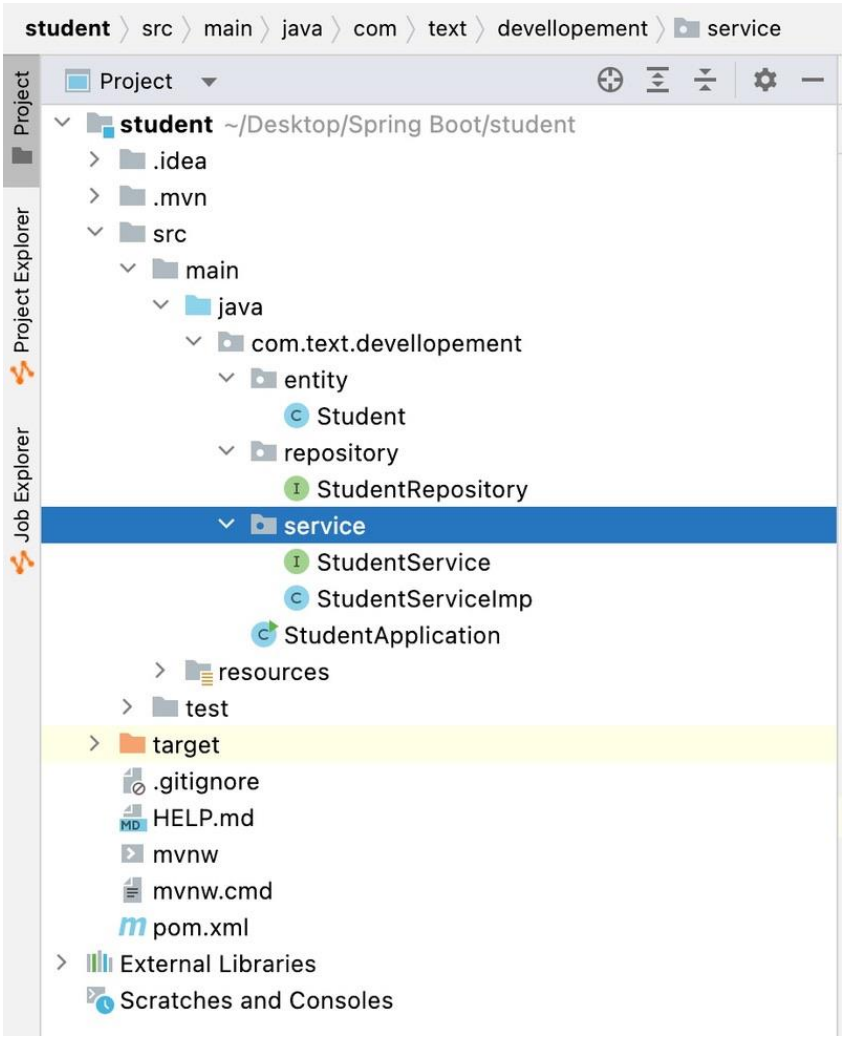
Hinane Aymane Rapport G5 IIR emsi centre

JpaRepository is a JPA (JAVA Persistence API) , c'est une extension de repository. Elle contient toute les api pour les requetes CRUD basic et aussi les api pour la pagination et le trie

Create 4 packages as listed below and create some classes and interfaces inside these packages as seen in the below image

- entity
- repository
- service
- controller



```
1 usage
@SpringBootApplication

public class StudentApplication  implements CommandLineRunner {
    12 usages
    @Autowired
    private StudentRepository studentRepository;

    public static void main(String[] args) {
        SpringApplication.run(StudentApplication.class, args);
    }
```

```
@Override
public void run(String ...args) throws Exception{
  System.out.println("********************Run********************");
    this.studentRepository.save(
            new Student( Id: null, registrationNumber: "A1", fullName: "Amine",new Date(), stillActive: true, lastConnection: null)
    );
    studentRepository.save(
            new Student( Id: null, registrationNumber: "A2", fullName: "Ilias",new Date(), stillActive: true, lastConnection: null)
    );
    studentRepository.save(
            new Student( Id: null, registrationNumber: "A3", fullName: "Saad",new Date(), stillActive: true, lastConnection: null)
    );
    System.out.println("********************Finish********************");
    System.out.println("Count: "+studentRepository.count());
    System.out.println("********************Display Rows ********************");
    List<Student> students = studentRepository.findAll();
    students.forEach(student -> {
        System.out.println(student.toString());
    });
    System.out.println("********************Get Elements By ID ********************");
    Student student  = studentRepository.findById(3).orElse( other: null);
    System.out.println(student);
    System.out.println("********************Update an Element ********************");
    student.setRegistrationNumber("S3");
    studentRepository.save(student);
    System.out.println("********************Delete an Element ********************");
    studentRepository.delete(student);
    System.out.println("Count: "+studentRepository.count());
    studentRepository.deleteById(2);
    System.out.println("Count: "+studentRepository.count());
    System.out.println("********************Search by Id Query ********************");
    Student student1 = studentRepository.SearchStudentById(1);
    System.out.println(student);
}
```


