Creating RESTful web services with Spring Boot

The Spring framework

- Free and open source
- Inversion of Control Container (IoC)
- Modules
  - DI / AOP
  - Data /Security
  - Web MVC/ REST
  - So much more +++

What is Spring Boot ?

virtual7

- Getting started with Spring Boot is easy
- Convention over configuration
- SPRING CLI
- Entry Point
- Starter POMs
- Production Ready
- Run anywhere!

Starter POMs

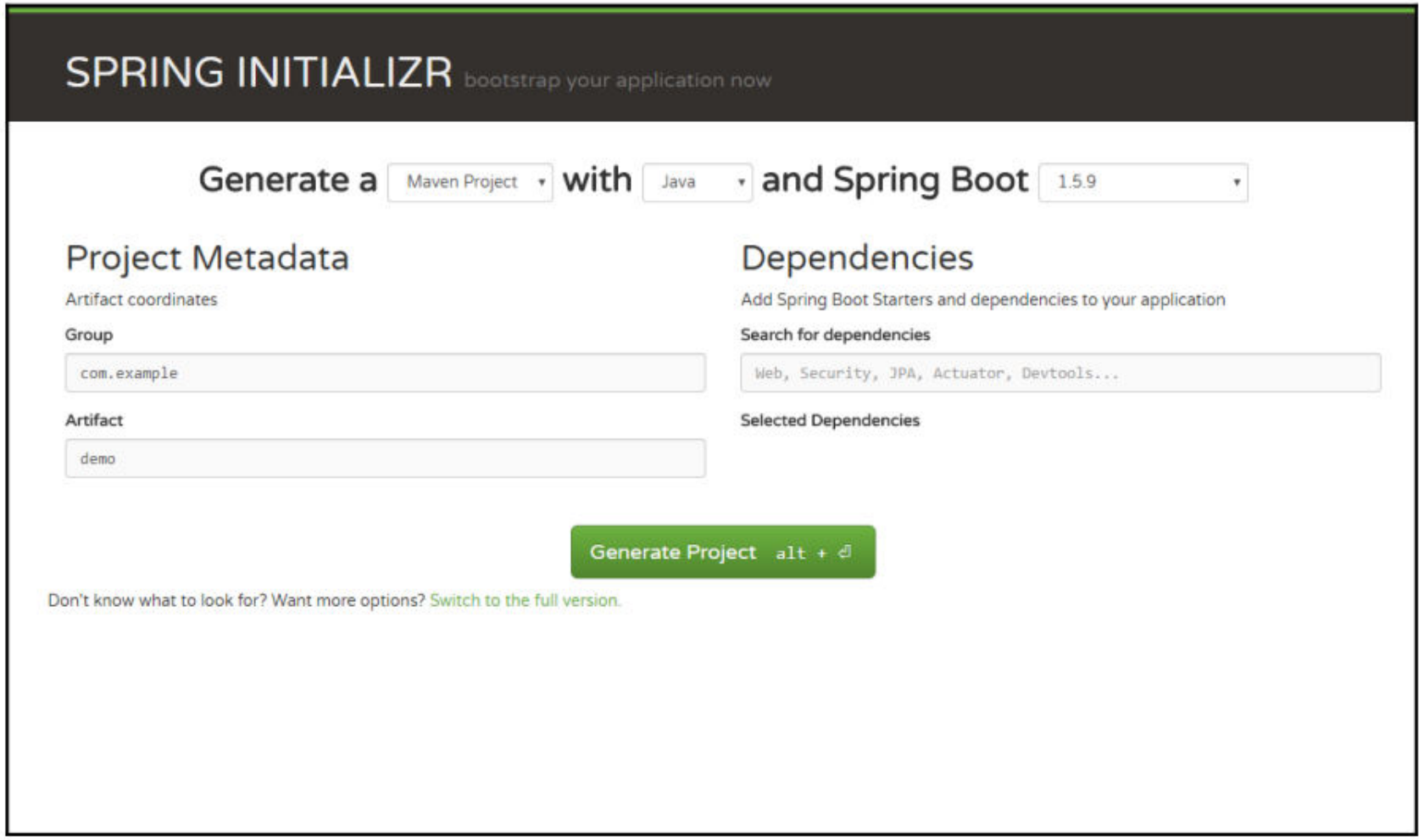
virtual7

- spring-boot-starter-*
- Set of convenient dependency descriptors
- So what starter POMs are available ?
  - docs.spring.io
  - The most common ones :

