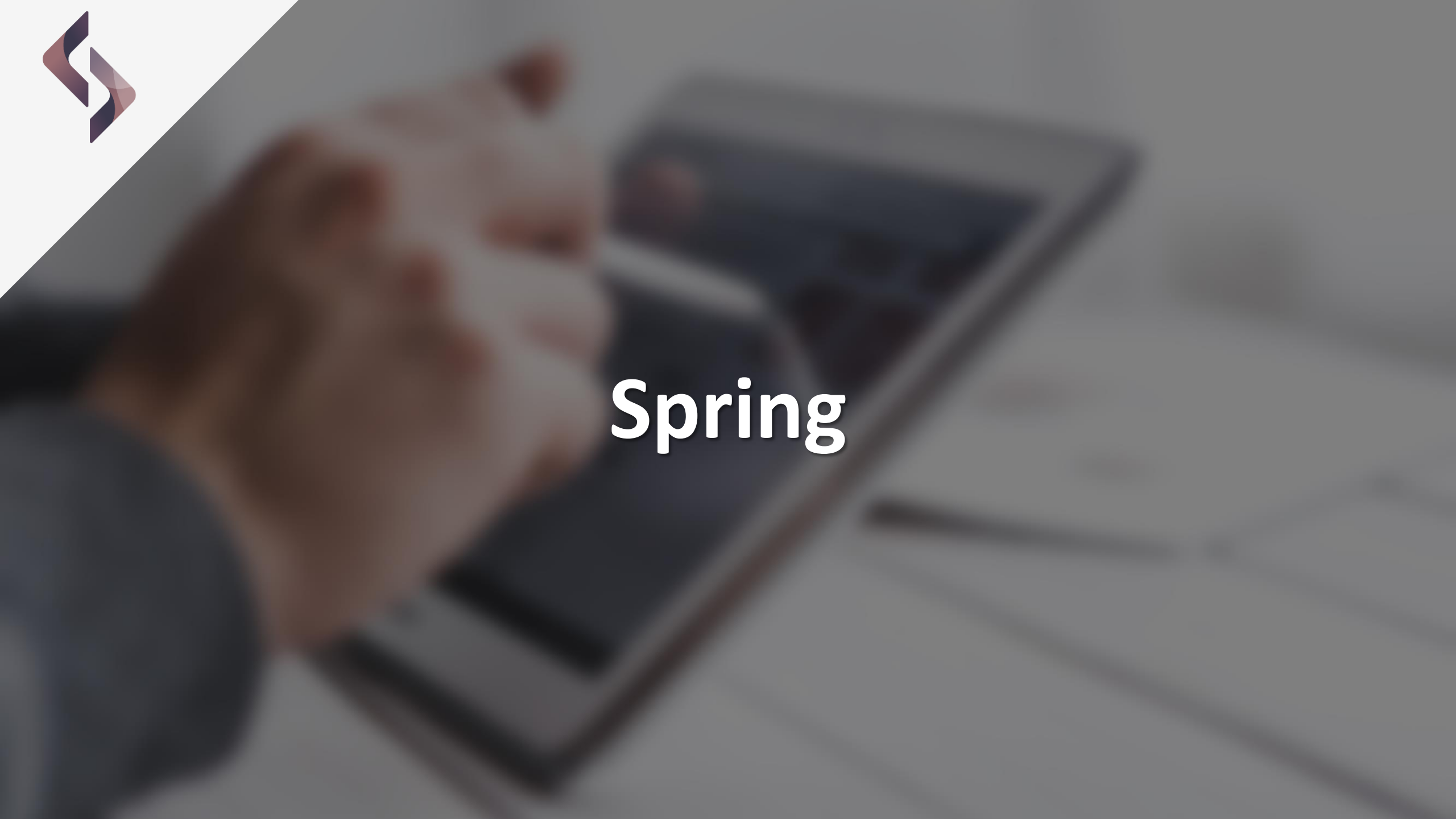


# Spring

# Agenda

1. Spring Framework
2. Spring Core and first application
3. Spring Boot
4. Spring Data
5. Spring MVC
6. Thymeleaf
7. Spring Security
8. Spring Boot REST API
9. Integration with Angular

