JAVA ACADEMY - XIDERAL MONTERREY, N.L

WEEK 4
SPRING DATA JPA

Made by:
Luis Miguel Sánchez Flores

INTRODUCTION

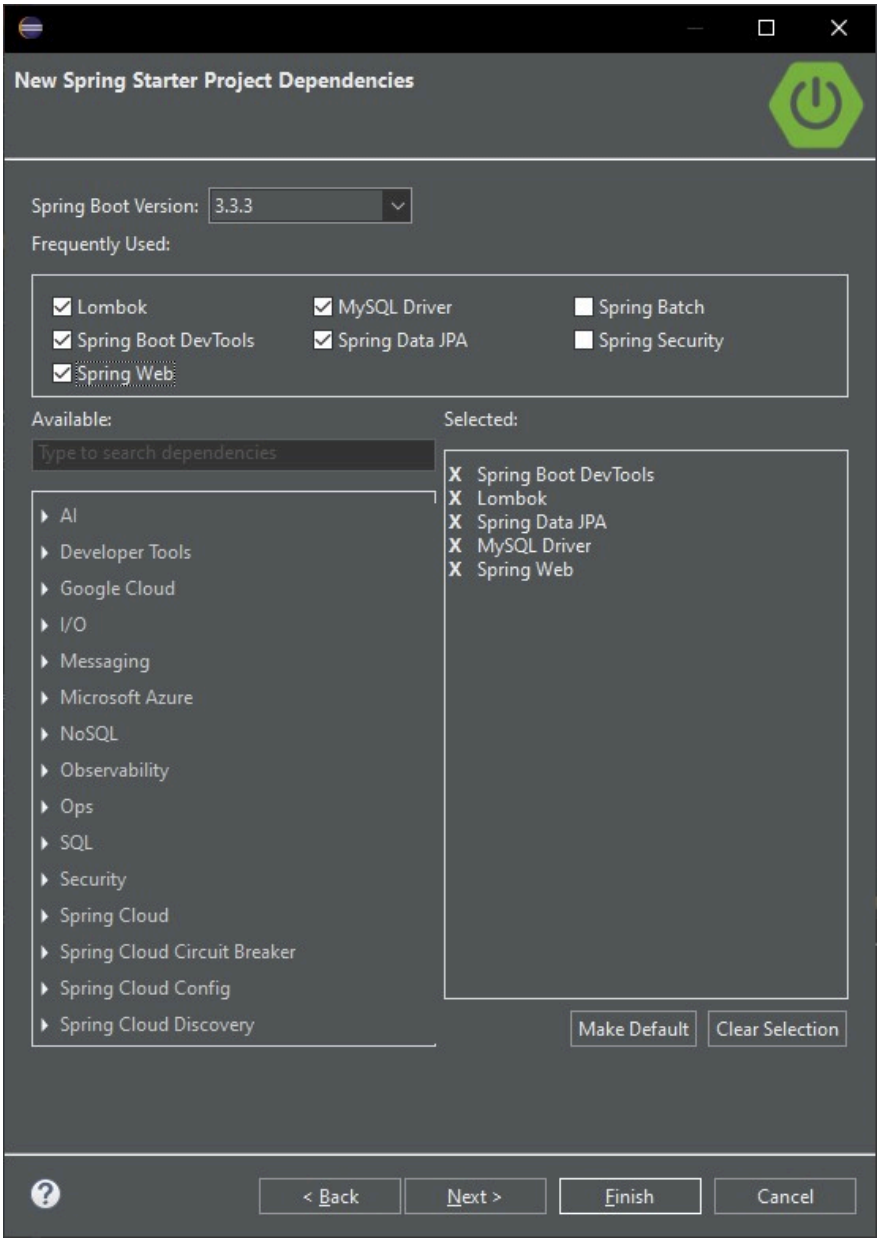
The Spring Data JPA framework stands out as a powerful solution for managing data persistence in Java REST applications, as it leverages the capabilities of the JPA (Jakarta Persistence API) while significantly simplifying the data access layer by reducing boilerplate code and automatically implement basic CRUD operations, as well as powerful query derivation mechanisms.
Spring Data JPA makes it possible to remove the DAO implementations entirely, with the DAO interface being the only artifact required by Spring to automatically implement the basic operations, promoting a clean and maintainable code.