    - `spring-boot-starter`
    - `spring-boot-starter-actuator`
    - `spring-boot-starter-jdbc`
    - `spring-boot-starter-data-*`
    - `spring-boot-starter-test`
    - `spring-boot-starter-security`
    - `spring-boot-starter-web`

Create a BOOTiful app on the fly

- ... With Spring Initializr Project -> http://start.spring.io



Executable JARs

- Spring Boot includes a Maven plugin that can package the project as an executable jar.
- Add the plugin to your <plugins> section if you want to use it

```
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```

Spring Boot: Auto Configuration

- Spring Boot Auto Configuration attempts to automatically configure your Spring application based on the jar dependencies that you have added

- Autoconfiguration is not invasive

REST

- architectural principle for managing state information

A few RESTful principles

- In REST principles, all resources are identified by the Uniform Resource Identifier (URI).
- HTTP REST resources are represented in some media types, such as XML, JSON, and RDF.
- RESTful resources are self-descriptive
- In REST principles, the server will not keep any state about the client session on the server side; hence, it's stateless
- With RESTful web services, a client can cache any response coming from the server. The server can mention how, and for how long, it can cache the responses.

HTTP Verbs

virtual7

HTTP METHOD	PATH	DESCRIPTION
GET	/users	Gets all users in the repository
GET	/users/{id}	Gets the user with an id of 1
POST	/users	Creates a new user
PUT	/users/1	Creates or updates an existing user
DELETE	/users/1	Deletes user with an id of 1

HTTP Status Codes

Status Code	Description
1xx	Informational
2xx	Success
3xx	Redirection
4xx	Client Error

Spring MVC

virtual7

- **FrontController = DispatcherServlet**
- **@EnableWebMvc** : Spring Boot adds it automatically when it sees Spring MVC on the classpath
- This flags the application as a webapp & activates features like : setting up the DispatcherServlet for us
- Web service components have special annotations like @Service, @RestController – provide hints of the role that the class has

CRUD operations with Spring Boot

virtual7

```java
@RestController
@RequestMapping("/user")
public class UserController {

    @Autowired
    UserService userService;

    @RequestMapping("")
    public List<User> getAllUsers() { return userService.getAllUsers(); }

    @GetMapping("/{id}")
    public User getUser(@PathVariable("id") Integer id) { return userService.getUser(id); }

    @PostMapping(value = "")
    public Map<String, Object> createUser(@RequestParam(value = "userid") Integer userid,
                                          @RequestParam(value = "username") String username) {
        Map<String, Object> map = new LinkedHashMap<>();
        userService.createUser(userid, username);
        map.put("result", "added");
        return map;
    }
```

CRUD operations with Spring Boot

virtual7

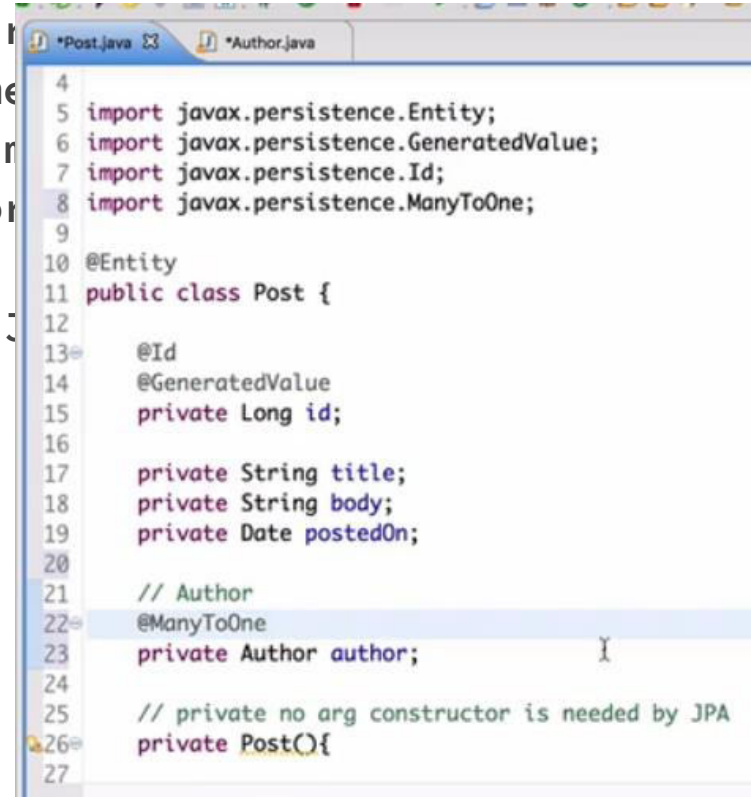
```
@PutMapping(value = "/{userid}")
public Map<String, Object> updateUser(@PathVariable Integer userid,
                                      @RequestParam(value = "username") String username) {
    Map<String, Object> map = new LinkedHashMap<>();
    userService.updateUser(userid, username);
    map.put("result", "updated");
    return map;
}