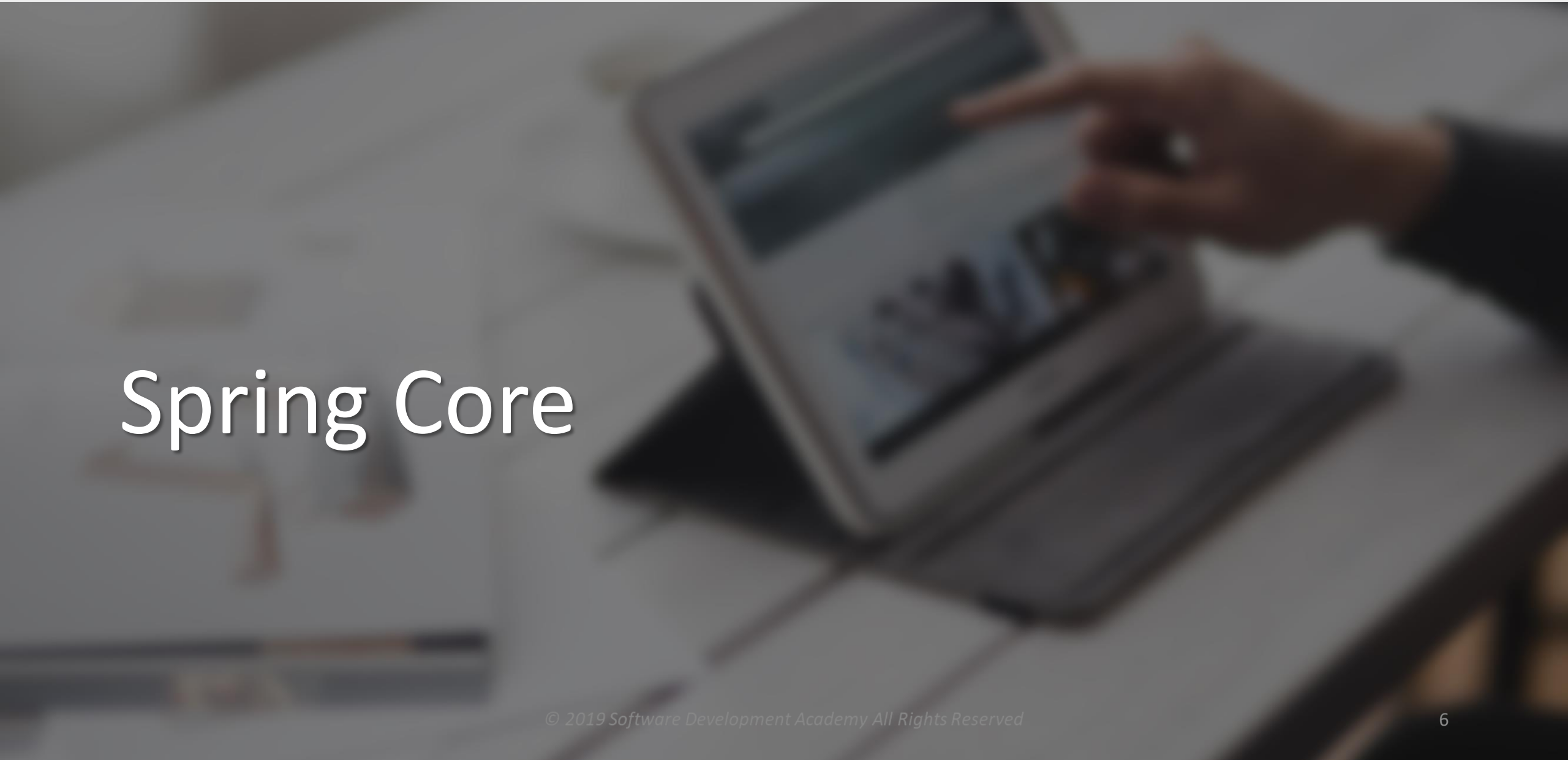
# Spring Framework

# What is Framework?

- Application framework is a tool used to implement standard structure of application.
- It lets developers create applications faster and easier because of variety of already created tools.

