CO559 A3 Reflections on A1/A2 - cssb2

Reflections on Assignment One

For this project, my group and I were assigned the task of implementing the functionality of a ‘Doctor’ in this system. Assignment one started with the group dividing up the eight user stories between the four of us. After a weekend to deliberate our user stories we gathered in a group Discord call to go through the conversations before they needed to be presented. During this call as a group we went through each user story and created a conversation centred on that story.

My individual contributions to other members of the group at this stage consisted of assisting the other members create their conversations suggesting potential questions and answers which could form the entire conversation, working through a shared Google Docs file over the space of a few hours the conversations were completed. At this stage of the conversations were completed and all that was left was to copy and paste all the conversations into a powerpoint ready to be presented.

Reflections on Sprint One

The first sprint consisted of the group implementing one authorisation user story as well as two other user stories that we had assigned priorities to. At this stage none of the additional deliverable documents had been assigned to any member of the group. As a group we went through and created the effort estimation and prioritisation documents all members giving input on what user stories should be implemented first and which ones can be left to a later stage. The same was done for the group organisational plan and it was at this stage where individual contributions were decided.

On the documentation side of things, I was assigned the ‘plan for version control’ document, in which I detailed how as a group we utilised Git, how often changes were going to be pushed to the repository and how the group was going to take advantage of branching during development. In the end the plan that was decided was to branch from a singular master branch for every user story implementation and from those branches, once more for testing of those features. Then after all was completed, merge on the last scrum meeting before the deadline to have a project to upload. Upon reflection of completing the document I could have researched further into other features of Git that we could have used, such as their ‘Issue Boards’ or added our requirements to the repository so we could have tracked each commit to exactly what user story it was satisfying.

A contribution that I made, which required very little effort to complete but played a pivotal role in organisation of files, was the creation of a shared Google Drive folder where all the documentation was to be stored and separated for each sprint. This made collaborations much easier to do and was better than sharing files on Discord as we had been doing before.

As well as the one page version control document, it was decided that I would implement all the jUnit tests for this sprint’s user stories as well as create test logs for each test. Throughout the project this ended up being the case for nearly all user stories. The methods that I had to test this sprint were ‘login’, ‘getNewMessages’ and ‘viewBookings’. Part of creating the test class involved in making test values that could be entered into the database

and be manipulated during the tests. The methods that had been created at this point used the ‘Doctors’, ‘Messages’, ‘Bookings’ and ‘Patient’ tables, in the ‘setUp’ and ‘tearDown’ methods of the test class values are inserted into the database before each test and removed after each test has passed.

Testing the ‘login’ method was simple, it involved passing through one of the usernames corresponding passwords from the values I entered into the database, and then asserting true the result of the method. The tests for ‘login’ were taken with correct data, partially correct data as well as incorrect data to fully test the system. For ‘viewBookings’ and ‘getNewMessages’ more work was required. Both of those methods returned lists containing messages or bookings in object form, to perform the tests I had to loop through the lists extracting the content of the objects that I needed to compare against strings which were initialised at the start of every test in the test body. Similar to ‘login’ I used data that was present in the database, as well as incorrect data and ensuring nothing gets returned if no user is logged into the system.

The last part of my contribution to the first sprint, was the creation of the test log document.In this document I listed all the test data that was used in the tests, and all the information gathered from testing. The logs of the testing were stored in a table, one row for each test stating the purpose of the test, expected and actual result, whether it passed or not and any rectifications that needed to be made to have a successful test. Any failed tasks were repeated and shown to pass after. Reflecting on the test logs, it might have been more beneficial to add more information about every test, perhaps showing what test values were used in every test, but due to the nature of how they were displayed in the table it would have been unfeasible to display large amounts of information like that.

