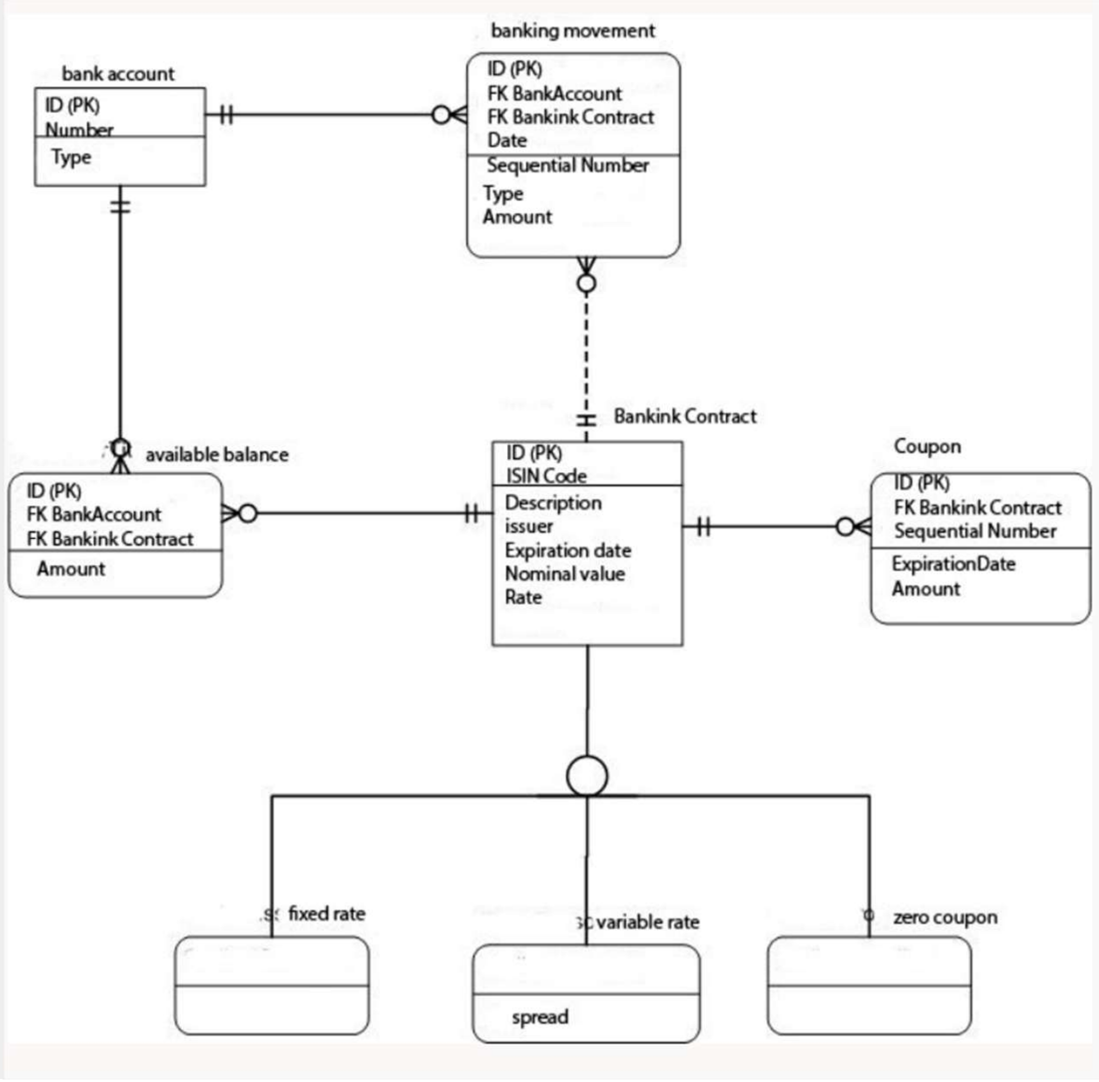
UNIVERSITÀ DELLA CALABRIA DIPARTIMENTO DI **MATEMATICA E INFORMATICA**

AGILE SOFTWARE DEVELOPMENT

Dr. Francesco Scalzo - francesco.scalzo@unical.it
Enterprise Solution Engineer at **Revelis s.r.l.**

Office hours & info: send a mail

Exercise



banking movement
ID (PK)
FK BankAccount
FK Bankink Contract
Date
Sequential Number
Type
Amount
bank account
ID (PK)
Number
Type
available balance
ID (PK)
FK BankAccount
FK Bankink Contract
Amount
Bankink Contract
ID (PK)
ISIN Code
Description
issuer
Expiration date
Nominal value
Rate
Coupon
ID (PK)
FK Bankink Contract
Sequential Number
ExpirationDate
Amount
fixed rate
variable rate
spread
zero coupon

Spring
Project

8 → 11 Java

spring boot

PRESENTATION	• Angular
REST	• Spring - @Controller
BUSINESS	• Spring - @Service
DATA ACCESS	• Spring – JPA, @Repository
DATA	• DB – MariaDB / HSQLDB

IJ

- https://www.jetbrains.com/idea/download/
- https://start.spring.io/

Spring

Controller → ResrController

- REST has quickly become the de-facto standard for building web services on the web because they’re easy to build and easy to consume.