# Spring Framework

- The most popular Framework for JVM developers
- Big community
- Wide variety of tools



# Spring Core

# Key features

- Dependency Injection
- Events
- Resources
- i18n
- Validation
- Data Binding

# Spring IoC container

- org.springframework.beans
- org.springframework.context
- Bean Factory
- Application Context
- Beans

# Spring IoC container

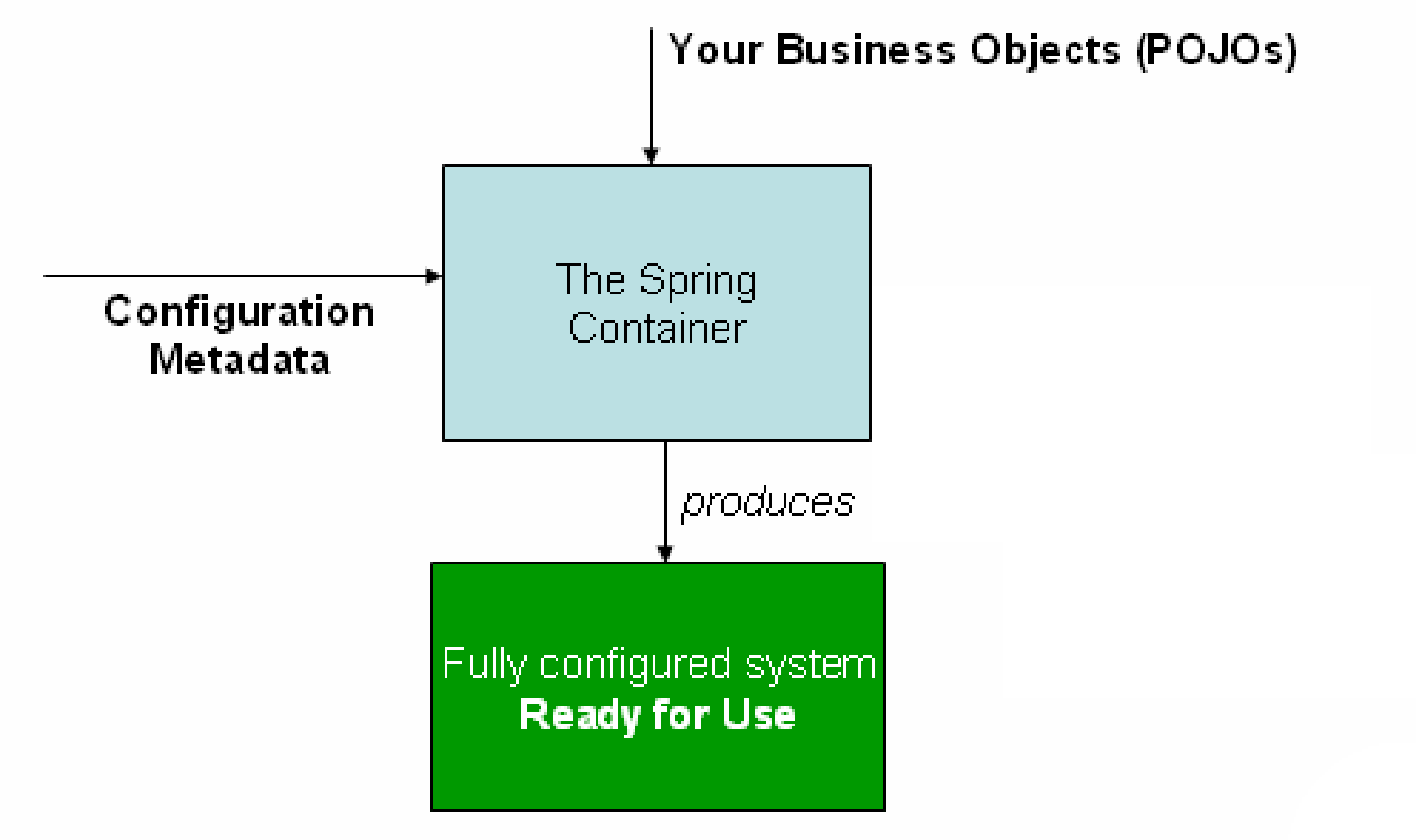



Diagram: https://docs.spring.io

# Configuration metadata

Formats:

- XML
- Annotations
- Java

# Configuration - XML

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:p="http://www.springframework.org/schema/p"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:jee="http://www.springframework.org/schema/jee" xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:task="http://www.springframework.org/schema/task"
    xsi:schemaLocation="http://www.springframework.org/schema/aop
      http://www.springframework.org/schema/aop/spring-aop-3.2.xsd
      http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
      http://www.springframework.org/schema/context
      http://www.springframework.org/schema/context/spring-context-3.2.xsd
      http://www.springframework.org/schema/jee
      http://www.springframework.org/schema/jee/spring-jee-3.2.xsd http://www.springframework.org/schema/tx
      http://www.springframework.org/schema/tx/spring-tx-3.2.xsd http://www.springframework.org/schema/task
      http://www.springframework.org/schema/task/spring-task-3.2.xsd">

    <context:component-scan base-package="com.sda.spring.example" />

    <bean id="myBean" class="com.sda.spring.example.beans.MyBean">
        <property name="name" value="SDA" />
    </bean>
</beans>
```

# Configuration - Java

## Explicit

```
@Configuration
public class AppConfig {

    @Bean
    public MyBean myBean(){
        MyBean bean = new MyBean();
        bean.setName("SDA");
        return bean;
    }
}
```

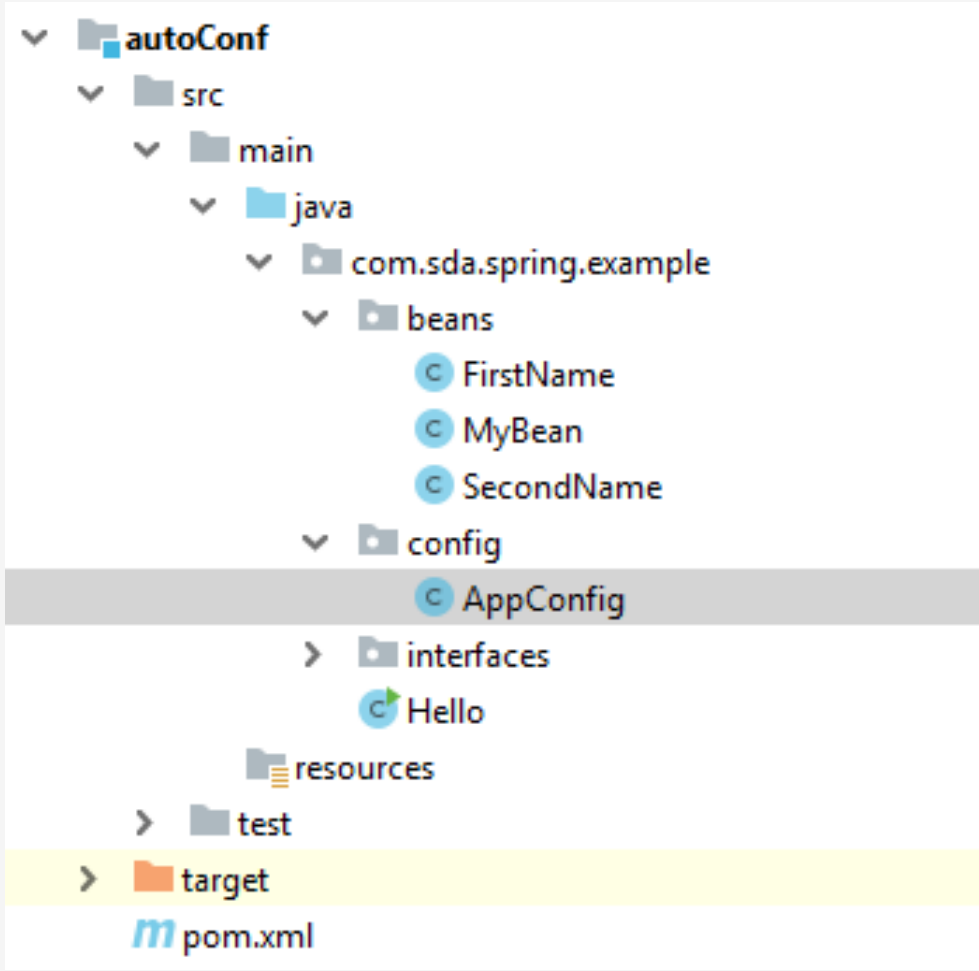
# Configuration - Java

## Automatic

