



Project: Learner's Academy

Online Management System

01/31/2021

Developed by:

Rafael Lopez Ambrosio

lopez.rafael08@gmail.com

<https://github.com/Rafael-Lopez>

www.linkedin.com/in/rafaellopezambrosio

Table of Contents

1- Product Description	2
1.1- Product Specifications	2
1.2- Application flowchart	3
1.3- Entity Relationship Diagram (ERD)	4
2.- Sprint Planning	5
2.1- Backlog	5
2.2- Task Descriptions	6
2.2.1- Sprint 1	6
2.2.2- Sprint 2	7
2.2.3- Sprint 3	8
3.- Core concepts & technologies	11
4.- Final Product	13
5.- Conclusion	17

1- Product Description

This project can be found through the following GitHub link

<https://github.com/Rafael-Lopez/PGFSD-Assignment-2>

The progress was recorded via GitHub, and every task has a corresponding issue created on GitHub in order to track the completion of the project.

<https://github.com/Rafael-Lopez/PGFSD-Assignment-2/issues?q=is%3Aissue+is%3Aclosed>

1.1- Product Specifications

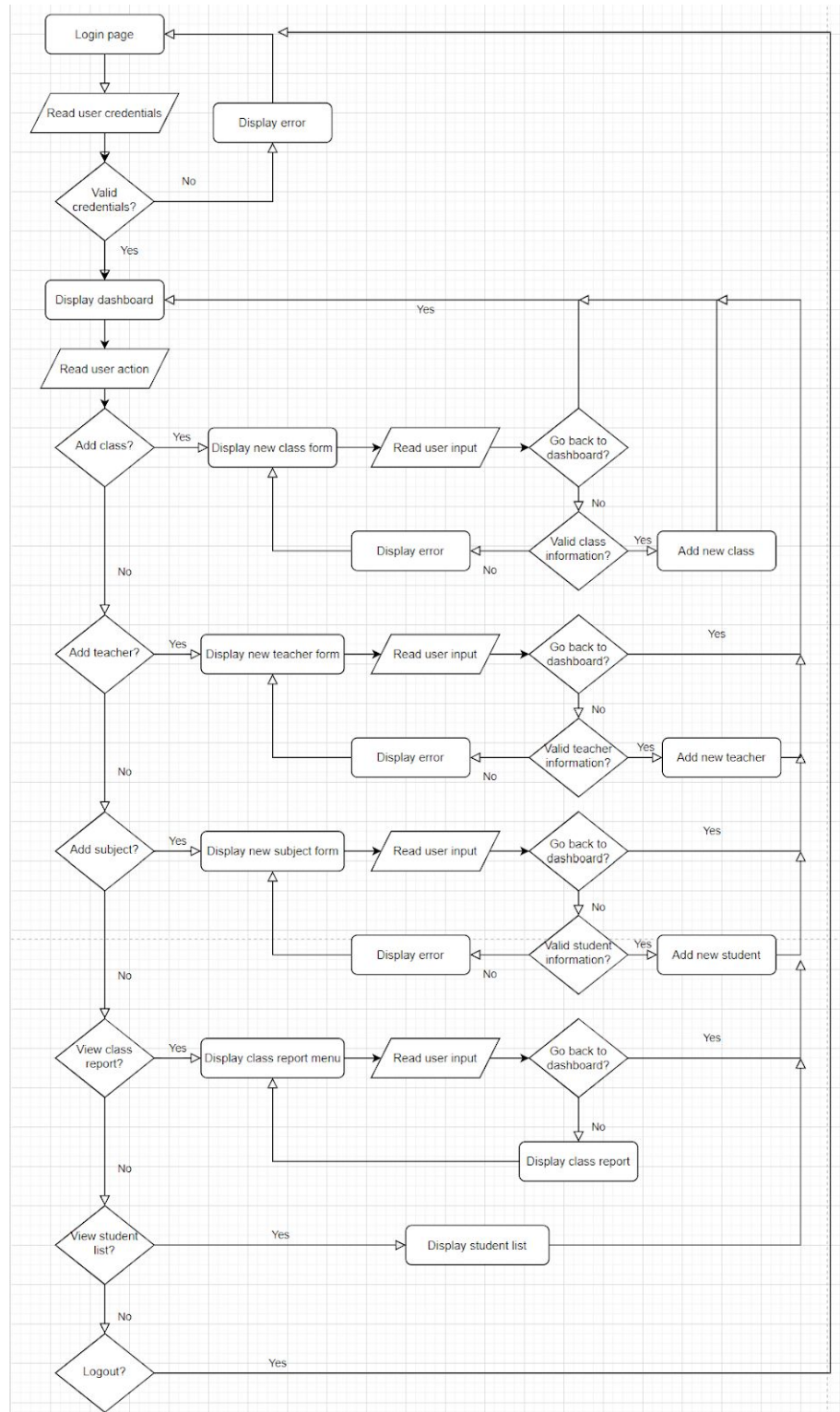
The system keeps track of its classes, subjects, students, and teachers. It has a back-office application with a single administrator login.