@DeleteMapping(value = "/{id}")
public Map<String, Object> deleteUser(
        @PathVariable("id") Integer userid) {
    Map<String, Object> map = new LinkedHashMap<>();
    userService.deleteUser(userid);
    map.put("result", "deleted");
    return map;
}
}
```

Data Access with Spring Boot

- Creating Entities & Repositories

  - Spring-boot-starter-jpa
  - we're using JPA , a way allows us to map objects to r
  - it has Hibernate, one of the most popular JPA impleme
  - it gives us spring-data-jpa which makes it easy to ir
  - it gives us spring-orm which gives us core ORM suppor

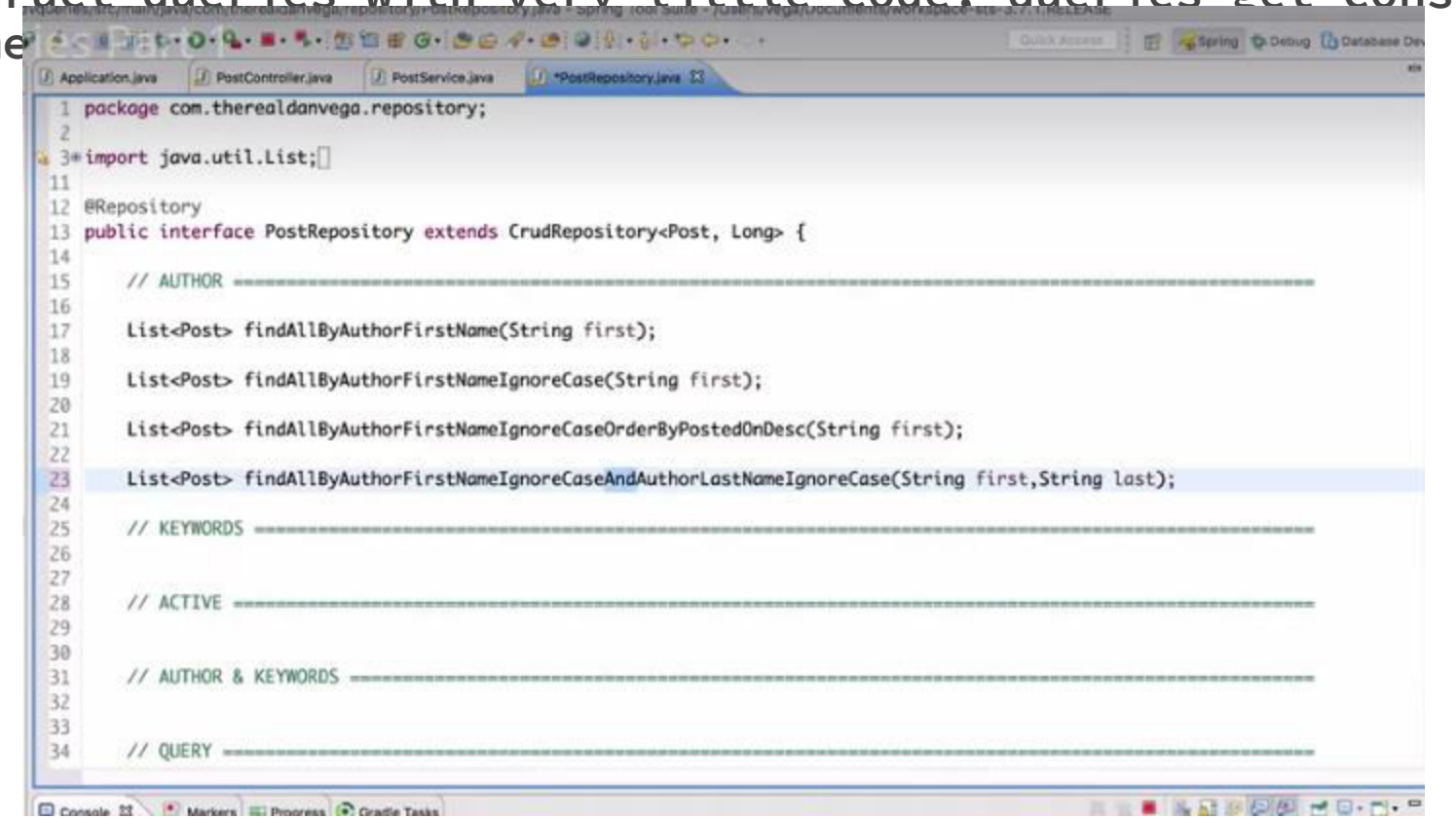
  - flag a POJO with @Entity if you want it to become a J



Data Access with Spring Boot