```
@Configuration
@ComponentScan("com.sda.spring.example.beans")
public class AppConfig {
}
```

```
@Component
public class MyBean {
    private BeanName name;
    public MyBean(@Qualifier("firstName") BeanName name){ this.name = name; }
    public void setName(BeanName name){ this.name = name; }
    public String sayHello(){ return "Hello! " + name.getName(); }
}
```

# Spring project structure



# Useful annotations

- @Bean
- @Autowired
- @Inject
- @Configuration
- @Service

# Spring – first application

DEMO

# Exercises

1. Create Spring project
2. Using one of the known methods, configure the beans Book

# Spring Boot

# Spring Boot

- Spring Initializr - Plug & Play kickstart your application
- Add dependencies easily
- Stand-alone Spring Apps
- Embed Tomcat, Jetty etc - no need to deploy WAR files
- No requirements for XML configuration

# Non-web cli application

```
@SpringBootApplication
public class Hello implements CommandLineRunner {

    public static void main(String[] args){
        SpringApplication.run(Hello.class, args);
    }

    public void run(String... args) {
        AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext();
        context.register(AppConfig.class);
        context.refresh();

        MyBean bean = context.getBean(MyBean.class);
        System.out.println(bean.sayHello());

        bean.setName(new SecondName());
        System.out.println(bean.sayHello());
    }
}
```

# Spring Boot non-web application

DEMO

# Spring Data

# Spring Data

- Powerful repository and custom object-mapping abstractions
- Dynamic query derivation from repository method names
- Implementation domain base classes providing basic properties
- Support for transparent auditing (created, last changed)

# Spring Data

- Possibility to integrate custom repository code
- Easy Spring integration via JavaConfig and custom XML namespaces
- Advanced integration with Spring MVC controllers
- Experimental support for cross-store persistence

# Use annotations

- @Entity
- @Id
- @GeneratedValue
- @OneToMany
- @ManyToMany
- @ManyToOne

# REPOSITORIES

- Repository
- CrudRepository
- PagingAndSortingRepository
- How to create Query methods?

# Database queries

```
@Repository
public interface UserRepository extends CrudRepository<User, Long> {

    List<User> findByName(String name);
    List<User> findAll();

}
```

# Spring MVC controllers useful annotations

- @GetMapping
- @PostMapping
- @RequestParam
- @PathParam

# Exercises

1. Create Spring Boot project
2. Using Spring Data, create a repository that supports tables with books

# Spring MVC

# Spring MVC

- Spring Web MVC is web framework based on Servlet API and available in Spring since beginning
- In Spring 5.0 we have Web-Flux which lets us to do reactive programming
- Servlets, DispatcherServlet - algorithms for processing HTTP requests

# Server response codes

| |
|---|
| 2XX SUCCESS<br>200 OK, 201 Created, 202 Accepted |
| 3XX REDIRECT<br>301 Moved Permanently, 302 Found, 303 See Other |
| 4XX CLIENT ERROR<br>401 Bad Request, 401 Unauthorized, 403 Forbidden |
| 5XX SERVER ERROR<br>500 Internal Server Error |

# Context hierarchy

- DispatcherServlet needs WebApplicationContext (which extends ApplicationContext)
- WebApplicationContext has dependency on ServletContext where Servlets reside
- We can define Servlets and Root Application Context - ServletContext contains Controllers and View resolvers RootContext contains Services and Repositories

# "Special Beans"

- HandlerMapping - request mappings
- HandlerExceptionResolver
- ViewResolver
- LocaleResolver
- ThemeResolver

# DISPATCHERSERVLET

- Finding WebApplicationContext
- Calling LocalResolver and ThemeResolver
- Searching for Handler - if found then execute chain (preprocessors, postprocessors, controllers)
- If Model is returned then generate view

# Request mapping

Generic Annotation with many attributes:

- URL
- HTTP method
- Request parameters
- Headers
- MediaTypes

Specialized annotations like:

- @GetMapping
- @PostMapping
- etc.

# Uri patterns

URI can contain wildcards

- ? - any character
- * - zero or more characters in path segment
- ** - zero or more path segment
- { } - used for defining @PathVariable

# What controllers returns?

- @ResponseBody
- HttpEntity, ResponseEntity
- HttpHeaders
- String - name of view for ViewResolver
- View
- @ModelAttribute
- ModelAndView and more

# What should i use for UI?

- JSP, JSTL
- Thymeleaf
- Angular

# Thymeleaf

# Thymeleaf

- Java template engine for XML, XHTML and HTML5 - app view
- Fully integrated with Spring
- Supports internationalization
- Has configurable, well performing cache

# How it looks like?

