Design Document

Bill Wang, Jacob Wallace, Yang Yang

Spinning Lightning Aggie Miner Doods (S.L.A.M.D.)

Project Overview

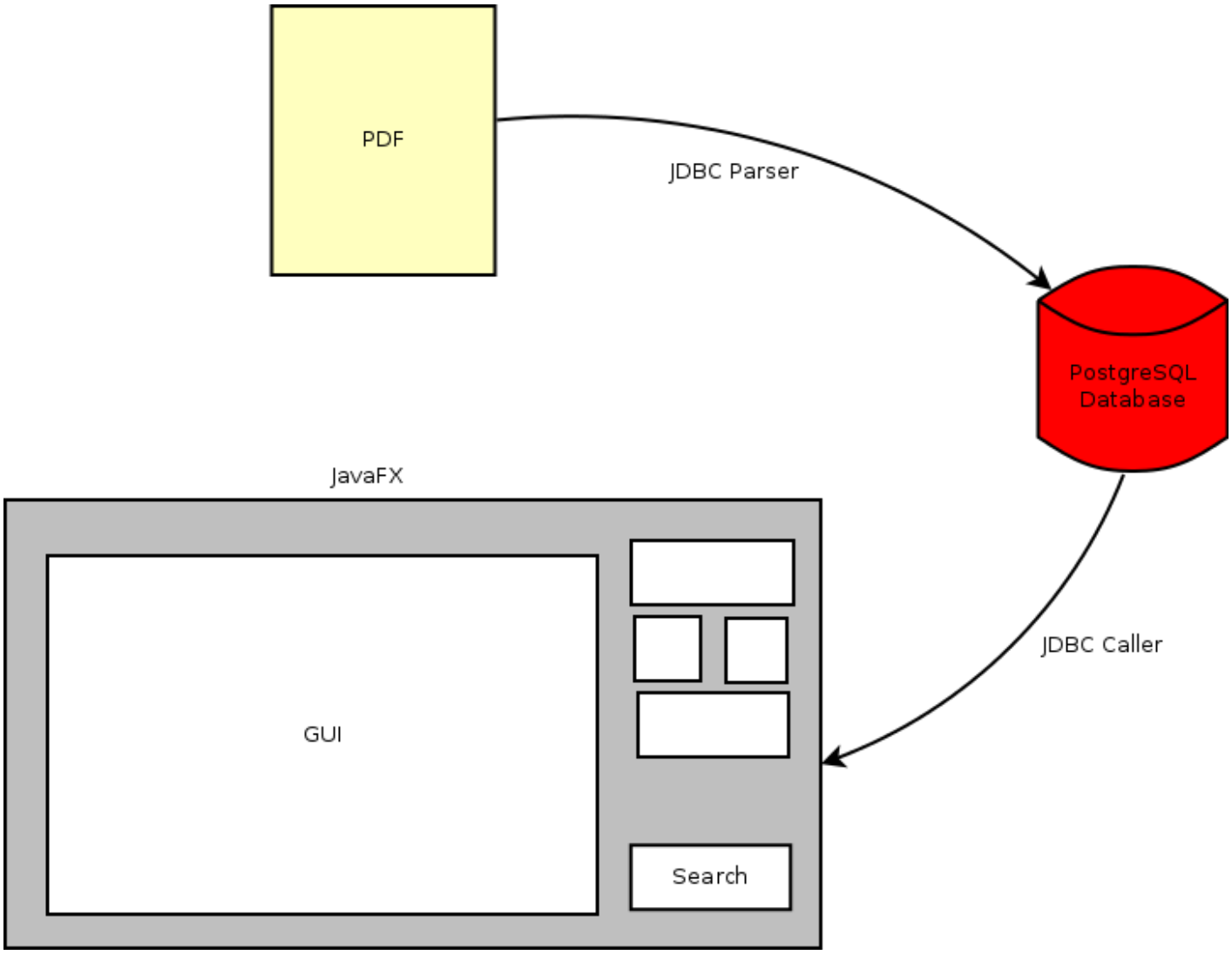
Our team is a subcontractor to a company named AYDB (“All your databases are belong to us”) with the goal of creating a prototype that will be used in place of Howdy’s class schedule system. This program will primarily be used by students and, will be able to search for courses based on a specific set of criterias as following,

- subject
- course number
- course name
- instructor
- start/end time
- class days
- capacity
- more..

