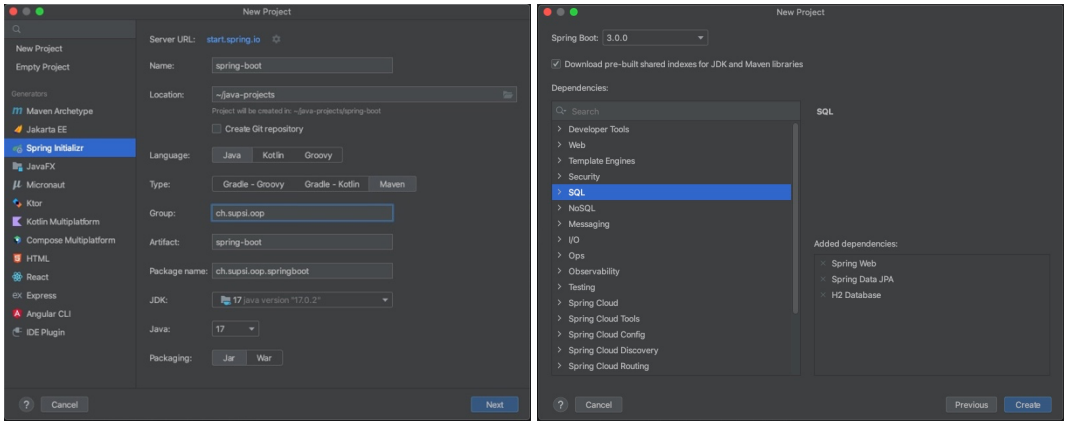
The goal of this assignment is to create a SpringBoot application that exposes REST endpoints and uses hibernate to persist data.

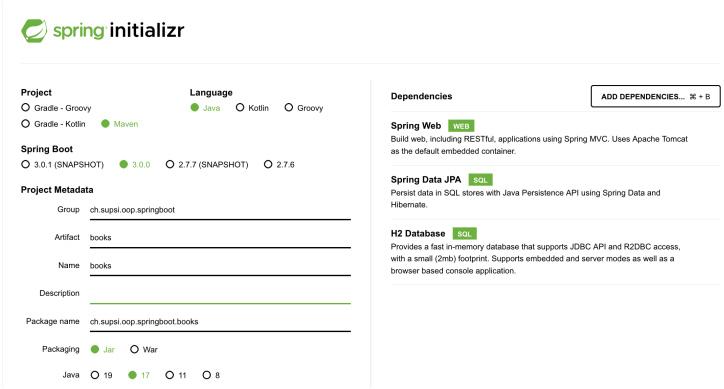
Exercise 1 – Books

1. SpringBoot project creation

Create a new SpringBoot project with *Spring Web, Data JPA and H2 Database* dependencies, using IntelliJ's wizard or the online SpringBoot initializer service available at: https://start.spring.io/.



IntelliJ new project Wizard



Spring initializr service

2. DB Configuration part

Open the *src/main/resources/application.properties* file and add the following lines required to configure the database connection (modify, if desired, the highlighted code to change from in-memory DB to file-based DB):

```
#jdbc:h2:file:[path/database-name] for disk-based database
spring.datasource.url=jdbc:h2:mem:books-db
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=

# dump the queries to standard out, enable this if needed
# spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.H2Dialect

# For production, this property should be 'validate'
spring.jpa.hibernate.ddl-auto=update
#start H2 administration tool accessible at: http://localhost:8080/h2-console
spring.h2.console.enabled=true
```

3. Model entities

Create a new package called "model" in your project and add the provided *Author.java* and *Book.java* classes. Add the minimal required annotation to the *Author* and *Book* classes:

1. Add @Entity annotation on each class.
2. Add @Id and @GeneratedValue(generator = "increment") to the id field.

4. CRUD Repository

Create a new package called "repository" in your project and add two new interfaces called *AuthorRepository* and *BookRepository*. Let each interface extend the *CrudRepository* interface and provide the corresponding types to correctly parametrize the generic type *CrudRepository*: the first parameter corresponds to the managed entity type while the second one is the type of the id.

5. REST Controller

Create a new package called "controller" in your project and implement the two controller classes to expose the REST API endpoints. Follow the next steps for each controller (in the instructions, Book will be used as an example):

1. Create a new Java class (e. g. BookController) and add the *@RestController* annotation to the class.
2. Add an instance field of the corresponding Repository class (e. g. BookRepository) and annotate the field using *@Autowired*.
3. Implement a ***get*** method (e. g. getAllBooks) that returns all instances of the repository.
    a. Annotate the method with *@GetRequest*, specifying its value attribute to the desired endpoint (e. g. /books)
    b. Collect all entities by calling the *repository.findAll()* method and store them in a new List that will be returned.
    c. Return a *ResponseEntity* using the *ResponseEntity.ok(… list …)* method.
4. Implement a ***getById*** method (e. g. getBookById) that returns the entity associated with the id.
    a. Annotate the method with *@GetRequest*, specifying its value to the desired endpoint **containing a variable name in curly brackets** (e. g. /books/**{id}**)
    b. Annotate the method's *id* parameter with *@PathVariable.*
    NOTE: Make sure that the variable name matches the name used in the endpoint.
    c. Let the method get the entity by calling the *repository.findById(id)* method and store the Optional result in a local variable.
    d. Return a *ResponseEntity* using *ResponseEntity.of* method passing the optional result instance from the repository.
5. Implement an ***add*** method (e. g. addBook) that adds a new entity (passed as a parameter) to the database.
    a. Annotate the method with *@PostRequest*, specifying its value attribute to the desired endpoint (e. g. /books).
    b. Annotate the method's parameter (e. g. Book book) with *@RequestBody*
    c. Add the parameter's variable to the repository by calling the *repository.save(param)* method and storing the returned entity to a local variable (e. g. storedInstance).
    d. Let the method return a *ResponseEntity* using the *ResponseEntity.ok(storedInstance)* method, where *storedInstance* is the return value of the save method call.

