Assignment 5 – Graphical User Interface (GUI) for a Medical Clinic System

Well, you have passed two different phases of the development of a software system. Both a model is implemented in code as well as a file layer to assure that user operations persist after system use. Now it is time to finish the system development with a graphical user interface (GUI).

The analyst got in touch with you again to discuss this next phase. Understanding that you do not have so much experience with GUI development, a prototype command-line interface (CLI) was developed in parallel to your second phase. Now you can connect the CLI with your code and see a prototype system working. The analyst explains that a prototype may be useful for both better understanding the users' point of view and for learning how a user interface interacts with a controller to manipulate a model.

The basics of a Model-View-Controller (MVC) design pattern is that your user interface (view) is decoupled from the model. The controller acts as an intermediate/proxy to allow the view to change the model. After changing the model, the view may be updated to reflect the new stated of the model. Decoupling the view from the model also means that you may replace one view by a different view without much effort. In your case, you may replace the prototype CLI by a real GUI meant for the end user.

The suggestion is that you handle this project phase in two different steps. First, you will experiment with the CLI to get used with how a user interface interacts with a controller object. Then you will develop the GUI for the medical clinic system.

Programming environment

For this assignment, please ensure your work executes correctly on Python 3.9.19 (you can use later Python versions, but recall that you cannot use any new features that depend on those later versions). You are welcome to use your own laptops and desktops for much of your programming as well as your favorite IDE. For coding the GUI, you will need the **`PyQt6`** framework. You will need to install it first with **`pip`**. If you do not have **`pip`** installed, find about it on the web and install it for your computer configuration. To install the **`PyQt6`** framework, you will run:

```
% pip install PyQt6
```

Start your Assignment 5 by copying the files provided on Brightspace (in Assignment 4) into your **`a5`** directory inside the working copy of your local repository (or, if you are working in pairs, inside the working copy of your group's local repository). Do not put your **`clinic`** and **`tests`** subdirectories in subdirectories of **`a5`**, put them in **`a5`** itself (this is important for grading).

Then add the files you developed in Assignment 4 into the correct places: **`controller.py`**, **`patient.py`**, **`patient_record.py`**, and **`note.py`** inside the **`clinic`** subdirectory, **`note_dao_pickle.py`**, **`patient_dao_json.py`**, **`patient_encoder.py`**, and **`patient_decoder.py`** inside the **`clinic/dao`** subdirectory, and **`patient_test.py`**, **`patient_record_test.py`**, and **`note_test.py`** inside the **`tests`** subdirectory.

In general, you will NOT need to run automated tests in this assignment. We suppose all the integration tests are passing as well as your unit tests. With GUI programming, you will mostly run manual tests, and check whether your application is correctly retrieving and updating data after calling controller methods. Recall that your integration tests delete your data files (patients and records) after running. In case you really need to run integration tests, first backup your data files so you do not need to retype data for your manual tests.

Commit your code frequently (**`git add`**, **`git commit`** and **`git push`**), so you do not lose your work. We will look at the code commits you did in this assignment. **We require at least four different commits** (you should split your work into parts, and if you are working in pairs, we expect that both group members commit code, at least two commits for each member). The final grading will take that into account. We recommend that you develop your changes in a separate new branch and only merge the changes to the master branch when you are confident your tests are passing. Do not forget to **`git push`** into your remote repo so that your new code is visible by the other stakeholders (e.g., collaborator member, grader).

Description of the task

As suggested by the systems analyst, handle the task in two different steps: 1) First experiment with a command-line interface to get used with how a user interface interacts with a controller object; 2) Then develop the graphical user interface (GUI) for your application.

1. Experimenting with a Command-Line Interface (CLI)

To start your work, get used to interacting with your controller by means of a CLI. We provided the CLI for you ready to use. It is located in the **`clinic/cli`** subdirectory. To run it, open a terminal inside your **`a5`** directory and type:

```
% python3 -m clinic cli
```

That will run the **`__main__.py`** file that is located inside the **`clinic`** directory, which will instantiate a **`ClinicCLI`** object and run the CLI application.

You may first interact with the CLI, log in the system, manipulate some patients, create an appointment with a patient to allow you to access the patient record and manipulate some notes, and log out of the system. Take a look at the files that are created or saved when you manipulate patients and notes.

Then, you may examine the CLI files inside the **`clinic/cli`** subdirectory to better understand how a CLI interacts with a controller object. Of course, your GUI application will be different since there will be no console interaction. But the process to invoke the controller, retrieve model data and the way to handle user errors (with either **`if`** or **`try-except`** blocks) will be a bit similar.