The administrator can:

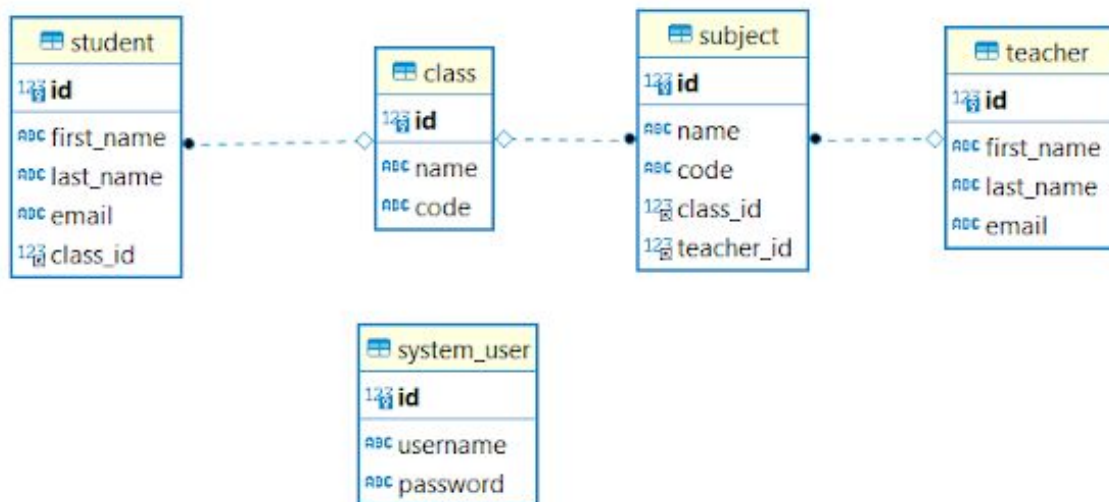
- Set up a master list of all the subjects for all the classes
- Set up a master list of all the teachers
- Set up a master list of all the classes
- Assign classes for subjects from the master list
- Assign teachers to a class for a subject (A teacher can be assigned to different classes for different subjects)
- Get a master list of students (Each student must be assigned to a single class)

There will be an option to view a Class Report which will show all the information about the class, such as the list of students, subjects, and teachers.

1.2- Application flowchart



1.3- Entity Relationship Diagram (ERD)



2.- Sprint Planning

Sprint length for this project will be 2 weeks, with a total of 2 sprints to have a final product. After the 2 sprints, all the tasks in the backlog are expected to be completed, resulting in a finished prototype to be presented to the stakeholders.

2.1- Backlog

Product Backlog	Sprint Backlog	Tasks
<ul style="list-style-type: none"> - Project setup - Database schema definition - Login page - Hibernate integration - New Subject form - New Class form - New Teacher form - New Student form 	Sprint 1	#1: As a Developer, I would like to initialize a GitHub repository #2: As a Developer, I would like to define a DB schema for the project #3: As a User, I would like to have a Login page #4: As a Developer, I would like to add a system_user DB table
	Sprint 2	#5: As a Developer, I would like to add Hibernate to the project #6: As a Developer, I would like for the Login page to connect to DB for authentication #7: As a User, I would like to have a form to add new subjects #8: As a Developer, I would like to add a Service layer to the project
	Sprint 3	#9: As a User, I would like to have a form to add new classes #10: As a User, I would like to have a form to add new teachers

	- Class report - Master list of students	#11: As a User, I would like to update the New Class form to be able to select a teacher when entering a new class #12: As a User, I would like to have a form to add new students #13: As a User, I would like to have an option to run a Class report #14: As a User, I would like to be able to get a master list of students #15: As a Developer, I would like to fix the entities and their relationship #16: As a Developer, I would like to convert DashboardServlet into a JSP file #17: As a User, I would like to have Go Back to Dashboard buttons in each registration panel
--	---	--

2.2- Task Descriptions

2.2.1- Sprint 1

#1: As a Developer, I'd like to initialize a GitHub repository

This is to create the repository where the code base will reside. Files such as .gitignore, README, etc., as well as the Maven project will be added as part of the project setup.

