SENG 300 FINAL REPORT - L01 GROUP 24

Erin Paslawski - edpaslaw@ucalgary.ca - T02 - TA: Pavan Kumar
Sydney Kwok - sydney.kwok@ucalgary.ca - T01 - TA: Colin Yeung
Arthur Franca - arthur.franca1@ucalgary.ca - T01 - TA: Colin Yeung
Shavonne Tran - shavonne.tran@ucalgary.ca - T01 - TA: Colin Yeung
Gerard Gabriel Dizon - gerardgabriel.dizon@ucalgary.ca - T01 - TA: Colin Yeung

Submission Date: 04/14/2020

Contents

1 Iteration 1: Product Backlog

The following stories have their priorities ranked in a scale of 1 to 10, where the 10 is the highest priority. These rankings are given in relation to the present state of development, and will be updated as more work are carried out.

- As a user, I want to create an account in the system, so that I can log in whenever I need to. (10)
- As a user, I want to log into my account, so that I can have access to what I am allowed to see and do. (10)
- As a patient, I want to view the availability of a doctor, so that I can request an appointment with them. (3)
- As a patient, I want to be able to request an appointment with a doctor, so that I can arrange consultations. (2)
- As a patient I want to view the information of my appointments and tests, so that I can know what and when they are. (2)
- As a patient, I want to cancel my appointment for tests and consultations, so I can adjust to a change in my availability. (1)
- As a patient, I want to view my test results, so that I can know my medical situation. (1)
- As a patient, I want to see my medical records and personal information that the hospital has on file, so that I can see if they are correct and up to date. (5)
- As a patient, I want to update my personal information, so that I can modify it if anything changes. (5)
- As a staff member, I want to see which nurses are working in which department, and with which doctor. (6)
- As a doctor, I want to cancel my appointments with patients, so that they do not come when I am unavailable. (7)
- As a doctor, I want to set my availability, so that my assistant or a receptionist can book me appointments. (7)

- As a doctor, I want to change my schedule, so that no one books appointments when I'm unavailable. (7)
- As a doctor, I want to view a patient's medical records, so I can know their situation when treating them. (4)
- As a doctor, I want to edit a patient's medical records, so that I can correct mistakes and note developments. (2)
- As a doctor, I want to view a patient's requests for tests, so that I can know if they have done what I prescribed, or if they requested the right test. (2)
- As a doctor, I want to refer my patients to tests, so that they're able to take the tests they need. (3)
- As a doctor, I want to be able to assign myself as a patient's physician, that way I can follow up on their situation. (8)
- As a doctor, I want to see how many patients I have, so that I know if I can take another one. (8)
- As a nurse, I want to view which doctor I am scheduled with, so that I know who I will be working with. (5)
- As a nurse, I want to view the department I will be working in, so that I know the location I am scheduled to work in. (5)
- As a nurse, I want to upload patient test results to their medical records, so that their information is up to date. (1)
- As an administrative staff, I want to see how many patients the hospital has had in the prior years, so that I better understand the necessities of the facilities. (8)
- As an administrative staff, I want to place doctors under different departments, that way I can organize shifts around departments. (9)

2 Iteration 1: Sprint Planning Meeting

Our sprint planning meeting took place on March 2 at Gallagher Library. The team members present were: Erin, Sydney, Arthur, and Shavonne. The duration of this meeting was roughly 1.5 hours.

Team Goals For This Sprint

- Be able to register for an account of a given classification into the system. The classifications being the various user roles: patient, doctor, admin, nurse, receptionist, assistant.
- Build a web-hosted application that displays a log in screen for users to log into their existing accounts.
- Upon logging in, be displayed a unique page that presents a message according to the account's classification.

2.1 Iteration 1: Sprint Backlog

Below is a list of all the user stories that we plan to implement in this iteration.

- As a user, I want to be able to create an account in the system, so that I can log in at a later time.
- As a user, I want to be able to log into my account, so that I can have access to whatever I am allowed to see and do.

2.2 Iteration 1: Estimated Tasklist

- Research web hosting and creation of web applications so that we can make an informed decision on what kind of technologies we want to use in creating our project. (Arthur and Erin)
- Code a registration page on a web-application. (Sydney)
- Be able to successfully log in an account on said page. (Sydney)
- Draft ideas for how we want the GUI to look like, what we want it to include, as well as what error messages will be required. (Arthur)

- Document our progress and goals. (Shavonne)
- Figure out how we want to implement the system database to store information for users (Arthur)
- Figure out how to connect the database with the front-end. (Sydney and Erin)

3 Iteration 1: Daily Scrum Meeting

Our first scrum meeting took place on March 2 at Gallagher Library. Takeaways from this meeting:

- Erin is going to research how to build a persistent DB/Data Store that will be used to store the login credentials.
- Erin will create a new discord server.
- Gerard is going to proof-read the iteration document.
- Sydney is going to upload basic login page and research basic applet development.
- Arthur is going to design the interface panels that are to be implemented, and plan the code structure.
- Shavonne is going to plan for the rest of the sprint and prioritize user stories.