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

   @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer Id;

  @Column(name="REGISTRATION_N",unique=true)
    private String registrationNumber;

  @Column(name="Name" , length = 30 ,nullable = false)
    private String fullName;

  @Temporal(TemporalType.DATE)
    private Date birthday;

  private Boolean stillActive;

  @Temporal(TemporalType.TIMESTAMP) @CreationTimestamp
    private Date lastConnection;
}
```

Create a simple interface and name the interface as StudentRepository. This interface is going to extends the JpaRepository

```
interface JpaRepository<T,ID>
```

- ***T:*** *Domain type that repository manages (Generally the Entity/Model class name)*
- ***ID:*** *Type of the id of the entity that repository manages (Generally the wrapper class of your @Id that is created inside the Entity/Model class)*

```
4 usages
@Repository

public interface StudentRepository extends JpaRepository<Student,Integer> {
    1 usage
    @Query("select s from Student as s  where s.id = :id")
    Student SearchStudentById(@Param("id") Integer Id);

    List<Student> findByFullNameAndStillActiveIsTrue(String fullame);
    List<Student> findByRegistrationNumberLike (String registrationNumber);
    List<Student> findByRegistrationNumberLikeAndFullNameLike(String registrationNumber, String fullName);
    List<Student> findByIdIsLessThan (Integer id);
    List<Student> findByIdIsLessThanEqual (Integer id);
    List<Student> deleteByFullNameAndStillActiveIsTrue(String fullName);
}
```

```
1 usage  1 implementation
public interface StudentService {

    1 implementation
    Student saveStudent(Student student);
    1 implementation
    List<Student> fetchStudentList();
    1 implementation
    Student updateStudent(Student student,Integer StudentId);
    1 implementation
    void deleteStudentById(Integer studentId);
}
```

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

    3 usages
    @Autowired
    private StudentRepository studentRepository;


    @Override
    public Student saveStudent(Student student) { return studentRepository.save(student); }

    @Override
    public List<Student> fetchStudentList() { return (List<Student>) studentRepository.findAll(); }

    @Override
    public Student updateStudent(Student student, Integer StudentId) { return null; }

    @Override
    public void deleteStudentById(Integer studentId) { studentRepository.deleteById(studentId); }
}
```

Some of the most important methods that are available inside the JpaRepository are given below

Method 1: saveAll(): Saves all given entities.
Syntax:

```
<S extends T> List<S> saveAll(Iterable<S> entities)
```

Parameters: Entities, keeping note that they must not be null nor must it contain null.
Return Type: the saved entities; will never be null. The returned Iterable will have the same size as the Iterable passed as an argument.
Exception Thrown: It throws IllegalArgumentException in case the given entities or one of its entities is null.
Method 2: getById(): Returns a reference to the entity with the given identifier. Depending on how the JPA persistence provider is implemented this is very likely to always return an instance and throw an EntityNotFoundException on first access. Some of them will reject invalid identifiers immediately.
Syntax:

```
T getById(ID id)
```

Parameters: id – must not be null.
Return Type: a reference to the entity with the given identifier.
Method 3: flush(): Flushes all pending changes to the database.
Syntax:

```
void flush()
```

Method 4: saveAndFlush(): Saves an entity and flushes changes instantly.
Syntax:

```
<S extends T> S saveAndFlush(S entity)
```

Parameters: The entity to be saved. Must not be null.
Return Type: The saved entity
Method 5: deleteAllInBatch(): Deletes the given entities in a batch which means it will create a single query. This kind of operation leaves JPAs first-level cache and the database out of sync. Consider flushing the EntityManager before calling this method.
Syntax:

```
void deleteAllInBatch(Iterable<T> entities)
```

Parameters*: The e*ntities to be deleted, must not be null.
Implementation: Let us consider a Spring Boot application that manages a Department entity with JpaRepository. The data is saved in the H2 database. We use a RESTful controller.