The prototype will have a GUI which allows the user to query for information from the SQL database and display the search results in a scrollable window. The GUI will display the information neatly in a table for ease of reading. The information in the database will come from PDF files. This requires the creation of a parser program that can convert the PDF to a text file that is parsable. After which it will store this information in tuples and push the sets into the SQL database uniquely to minimize duplicates.

After the data is inserted into the database by the parser using JDBC, user will be able to use GUI to generate query to the database, which will return the corresponding result to the GUI, and the interface will display it in the window. An additional log system is also included in the GUI, which will request data base to forward previous search results.

System Description



As presented in the diagram above, the system is implemented separately in three parts. We are given a text file with the information of all classes at Texas A&M University. From there the information on each class needs to be inputted into the database as a seperate entry.

The first step to accomplish this task is to implement a parser to pull the information we require to a format usable by the JDBC to query the Database with an INSERT call.

The second step in the process will be the database itself. The database is designed as a PostgreSQL database to hold the information. There will be a single table that has multiple attributes per entry in the table. Attributes will consist of Subject, Course Number, Title, Start Time, End Time, Days, and Professor. From these attributes we may create other relationship tables based on these attributes when needed for queries.

Once the database has been loaded with information from the parser, the GUI can take contorl. For the graphical interface to display information from the database, JavaFX is used. The UI will consist of many different types of input methods and buttons to encompass a search query. From this query we will utilize the JDBC to query the database with SELECT statements that return all entries with the specified parameters. To finally display the information to the student who is searching, the data will be input into a table for ease of reading.

Subsystem Description

The following section will describe the subsystems implemented

Parser

Usage

Parser is responsible for extracting class information from the text file and inserting it into the database using JDBC. Java is going to be used to implement the parser.

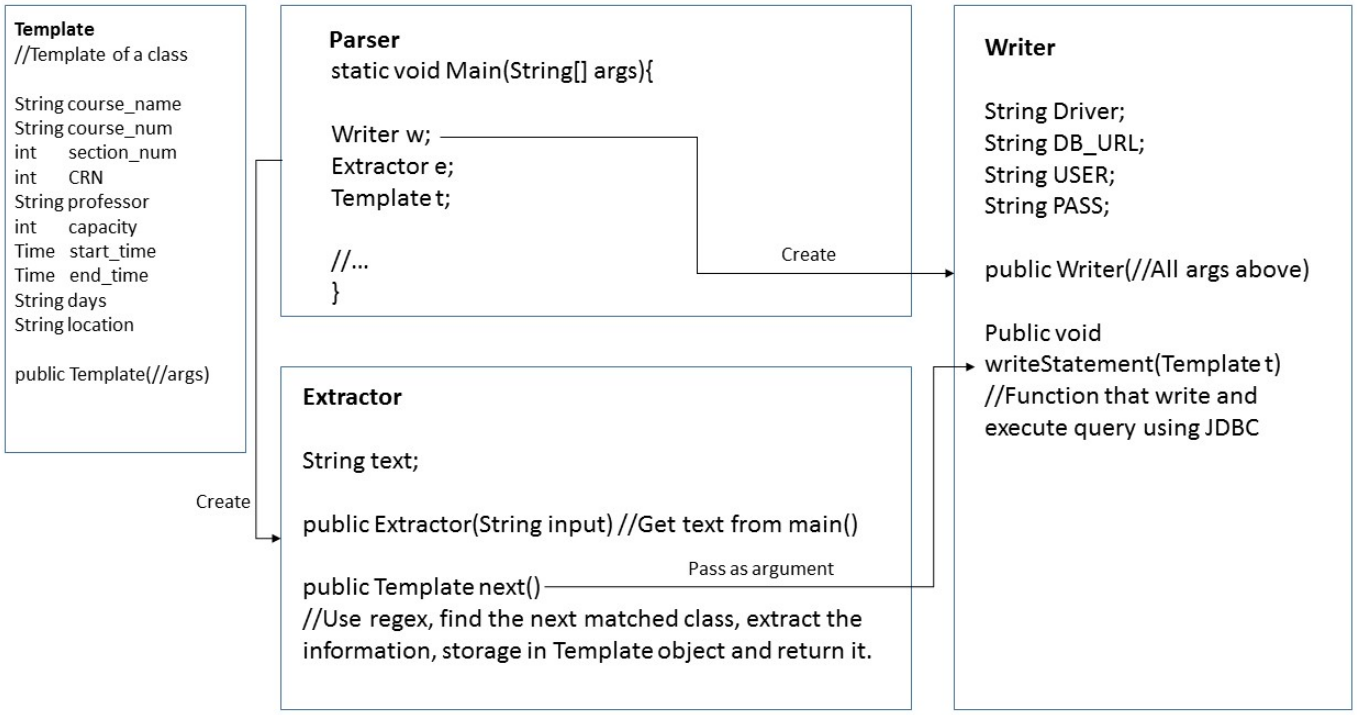
The parser has four main components, the main **parser** class, the **template** class, the **extractor** class, and the **writer** class.