Our second scrum meeting took place in class, March 3. Takeaways from this meeting:

- Erin discussed with others about the database implementation. She had issues with deciding where the database will be hosted, locally or otherwise. She will inquire the instructor about it.
- Sydney worked on trying to get a login applet working so that we could export it as a JAR file and integrate it into our webpage with the HTML applet tag. However, she had issues when she found out that the HTML applet tag has been deprecated, and is no longer supported on many browsers. She will continue to work on it, and consider alternatives.
- Arthur designed several panels to be implemented with windows builder, and brainstormed the necessary classes for this project. He had issues with the representation of dates, given the inconsistency of dates-weekdays relationships in the Gregorian Calendar, as well as differing length of months. He will inquire instructor when possible, and define which JSON files our database will consist of, and how they are structured.
- Shavonne went over the user stories produced by the group, and highlighted the ones that should be implemented in the coming days. She's had no troubles since the last meeting. She will also research how to implement our database.

Our third scrum meeting took place after class, March 5 in the Science Collaborative Space. Takeaways from this meeting:

- Erin built two Java classes that create JSON files and write basic new user info to that new JSON file.
- Sydney implemented JSPs and servlets into the webpage so that information from user login/registration requests could be passed from the HTML front-end to the Java back-end. During the meeting she worked with Erin to integrate the front and back ends together. However, a few problems arose with the classpath and writing to the JSON. They will continue to work on resolving these issues.
- Arthur continued working on panels, obtained information with regards to the Calendar issue, and designed the model our JSON files will take.
- Shavonne read up on what JSON files were and organized the user stories based on importance of implementation.

4 Iteration 1: Sprint Review Meeting

We had our Sprint Review Meeting on March 5, in the Science Collaborative space. The duration of this meeting was approximately 1.5 hours.

We found that a web-application is not strictly necessary, but still worth trying. At the moment, even if we decide to switch to a desktop application, most of our work can be converted without changes.

We've concluded that JSON will be used for a system database that doubles as a backup.

We've also decided that we will use Servlets and Java Server Pages (JSPs) to generate our web application, in the case that we do choose to continue on with the web application implementation of the project.

As of now, our application can successfully take us to a login and registration page when launched. Upon logging in, we are able to select from a list of different users such as patient, doctor and etc. It currently accepts any login credentials and in doing so, we are taken to another page that displays a message with the user's username. We are also able to register for an account successfully but the authentication only checks if the two passwords match as of now. In short, our product can properly run and display the basic features of the application.

We also updated our product backlog based on which user stories were most important to implement for each actor involved.

5 Iteration 1: Retrospective Meeting

- We found that we should continue having scrum meetings with frequency, that way we're always aware of how much progress has been done, and what needs to be done. At the present no conduct was deemed unsuccessful.
- We should also continue working on the project frequently so that we do not fall behind and ensure that everything gets done in time.
- We have also realized that we soon must decide whether to build a desktop application or a web-based software. While we can re-use our current work, after the next iteration it will be time-consuming to switch over to the other, if necessary.
- We have realized that we often decide last minute when and where to meet, this often results in wasting time looking for a place where we can set ourselves up. What we aspire to do, in the coming weeks, is better planning our meetings. We should select at least one day in the week where everyone can meet up to work on the project together.
- We need to make sure to create branches when working on implementing new features, and not push directly to master until the branch has been tested and the new changes are integrated properly.

6 Iteration 2: Updated Product Backlog

The following stories have their priorities ranked in a scale of 1 to 10, where the 10 is the highest priority. These rankings are given in relation to the present state of development, and will be updated as more work are carried out.

