**Learner’s Academy**

A Back-End,

Administrative Portal.

Run Index.html!

1. **Project Statement:**

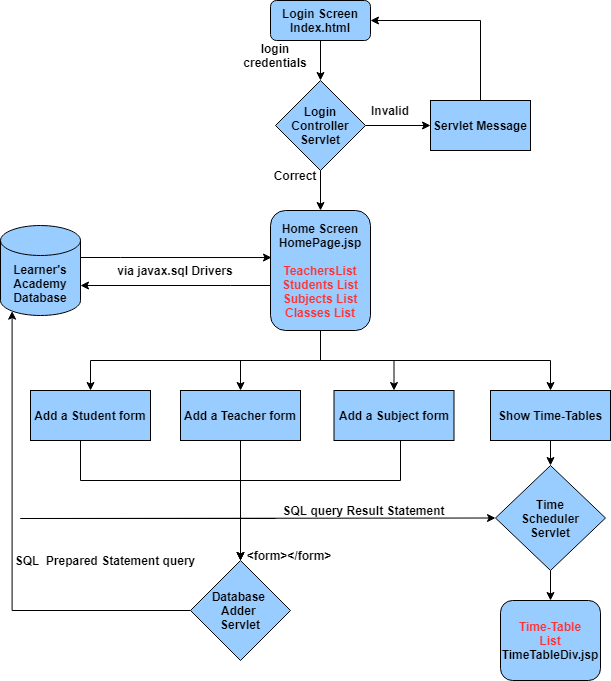
design and develop a backend administrative portal for the Learner’s Academy. Learner’s Academy is a school that has an online management system. The system keeps track of its classes, subjects, students, and teachers. It has a back-office application with a single administrator login.

1. **Sprint Planning:**
2. Product Backlog:
3. **Plan Flow** of the Back End.
4. Algorithm for Login, Database Connectivity, Dynamic Page Generator.
5. Sorting Algorithms. Collections, Exception Handling.
6. Adding of Teachers, Students, Subjects from HTML interface (additional).
7. Plan **JAVA Classes, Servlets, .HTMLs** and **.JSPs**.
8. Algorithm for LoginController Servlet.
9. Algorithm for HomePage Servlet.
10. Algorithm for DatabaseAdder Servlet.
11. Algorithm for TimeScheduler Servlet.
12. Structures of Class, Teacher, Student, Subject classes.
13. Create **SQL file** to create schemas, tables and fill random values.
14. Create a ‘**Create Schema and Tables.sql**’ file.
15. Create a ‘**Add Random Values.sql**’ file.
16. Create **Classes, Servlets, .JSPs** and **Servlet** codes to display data from **Database**.
17. Create Classes, Teachers, Students, Subjects and Time Table classes.
18. Create ‘HomePage.jsp’, ‘WelcomeDiv.html’, ‘Index.html’, ‘TimeTable.html’ files. Configure ‘web.xml’ file.
19. Create LoginController Servlet.
20. Create HomePage Servlet.
21. Create DataBaseAdder Servlet.
22. Create TimeScheduler Servlet.
23. Test Run App.
24. **Run** and **Debug** Application.

1. Sprint Table:

|  |  |  |  |
| --- | --- | --- | --- |
| Sprint No. | Tasks | Estimation | Status |
| 1. | **Plan Flow** of the Back End | 4 hours | Pending |
| 2. | Plan **JAVA Classes, Servlets, .HTMLs** and **.JSPs** | 2 hours | Pending |
| 3. | Create **SQL file** to create schemas, tables and fill random values | 2 hours | Pending |
| 4. | Create **Classes, Servlets, .JSPs** and **Servlet** codes to display data from **Database –- PART I** | 8 hours | Pending |
| 5. | Create **Classes, Servlets, .JSPs** and **Servlet** codes to display data from **Database – PART II** | 8 hours | Pending |
| 6. | Test Run App | 4 hours | Pending |

1. Sprints:
2. **Plan Flow** of the Back End:
3. Algorithm for Login, Database Connectivity, Dynamic Page Generator:



1. Sorting Algorithms. Collections, Exception Handling:

In this we use ArrayLists<TimeTable> because using TimeTable[] timeTableArrays with

Databases is difficult as getting the number of entries is harder with ResultStatement rs as it is a Buffered writer type output. i.e. it is in different amounts of bytes rather than an explicit amount. We also use the comparator interface so that we can sort the ArrayList according to timeslot or dayOfTheWeek datamembers of the TimeTable class.