When Spring Data JPA creates a new Repository implementation, it analyzes all the methods defined by the interfaces and tries to automatically generate queries from the method names. While this has some limitations, it's a powerful method of defining new custom access methods with less effort.

Spring Data JPA has become an invaluable tool for developers when building REST applications thanks to its streamlining of the data access process, allowing developers to focus on crafting well-defined API endpoints and handling the business logic.
The following example will showcase how Spring Data JPA differentiates from the use of Spring and JPA separately in code, and how it allows the creation of more robust, efficient and scalable applications.

Setting up the project

As always, the Spring project is generated through the use of the **Spring Initializr** web tool or the **Spring Tool Suite plugin** included in popular IDEs, setting up the appropriate metadata and including the necessary dependencies for the project, such as Spring Web for the REST capabilities and Spring Data JPA for implementing a data access layer towards a relational database such as MySQL:



As with the Spring + JPA example project, the example of the school database with student information will be used, which will be subjected to CRUD operations. However, the table creation will be delegated to Spring with the `spring.jpa.generate-ddl` and `spring.jpa.hibernate.ddl-auto` properties, that will tell Spring whether it should take care of the initialization, and how it should be done, respectively. In this case, the `create` value of the property lets Spring know that it should generate the schema every time the application is executed, but avoid deleting it once the application is finished.

It is here where Hibernate (JPA) makes use of the mapping information in the created entities to generate the DDL statements needed to initialize the table:

```
# SPRING DATA JPA project

spring.application.name=Spring_Data_JPA

# MySQL Datasource
spring.datasource.url=jdbc:mysql://localhost:3306/school_example
spring.datasource.username=springstudent
spring.datasource.password=springstudent
server.port=8090

# JPA properties
spring.jpa.show-sql=true
spring.jpa.generate-ddl=true
spring.jpa.hibernate.ddl-auto=create
spring.sql.init.data-locations=classpath:data.sql
spring.jpa.defer-datasource-initialization=true
spring.sql.init.mode=always
```

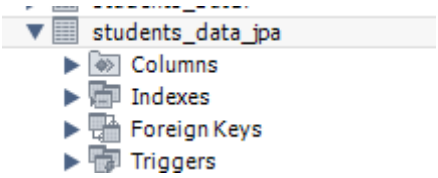
In the same way as in the Spring REST + JPA project, initial data is entered to the newly created table via the `data.sql` file entered in the resource folder, adding the following rows:

```sql
insert into students_data_jpa (id, first_name, last_name, email, gender, date_of_birth, phone_number) values
                    (1, 'Chad', 'Barfitt', 'cbarfitt0@simplemachines.org', 'Other', '27-06-2002', '1171887923'),
                    (2, 'Millard', 'Simonich', 'msimonich1@google.ca', 'Male', '29-08-2004', '4977160910'),
                    (3, 'Mick', 'MacMeanma', 'mmacmeanma2@blinklist.com', 'Male', '25-05-2001', '3294515237'),
                    (4, 'Etti', 'Hendrikse', 'ehendrikse3@jigsy.com', 'Female', '10-02-2004', null),
                    (5, 'Dunn', 'Meneur', 'dmeneur4@photobucket.com', 'Male', '08-07-2003', null),
                    (6, 'Selia', 'Furneaux', 'sfurneaux5@i2i.jp', 'Female', '16-11-2002', '8747941454'),
                    (7, 'Nigel', 'Humfrey', 'nhumfrey6@over-blog.com', 'Male', '04-07-2005', '6722010693'),
                    (8, 'Alikee', 'Hugonnet', 'ahugonnet7@marriott.com', 'Female', '13-06-2004', '8615206281'),
                    (9, 'Willey', 'Struys', 'wstruys8@mail.ru', 'Male', '12-04-2004', '6144725276'),
                    (10, 'Farrand', 'Kolinsky', 'fkolinsky9@blogtalkradio.com', 'Female', '02-12-2003', '6638043043');
```