- As a user, I want to create an account in the system, so that I can log in whenever I need to. (Completed)
- As a user, I want to log into my account, so that I can have access to what I am allowed to see and do. (Completed)
- As a patient, I want to view the availability of a doctor, so that I can request an appointment with them. (6)
- As a patient, I want to be able to request an appointment with a doctor, so that I can arrange consultations. (8)
- As a patient I want to view the information of my appointments and tests, so that I can know what and when they are. (9)
- As a patient, I want to cancel my appointment for tests and consultations, so I can adjust to a change in my availability. (5)
- As a patient, I want to view my test results, so that I can know my medical situation. (1)
- As a patient, I want to see my medical records and personal information that the hospital has on file, so that I can see if they are correct and up to date. (2)
- As a patient, I want to update my personal information, so that I can modify it if anything changes. (2)
- As a staff member, I want to see which nurses are working in which department, and with which doctor. (5)
- As a doctor, I want to cancel my appointments with patients, so that they do not come when I am unavailable. (10)
- As a doctor, I want to set my availability, so that my assistant or a receptionist can book me appointments. (7)

- As a doctor, I want to change my schedule, so that no one books appointments when I'm unavailable. (6)
- As a doctor, I want to view a patient's medical records, so I can know their situation when treating them. (3)
- As a doctor, I want to edit a patient's medical records, so that I can correct mistakes and note developments. (2)
- As a doctor, I want to view a patient's requests for tests, so that I can know if they have done what I prescribed, or if they requested the right test. (2)
- As a doctor, I want to refer my patients to tests, so that they're able to take the tests they need. (5)
- As a doctor, I want to be able to assign myself as a patient's physician, that way I can follow up on their situation. (9)
- As a doctor, I want to see how many patients I have, so that I know if I can take another one. (9)
- As a nurse, I want to view which doctor I am scheduled with, so that I know who I will be working with. (5)
- As a nurse, I want to view the department I will be working in, so that I know the location I am scheduled to work in. (4)
- As a nurse, I want to upload patient test results to their medical records, so that their information is up to date. (1)
- As an administrative staff, I want to see how many patients the hospital has had in the prior years, so that I better understand the necessities of the facilities. (8)
- As an administrative staff, I want to place doctors under different departments, that way I can organize shifts around departments. (9)

7 Iteration 2: Sprint Planning Meeting

Our second sprint planning meeting took place on March 9th in the Math Sciences building. The team members present were: Sydney, Arthur, Shavonne and Gerard. Erin had work at this time so she gave her input via Discord. The duration of this meeting was roughly 45 minutes.

Team Goals For This Sprint

- Make the login and registration more realistic. This includes making certain registration fields mandatory, and actually storing usernames and passwords from registration into JSON files that we check when a user logs in.
- When a user logs in, we want to display a screen that is customized to their account type. The various user roles include: patient, doctor, admin, nurse, receptionist, assistant. The goal for this sprint is work on the perspective of a doctor and patient.
- Implement the booking of appointments. To be completed in full, it would be necessary to implement a doctor's availability, and a form to check if a time-slot is taken, and a form to sort these by date and time. These particularities, for the the time being, are considered secondary.
- Implement certain components of the main menu in a doctor's perspective. That is, we want to implement the ability of a doctor to see his appointments, see his patients, and add a new patient from the system as a personal patient. With these things implemented, it would be significantly quicker to implement a patient's perspective. If no major setback appears, we would like to implement the patient's perspective in full, excluding the test-related stories, and without intricacies that would require another user type, such as receptionist.

7.1 Iteration 2: Sprint Backlog

Below is a list of all the user stories that we plan to implement in this iteration.

- As a doctor, I want to be able to assign myself as a patient's physician, that way I can follow up on their situation.

- As a doctor, I want to see how many patients I have, so that I know if I can take another one.
- As a patient, I want to be able to request an appointment with a doctor, so that I can arrange consultations.
- As a patient I want to view the information of my appointments and tests, so that I can know what and when they are.

7.2 Iteration 2: Estimated Tasklist

- JSON structure - Erin and Arthur
  - Appointments (in general)
  - User accounts (all types) with user data, appt id's etc.
  - Departments- doctors, nurses, appt id's
- WindowsBuilder Panels
  - Log in - Shavonne.
  - Failed log in - Shavonne.
  - New registration - Shavonne.
  - Doctor Panels (Implement a few of the doctor functionalities): - Arthur, Gerard
    - Viewing patients of a specific doctor
    - Doctor booking an appointment for a patient
    - Viewing appointments of a specific doctor
- Event handlers for the panels - Sydney, Gerard
  - Login button + show new page with that user type OR show failed login page.
  - Registration button -> registration page. Add new user function.
  - View patients button handler -> page with list of patients for that individual doctor.