- What benefits? The web and its core protocol, HTTP, provide a stack of features:
  - Suitable actions (GET, POST, PUT, DELETE, …)
  - Caching
  - Redirection and forwarding
  - Security (encryption and authentication)

- REST is not a standard but an approach, a style, a set of constraints on your architecture that can help you build web-scale systems.

Spring

RestController

To wrap your repository with a web layer, you must turn to Spring MVC

- **@RestController** indicates that the data returned by each method will be written straight into the response body instead of rendering a template.
- The **@RequestBody** annotation is used to define the request body content type.
- The **@PathVariable** annotation is used to define the custom or dynamic request URI. The Path variable in request URI is defined as curly braces {}
- The **@RequestParam** annotation is used to read the request parameters from the Request URL. By default, it is a required parameter. We can also set default value for request parameters
- We have routes for each operations (**@GetMapping**, **@PostMapping**, **@PutMapping** and **@DeleteMapping**, corresponding to HTTP GET, POST, PUT, and DELETE calls).
- *ResponseEntity represents the whole HTTP response: status code, headers, and body. As a result, we can use it to fully configure the HTTP response.*

Spring

RestController

- The default **HTTP** request method is **GET**. This method does not require any Request Body. You can send request parameters and path variables to define the custom or dynamic URL.

- The **HTTP POST** request is used to create a resource. This method contains the Request Body. We can send request parameters and path variables to define the custom or dynamic URL.

- The **HTTP PUT** request is used to update the existing resource. This method contains a Request Body. We can send request parameters and path variables to define the custom or dynamic URL.

- The **HTTP Delete** request is used to delete the existing resource. This method does not contain any Request Body. We can send request parameters and path variables to define the custom or dynamic URL.

Spring

Naming Convention for REST API Resources/Endpoints

1. Use nouns to represent resources and not verbs (/folders - ~~/createFolder~~)
2. Use plural resources (/users/21 - ~~/user/21~~)
3. Use lower-case

Resources	GET read	POST create	PUT update	DELETE
/books	Return a book list	Create a new book	Update all books	Delete all book
/books/145	Return a single book	not allowed (405)	Update only a book	Delete only a book

4. Use hyphens (spinal case) to improve readability of URIs. (/users/noam/reset-password - ~~/users/noam/resetPassword)~~
5. uses the PUT, POST and DELETE methods to alter the state of a resource. Do not use the GET method for state changes
6. Use the sub-resources to describe the relationships
   (GET /books/411/authors/1 → Returns the author #1 of the book411)

Spring

Naming Convention for REST API Resources/Endpoints

7. Implements filtering, sorting, selecting specific fields and paging for collections:

   GET /books?author=Franz+Kafka → Returns The list of books written by Kafka
   GET /books?pages<=200 Return a list of books that have a maximum of 200 pages

8. Allow sorting based on one or more fields:

   GET /books?sort=-pages,+author

9. Selection of fields:

   GET /books?fields=title,author,id

10. Handle errors using HTTP status codes:

200 - OK - All right
201 - OK - A new resource has been created
204 - OK - The resource was successfully deleted
304 - Not modified - The data has not changed. The customer can use the cached data
400 - Bad Request - Invalid request. The exact error should be explained in the error payload (which we will discuss shortly). For example. "The JSON is invalid"
401 - Unauthorized - The request requires user authentication
403 - Forbidden - The server has understood the request, but according to the rights of the applicant, access is not allowed.
404 - Not Found - There is no resource behind the requested URI.
422 - Unprocessable Entity - must be used if the server cannot process the entity, for example if an image cannot be formatted or required fields are missing in the payload.
500 - Internal Server Error - API developers should avoid this error. If a global application error occurs, the stacktrace must be logged and not sent in the response to the user.

Spring

ControllerAdvice

- @ResponseBody signals that this advice is rendered straight into the response body.
- @ExceptionHandler configures the advice to only respond if an EmployeeNotFoundException is thrown.
- @ResponseStatus says to issue an HttpStatus.NOT_FOUND, i.e. an HTTP 404.
- The body of the advice generates the content. In this case, it gives the message of the exception.