Reflections on Sprint Two

For the second sprint, my contributions were similar to the first sprint, I was assigned all testing for the user stories that gave the methods ‘getOwnPatinets’ and ‘enterDetails’. As well as this, due to a group member not understanding the user story that relates to ‘enterDetails’ I took it upon myself to rewrite that method to fully implement all the functionality as we were pressed for time, in addition to doing the regular testing of that method. However, before writing the tests for ‘getOwnPatinets’ I had to fix an issue with the method which resulted in no data being retrieved from the database. After inspecting the method I found the solution was to add a space in the SQL statement as there was one missing which caused half of the SQL statement executed to appear all as one word instead of multiple words.

Writing the tests for the method ‘getOwnPatinets’ was simple, similar to the previous methods, a list is returned with all the patients assigned to a specific doctor. To test the functionality performed as expected, I went through the list using the accessor methods, made by the group member who wrote the method, and built strings containing all the fields that were returned from the database, which were the ‘patinetID’, ‘doctorName’, ‘patientName’, ‘phoneNumber’, ‘dateOfBirth’ and ‘lastBooking’. This string was then compared to an expected string which was initialized in the test body. Similar to the other tests a variety of test data was used to fully ensure the method worked as intended, including calling the method when no doctor was logged in to make sure nothing was

returned, similarly ensuring that no patient not registered to the doctor was added to the list of patients.

To correct the ‘enterDetails’ method, I first looked at the user story which the method had to satisfy, the user story wanted the user to be able to edit already existing visit and prescription information about an existing booking then send confirmation messages to both the user and patient whose information had been updated. All I had to do in this case was update the ‘Bookings’ table with the new values that get passed as arguments to the method. A simple update SQL query was used to complete this. To send the confirmation messages, for the doctor a new message was entered to the ‘Messages’ table using a SQL statement and the new prescription and visit details inserted into the message portion, for the patient a string returned which contains the new information that has been updated.

Testing ‘enterDetails’ followed the same format as the testing done before, however for this method there were more aspects than others. Three expected strings were initialised, one for the amended booking, one for the doctors confirmation message and finally one for the patients confirmation message. To build the actual strings for the amended booking and messages, I looped through the lists returned after calling ‘getNewMessages’ and ‘getBookings’ and built the strings similar to how I did in the tests for those methods. The final actual string is simply returned from the method call which was assigned to its own string. Finally all the strings were compared to see if the test was successful. As with every other set of tests, variations of the test data were used to test all possibilities of use cases, such as making sure the right booking is amended if a doctor has more than one booking with the same patient, and ensuring nothing gets changed in the database if no user is logged into the system.

Similar to the first sprint, to log the tests, a document was made which contained new test data which I used for the ‘Patient’ table. In the same format as before a table was used to show the information for every test, showing the purpose of the test, expected and actual results, whether the test passed and any rectifications that would have to be made to make the test successful.

Reflections on Sprint Three

For the third and final sprint, my contributions to the project varied. I was assigned to implement the final authentication user story regarding logging all aspects of using the system, and naturally complete the testing that came with that. However in this sprint a few issues appeared across the project which I volunteered to sort out.

To implement the authentication user story first I had to create a table into the database to store the logs of each action that the user may take. In the table it had the fields, ‘logID’, ‘doctor_name’, ‘action’ and ‘time’ all of these fields used to log an action within the system. Once the database was in, it was simply a case of writing a method that once called takes the ID of the current doctor logged into the system and the action they have just taken, such as login or retrieved messages, and enter those details in the database. Three queries are used in the method, the first to see if there is any entry already in the table, if there is no entry then it starts the log at ‘logID 1’ if there is then the second query retrieves the ID of the last entered record, which is then incremented and added into the database using the final

query. I then placed a call to this method in each method in the project, so that every aspect of the system is logged, satisfying the requirements. Testing of this method brought to my attention that before each test is run the entire log table in the database must be cleared before anything is added. This was due to the fact that there was no way to know at the time of each test, what ID is next in the database when building the expected string, without first pulling from the database which defeats the whole purpose of the test, as in that case the expected string would be built using the value from the database instead of a value that I have assigned to it. Clearing the database before each test ensures that every test log starts at 1 and the IDs can be predicted from there based on how many actions have been taken. As with every other testing phase I filled out the test logs document for this method, testing all uses of the system were logged correctly.