8 Iteration 2: Daily Scrum Meeting

Our first scrum meeting took place on March 13 remotely with all group members present. This meeting was conducted before any work was carried out, hence the absence of difficulties encountered. Takeaways from this meeting:

- Erin is going to build the JSON's and their structure.
- Gerard is going to implement event handlers for the panels.
- Sydney is going to add functionality to the panels by implementing event handlers.
- Arthur is going to create the WindowsBuilder panels for the doctors perspective.
- Shavonne is going to create the WindowsBuilder panels for the login and registration page.

Our second scrum meeting took place on March 17 remotely with all group members present. Takeaways from this meeting:

- Erin created the JSON files for storing back end data for accounts, departments, and appointments. Erin will continue to work on integrating the login page and new registration page and event handlers with the back end JSON files. She will also continue to update the JSON files.
- Gerard worked with Sydney to implement event handlers to go from one pane to another using the appropriate buttons. A specialized welcome page is shown according to the user type used when logging in. Gerard will continue the work on the perspectives for the other user types.
- Sydney worked with Gerard to implement connection between frames and add functionality to buttons. The program now recognizes different user types based off of what the user chooses when logging in, and will show a basic welcome page of the user's type. Sydney will continue to work on developing all of the different user views and will try to implement writing to the JSON for the book appointment functionality if we get to it for this iteration.

- Arthur created the panels successfully and relayed them for teammates to implement their code. He will now work on integrating JSON with the viewing of appointments, as well as the panel's event handlers and functionalities.
- Shavonne added in some Javadoc comments to the files and additional buttons for the patient perspective panel that Sydney created. The buttons have no event handlers yet. She will start adding some basic functionality to the other user panels.

9 Iteration 2: Sprint Review Meeting

We had our Sprint Review Meeting on March 21 over discord. The duration of this meeting was approximately 1.5 hours. All team members were present.

We successfully made the switch from web-hosted application to a desktop application. We began creating the panels and functionalities of the system through the use of windows builder in Eclipse. There have been a minimal amount of bugs with compatibility, but most machines we tested our system in function properly.

The only difficulties we found were with integrating JSON with Eclipse. It was not so much the syntax, as it was getting the software to recognize the imported utilities. This in turn delayed the implementation of more sophisticated features we intended to include in this iteration.

At the moment, no changes to were made to our development approach, or the functionalities required. Therefore we only updated the priority of stories in accordance to the state of development.

10 Iteration 2: Retrospective Meeting

Our retrospective meeting took place remotely on March 23. The duration of this meeting was approximately 20 minutes. All team members were present.

- We found that our communication has decreased since the university closure, which did affect the degree of cohesion of our work. This is something we've all agreed to rectify by paying a closer eye to our group chat and making the effort to have weekly conferences.
- We maintain that we continue working on the project frequently, for we have learned that seemingly easy tasks can be bug-riddled or time-consuming.
- We stand by the planning that was made. The approach we have to the system seems to appropriate, and much of the work we produce may be re-used in the implementation of other functionalities.
- We will be more careful with working on our own branches and only pushing to our main branch when the changes are discussed with the group.

11 Test Suites

To test our GUI, we did manual testing using test cases. We manually checked the GUI panels according to the requirements provided by the client. The following is a list of some of the test cases we used for manual testing:

1. Verify that the login page has buttons for the user to log in and create and account.

   Purpose: Check that the user can choose between logging into their account or creating a new account.

   Setup: Login page has textfields for the user to enter their login credentials and buttons to login or register for an account.

   Action: Click either the "Login" or "Register" button.

   Expected result: The GUI should display a button with the label "Login" and another button labeled "Register".

2. Verify that the user can select which user type they want to log into as.

   Purpose: Check that the user is able to select their user type so they can have access to the appropriate information.

   Setup: The login page is displayed with a drop down menu of user types.

   Action: Select a user type from a drop down menu.

   Expected Result: The user type that the user selects should be chosen.

3. Verify that the user is navigated to the account registration page after clicking the "Register" button from the login page.

   Purpose: Check that the user is able to get to the page where they can create new accounts.

   Setup: The login page is displayed with a button labeled "Register".

   Action: Click the "Register" button.

   Expected Result: The panel for registering a new account should be displayed with text fields and drop down menus for the user to enter their new account information and select certain options.

4. Verify that the user is navigated to the appropriate user perspective page after clicking the "Login" button.

   Purpose: Check that the user is able to get to the page corresponding to their user type they selected during the login prompt.

   Setup: The login page is displayed with all the login credentials filled out from the user and a user type is selected from the drop down menu.

   Action: Enter login credentials of the account and click the "Login" button.