The **parser** class will read in the text file and extract the text into a string, create an extractor object, pass the string as parameter during construction, a writer object and a template object.

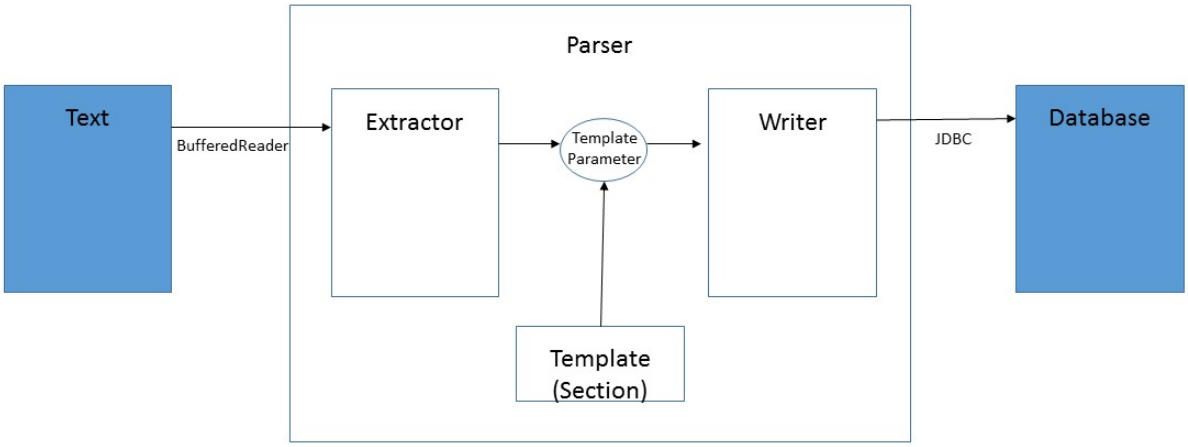
The **extractor** object will parse the text string and extract class information, using regular expression, and store information of each section in a **template** class object, and pass it to the **writer**, through the template object in parser, which will be responsible for calling the writing function in **writer** object, and pass the template object containing the current section's information as parameter.

The **writer** object, will read section information from the template object passed in as parameter, and insert the information to the database using JDBC, section by section.