Another involvement I had in this sprint was assisting another group member with their user story. The user story which dictated the ability for a doctor to assign a patient to themselves did not behave as expected, after looking at the code I noticed in the SQL statements to modify the database some punctuation was missing when concatenating strings, resulting in the query executed on the database being incorrect SQL. After fixing this, another fault in the method was discovered, the method updated the ‘Assign’ table in the database which has every doctor and patient stored, however this table was never used outside of this method, and for other methods where patients are retrieved the ‘Patient’ table is accessed. The solution was simple, which was to change the SQL statement to update the ‘Patient’ table rather than the ‘Assign’ table, and whilst another group member altered the code I removed the ‘Assign’ table from the database as it was no longer used at all. After fixing the method, I then completed the testing for this method. The member who was meant to do the testing missed out the assertTrue or assertEquals comparisons in the test methods by mistake, so I added the comparisons in to ensure the test would compare something and not pass by default. I also filled out the test logs for this method. As the database had changed during this sprint, I updated the database design document to add the ‘Logs’ table and all the SQL statements used to create the table, as well as remove the ‘Assign’ table from the document as it was no longer in the database.

My final contribution was going through the backend portion of the project and trimming lines of code so they all were under a certain amount of characters as well as a general clean up of code, removing unnecessary lines of code, adding comments and ensuring indentation was correct.

Reflections on Group Work

Reflecting on the project management practices used by the group showed that there were several good and bad practices that we used during this development. During the week we had two scrum meetings on Wednesday and Friday and an additional one on Tuesday during our classes, this was both good and bad practice as we left enough time between meetings from Wednesday to Friday so that work could be completed, however the gap between Tuesday and Wednesday often meant that not a lot of work had been completed since the last meeting.

Our communication was effective, we used Discord for all communications which meant we had a text channel to speak about issues we were having as well as the audio channel

where we could share our screens which came in useful during debugging of certain errors. As everyone was mostly always online on Discord, getting a group member's attention or assistance on a topic did not require much effort, usually a message was sent to ask a certain member to join the audio channel and from there conversation started. In the text chat we also pinned organisational messages so everyone knew what each group member was responsible for.

However during the second sprint, we started creating more branches than we needed, which cluttered up the repository, and caused continuity errors in our IntelliJ projects resulting in having to clone the entire project again. This was rectified in the third sprint by removing all the excess branches and sticking to one branch per user story.

Reflections on Individual Performance

Looking at my individual performance and contributions to the project there are certainly areas where I have not contributed at all or much. For example during the first sprint, the creation of the Git and database were solely done by other members of the group, while I worked on documentation and testing.

Throughout the entire project all of the front end GUI aspects of the system were handled by a single member of the group, meaning I had no input on the GUI. This is something that if the project was to be redone, I would change as there was a lot of work to do for the GUI for one person, as well as being able to develop my own skills working on the frontend side of things rather than the backend.

Future Improvements

If the project was to continue, most of the future improvements that I would have wanted to implement would centre around the GUI, making it appear more refined, mimicking a real doctors system rather than what it looks like now. All of the backend functionality has been implemented as we have wanted, however the GUI could have been improved.

In addition to this, if the project was to go on, I would have wanted to figure out a way to correctly increment the logID in the ‘Logs’ table using the format ‘Lx’ with ‘x’ being the current ID. At first all the IDs followed that format however when incrementing the ID it would increment into double digits when converting from a string to an int then back to a string. I would have wanted to change this to fit in form with all other tables as their primary key IDs share the format of a letter and number, for example ‘D001’ for the ‘Doctors’ table and ‘P001’ for the ‘Patient’ table.

Another aspect that I would want to improve if the project continued would be to refine the test methods, some tests which have multiple loops to search through lists or have to make contact with the database take a few seconds to complete, in future revisions I would have wanted to reduce the compute time of each test.

Finally I would have wanted to alter the test log document as mentioned earlier in the sprint reflections to show more information about each test, stating what values have been used for each test as well as more details about the expected and actual results of the test.