- This is the way to verify the quality and demand for the shared code 
- Otherwise, there is a chance of living inside the illusion of a good job done 

# Conclusions

- (1/6) Make sure you need a shared library
- (2/6) Don't include anything unnecessary
- (3/6) Create a stable and understandable structure
- (4/6) Help people navigate and contribute via well-written documentation
- (5/6) Choose the easiest versioning strategy
- (6/6) If people want to use it, you have done a good job

# Thank You!

Presentation Slides



 vadim-prudnikov

