Gerardo Andres Almaguer Ramos

Business Case: Student Enrollment and Data Management System

1. Introduction

This business case outlines the implementation of a system for managing student enrollment, deregistration, and data updates within a school. The system aims to streamline the processes associated with student records management, ensuring accuracy, efficiency, and ease of access.

2. Project Overview

The project is designed to handle three primary functions:

- **Student Registration:** The system allows for the enrollment of new students, capturing essential data such as personal information, contact details, and academic records.
- **Student Deregistration:** It provides a mechanism for removing students from the system, whether due to graduation, transfer, or other reasons.
- **Student Data Update:** The system enables updates to existing student records, ensuring that all information remains current and accurate.

3. Technical Approach

The project is implemented using Java Persistence API (JPA), a specification in Java that simplifies the management of relational data in applications using Java. JPA enables the mapping of Java objects to database tables and provides an abstraction layer over the database, making data persistence and retrieval more efficient and less error-prone.

4. Database Integration

The system connects to a local database where all student data is stored. The database is designed to handle large volumes of student information, with tables representing various entities such as students, courses, and enrollments. JPA handles the interaction between the Java application and the database, ensuring that operations such as adding, deleting, and updating records are performed efficiently.

5. Project Benefits

- **Efficiency:** Automates the enrollment and deregistration process, reducing the manual effort required by administrative staff.
- **Accuracy:** Ensures that student data is consistently updated and maintained, reducing errors and inconsistencies.
- **Scalability:** The system is designed to scale with the school's needs, handling increasing volumes of data as the student population grows.

- **User-Friendly Interface:** Provides an intuitive interface for administrators to manage student data with minimal training.

6. Conclusion

This project provides a robust and efficient solution for managing student data in a school environment. By leveraging JPA for database integration, the system ensures that all operations are performed with high efficiency and accuracy, ultimately contributing to better data management and overall operational efficiency within the school.

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

 :: Spring Boot ::                (v3.3.2)

Creating new student object ...
Saving the student ...
Save Estudiante
Saved student. Generated id: 13
Creating 3 student objects ...
Saving the students ...
Save Estudiante
Save Estudiante
Save Estudiante
Creating new student object ...
Saving the student ...
Save Estudiante
Saved student. Generated id: 17
Retrieving student with id: 17
Found the student: Student{id=17, firstName='Epeneto', lastName='Duck', email='daffy@luv2code.com'}
```

This is some text that the console prints, it's just a brief “explanation” of what the program is processing.

```
Found the student: Student{id=17, firstName='Epeneto', lastName='Duck', email='daffy@luv2code.com'}
Student{id=1, firstName='John', lastName='Doe', email='john.doe@foo.com'}
Student{id=2, firstName='Mary', lastName='Public', email='mary.public@foo.com'}
Student{id=3, firstName='Susan', lastName='Queue', email='susan.queue@foo.com'}
Student{id=4, firstName='David', lastName='Williams', email='david.williams@foo.com'}
Student{id=5, firstName='Lisa', lastName='Johnson', email='lisa.johnson@foo.com'}
Student{id=6, firstName='Paul', lastName='Smith', email='paul.smith@foo.com'}
Student{id=7, firstName='Carl', lastName='Adams', email='carl.adams@foo.com'}
Student{id=8, firstName='Bill', lastName='Brown', email='bill.brown@foo.com'}
Student{id=9, firstName='Susan', lastName='Thomas', email='susan.thomas@foo.com'}
Student{id=10, firstName='John', lastName='Davis', email='john.davis@foo.com'}
Student{id=11, firstName='Mary', lastName='Fowler', email='mary.fowler@foo.com'}
Student{id=12, firstName='David', lastName='Waters', email='david.waters@foo.com'}
Student{id=13, firstName='Pedro', lastName='Doe', email='pedro@luv2code.com'}
Student{id=14, firstName='John', lastName='Doe', email='john@luv2code.com'}
Student{id=15, firstName='Mary', lastName='Public', email='mary@luv2code.com'}
Student{id=16, firstName='Bonita', lastName='Applebum', email='bonita@luv2code.com'}
Student{id=17, firstName='Epeneto', lastName='Duck', email='daffy@luv2code.com'}
Student{id=1, firstName='John', lastName='Doe', email='john.doe@foo.com'}
Student{id=13, firstName='Pedro', lastName='Doe', email='pedro@luv2code.com'}
Student{id=14, firstName='John', lastName='Doe', email='john@luv2code.com'}
Getting student with id: 12
Updating student ...
Updated student: Student{id=12, firstName='John', lastName='Waters', email='david.waters@foo.com'}
Deleting student id: 15
Deleting all students
Deleted row count: 16
```

Now with the help of hardcode we can easily start the JPA project and run it as required.

```java
@Bean
public CommandLineRunner commandLineRunner(StudentDAO studentDAO) {

    return runner -> {
        createStudent(studentDAO);

        createMultipleStudents(studentDAO);

        readStudent(studentDAO);

        queryForStudents(studentDAO);

        queryForStudentsByLastName(studentDAO);

        updateStudent(studentDAO);

        deleteStudent(studentDAO);

        deleteAllStudents(studentDAO);
    };
}
```

Here are the principal methods that the program used to did, according to the principal needs of the school.