#2: As a Developer, I would like to define a DB schema for the project

This task is to create a database called "academy", considering the following entities:

- Subject
- Class
- Student
- Teacher

Where:

- Classes can be assigned to the master subject list
- Teachers can be added to a class for a subject (A teacher can be assigned to different classes for different subjects)
- Each student must be assigned to a single class

#3: As a User, I would like to have a Login page

The portal needs a single administrator login. Therefore, this task is to create a simple Login page that will interact with the DB for user authentication.

#4: As a Developer, I would like to add a system_user DB table

As part of the Login functionality, we need to be able to store system users, where we can keep track of their login information. Therefore, this task is to create such a table, which will be used for the login page.

2.2.2- Sprint 2

#5: As a Developer, I would like to add Hibernate + Log4j to the project

This project will use Hibernate (annotation-based) for the data access layer, as well as Log4j for logging purposes. This task is to add the Hibernate and Log4j dependencies to the Maven project.

#6: As a Developer, I would like for the Login page to connect to DB for authentication

When an admin tries to login, the credentials (username + password) are to be retrieved from the DB in order to authenticate registered system users.

#7: As a User, I would like to have a form to add new subjects

In this task, we need to create a form to allow users to add new subjects into the system. First, in the Dashboard page, after an admin has signed in, we need a "Register New Subject" button, which will redirect the user to a form where they can enter the subject's details (name and code). After the subject is added, the user is sent back to the Dashboard.

#8: As a Developer, I would like to add a Service layer to the project

A service layer is a good idea to separate the rest of the application from Hibernate so that unnecessary dependencies are not created by the data access layer. The service layer should be responsible for interfacing with the Hibernate data retrieval APIs.

2.2.3- Sprint 3

#9: As a User, I would like to have a form to add new classes

We need to create a form to allow users to add new classes into the system. First, in the Dashboard page, after an admin has signed in, we need a "Register New Class" button, which will redirect the user to a form where they can enter the class' details (name, code and subject). After the class is added, the user is sent back to the Dashboard.

Note that there should be a list of subjects in the form, which will be populated with the subjects previously registered by a user.

#10: As a User, I would like to have a form to add new teachers


In this task, we need to create a form to allow users to add new teachers into the system. First, in the Dashboard page, after an admin has signed in, we need a "Register New Teacher" button, which will redirect the user to a form where they can enter the teacher's details (first name, last name, and email). After the teacher is added, the user is sent back to the Dashboard.

#11: As a User, I would like to update the New Class form to be able to select a teacher when entering a new class

When adding a new class, there's no way to select the teacher. This task is to update the New Class form by adding a Teacher drop-down menu, which will be populated with the existing teachers in the DB.

#12: As a User, I would like to have a form to add new students

We need to create a form to allow users to add new students into the system. On the Dashboard page, after an admin has signed in, we need a "Register New Student" button,



which will redirect the user to a form where they can enter the student's details (first name, last name, emails and class). After the student is added, the user is sent back to the Dashboard.

Note that there should be a list of classes in the form, which will be populated with the classes previously registered by a user.

#13: As a User, I would like to have an option to run a Class report

A "View Class Report" option needs to be added to the main dashboard. If a user selects this option, a drop-down menu will be displayed, containing all the different classes available. Once the user selects a class, a report displaying the following information will be presented on the screen:

- Class name
- Subject
- Teacher
- List of students enrolled

#14: As a User, I would like to be able to get a master list of students

Users should be able to get a master list containing all students registered. The report will contain the following details:

- First Name
- Last Name
- Email
- Class

#15: As a Developer, I would like to fix the entities and their relationship

The original idea when designing the system was as follows:

- A subject can have many classes

However, after further analysis, I realized the way it's supposed to work is as described below:

- Class examples: Grade 1, Grade 2, etc.