6. Execute SpringBook application

Test your SpringBoot application by starting it directly from your IDE or by generating the jar file using maven and then executing it using the following terminal commands from the project's root folder:

```
mvn package

java –jar target/ARTIFACTNAME–0.0.1–SNAPSHOT.jar
```

7. Test endpoints

Use the following curl commands to test your endpoints by populating and fetching data using the REST endpoints.

Insert authors

```
curl –v –X POST –H "Content–Type: application/json" –d '{"firstName":"Stephen", "lastName": "King",
   "birthYear":1974}' http://localhost:8080/authors
curl –v –X POST –H "Content–Type: application/json" –d '{"firstName":"Herman", "lastName":
   "Melville", "birthYear":1819}' http://localhost:8080/authors
curl –v –X POST –H "Content–Type: application/json" –d '{"firstName":"Herman", "lastName": "Hesse",
   "birthYear":1877}' http://localhost:8080/authors
```

Insert books

```
curl –v –X POST –H "Content–Type: application/json" –d '{"title":"Carrie", "publishingYear":1974,
   "publisher": "Doubleday"}' http://localhost:8080/books
curl –v –X POST –H "Content–Type: application/json" –d '{"title":"Shining", "publishingYear":1977,
   "publisher": "Doubleday"}' http://localhost:8080/books
curl –v –X POST –H "Content–Type: application/json" –d '{"title":"IT", "publishingYear":1986,
   "publisher": "Viking"}' http://localhost:8080/books
curl –v –X POST –H "Content–Type: application/json" –d '{"title":"Moby dick", "publishingYear":
   1851, "publisher": "Richard Bentley"}' http://localhost:8080/books
curl –v –X POST –H "Content–Type: application/json" –d '{"title":"Siddhartha", "publishingYear":
   1922, "publisher": "Suhrkamp Verlag"}' http://localhost:8080/books
```

Get authors / books with id 1

```
curl –v http://localhost:8080/authors/
curl –v http://localhost:8080/authors/1

curl –v http://localhost:8080/books/
curl –v http://localhost:8080/books/1
```

Exercise 2 – Address book

Implement a SpringBoot application to manage an address book. For each contact the firstname, lastname, phone number, mobile phone number and email address should be stored to a H2 database using Spring Data JPA. Add the following REST API endpoints:

- **`GET /contacts/`** returns all contacts
- **`GET /contacts/{id}`** returns the contact associated with the specified id
- **`POST /contacts/`** receives a json representing a contact and adds it to the address book
- **`GET /contacts/findAllByFirstname/{name}`** returns a json representing all contacts that match the given firstname
- **`DELETE /contacts/{id}`** deletes the contact with id from the address book.

HINTS:

To implement the findAllByFirstname functionality, add a query method to the CrudRepository and call it from the Controller or Service.

Exercise 3 – Books relation (Optional)

Extend the Exercise 1 by modelling the Author – Book relation.

Add references to the model entities:

1. Add a list of books (e. g. List<Book> books) to the *Author* class and implement the getter and setter methods.
2. Add a reference to the author in the *Book* class (e. g. Author author) and implement the getter and setter methods.

Model the *Author.books* and *Book.author* relation:

1. Add the *@OneToMany* annotation to the *Author.books* field, by specifying the mappedBy property of the annotation to reference the "author" field
2. Add the *@ManyToOne* and *@JoinColumn* annotations to the Book.author field and specify the name of the join column to "author_id"

Prevent stack-overflow while rendering response:

1. Add *@JsonManagedReference* annotation to the *Author's* books list field
2. Add *@JsonBackReference* annotation to the *Book's* author reference

Introduce BookService:

1. Create a new package called "service" and add a new Java class called *BookService*.
2. Annotate the *BookService* class with @Service.
3. Add both *BookRepository* and *AuthorRepository* fields and annotate them with *@Autowired*.
4. Implement a method (e. g. assignBookAuthor) to assign the author reference of a book, starting from the bookId and authorId:
    a. Requested both entities using the related *findById* method of the repository.
    b. Assign the author to the book using the setter method.
    c. Store the updated book instance, calling the *save* method of the repository.
    d. Return the returned instance from the save method.
5. Annotate the method with the *@Transactional*, specifying the isolation level as read committed and the propagation as required.

Introduce a new endpoint to assign an author to a book (e. g.: POST /books/{bookID}/author/{authorID}).

1. Add a reference to the *BookService* in the *BookController* class and annotate it with *@Autowired*.
2. Add a new method with two id parameters: one for the book and one for the author.
3. Annotate the method with *@PostRequest*, specifying its value attribute to the desired endpoint (e. g. /books/{bookID}/author/{authorID}).
4. Annotate the method's parameters with *@PathVariable*
5. Call the service's method to assign the book's author (e. g. assignBookAuthor).
6. Return a *ResponseEntity* with the updated entity.

Test your implementation by populating the DB and finally assigning the authors to the books using the new endpoint.

E. g: `curl –v –X POST http://localhost:8080/books/3/author/1`