For this case, Hibernate will create the table based on the `Student` entity shown below, where it utilizes the JPA annotations such as `@Table`, `@Id` and `@Column` to map the Java class and its attributes with the MySQL database table and columns to be generated by Spring. The class also uses Lombok annotations to set up the constructors, getters and setters of the class automatically:

```java
@NoArgsConstructor
@AllArgsConstructor
@Data
@Entity
@ToString
@Table(name = "students_data_jpa")
public class Student {

    public enum Gender {
        Male,
        Female,
        Other
    }

    // Attributes
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    @Column(name="id")
    private int id;

    @Column(name="first_name", nullable=false)
    private String firstName;

    @Column(name="last_name", nullable =false)
    private String lastName;

    @Column(name="date_of_birth", nullable = false)
    private String dateOfBirth;

    @Column(name="email", unique =true)
    private String email;
```

CREATED TABLE WITH INITIAL DATA:



id	date_of_birth	email	first_name	last_name	phone_number	gender
1	27-06-2002	cbarfitt0@simplemachines.org	Chad	Barfitt	1171887923	Other
2	29-08-2004	msimonich1@google.ca	Millard	Simonich	4977160910	Male
3	25-05-2001	mmacmeanma2@blinklist.com	Mick	MacMeanma	3294515237	Male
4	10-02-2004	ehendrikse3@jigsy.com	Etti	Hendrikse	NULL	Female
5	08-07-2003	dmeneur4@photobucket.com	Dunn	Meneur	NULL	Male
6	16-11-2002	sfurneaux5@i2i.jp	Selia	Furneaux	8747941454	Female
7	04-07-2005	nhumfrey6@over-blog.com	Nigel	Humfrey	6722010693	Male
8	13-06-2004	ahugonnet7@marriott.com	Alikee	Hugonnet	8615206281	Female
9	12-04-2004	wstruys8@mail.ru	Willey	Struys	6144725276	Male
10	02-12-2003	fkolinsky9@blogtalkradio.com	Farrand	Kolinsky	6638043043	Female
NULL	NULL	NULL	NULL	NULL	NULL	NULL

As with the Spring + JPA project, **a Service layer and a Controller layer** were developed, taking care of the business logic when interacting with the database and handling the HTTP requests that users send to then provide them with the appropriate response, respectively:

```java
@Service
public class StudentServiceImpl implements StudentService{

    @Autowired
    private StudentRepository studentDAO;

    @Override
    @Transactional
    public Student saveStudent(Student newStudent) {

        Student addedStudent = studentDAO.save(newStudent);

        return addedStudent;

    }

    @Override
    public Optional<Student> findByID(int id) {

        Optional<Student> retrievedStudent = studentDAO.findById(id);

        if(retrievedStudent.isEmpty())
            throw new RuntimeException("No student with ID #"+id+" was found at the database...");

        return retrievedStudent;
    }
```

```java
// REST CONTROLLER
@RestController
public class StudentController {

    @Autowired
    private StudentService studentService;

    @GetMapping("/students")
    public List<Student> findAllStudents() {
        return studentService.findAll();
    }

    @GetMapping("/students/size")
    public String countStudents() {
        return "Number of students in database : " + studentService.countNumberOfStudents();
    }

    @GetMapping("/students/gender/{gender}")
    public List<Student> findStudentsByGender(@PathVariable Gender gender) {
        return studentService.findStudentsByGender(gender);
    }

    @GetMapping("/students/no_phone_number")
    public List<Student> findStudentsWithoutNumber() {
        return studentService.findStudentsWithoutPhoneNum();
    }

    @GetMapping("/student/id/{id}")
    public Student findStudentById(@PathVariable int id) {
        Optional<Student> retrievedStudent =  studentService.findByID(id);
```