- Subject examples: Math 1, Math 2, Biology 1, Biology 2, etc.
- A class can have many subjects
- A class can have many students
- A subject has a teacher
- A teacher can have different subjects in different classes
- A student is enrolled in one one class

Therefore, this task is to fix the way classes and subjects are working up to this point, since the relationships are not correct.

#16: As a Developer, I would like to convert DashboardServlet into a JSP file

The dashboard should be a JSP file since it's just a view. This task is to convert the DashboardServlet into dashboard.jsp

#17: As a User, I would like to have Go Back to Dashboard buttons in each registration panel

At the moment there is no way to go back to the dashboard when the user is on any of the registration panels. This task is to add a button to allow users to go back to the dashboard if they don't want to continue with the registration process.

3.- Core concepts & technologies

The following core concepts and technologies were used in the development of this project:

- a) HTML & CSS: the index page was purely coded in HTML. As for the views, even though they were done via JSP, they still have a lot of HTML. Along with this, CSS was used to give a nice look to certain pages and buttons. The List of Students and Class Report pages have a lot of CSS in particular. The Dashboard as well as the buttons throughout the entire application used some CSS as well.
- b) Maven: dependency management (Hibernate, MySQL, Log4j, etc) and WAR file generation was handled by using Maven. There were some challenges involved, such as no web.xml file being generated or updated automatically. Nevertheless, the benefits overcome the cons of choosing Maven as the tool to manage the project.
- c) Model View Controller (MVC) architecture: the different components of the application were organized by following the MVC architecture. All the entities (Class, Subject, Student, Teacher, etc) were added to a “model” package. Similarly, the servlets that act as the bridge between the views and the models, were placed in a “controller” package. Finally, even though there is no folder called “views”, all JSP and HTML files, which act as the presentation layer, are grouped in the same folder, keeping a clear separation of concerns among models, views, and controllers.
- d) Java Servlets: for the “controller” portion of the app. They act as the connection between the views and models. All business logic and data access is handled inside these servlets. There is a package called “services”, that contains classes in charge of taking care of all the Hibernate code (opening sessions, transactions, etc). In this way, the servlets are not crowded with Hibernate code, and all data-access-related code. This also allows switching ORM’s easily since all the Hibernate code is in a few places, which can be easily replaced if necessary, leaving the servlets untouched.
- e) JSP: as mentioned above, most of the views were created using JSP. This is because it allows access to Java beans in a quick and easy manner, while having a clean and easy-to-read HTML structure.
- f) JSTL: for cases where we had to iterate through a collection of beans in a view (list of students, class reports, form with drop-down menus), JSTL was very handy, the “<c:forEach>” tag in particular.
- g) MySQL: all data is stored in a MySQL Server database. However, this time I decided to experiment a little bit. Instead of a conventional local database, I used an Amazon Relational Database Service (Amazon RDS), where I mounted a MySQL Server instance. This allowed me to connect to it from anywhere, and not depend on a local database running. The process followed is described in this link <https://aws.amazon.com/getting-started/hands-on/create-mysql-db/>

Even though the database is up and running, a SQL script is located in the project (\src\main\resources\initial_schemas.sql) to show the SQL work done to create the database and tables, as well as the INSERT statement to create an admin user. Also, the username, password and URL for the connection were not added to the repository for security purposes. These will be provided in the Additional Remarks section, when submitting the assignment via the LMS portal.

- h) Hibernate: all Hibernate work was done using annotations. From declaring the entities in the code to the different entity relationships; the whole application is annotation-based. There were a few challenges faced during the implementation, one of the biggest ones was:

org.hibernate.LazyInitializationException : could not initialize proxy - no Session.

Due to the fact that some of the entity relationships were marked as Lazy (fetch type) and the way the different service methods open and close a session immediately; when using such entities, this error would pop up since Access to a lazy-loaded object outside of the context of an open Hibernate session will result in this exception. The problem was solved by using the following Hibernate property:

```
<property name="hibernate.enable_lazy_load_no_trans" value="true"/>
```

- i) Collections: List (with ArrayList being the specialization) and Set (HashSet implementation) were the collections used for this project. Set was used in some of the entities due to the following problem:

org.hibernate.loader.MultipleBagFetchException: cannot simultaneously fetch multiple bags

The solution for this problem, as described in this article

<https://vladmihalcea.com/hibernate-multiplebagfetchexception/> is to use Set instead of List.

