Project DB2: TDT4145 - Data Modelling, Databases and Database Management Systems

Alexander J. Ohrt, Martin Olderskog, Jim Totland

March 25, 2021

Prerequisites and Instructions

We have chosen to use Python (Version 3.8.5) to implement the Piazza-like forum. Consequently we have used this MySQL connector. The connector can be installed with `pip`, by writing

```
> pip install mysql-connector-python
```

in your terminal. We have also used the package `prettytable`, which can be installed by typing

```
> pip install prettytable
```

in the terminal. To get the program up and running, you need to follow these instructions

1. Run the SQL-script `createTables.txt` to create the database and insert some data.
2. In the `DB_connector`-file, match the username and password in the `self._config`-attribute with the username and password of your local MySQL Server.
3. Run the `main` file.

If you so desire, we have added a `requirements.txt`-file that may be used to make a virtual environment with Python. This way, both `mysql-connector-python` and `prettytable` are installed at the same time. A virtual environment should be made with

```
> python -m venv piattsa
```

which makes a folder in your current directory containing the virtual environment. Then, it should be activated. On Unix or MacOS, run

```
> source piattsa/bin/activate
```

Finally, once the virtual environment has been activated, run

```
> python -m pip install -r requirements.txt
```

to install the packages. Consult the documentation for more information. As a sidenote, another advantage of using a virtual environment is that this way one can eliminate some dependency-related issues, e.g. that different applications require different versions of the same libraries. Since the virtual environment is self-contained, it can be easily deleted without affecting other environments.

API Reference

The classes that exist in our program are documented in this section. A simple UML-diagram is shown in figure 1, in order to give a graphical display of the classes and their relationships. They are the following

1. `DB_connector`
2. `UI_ctrl`
3. `User_login_ctrl`
4. `Make_post_ctrl`
5. `Make_reply_ctrl`
6. `Search_post_ctrl`
7. `Statistics_ctrl`

`DB_connector` is a class that handles connecting to the MySQL database, using `mysql.connector`. When an object of this class is instantiated, it connects to the database with the attributes specified in its constructor. The program has some simple exception-handling in order to make sure that the database is correctly connected before continuing. The connection to the database is closed when Python calls the destructor of the `DB_connector`-object. One main `DB_connector`-object is made and used throughout the entire running time of the program. In the following, we refer to the `DB_connector`-object instantiated in the main file as the *connection* (as it is called in the code).

`UI_ctrl` is a class that controls the textual user interface. The connection is passed to a `UI_ctrl`-object that is made in the main-file. From here, the `UI_ctrl`-object passes the connection on to the other necessary controllers for different use cases in the forum.

`User_login_ctrl` is a controller that handles user-logins to the system. The connection is passed to the constructor in order for the suitable queries to be made, i.e. to check the credentials (email and password) passed to the constructor. If they match, the user with the given credentials is logged in. Also, a cursor, which is an object from `mysql.connector` that is used to query the database, is instantiated in the constructor. The class also contains methods to get the type of the user (instructor or student), to get the UserID and to insert user- and post-tuples into ViewedBy. These methods are used in some of the other classes, in order to keep statistics about which posts each user have viewed. Finally, when the destructor is called, the cursor is closed to avoid any loose ends.

`Make_post_ctrl` is a controller that handles insertion of posts into the database. The connection is passed to the constructor in order for the suitable queries to be made, i.e. to insert posts into the Post-table. Also, a cursor is instantiated in the constructor. Moreover, an already logged in `User_login_ctrl`-object, a folder name and a course-id is passed to the constructor. The class contains a method to insert a post, with the given summary, text and tag from the user, i.e. the author of the post. Furthermore, a method to query the database for the FolderID of the given folder name is implemented, since it is needed to insert the post. Finally, when the destructor is called, the cursor is closed to avoid any

loose ends.

`Make_reply_ctrl` is a controller that handles insertion of replies to other posts into the database. The connection is passed to the constructor in order for the suitable queries to be made, i.e. to insert replies into the ReplyPost-table. Also, a cursor is instantiated in the constructor. Moreover, an already logged in `User_login_ctrl`-object and a PostID is passed to the constructor. The class contains a method to insert a reply, with the given reply-text from the user, i.e. the author of the reply. Furthermore, a method to give different outputs in the UI based on the user-type, and to simultaneously change the ColorCode of the post, is implemented. Finally, when the destructor is called, the cursor is closed to avoid any loose ends.

`Search_post_ctrl` is a controller that handles searching for posts that contain a given keyword. The connection is passed to the constructor in order for the suitable queries to be made, i.e. to search for rows in Post, Followup, ReplyPost and ReplyFollowup containing the specified keyword. Also, a cursor is instantiated in the constructor. Methods for querying and displaying results from each of the four specified tables are implemented in the class. The method `total_search` takes the keyword as an argument and calls all the other methods in order to search through each table. Finally, when the destructor is called, the cursor is closed to avoid any loose ends.

`Statistics_ctrl` is a controller that handles compilation of statistics for an instructor. The connection is passed to the constructor in order for the suitable queries to be made, i.e. query the necessary tables to compile statistics. Also, a cursor is instantiated in the constructor. Moreover, an already logged in `User_login_ctrl`-object is passed to the constructor, in order to check if the user is in fact an instructor. The method `compile_stats` searches the necessary tables and displays the statistics for the instructor that asked for it. Finally, when the destructor is called, the cursor is closed to avoid any loose ends.

