**SPRING REST:**

**Theory:**  
Spring Boot accelerates development by reducing boilerplate, providing starter templates, auto-configuration, and embedded servers.  
A minimal app uses @SpringBootApplication and can be run using SpringApplication.run().

**Code:**

java

**package** com.example.demo;

**import** org.springframework.boot.SpringApplication;

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

@SpringBootApplication

**public** **class** DemoApplication {

**public** **static** **void** main(String[] args) {

SpringApplication.run(DemoApplication.**class**, args);

}

}

**2. Loading Beans from Spring XML Configuration**

**Theory:**  
Although annotation-based config is standard in Spring Boot, you can still use XML config for beans for legacy or specific needs.  
Use @ImportResource("classpath:beans.xml") to import the XML file.

**Classic XML (src/main/resources/beans.xml):**

xml

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id="country" class="com.example.Country">

<property name="code" value="IN"/>

<property name="name" value="India"/>

</bean>

</beans>

**Java Config:**

java

@Configuration

@ImportResource({"classpath:beans.xml"})

**public** **class** XmlConfig { }

**Accessing Bean:**

java

ApplicationContext context = **new** ClassPathXmlApplicationContext("beans.xml");

Country country = (Country) context.getBean("country");

**3. Spring RESTful Web Service – Basic GET Method**

**Theory:**  
Use @RestController and @GetMapping for REST endpoints.  
Spring Boot handles JSON serialization automatically for objects.

**Code:**

java

@RestController

@RequestMapping("/countries")

**public** **class** CountryController {

@GetMapping("/{code}")

**public** Country getCountry(@PathVariable String code) {

*// fetch country from service or list*

**return** **new** Country("IN", "India");

}

}

**4. Spring RESTful Web Service – POST, PUT, DELETE Methods**

**Theory:**

* **@PostMapping**: create resource
* **@PutMapping**: update resource
* **@DeleteMapping**: delete resource

**Code:**

java

@RestController

@RequestMapping("/employees")

**public** **class** EmployeeController {

@PostMapping

**public** Employee create(@RequestBody Employee emp) { */\*...\*/* **return** emp; }

@PutMapping("/{id}")

**public** Employee update(@PathVariable **int** id, @RequestBody Employee emp) { */\*...\*/* **return** emp; }

@DeleteMapping("/{id}")

**public** **void** delete(@PathVariable **int** id) { */\*...\*/* }

}

**5. Handling Not Found and Input Validation in REST**

**Theory:**  
Use @ResponseStatus(HttpStatus.NOT\_FOUND) for not found errors.  
Validation is via annotations like @NotNull, @Size, and @Valid.

**Sample Exception:**

java

@ResponseStatus(value = HttpStatus.NOT\_FOUND, reason = "Country not found")

**public** **class** CountryNotFoundException **extends** RuntimeException { }

**Validation:**

java

**public** **class** Employee {

@NotNull @Size(min=2, max=30)

**private** String name;

}

@PostMapping("/employees")

**public** ResponseEntity<?> add(@Valid @RequestBody Employee emp, BindingResult result) {

**if** (result.hasErrors()) { */\* handle errors \*/* }

*// Save employee*

}

**6. Spring Boot JWT Authentication with Spring Security**

**Theory:**  
JWT (JSON Web Token) authenticates REST APIs in a stateless way.  
Use @EnableWebSecurity, configure filters, and create token utilities.

**Configuration:**

java

@Configuration

@EnableWebSecurity

**public** **class** SecurityConfig **extends** WebSecurityConfigurerAdapter {

@Override

**protected** **void** configure(HttpSecurity http) **throws** Exception {

http.csrf().disable()

.authorizeRequests().antMatchers("/api/auth/\*\*").permitAll()

.anyRequest().authenticated()

.and()

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);

*// add JWT filter*

}

}

**JWT Token Generation (with jjwt):**

java

String jwt = Jwts.builder()

.setSubject(username)

.setIssuedAt(**new** Date())

.setExpiration(**new** Date(System.currentTimeMillis() + 86400000))

.signWith(SignatureAlgorithm.HS512, secretKey)

.compact();

**7. Sample Employee List in XML and Loading in Spring**

**Theory:**  
Static data can be loaded from XML using <list> in bean definition.

**employee.xml example:**

xml

<beans ...>

<bean id="employeeList" class="java.util.ArrayList">

<constructor-arg>

<list>

<bean class="com.example.Employee">

<property name="id" value="1"/>

<property name="name" value="Jack"/>

</bean>

*<!-- More employees... -->*

</list>

</constructor-arg>

</bean>

</beans>

**Usage:**

java

@Autowired

List<Employee> employeeList;

**8. Spring Bean Scopes: Singleton vs Prototype**

**Theory:**

* **Singleton** scope: one bean instance per Spring IoC container (default).
* **Prototype** scope: new instance for each injection.

**Definition Example:**

xml

<bean id="country" class="com.example.Country" scope="prototype"/>

**Access in Java:**

java

Country c1 = context.getBean("country", Country.**class**);

Country c2 = context.getBean("country", Country.**class**);

*// c1 and c2 are different instances if 'prototype'*

**9. Sample REST URL Naming Table**

| **HTTP Method** | **Sample URL** | **Description** | **Annotation** |
| --- | --- | --- | --- |
| GET | /countries | Get all countries | @GetMapping |
| GET | /countries/{code} | Get country by code | @GetMapping("/{id}") |
| POST | /countries | Create country | @PostMapping |
| PUT | /countries | Update country | @PutMapping |
| DELETE | /countries/{code} | Delete country by code | @DeleteMapping("/{id}") |

**10. What Happens When context.getBean() Is Invoked?**

**Theory:**

* Spring checks the ApplicationContext for the bean definition.
* If scope is singleton, returns the same instance.
* If scope is prototype, creates a new instance.
* Handles autowiring, dependency injection, and returns the bean.

These code snippets and explanations should help you implement the key features from setting up a Spring Boot app to securing it with JWT, along with related RESTful and configuration concepts.