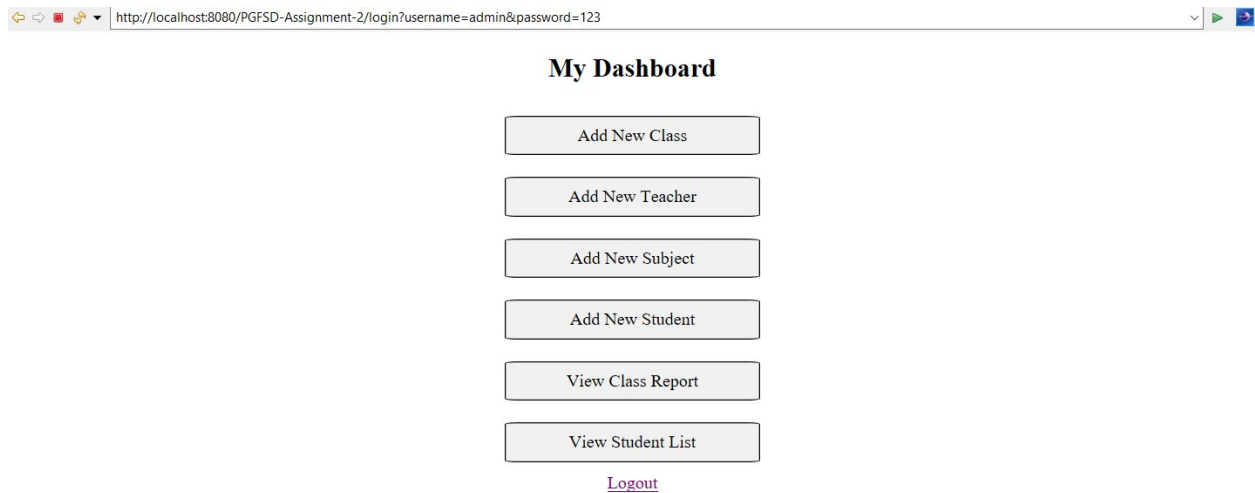
4.- Final Product

Screenshots are used in this section to present the final product. First, the login page is presented once the application is started.



The screenshot shows a web browser window with the address bar displaying `http://localhost:8080/PGFSD-Assignment-2/`. The page title is "Learner's Academy". Below the title, the heading "User Login" is centered. There are two input fields: "Username:" and "Password:". Below the password field is a "Login" button.

If the valid credentials are entered (see initial_schema.sql script for username and password), the user is presented with a dashboard page.




The screenshot shows a web browser window with the address bar displaying `http://localhost:8080/PGFSD-Assignment-2/login?username=admin&password=123`. The page title is "My Dashboard". Below the title, there are six buttons stacked vertically: "Add New Class", "Add New Teacher", "Add New Subject", "Add New Student", "View Class Report", and "View Student List". Below these buttons is a "Logout" link.