Model



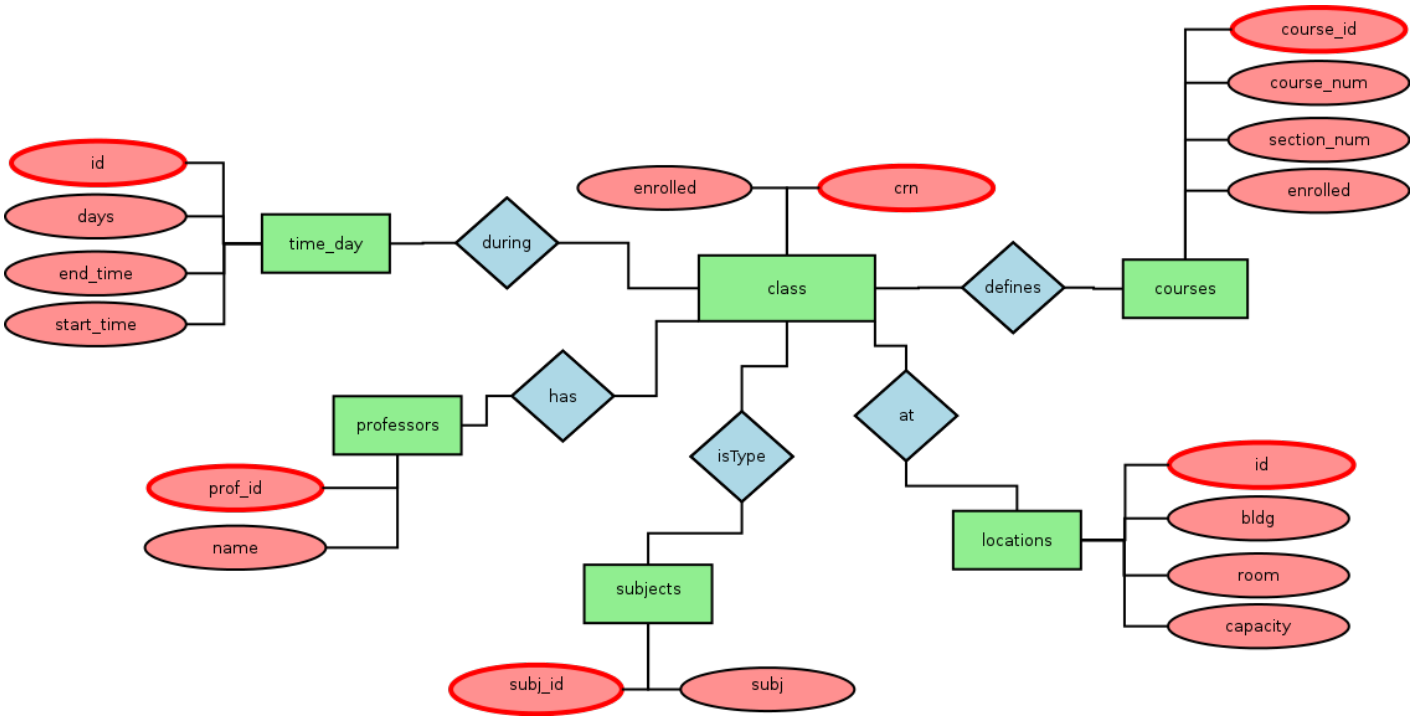
Interaction



Database

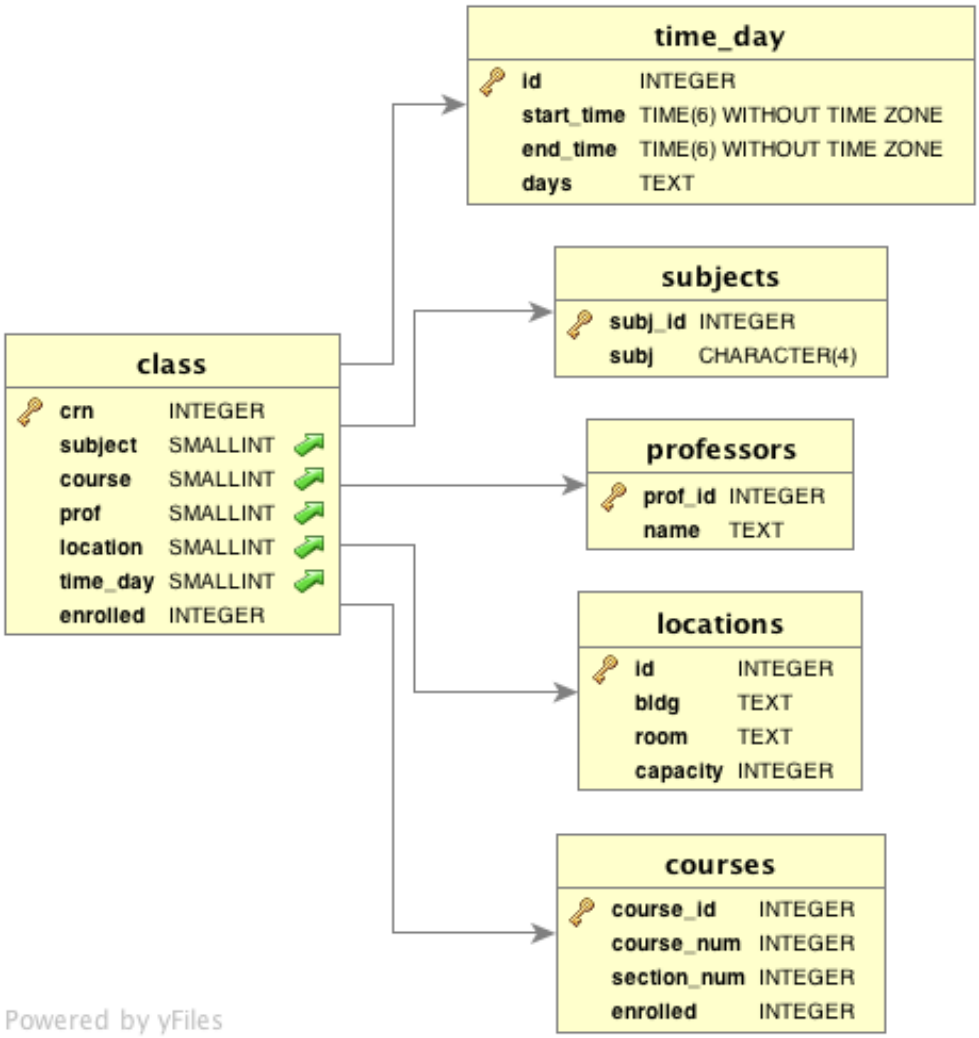
The database will act as the central unit for all data in the system. To organize the data into a more workable and readable form it will be stored in tuples. Each tuple of information will be used to define a table. Each entry in a single table will store data with mathching tuples. For classes schedule, data had to be seperated into multiple tuple sets to allow for a relationship between these tuples to occur. In our case, professor table is defined as having an id, and a name. The tuple is the prof_id, and name together.