We use Exception Handlings such as SQLException, ArrayIndexOutOfBounds Exception, IOException, ServletException, etc. These are important for reducing known crashes as is the use of Exception Handling. Also Errors are conveyed to user so that App can work as intended. Although we also use Types sparsely and correct TypeCasting of HTML sent requestParameters so that Exceptions can be avoided.

1. Adding of Teachers, Students, Subjects from HTML interface (additional):

We add some additional features to improve the quality of our product and save time as well

as deliver more with this product. We have added HTML forms that send requests to a Servlet which then connects with database to insert additional entries. The features are:

* **Add a new Student.**
* **Add a new Teacher.**
* **Add a new Subject.**
* **Three different classes and their Time-Tables.**

1. Plan **JAVA Classes, Servlets, .HTMLs** and **.JSPs:**
2. Algortithm for LoginController Servlet:

We use loginController servlet to check if the login credentials entered are the correct ones or

invalid ones so that they can be redirected accordingly. If the credentials are wrong ones, we use RequestDispatcher to redirect the user back to index.html and include some other .html file or some generated text as feedback. If the credentials are right, we still use RequestDispatcher to forward the request to another servlet so that we can view the content that is usually sensitive and hidden to everyone but the user with correct login credentials.

1. Algorithm for HomePage Servlet:

We use HomePage servlet so that we can connect and communicate with the database and

get the sensitive data stored in it for display for the user with correct login. Here we use RequestDispatcher and include all the different elements of pages which build the page that the user sees. Hence we use .include() method again here. We can alternatively use Java Bean and send all the objects to the user itself. But that would be rather tedious and contains the risk of data leaking if the login device was a public one.

The HomePage servlet uses Connection for connecting to database link, PrintWriter to generate dynamic response, ArrayList to store all the data got from the database, ResultSet as response from database then convert it to objects using constructors. It also uses ExecuteQuery to send commands to database, .sort() methods of ArrayList is used and Comparator is provided in the class itself wherever required. It also has the additional methods required so that the request and data the forms provide in the .html can be used to forwarded to another Servlet for breaking down, making objects for use and storing it in database in another servlet.

1. Algorithm for DatabaseAdder Servler:

We use DatabaseAdder servlet to store additional objects such as Teachers, Students and

Subjects. Data we get from the post method via the parameters is used to make Objects using the Teacher, Student or Subject constructors. These objects are then stored in memory as well as in database using the Connection, PreparedStatement objects and .execute() methods.

We use HTML hidden form field to locate the form number and hence the method that needs to be executed to be saved in the right table in the database. Form number also helps us change statements because every statement is different as every table is different and so are the incoming parameters. We also use HTML to fool proof the methods as the forms cannot be submitted without submitting all the required fields which are NOT NULL in the database and hence if the form is sent, the database is updated as user error has been removed. After submitting the form we use the RequestDispatcher, namely the .forward() method so that we can redirect the user to the same page but with the tables updated. We only show Teachers, Subjects, Classes and Students table on this page.

1. Algorithm for TimeScheduler Servlet:

We use this servlet to display all the Time-Tables of the three classes with the subjects and

the assigned teachers we use generate most of this page dynamically. We use ResultStatement, Connection and .executeQuery() methods to get the information from database, convert them into objects so they remain in data pool, use constructors to turn them into objects i.e. TimeTable. Now we use for-each loops and PrintWriter to dynamically generate the web-page according to the amount of data.

1. Structures of Student, TimeTable, Subject, Teacher and Class:

We use create most of the data members as we want them to exist in the database but the

Primary key which is identification is generated in the database using Auto-Increment and is given to the object hence the object does not have a setter method for the unique identification. We implement the Comparator method in TimeTable class as we need to sort it when we have it in a collection ArrayList.

1. Create SQL files to create schema and fill random values:
2. Create a ‘Create Schema and Tables.sql’ file:

We create this file thinking that it will create a schema named ‘learnersacademy’ and also

add tables students, teachers, subjects, classes and timetables. We create this file as fool proof so that if a schema already exists and/or the tables do too, the file is executed it first drops the schema as well as the tables then proceeds to create a new one along with the new tables. This file can also be used as reset if the tables are altered and/or functioning of app stops.