Otherwise, an error is displayed indicating the credentials are invalid

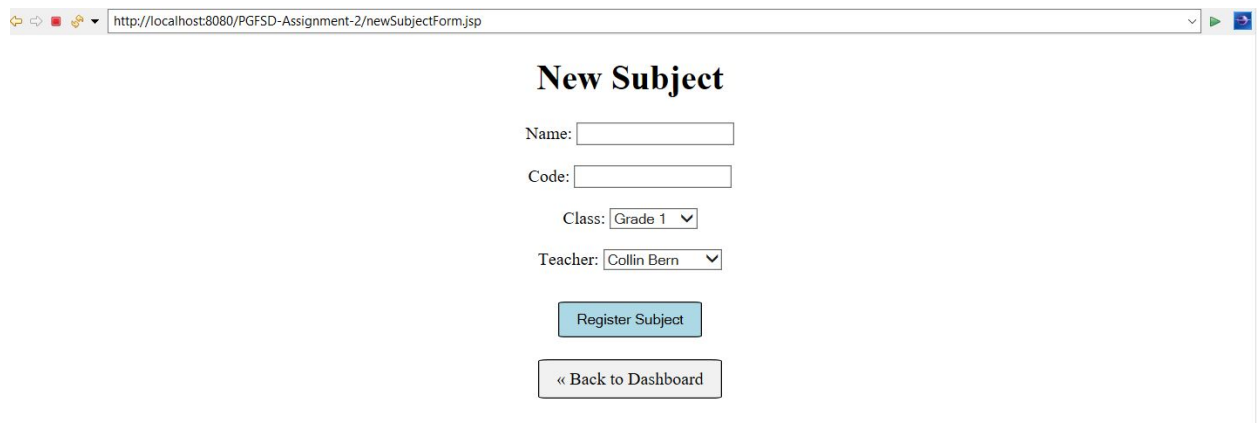


The screenshot shows a web browser window with the address bar displaying `http://localhost:8080/PGFSD-Assignment-2/login?username=ef&password=f`. Above the page title "Learner's Academy", there is a red error message: "Invalid Credentials!". Below the error message, the heading "User Login" is centered. There are two input fields: "Username:" and "Password:". Below the password field is a "Login" button.

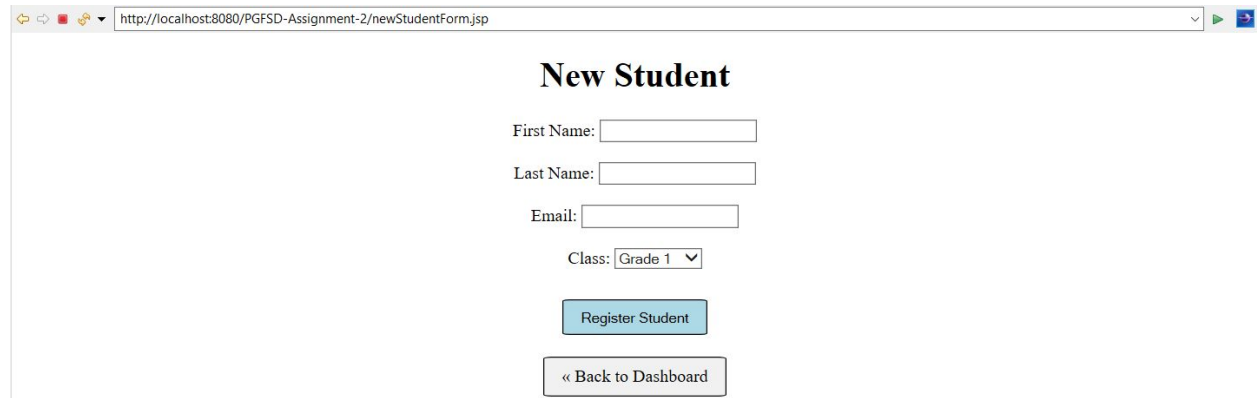
The dashboard has the options to add new classes, teachers, subjects, students, view a class report and display a master list of students. Each option has a respective “Go Back” button in case the user decides to not continue with the action previously chosen. Below are screenshots of each page:



The screenshot shows a web browser window with the URL `http://localhost:8080/PGFSD-Assignment-2/newClassForm.jsp`. The page title is "New Class". It contains two text input fields: "Name:" and "Code:". Below these fields are two buttons: a blue "Register Class" button and a grey "« Back to Dashboard" button.



The screenshot shows a web browser window with the URL `http://localhost:8080/PGFSD-Assignment-2/newSubjectForm.jsp`. The page title is "New Subject". It contains three text input fields: "Name:", "Code:", and "Class:". The "Class:" field is a dropdown menu showing "Grade 1". Below these fields are two buttons: a blue "Register Subject" button and a grey "« Back to Dashboard" button.



The screenshot shows a web browser window with the URL `http://localhost:8080/PGFSD-Assignment-2/newStudentForm.jsp`. The page title is "New Student". It contains three text input fields: "First Name:", "Last Name:", and "Email:". Below these fields are two buttons: a blue "Register Student" button and a grey "« Back to Dashboard" button.

Screens with drop-down menus (such as New Subject and New Student) are particularly interesting because they need to fetch the corresponding data from the database, in order

to populate the different menus. In other words, there are SELECT statements run behind the scenes by Hibernate, to retrieve, for example, the different classes available, when adding a new subject. It's important to mention that there is also validation in place for each screen, if the user does not provide all the information required, they are notified immediately and the registration process is cancelled.

http://localhost:8080/PGFSD-Assignment-2/registerStudent

All fields are required!