- We get a lot of methods out of the box from CrudRepository : save(entity), findOne(id), exists(id), delete(id), delete(entity)

- construct queries with very little code, queries get constructed by Spring from the me



Testing RESTful web services

- Junit
- MockMvc
- @MockBean
- JPA components Tests
- WebMVC component Test

Security & JWT

- JWT tokens are URL-safe and web browser-compatible especially for Single Sign-On (SSO) contexts.

- JWT has three parts:
- Header
- Payload/body
- Signature

- OAuth2.0

Building a REST Client

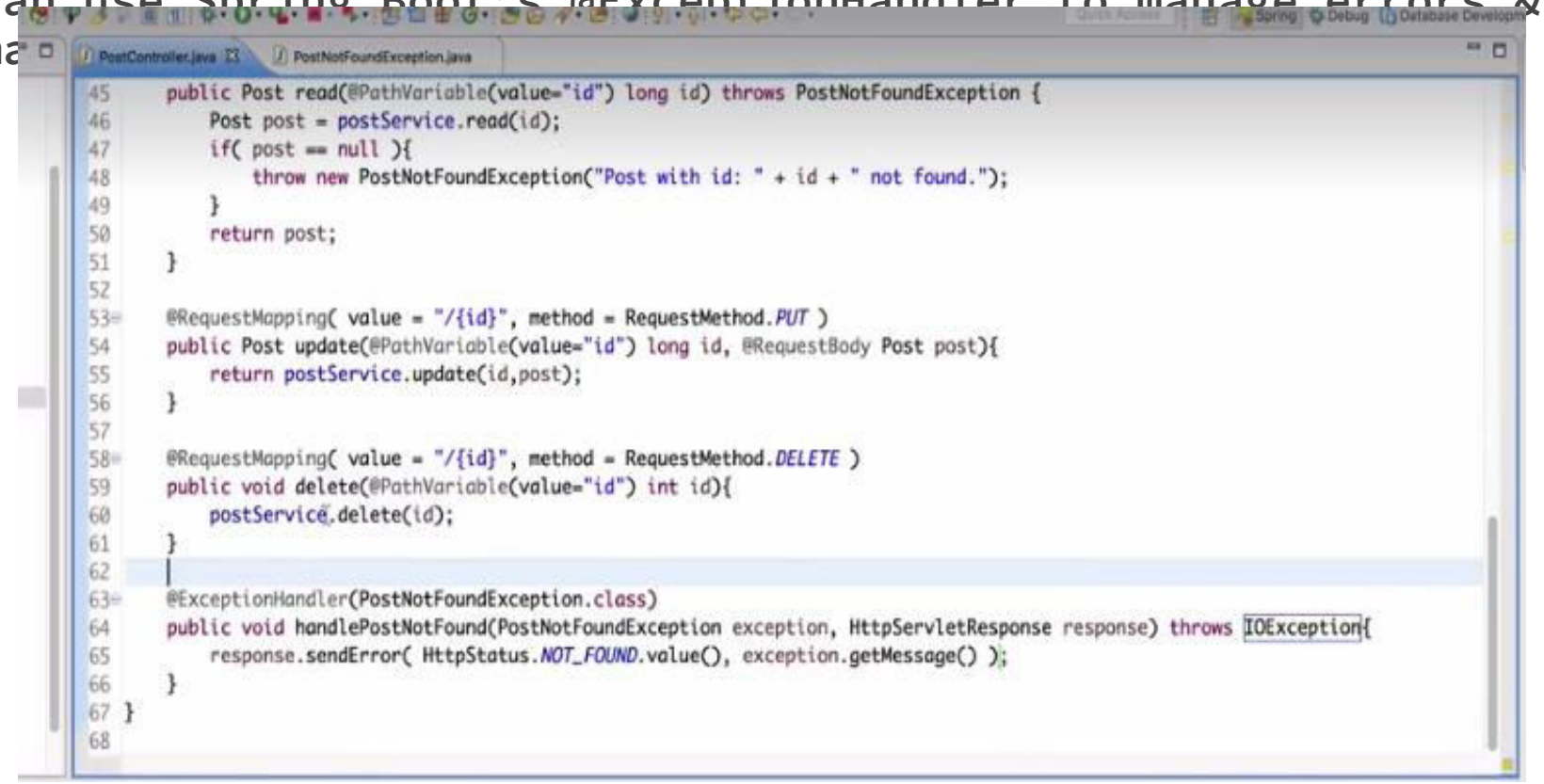
- If we want to build a REST Client to consume another REST API, SpringTemplate is a Spring class used to consume REST API from the client side .

- RestTemplate can be used use to call GET, POST, PUT, DELETE, and other advanced HTTP methods (OPTIONS, HEAD)