1. Create a ‘Add Random Values.sql’ file:

We create this file thinking that it will truncate the already existing values in the tables and

Add the random values that we create so that the user can experience the app the way it was first given. Although beware the entered data will be lost if the file is executed. We create this file fool proof so that if the file is executed the tables are first truncated. This step is also necessary for proper functioning of the TimeScheduler servlet. This file can also be used as reset if the tables are altered and/or functioning of app stops.

1. Create Classes, Servlets, .JSPs and Servlet codes to display data from Database:

We create all the necessary java classes, java HttpServlets, .JSP files and .HTML files and

also debug the application where necessary to ensure proper working of the portal. The coding is done in Eclipse IDE using jdk 16 and JRE 1.8, the codes are uploaded to GitHub. GitHub link is at the start and end of document and footer of every page.

|  |  |  |  |
| --- | --- | --- | --- |
| Sprint No. | Tasks | Estimation | Status |
| 1. | **Plan Flow** of the Back End | 4 hours | Done |
| 2. | Plan **JAVA Classes, Servlets, .HTMLs** and **.JSPs** | 2 hours | Done |
| 3. | Create **SQL file** to create schemas, tables and fill random values | 2 hours | Done |
| 4. | Create **Classes, Servlets, .JSPs** and **Servlet** codes to display data from **Database –- PART I** | 8 hours | Done |
| 5. | Create **Classes, Servlets, .JSPs** and **Servlet** codes to display data from **Database – PART II** | 8 hours | Done |
| 6. | Test Run App | 4 hours | Done |

1. Test Run App:
2. Test and Debug the App:

The application was run and debugged several times. It was tweaked until all the

grammatical errors and logical errors were sorted. A new feature was added where user can now add Teachers, Students and Subjects from the database itself.

1. **Working of the App:**
2. Logging in to the portal:

Login Screen:



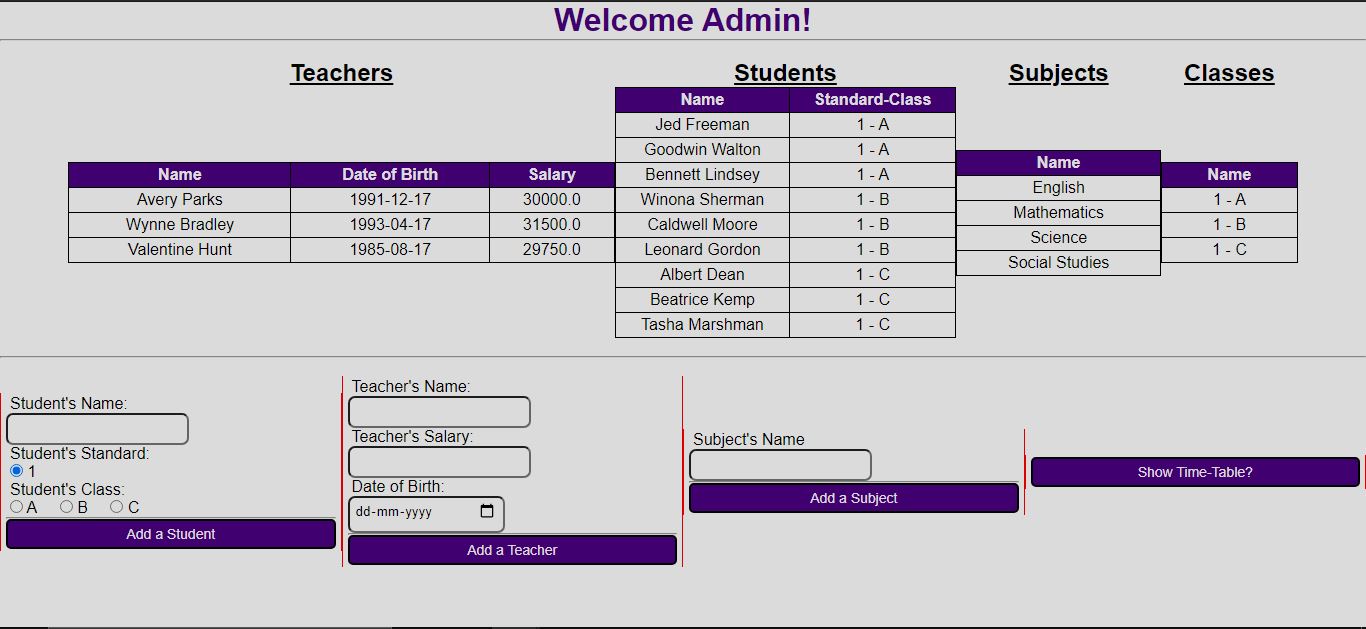
Login Box

Login Fail:



Login Fail

Home Page:



Show Time-Table Button

Add a Subject form

Add a Teacher form

Add a Student form

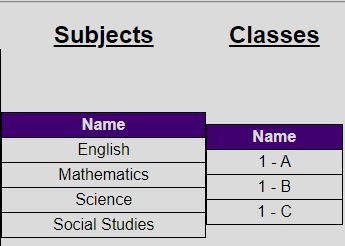
Classes List

Subjects List

Students List

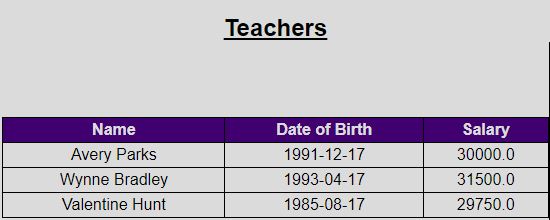
Teachers List

1. Master List of all Subjects for all Classes:

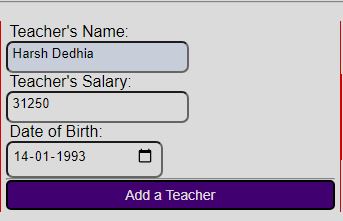


1. Master List of all Teachers:

Teacher List

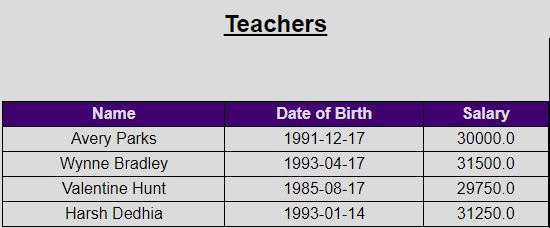
****

Adding Teacher:

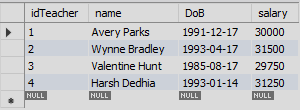


**Add a Teacher Form**

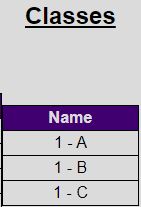
Updated Portal List:



Updated SQL List:

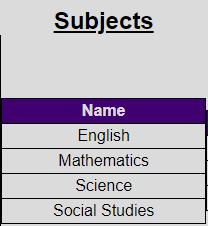


1. Master List of all classes:



1. Assigning Subjects:

Subjects List:

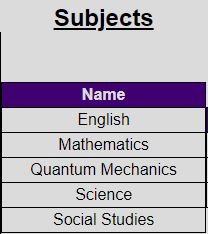


Adding Subject:

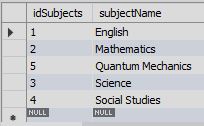
**Subject Form**



Updated Portal List:

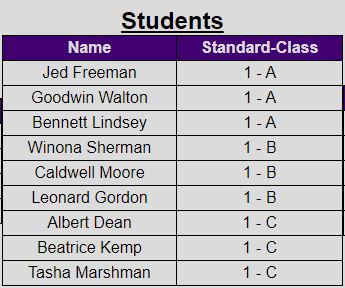


Updated SQL List:

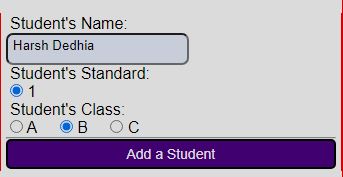


1. Student Master List:

Students List:

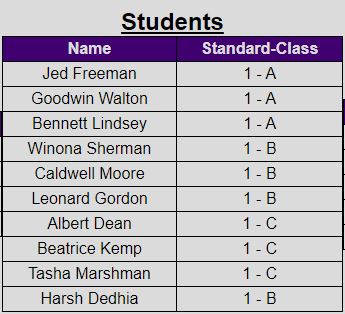


Adding Student:

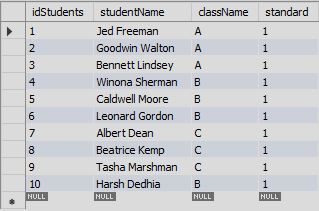


**Add a Student Form**

Updated Portal List:



Updated SQL List:



1. **Git and GitHub:**

Now we link git on PC and git repository on git hub using git bash. Then we use git commit and git push commands to upload to repository and give a commit message.