2. Developing a Graphical User Interface (GUI)

Now you will start to develop your GUI for the medical clinic system. The systems analyst did not put many constraints on how you will organize your GUI. But there are still some constraints. *First, you will have to run your application from the* ***`__main__.py`*** *file inside the* ***`clinic`*** *directory*. To run your GUI, open a terminal inside your **`a5`** directory and type:

```
% python3 -m clinic gui
```

That will allow you to instantiate a **`ClinicGUI`** object and run the GUI application. The **`ClinicGUI`** class is in the **`clinic_gui.py`** file, which is inside the **`clinic/gui`** subdirectory. And that is the second constraint on your application: *your main window has to be the* ***`ClinicGUI`*** *class*. Change the stub that was given and write your application code starting from that class. Of course, you may create other GUI files with windows, tabs or widgets.

Then, the third constraint is that *all your GUI classes and files must be inside the* ***`clinic/gui`*** *subdirectory.*

How you organize your GUI is mostly up to you. Do you want to use a single window with a stacked layout? Fine. Do you prefer to use multiple windows? Fine as well. As long as all the user stories are implemented correctly in the GUI and your data remains consistent after system use, fine. It is important to make your GUI usable, since usability is an important quality of any software system.

Finally, there are two other constraints that the analyst demanded: *The* ***list patients*** *and* ***retrieve patients****' results must be displayed in a* ***`QTableView`*** *widget. And the* ***list notes*** *and* ***retrieve notes****' results for a given patient must be displayed in a* ***`QPlainTextEdit`***.

Implementing your program

If you are working in pairs, you can either work together doing pair programming (even online pair programming is a good idea, as one of the developers may share their IDE screen and the pair may chat over an audio channel).

To check that you are advancing correctly, run the GUI application and test the user stories. Start from the simplest user stories: first login and logout. Then handle patients. Finally, handle notes.

You should commit your code whenever you finish some functional part of it. We recommend that you develop your changes in a separate new branch and only merge the changes to the master branch when you are confident your tests are passing. That helps you keep track of your changes and return to previous snapshots in case you regret an unfortunate change. When you are confident that your code is working correctly, merge it to the master, and push your local master commits to the remote repo. You could do this various times during the development of your project.

What you must submit

- You must submit a solution to this assignment containing at all your Python GUI source files below within your individual **`git`** repository (and within its **`a5`** subdirectory) if you are working individually or within your group's **`git`** repository (and within its **`a5`** subdirectory) if you are working in pairs.
  - Inside **`clinic`**, you will submit **`controller.py`**, **`patient.py`**, **`patient_record.py`**, and **`note.py`**;
  - Inside **`clinic/dao`**, you will submit **`patient_encoder.py`**, **`patient_decoder.py`**, **`patient_dao_json.py`**, and **`note_dao_pickle.py`**;
  - Inside **`clinic/gui`**, you will submit **`clinic_gui.py`** and all the other **`.py`** files with GUI code that are needed to fully run your application.
- Ensure your work is committed to your local repository and pushed to the remote before the due date/time. (You may keep extra files used during development within the repository, there is no problem in doing that. But notice that the graders will only analyze the GUI files and, if needed, the other files described above.)
- Do not forget to submit on Brightspace your project's final commit hash and the repository name (either your NetLinkID if you are working individually or your group ID **`groupXXX`**, where XXX is your group number, if you are working in pairs).

Evaluation

Our grading scheme is relatively simple and is out of 100 points. Assignment 5 grading rubric is split into five parts.

1. User stories implemented as a GUI - 70 points, being 5 points for each user story.
   1.1. Log in;
   1.2. Log out;
   1.3. Search patient;
   1.4. Create new patient;
   1.5. Retrieve existing patients (in a **`QTableView`** widget);
   1.6. Update existing patient;
   1.7. Delete existing patient;
   1.8. List all patients (in a **`QTableView`** widget);
   1.9. Choose current patient;
   1.10. Create new note;
   1.11. Retrieve existing notes (in a **`QPlainTextEdit`** widget);
   1.12. Update existing note;
   1.13. Delete existing note;
   1.14. List the full patient record (in a **`QPlainTextEdit`** widget).
2. Modularization - 10 points - the code should have appropriate modularization, i.e., your GUI code is also split into different classes and files, and the split is made of cohesive and decoupled code;

3. Documentation - 6 points – on documentation, we will only additionally document the new classes inside **`clinic/gui`** since the other classes have already been documented. We will check for code comments (enough comments explaining the hardest parts, no need to comment each line), function comments (explain function purpose), and adequate indentation;
4. Version control - 4 points – Appropriate committing practices will be evaluated, i.e., your work cannot be pushed in just one single commit, its evolution will have to happen gradually; at least four commits are expected (if working in pairs, each collaborator must have authored at least two commits);
5. Usability - 10 points – Good usability is expected from the point of view of the end user, i.e., the end user finds it easy to learn to use the system as well as to use it in the long run after learned.

We will only assess your final submission sent up to the due date (previous submissions will be ignored). On the other hand, late submissions after the due date will not be assessed.