Easy Repository Setup

What differentiates Spring Data JPA from using Spring and JPA separately is the former's addition of an layer of abstraction that provides a more convenient and consistent data access to the database, in which a simple extension to the `JPARepository` interface **will prompt Spring to automatically generate a repository implementation at runtime**, meaning that **no code is required to implement basic CRUD operations**, eliminating the need to define an implementation DAO class as with the Spring REST + JPA project:

```
// SPRING DATA JPA REPOSITORY
@Repository                                  // Automatically generates a repository implementation!
public interface StudentRepository extends JpaRepository<Student, Integer> {

```

With Spring Data JPA, methods such as `save()`, `findById()`, `deleteById()` and `count()` are ready to be used by the service layer to perform basic CRUD operations in a simple yet effective implementation:

```
@Override
@Transactional
public Student saveStudent(Student newStudent) {

    Student addedStudent = studentDAO.save(newStudent); // <- Method provided by Spring Data JPA

    return addedStudent;

}
```

Spring also allows additional CRUD methods through a convention-based approach, which infers the underlying query based on the method's name. For example, the `findByFirstName()` and `findByEmail()`functions will be assigned with a query that retrieves the list of students that matches the

provided `firstName` and the `email` argument, respectively, without the need to write out the SQL or JPQL queries manually:

```
// Query method based on the naming convention (Retrieve students based on the firstName attribute)
List<Student> findByFirstName(String firstName);

List<Student> findByEmail(String email);
```

For more complex / sophisticated search operation, Spring Data JPA offers the `@Query` annotation to specify the exact SQL or JPQL query to run onto the database when the method is called. In this case, the `findStudentsWithoutPhoneNum()` has a custom query that retrieves the list of students who have their phone number set as a null value, without the need of implementing the code onto the method:

```
// Custom JPQL Query without the need of an concrete method:
@Query("SELECT s FROM Student s where s.phoneNumber IS NULL")
List<Student> findStudentsWithoutPhoneNum();
```