Spring

Utility - Spring Boot Actuator

Spring Boot includes a number of additional features to help you monitor and manage your application when you push it to production. You can choose to manage and monitor your application by using HTTP endpoints or with JMX. Auditing, health, and metrics gathering can also be automatically applied to your application.

1. Add dependency

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

2. Go to: http://localhost:8080/actuator and choose the enabled endpoint
3. Since Endpoints may contain sensitive information, careful consideration should be given about when to expose them. The table in the follow slide shows the default exposure for the built-in endpoints
4. To change which endpoints are exposed, use the following technology-specific include and exclude properties:

Property	Default
management.endpoints.jmx.exposure.exclude	
management.endpoints.jmx.exposure.include	*
management.endpoints.web.exposure.exclude	
management.endpoints.web.exposure.include	info, health

https://docs.spring.io/spring-boot/docs/current/reference/html/production-ready-features.html

Spring

Utility - Spring Boot Actuator

ID	Description	JMX	Web
auditevents	Exposes audit events information for the current application. Requires an AuditEventRepository bean.	Yes	No
beans	Displays a complete list of all the Spring beans in your application.	Yes	No
caches	Exposes available caches.	Yes	No
conditions	Exposes available caches.	Yes	No
configprops	Displays a collated list of all @ConfigurationProperties.	Yes	No
env	Exposes properties from Spring’s ConfigurableEnvironment.	Yes	No
flyway	Shows any Flyway database migrations that have been applied. Requires one or more Flyway beans.	Yes	No
health	Shows application health information.	Yes	Yes
heapdump *	Returns an hprof heap dump file	N/A	No
httptrace	Displays HTTP trace information (by default, the last 100 HTTP request-response exchanges). Requires an HttpTraceRepository bean.	Yes	No
info	Displays arbitrary application info.	Yes	Yes
integrationgraph	Shows the Spring Integration graph. Requires a dependency on spring-integration-core.	Yes	No
Jolokia *	Exposes JMX beans over HTTP (when Jolokia is on the classpath, not available for WebFlux). Requires a dependency on jolokia-core.	N/A	No
Logfile *	Returns the contents of the logfile (if logging.file.name or logging.file.path properties have been set). Supports the use of the HTTP Range header to retrieve part of the log file’s content.	N/A	No
loggers	Shows and modifies the configuration of loggers in the application.	Yes	No
liquibase	Shows any Liquibase database migrations that have been applied. Requires one or more Liquibase beans.	Yes	No
metrics	Shows ‘metrics’ information for the current application.	Yes	No
mappings	Displays a collated list of all @RequestMapping paths.	Yes	No
Prometheus *	Exposes metrics in a format that can be scraped by a Prometheus server. Requires a dependency on micrometer-registry-prometheus.	N/A	No
scheduledtasks	Displays a collated list of all @RequestMapping paths.	Yes	No
sessions	Allows retrieval and deletion of user sessions from a Spring Session-backed session store. Requires a Servlet-based web application using Spring Session.	Yes	No
shutdown	Lets the application be gracefully shutdown. Disabled by default.	Yes	No
startup	Shows the startup steps data collected by the ApplicationStartup. Requires the SpringApplication to be configured with a BufferingApplicationStartup.	Yes	No
threaddump	Performs a thread dump.	Yes	No

* For only web application (Spring MVC, Spring WebFlux, or Jersey)

Spring

Utility - OpenAPI 3.0 (Swagger)

- Documentation is an essential part of building REST APIs.
- SpringDoc is a tool that simplifies the generation and maintenance of API docs, based on the OpenAPI 3 specification
- To have springdoc-openapi automatically generate the OpenAPI 3 specification docs for our API, we simply add the springdoc-openapi-ui dependency to our pom.xml

```xml
<dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-ui</artifactId>
    <version>1.5.0</version>
</dependency>
```

- To view the documentation go to http://localhost:8080/swagger-ui.html

Spring

DTO & ModelMapper

- DTO, which stands for Data Transfer Object, is a design pattern conceived to reduce the number of calls when working with remote interfaces.

- For example, lets say that we were communicating with a RESTful API that exposes our banking account data. In this situation, instead of issuing multiple requests to check the current status and latest transactions of our account, the bank could expose an endpoint that returned a DTO summarizing everything.

- Another advantage of using DTOs on RESTful APIs written in Java (and on Spring Boot), is that they can help hiding implementation details of domain objects (aka. entities). Exposing entities through endpoints can become **a security issue** if we do not carefully handle what properties can be changed through what operations. With DTOs, we can only expose what is needed.

- To avoid having to write cumbersome/boilerplate code to map DTOs into entities and vice-versa, we are going to use a library called ModelMapper. The goal of ModelMapper is to make object mapping easy by automatically determining how one object model maps to another.