```
form action="#" th:action="@{/adduser}" th:object="${user}" method="post"
    div class="row"
        div class="form-group col-md-6"
            label for="name" class="col-form-label"       label
            input type="text" th:field="*{name}" class="form-control" id="name" placeholder="Name"
            span th:if="${#fields.hasErrors('name')}" th:errors="*{name}" class="text-danger"></span
         div
        div class="form-group col-md-6"
            label for="email" class="col-form-label"        label
            input type="text" th:field="*{email}" class="form-control" id="email" placeholder="Email"
            span th:if="${#fields.hasErrors('email')}" th:errors="*{email}" class="text-danger"   span
         div
     div
    div class="row"
        div class="col-md-6 mt-5"
            input type="submit" class="btn btn-primary" value="Add User"
         div
     div
 form
```

# Why use Thymeleaf?

- Make the mapped methods in your Spring MVC @Controller objects forward to templates managed by Thymeleaf, exactly like you do with JSPs.
- Use Spring Expression Language (Spring EL) instead of OGNL in your templates.
- Create forms in your templates that are completely integrated with your form-backing beans and result bindings, including the use of property editors, conversion services and validation error handling.
- Display internationalization messages from messages files managed by Spring (through the usual MessageSource objects).

# Thymeleaf MVC

DEMO

# Exercises

1. Create Spring Boot MVC project using Thymeleaf

# Spring Security

# Spring Security

- Lets you configure security with a little code
- Based of filters for HTTP request
- Authorization, Authentication

# Authentication

- Identifying user