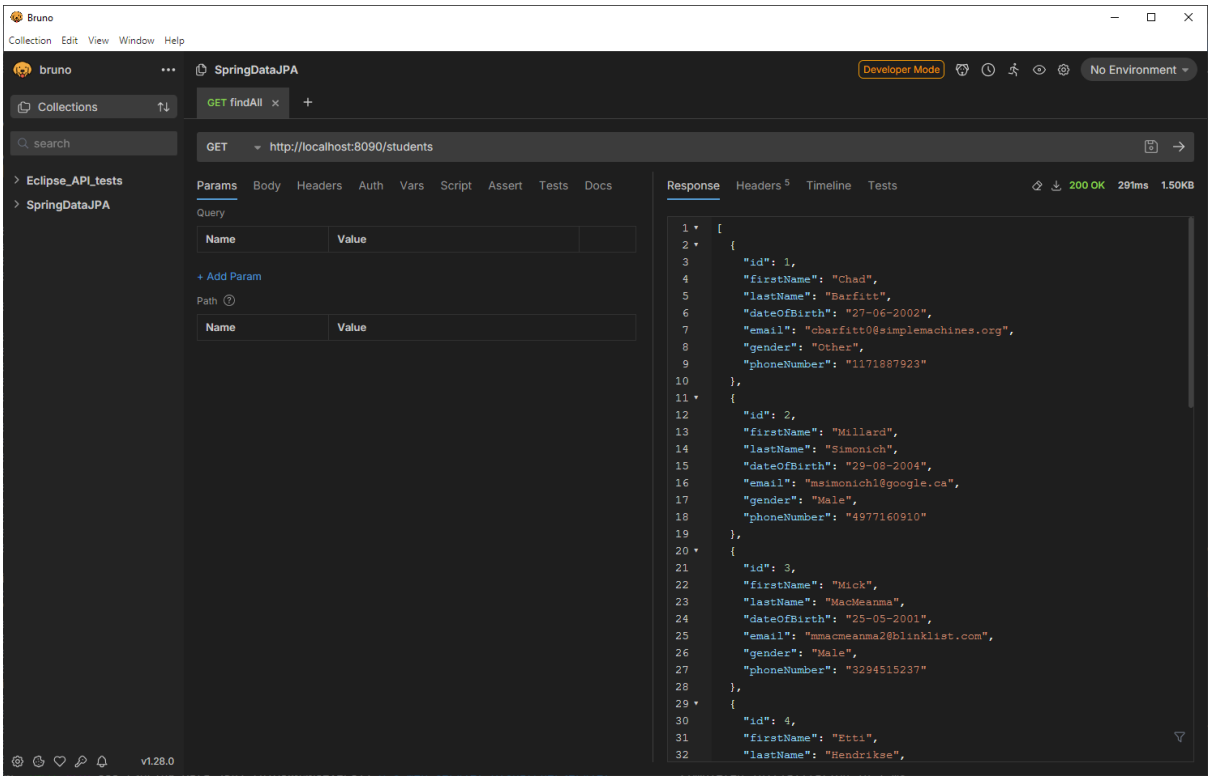
Testing out Spring Data JPA

With all the necessary components completed, we can run the application and deploy the embedded server to be able to make requests to the REST application, testing if the JPA repository generated by Spring Data JPA is able to complete the requests successfully:

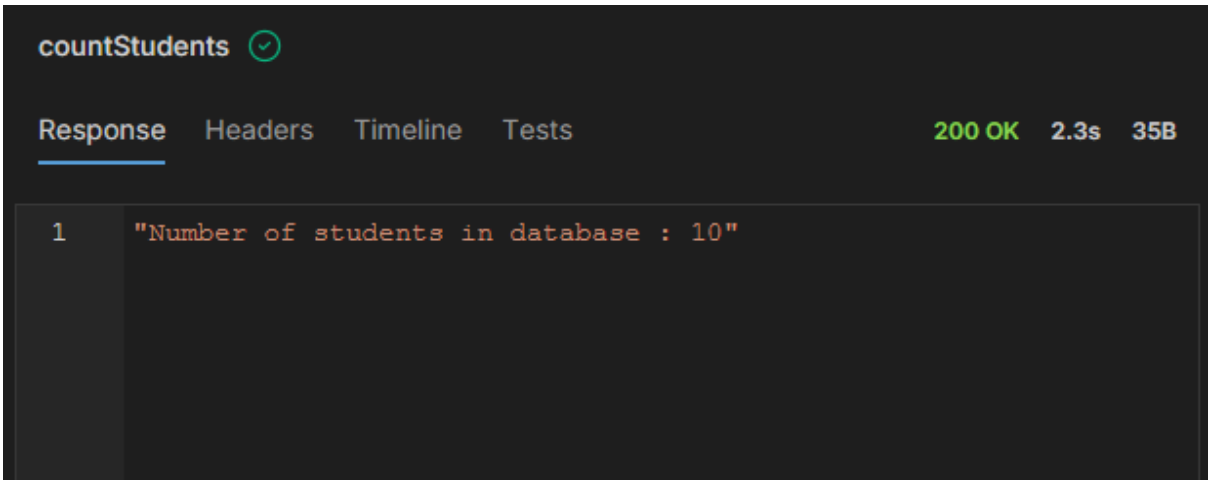
```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/

 :: Spring Boot ::                (v3.3.3)

2024-09-07T18:24:10.386-06:00  INFO 5476 --- [Spring_Data_JPA] [  restartedMain] c.l.s.SpringDataJpaApplication           : Starting SpringDataJpaApplication using Java 21.0.4 with PID 5476 (C:\Users\HP\Documents
\Academia_Java\eclipse\ProyectosAcademiaJava\semana_4\Spring_Data_JPA\target\classes started by HP in C:\Users\HP\Documents\Academia_Java\eclipse\ProyectosAcademiaJava\semana_4\Spring_Data_JPA)
2024-09-07T18:24:10.388-06:00  INFO 5476 --- [Spring_Data_JPA] [  restartedMain] c.l.s.SpringDataJpaApplication           : No active profile set, falling back to 1 default profile: "default"
2024-09-07T18:24:10.425-06:00  INFO 5476 --- [Spring_Data_JPA] [  restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults active! Set 'spring.devtools.add-properties' to 'false' to
disable
2024-09-07T18:24:10.425-06:00  INFO 5476 --- [Spring_Data_JPA] [  restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider setting the 'logging.level.web' property to
'DEBUG'
2024-09-07T18:24:10.855-06:00  INFO 5476 --- [Spring_Data_JPA] [  restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2024-09-07T18:24:10.894-06:00  INFO 5476 --- [Spring_Data_JPA] [  restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 34 ms. Found 1 JPA repository interface.
2024-09-07T18:24:11.262-06:00  INFO 5476 --- [Spring_Data_JPA] [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port 8090 (http)
2024-09-07T18:24:11.274-06:00  INFO 5476 --- [Spring_Data_JPA] [  restartedMain] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2024-09-07T18:24:11.274-06:00  INFO 5476 --- [Spring_Data_JPA] [  restartedMain] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.28]
2024-09-07T18:24:11.308-06:00  INFO 5476 --- [Spring_Data_JPA] [  restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2024-09-07T18:24:11.308-06:00  INFO 5476 --- [Spring_Data_JPA] [  restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 882 ms
2024-09-07T18:24:11.419-06:00  INFO 5476 --- [Spring_Data_JPA] [  restartedMain] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing PersistenceUnitInfo [name: default]
2024-09-07T18:24:11.456-06:00  INFO 5476 --- [Spring_Data_JPA] [  restartedMain] org.hibernate.Version                    : HHH000412: Hibernate ORM core version 6.5.2.Final
2024-09-07T18:24:11.478-06:00  INFO 5476 --- [Spring_Data_JPA] [  restartedMain] o.h.c.internal.RegionFactoryInitiator    : HHH000026: Second-level cache disabled
2024-09-07T18:24:11.669-06:00  INFO 5476 --- [Spring_Data_JPA] [  restartedMain] o.s.o.j.p.SpringPersistenceUnitInfo      : No LoadTimeWeaver setup: ignoring JPA class transformer
2024-09-07T18:24:11.686-06:00  INFO 5476 --- [Spring_Data_JPA] [  restartedMain] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
2024-09-07T18:24:11.925-06:00  INFO 5476 --- [Spring_Data_JPA] [  restartedMain] com.zaxxer.hikari.pool.HikariPool        : HikariPool-1 - Added connection com.mysql.cj.jdbc.ConnectionImpl@1f0a5eb
2024-09-07T18:24:11.927-06:00  INFO 5476 --- [Spring_Data_JPA] [  restartedMain] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
2024-09-07T18:24:12.553-06:00  INFO 5476 --- [Spring_Data_JPA] [  restartedMain] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000489: No JTA platform available (set 'hibernate.transaction.jta.platform' to enable
JTA platform integration)
Hibernate: drop table if exists students_data_jpa
Hibernate: create table students_data_jpa (id integer not null auto_increment, date_of_birth varchar(255) not null, email varchar(255), first_name varchar(255) not null, last_name varchar(255) not null, phone_number
varchar(255), gender enum ('Female','Male','Other'), primary key (id)) engine=InnoDB
Hibernate: alter table students_data_jpa add constraint UKmr5j07khutlw739ykjy806fwl unique (email)
2024-09-07T18:24:12.640-06:00  INFO 5476 --- [Spring_Data_JPA] [  restartedMain] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2024-09-07T18:24:12.805-06:00  INFO 5476 --- [Spring_Data_JPA] [  restartedMain] o.s.d.j.r.query.QueryEnhancerFactory     : Hibernate is in classpath; If applicable, HQL parser will be used.
2024-09-07T18:24:13.151-06:00  WARN 5476 --- [Spring_Data_JPA] [  restartedMain] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed
during view rendering. Explicitly configure spring.jpa.open-in-view to disable this warning
2024-09-07T18:24:13.411-06:00  INFO 5476 --- [Spring_Data_JPA] [  restartedMain] o.s.b.d.a.OptionalLiveReloadServer       : LiveReload server is running on port 35729
2024-09-07T18:24:13.440-06:00  INFO 5476 --- [Spring_Data_JPA] [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8090 (http) with context path '/'
2024-09-07T18:24:13.446-06:00  INFO 5476 --- [Spring_Data_JPA] [  restartedMain] c.l.s.SpringDataJpaApplication           : Started SpringDataJpaApplication in 3.309 seconds (process running for 3.874)
```