The following is the ER Diagram the portrays the PostgreSQL Database designed.



As you can see in the diagram, there is a master table called class. All other tables will be referenced from that table and point to the row where the data matches the INSERT called in the parser. It was setup in this way to avoid duplicates in the data of tables. This helps minimize size and speed up look up times by decreasing elements to look through. To

connect these tables to the master table was easy with the use of foreign keys. Foreign keys are not visible in the ER Diagram so the creation of a Database schema.



The foregin key in the master table references to the primary keys of the other tables it connects. This allows the master table to pull the other data in the tuple attached to this unique id in the other tables. All this information together defines each unique class that we will read from the PDF document of classes.

UI

The UI will display the buttons and input boxes the user will interact with to produce a search query. It will also display the search results, and a log showing the user's previous search criteria. The core of the application will be a BorderPane layout pane, which consists

of five regions to place nodes (top, left, center, right, and bottom).

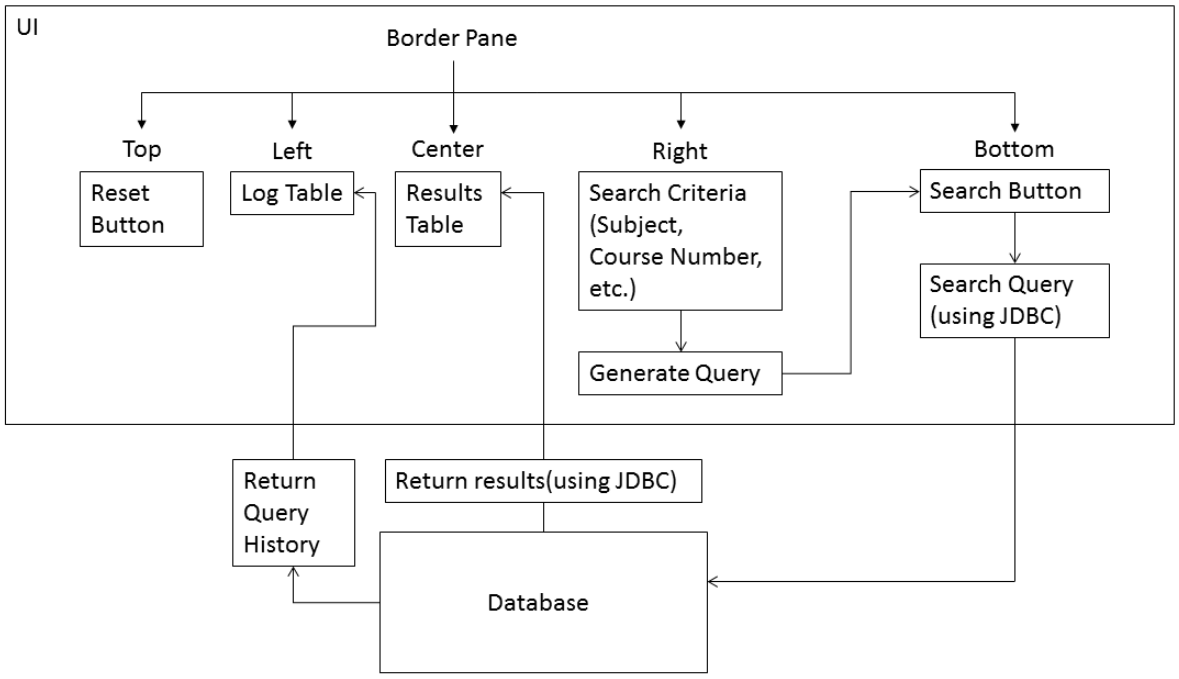
The right region will hold the buttons and input boxes that will be drawn using the JavaFX UI controls Combo Box, Text Field, Checkbox, and Button. These buttons and text fields allow the user can construct a search query.