Solution of the Use Cases

An overview of the five use cases that have been solved and how they are realized in our program follows in this section. Every use case is resolved with the `UI_ctrl`-object, which collects the necessary input from the user and delegates it to the other case-specific controllers.

1. For use case one, the `UI_ctrl`-object queries the user for an email and a password, and then instantiates a `User_login_ctrl`-object, which handles the credential-checking, as described in the API reference.

2. For use case two, the user can make a post by typing '1' in the 'main menu' of the UI. The `UI_ctrl`-object then queries the user to input summary, text and a tag, and instantiates a `Make_post_ctrl`-object, which handles the insertion of the post into the database, as described in the API reference.

3. For use case three, the user can reply to a post by typing '2' in the 'main menu'. The `UI_ctrl`-object retrieves a list of available posts in the database and displays them

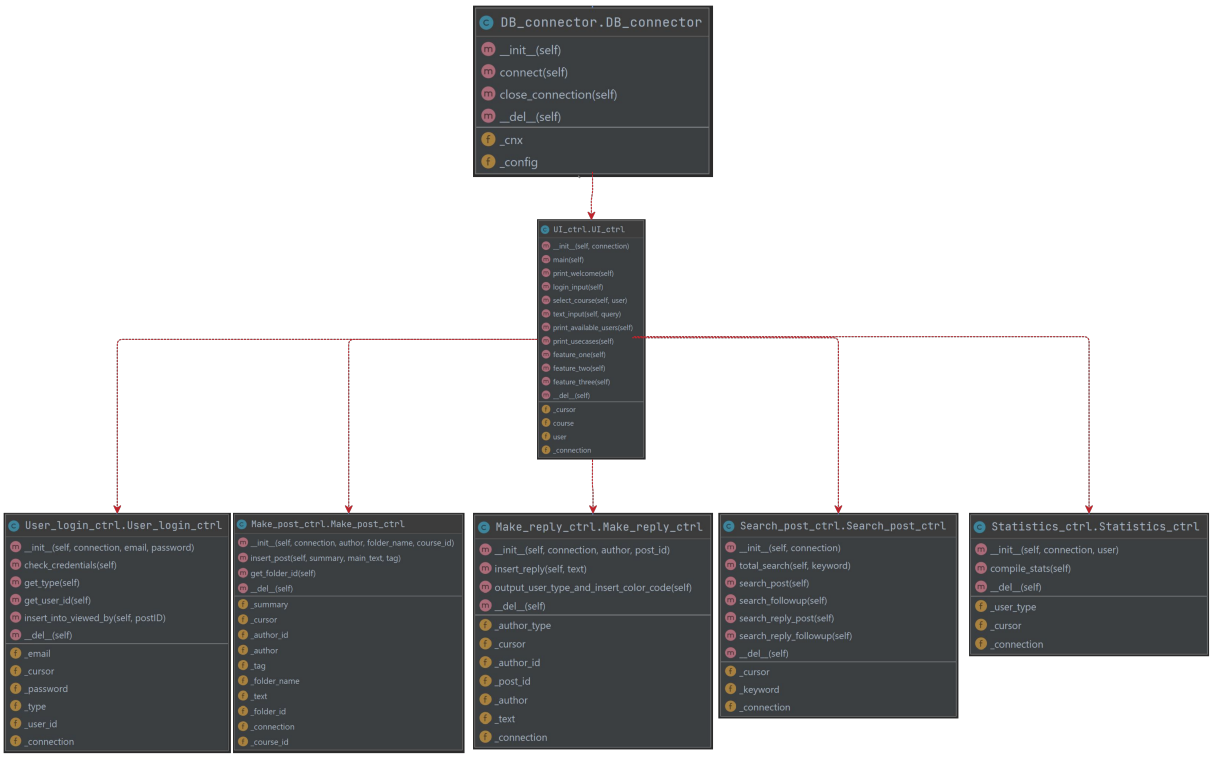

Figure 1: UML diagram showing all seven class controllers with their corresponding name, methods and fields.

to the user, before querying the user for an ID of the post to reply to, as well as the reply-text. The `UI_ctrl`-object then instantiates a `Make_reply_ctrl`-object, which handles the insertion of the reply into the database, as described in the API reference.

4. For use case 4, the user can search for a keyword in all posts, followups and replies by typing '3' in the 'main menu'. In other words, we have made a slightly more general version of the use case than what is specified in the task description. The `UI_ctrl`-object queries the user for a keyword, and then instantiates a `Search_post_ctrl`-object, which handles the searching in the database, as described in the API reference.

5. For use case five, the user can compile and view statistics by typing '4' in the 'main menu'. If the (logged in) user is not an instructor, the statistics query will be rejected. The `UI_ctrl`-object instantiates a `Statistics_ctrl`-object, which handles the queries against the database as described in the API reference. The compiled statistics have a

slightly different format compared to what is specified in the task description. A 'user name' is not displayed, but rather the full name and email of the user is displayed. This is how statistics are displayed in Piazza, which is why we wanted to replicate this.