The Bruno client API is used to check the requests to the application, executing the following requests:

findAll - Retrieve all the students from the database:



countStudents - get the number of students included in the database:



getStudentByID - Retrieve the student's data by their ID:

getStudentByID

Response Headers Timeline Tests 200 OK 2.4s 150B


```
{
  "id": 6,
  "firstName": "Selia",
  "lastName": "Furneaux",
  "dateOfBirth": "16-11-2002",
  "email": "sfurneaux5@i2i.jp",
  "gender": "Female",
  "phoneNumber": "8747941454"
}
```


getStudentsByGender - Retrieve the list of students of a specific gender (Female in this case):

getStudentsByGender

Response Headers Timeline Tests 200 OK 2.6s 622B


```
12      "id": 6,
13      "firstName": "Selia",
14      "lastName": "Furneaux",
15      "dateOfBirth": "16-11-2002",
16      "email": "sfurneaux5@i2i.jp",
17      "gender": "Female",
18      "phoneNumber": "8747941454"
19    },
20    {
21      "id": 8,
22      "firstName": "Alikee",
23      "lastName": "Hugonnet",
24      "dateOfBirth": "13-06-2004",
25      "email": "ahugonnet7@marriott.com",
26      "gender": "Female",
27      "phoneNumber": "8615206281"
28    },
29    {
30      "id": 10,
31      "firstName": "Farrand",
32      "lastName": "Kolinsky",
33      "dateOfBirth": "02-12-2003",
34      "email": "fkolinsky9@blogtalkradio.com",
35      "gender": "Female",
36      "phoneNumber": "6638043043"
37    }
38  ]
```


getStudentsWithNoPhone - Retrieve the list of students without a phone number:

getStudentsWithNoPhone

Response Headers Timeline Tests 200 OK 2.3s 293B


```
[
  {
    "id": 4,
    "firstName": "Etti",
    "lastName": "Hendrikse",
    "dateOfBirth": "10-02-2004",
    "email": "ehendrikse3@jigsy.com",
    "gender": "Female",
    "phoneNumber": null
  },
  {
    "id": 5,
    "firstName": "Dunn",
    "lastName": "Meneur",
    "dateOfBirth": "08-07-2003",
    "email": "dmeneur4@photobucket.com",
    "gender": "Male",
    "phoneNumber": null
  }
]
```


findStudentsByFirstName - Retrieve a list of students match by a first_name argument:

findStudentsByFirstName

Response Headers Timeline Tests 200 OK 2.3s 148B


```
[
  {
    "id": 9,
    "firstName": "Willey",
    "lastName": "Struys",
    "dateOfBirth": "12-04-2004",
    "email": "wstruys8@mail.ru",
    "gender": "Male",
    "phoneNumber": "6144725276"
  }
]
```


addStudent - Add a new student to the database:

addStudent

Response Headers Timeline Tests 200 OK 2.10s 149B


```
{
  "id": 11,
  "firstName": "Willey",
  "lastName": "Man",
  "dateOfBirth": "18-02-2002",
  "email": "willeyclone@gmail.com",
  "gender": "Male",
  "phoneNumber": "9999999999"
}
```