The bottom region will have the search button, which when pressed, takes information placed in each UI control (Text Fields, Buttons, etc.) and converts it to JDBC. Using JDBC, the search query will be sent to the SQL database.

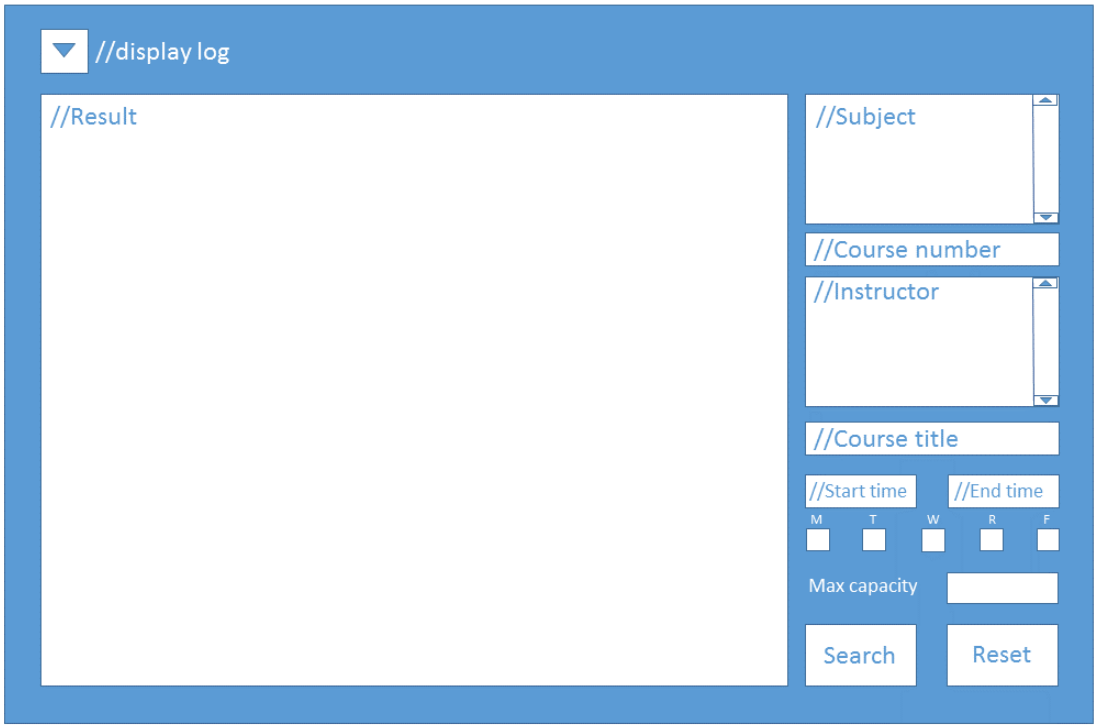
Once the database finds the information, we will use a ListView to take the output of the database and display it in the center region of the BorderPane.

In the top region there will also be a function used to reset all the search results. This includes unchecking boxes, clearing text fields, and resetting pull-down menus.

The left region will have a log for previous search criteria, which will be extracted from the SQL database using JDBC.



Preview of the UI



Benefits/Risks/Issues/Assumptions

Benefits

- Compare to Howdy, users can have more accessible approach to search class schedules, with more useful search keys like class-level, seat number, etc, and user-friendly option like multiple search keys and wildcards.
- Quick response, all search results are display on the panel next to the search options.
- Search log is a great tool for user to save time without doing the same search repetitively.
- If the program works as intended it will provide an easier method of searching for classes with maintaining a similar feel of environment of what the student is used to using. Lastly it will be a local copy so the user will be able to search classes when he/she is out of range of internet connection.

Risks/Issues

- Modification of the database without permission will cause errors, as the database is not prepared for unrestricted access. The database will need to be updated at some point as well, since several times are labeled “TBA”.
- Currently, regular expression is used to implement the parser, which is not robust. If there are any changes in the structure class text file (for example, a new column), the parser might not function. A possible solution would be using external parser libraries that use formal grammar to parse. Performance of regex is another issue.

Assumptions

We assume that the PDF used is accurate and that the user should not be able to access or modify the database directly. We also assume that the no additional search methods are needed, such as searching by CRN or searching by the availability of the course.