- May be password based or token basen

# Authorization

- Happens after authentication
- Checks if user has permissions to perform given action
- In practice - checking roles assigned to user

# How to configure?

- @EnableWebSecurity annotation
- Extend WebSecurityConfigurerAdapter
- Override 'configure' method where you can do specific configuration

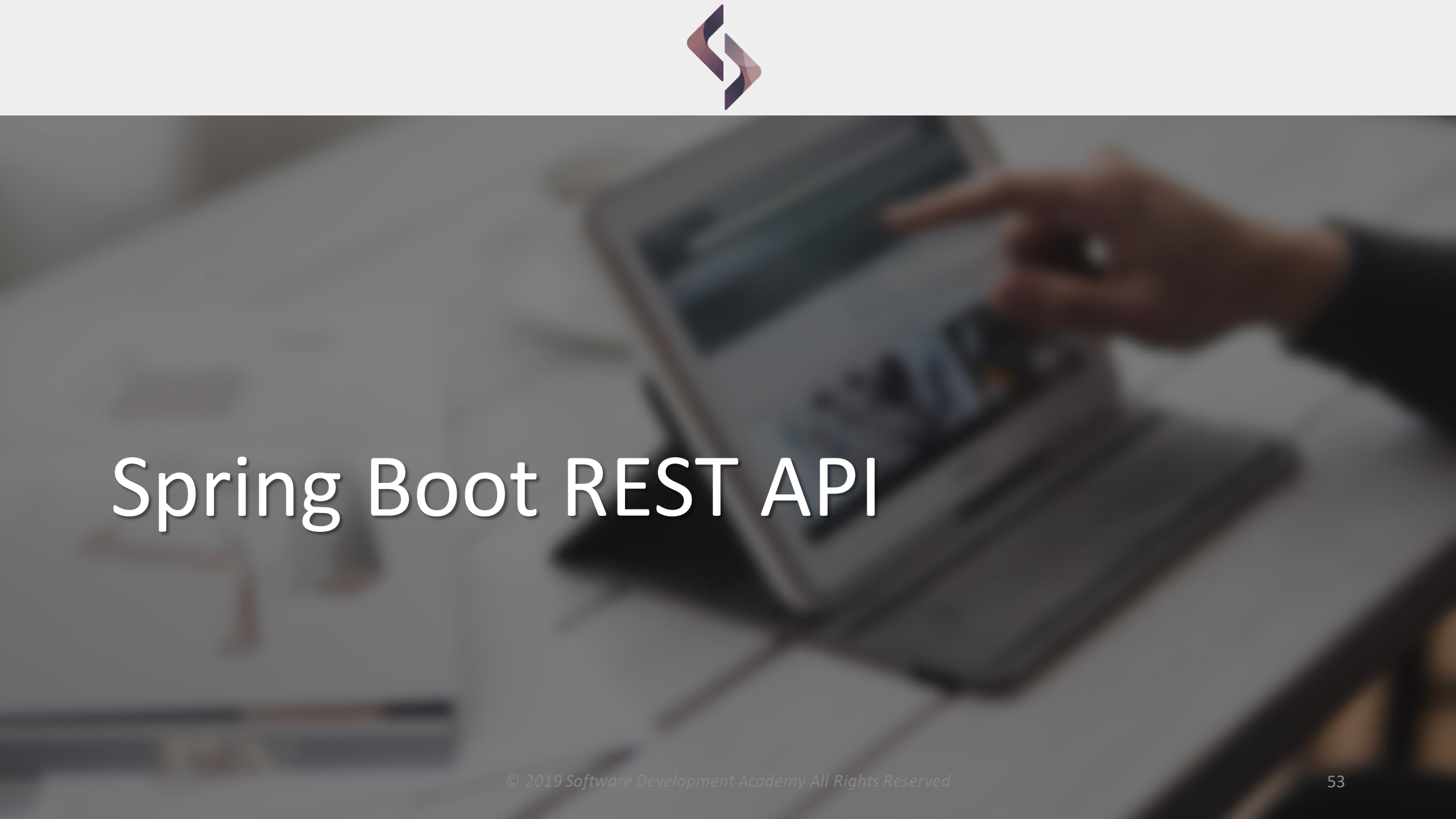
# How to configure?

How to check roles?

- @Secured(nameOfRole) on REST endpoint
- Available only for users with given role

# Login form

DEMO



# Spring Boot REST API

# Spring Boot Rest

- In Spring, a controller class, which is capable of serving REST API requests, is called rest controller. It should be annotated with **@RestController** annotation.
- The resource uris are specified in **@RequestMapping** annotations. It can be applied at class level and method level both. Complete URI for an API is resolved after adding class level path and method level path.
- We should always write **produces** and **consumes** attributes to specify the mediatype attributes for the API. Never reply on assumptions.

# Login form

DEMO

# Exercises

1. Create Spring Boot REST API project
2. Create books repository
3. Use Spring Security to secure your API

# Integration with Angular

# Angular

Angular is a JavaScript framework which makes you able to create reactive Single Page Applications (SPAs). This is a leading front-end development framework which is regularly updated by Angular team of Google. Angular is completely based on components. It consists of several components forming a tree structure with parent and child components.

# Install Angular 7

1. Install Node.js (https://nodejs.org)
2. Install Angular 7