```
@RestController
@RequestMapping("/client")
public class ClientController {

	private final Logger _log = LoggerFactory.getLogger(this.getClass());

	@Autowired
	private RestTemplate template;

	@ResponseBody
	@RequestMapping("/test")
	public Map<String, Object> test(){
		Map<String, Object> map = new LinkedHashMap<>();

		String content = template.getForObject("http://localhost:8080/", String.class);
		map.put("result", content);

		return map;
	}
```

Error Handling

- usually when we deal with an unexpected situation in REST APIs, it will automatically throw an HTTP error such as 404
- we might need a JSON format result regardless of whether things go right or wrong
- we can use Spring Boot's @ExceptionHandler to manage errors & display them in JSON forma



```
45    public Post read(@PathVariable(value="id") long id) throws PostNotFoundException {
46        Post post = postService.read(id);
47        if( post == null ){
48            throw new PostNotFoundException("Post with id: " + id + " not found.");
49        }
50        return post;
51    }
52
53    @RequestMapping( value = "/{id}", method = RequestMethod.PUT )
54    public Post update(@PathVariable(value="id") long id, @RequestBody Post post){
55        return postService.update(id,post);
56    }
57
58    @RequestMapping( value = "/{id}", method = RequestMethod.DELETE )
59    public void delete(@PathVariable(value="id") int id){
60        postService.delete(id);
61    }
62
63    @ExceptionHandler(PostNotFoundException.class)
64    public void handlePostNotFound(PostNotFoundException exception, HttpServletResponse response) throws IOException{
65        response.sendError( HttpStatus.NOT_FOUND.value(), exception.getMessage() );
66    }
67 }
68
```

QUESTIONS & ANSWERS