New Student

First Name:

Last Name:

Email:

Class:

This is an example of the View Class report option. First, the user needs to choose a class:

http://localhost:8080/PGFSD-Assignment-2/classReportMenu.jsp

Class Report

Select a class:

Once the class is chosen, the report is displayed:

http://localhost:8080/PGFSD-Assignment-2/classReport

Class: Grade 1

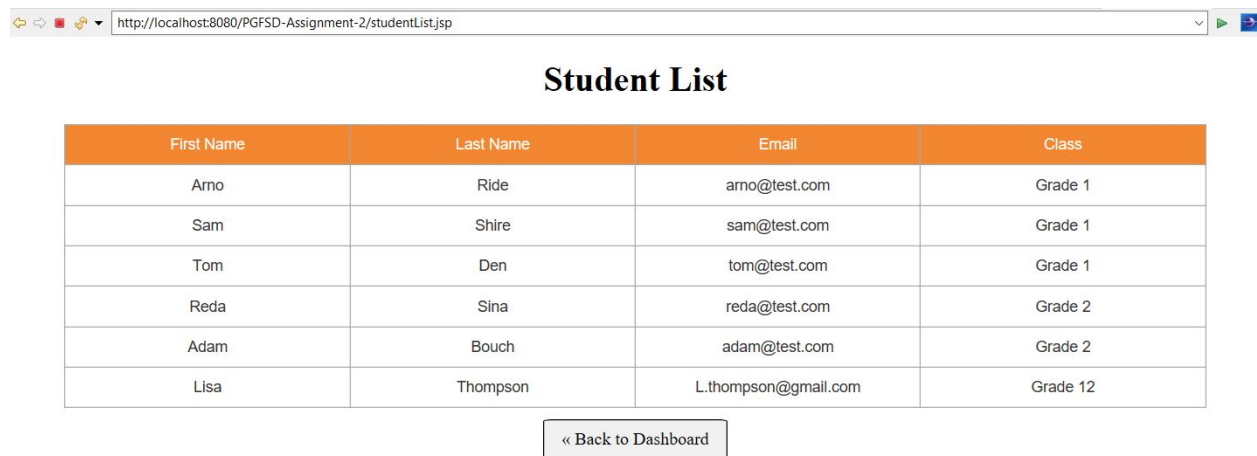
Subjects & Teachers

Subject	Teacher
Biology 1	Jannis Kampa
Math 1	Collin Bern

Students Enrolled

First Name	Last Name	Email
Arno	Ride	arno@test.com
Tom	Den	tom@test.com
Sam	Shire	sam@test.com

A screenshot of the master list of students:



First Name	Last Name	Email	Class
Arno	Ride	arno@test.com	Grade 1
Sam	Shire	sam@test.com	Grade 1
Tom	Den	tom@test.com	Grade 1
Reda	Sina	reda@test.com	Grade 2
Adam	Bouch	adam@test.com	Grade 2
Lisa	Thompson	L.thompson@gmail.com	Grade 12

[« Back to Dashboard](#)

Finally, if the user clicks on the Logout option in the Dashboard, they are sent back to the initial Login screen.

5.- Conclusion

While this project uses Hibernate and annotations for the data access layer, the transaction management aspect and setup can be greatly improved by using a higher level framework such as Spring. Spring Data in particular would be extremely beneficial, especially if it is used in conjunction with Spring Boot. The initial setup would be almost done in a couple of minutes, not to mention the introduction of repositories and transaction management through annotations.

Another area to improve is the front end work. Using React or Angular could make the development of the pages easier and more professional. CSS can be used more (even frameworks such as Bootstrap) to enhance the look of the whole application. This, in conjunction with more of an API-driven back end, could make the project more maintainable, reusable and scalable.

Unit and integration testing is another aspect to not be forgotten, especially because the project has a few external dependencies that can change overtime. Integration testing for the ORM is a good example, it would help prevent issues when the time comes to update it to a newer version.

Ultimately, as we move on to the next modules in the course, these concerns will probably be addressed as we learn more about Spring, Angular, React, testing, and DevOps.