```
npm install -g @angular/cli@7
```

3. Verify Angulars version

```
ng --version
```

4. Create project

```
ng new [project_name]
```

5. Create Service and Components

```
ng g s user
ng g c create-user
ng g c user-list
```

6. Implementation of the Service and Components
7. Run App

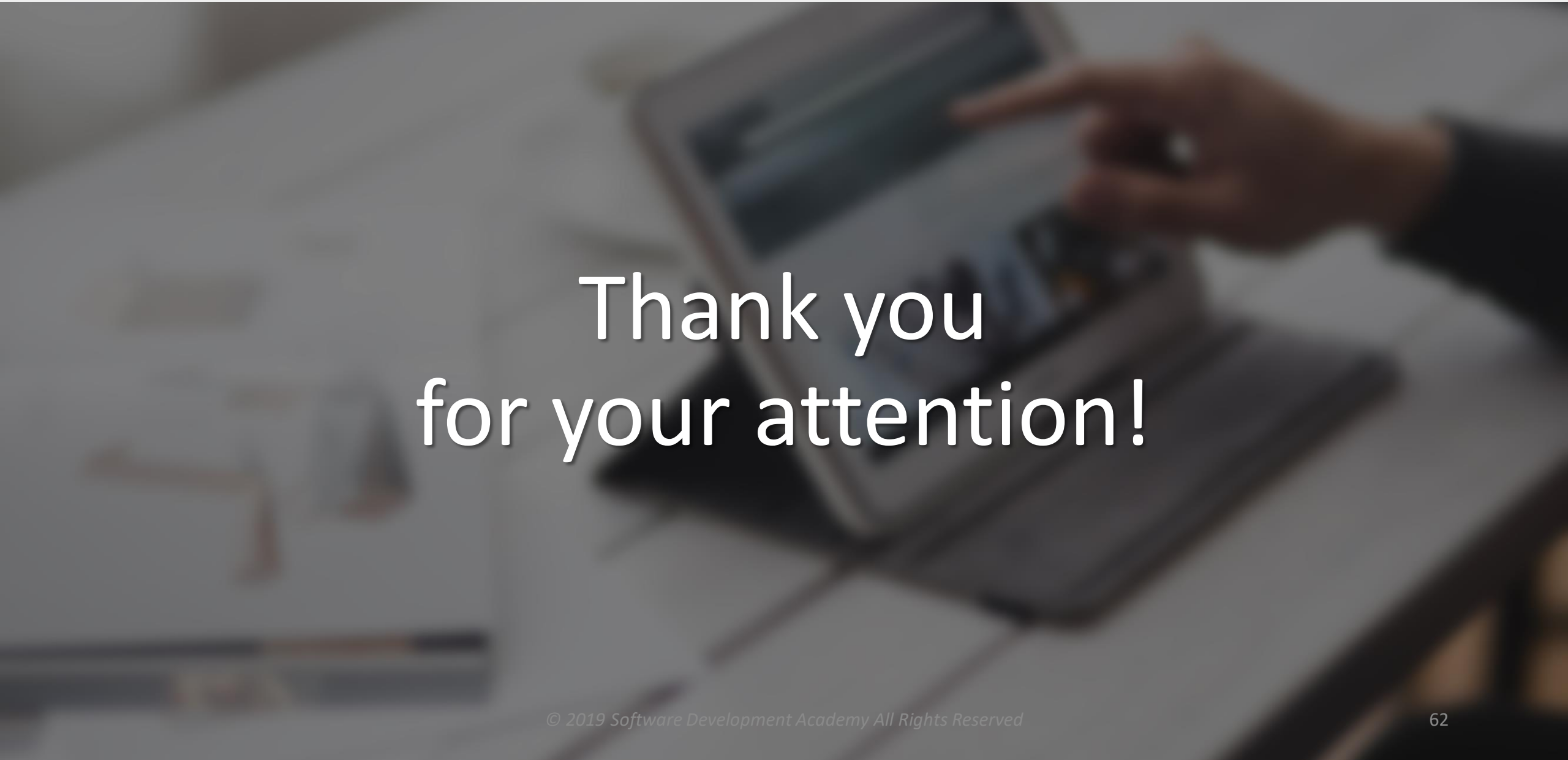
```
npm start
```

# Angular App

DEMO

# Exercises

1. Create an interface in Angular 7 that supports the created REST API
2. Create a login form and book registration



# Thank you for your attention!