BEFORE:

8	13-06-2004	ahugonnet7@marriott.com	Alikee	Hugonnet	8615206281	Female
9	12-04-2004	wstruys8@mail.ru	Willey	Struys	6144725276	Male
10	02-12-2003	fkolinsky9@blogtalkradio.com	Farrand	Kolinsky	6638043043	Female
NULL	NULL	NULL	NULL	NULL	NULL	NULL

AFTER:

8	13-06-2004	ahugonnet7@marriott.com	Alikee	Hugonnet	8615206281	Female
9	12-04-2004	wstruys8@mail.ru	Willey	Struys	6144725276	Male
10	02-12-2003	fkolinsky9@blogtalkradio.com	Farrand	Kolinsky	6638043043	Female
11	18-02-2002	willeyclone@gmail.com	Willey	Man	9999999999	Male
NULL	NULL	NULL	NULL	NULL	NULL	NULL

updateStudent - Update an existing student in the database:

updateStudent

Response Headers Timeline Tests 200 OK 2.4s 143B


```
{
  "id": 6,
  "firstName": "Selia",
  "lastName": "Cruz",
  "dateOfBirth": "16-11-2002",
  "email": "newmail@i2i.jp",
  "gender": "Female",
  "phoneNumber": "8888888888"
}
```

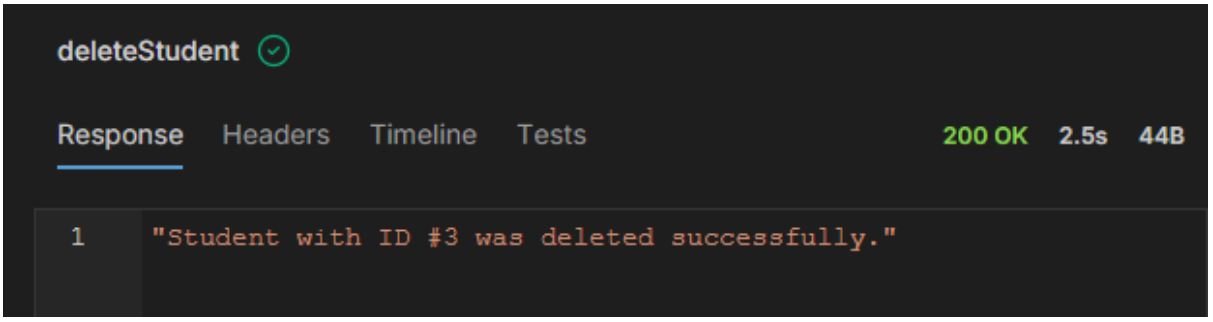

BEFORE:

6	16-11-2002	sfurneaux5@i2i.jp	Selia	Furneaux	8747941454	Female
7	04-07-2005	phumfrey6@over-blog.com	Nigel	Humfrey	6722010693	Male

AFTER:

6	16-11-2002	newmail@i2i.jp	Selia	Cruz	8888888888	Female

deleteStudent - Delete an existing student by their ID:



BEFORE:

3	25-05-2001	mmacmeanma2@blinklist.com	Mick	MacMeanma	3294515237	Male

AFTER:

2	29-08-2004	msimonich1@google.ca	Millard	Simonich	4977160910	Male
4	10-02-2004	ehendrikse3@jigsy.com	Etti	Hendrikse	NULL	Female
5	08-07-2003	dmeneur4@photobucket.com	Dunn	Meneur	NULL	Male