   Expected Result: The perspective page corresponding to the user type selected at login time should be displayed. For example, if the user selected Doctor as their user type, the doctor perspective page should be displayed.

5. Verify that the user cannot log into an account without entering login credentials.

   Purpose: Check whether the user has entered any login credentials or not.

Setup: The login page is displayed with either all or some of the text fields for the login credentials being empty.

Action: Leave the text fields empty and click the "Login" button.

Expected Result: An error message should pop up to the screen after clicking the "Login" button. The user should not be able to log into their accounts and get an error message if there is at least one empty text field.

6. Verify that after a user registers for a new account, they can use that account information to log into the system.

   Purpose: Check if users can successfully register for a new account and store that information in the database to log into the system.

   Setup: The user has registered for a new account.

   Action: Select the account type from the drop down menu and enter the login credentials they registered with during the registration process. After doing so, the user clicks the button labelled "Login".

   Expected Result: The user should be navigated to the appropriate page according to which account type they signed in as. The account information the user signed up with should be stored in the accounts JSON file.

7. Verify that a patient can book an appointment.

   Purpose: Check if a patient can book appointments when they want to make a visit to the hospital.

   Setup: The patient has logged into their account and navigated to the panel for booking appointments.

   Action: Select a time slot and appointment type on the book appointments panel and confirm by clicking the button labeled "Book".

   Expected Result: A drop down menu with the time slot and appointment type should be displayed. After confirming, the appointment details are stored in the database in which the patient can view this appointment information.

8. Verify that a patient can view their personal information with the option to edit it.

Purpose: Check if the patient can see their personal information so they know if it is up to date and can make changes if necessary.

Setup: The patient has logged into their account.

Action: Click on the button labelled "View personal info".

Expected Result: The patient can view a page with all the information they registered with. This information should include: first name, last name, age, email, gender, and password. There should be a button next to each field for the patient to edit the information.

9. Verify that a doctor can view which patients they have.

   Purpose: Check if a doctor can see which patients have booked them for an appointment so they know how many patients will visit them.

   Setup: The doctor has logged into their account.

   Action: Click on the button labeled "View Patients".

   Expected Result: The doctor can view a list with all the patients who have booked for an appointment after clicking the "View Patients" button.

10. Verify that the nurse can view their schedules.

    Purpose: Check if the nurse can see which days, time, and department they are working in.

    Setup: The nurse has logged into their account.

    Action: Click on the button labelled "View Schedule".

    Expected Result: The nurse can view a page with the date, time and department they will working in after clicking the "View Schedule" button.

11. Verify that the assistant can approve appointments.

    Purpose: Check if the assistant is able to approve of appointments requested by patients if they fit the doctor's schedules.

    Setup: The assistant has logged into their account.

    Action: Click on the button labelled "Approve Appointments" and select an appointment from the list displayed to approve. After selecting, the assistant clicks on the button labelled "Approve".

Expected Result: The assistant can view a page listing all the appointments requested by patients. After selecting an appointment and clicking the "Approve" button, a confirmation message should be displayed and the selected appointment should be deleted from the list. If the no appointment is selected, an error message will be displayed.

12. Verify that the admin can view statistics of the hospital.

    Purpose: Check if the admin can see the statistics of how many patients have visited the hospital in prior years.

    Setup: The admin has logged into their account.

    Action: Click on the button labelled "View Statistics". Then the admin will select from a drop down menu containing different years and click "Next".

    Expected Result: The admin can view a page with a drop down menu with all the years the hospital has been in business for. After selecting one, the admin should see a page that shows the number of patients that visited in the selected year.

In addition to manual testing, we also used automated JUnit tests to test our Account class. When a user registers into the system, an instance of the Account class is created that stores the user's information (name, username, password, etc). The JUnit_Account.java file contains unit tests that ensure the Account class is saving this information correctly. The following is a list of the test cases we used for testing:

1. Test the Account constructor that takes in an age parameter.

    Test Name: test_ageConstructorAccount

    Purpose: Check that the Account constructor that takes in an age parameter (that we use to store patient information) is working as we would expect it to.

    Setup: Create an Account object with an age parameter to test the targeted constructor.

    Action: Use the accessor methods to confirm that the correct account type, first name, last name, age, email, gender, and password are stored.

Expected Result: An instance of the Account class with the specified account type, first name, last name, age, email, gender, and password has been created.

2. Test the account type accessor method for an Account that was created using the constructor that takes in an age parameter.

   Test Name: test_getAccountType_ageConstructor

   Purpose: Check that the accessor method is working for accounts created with the Account constructor that takes in an age parameter (patient accounts).

   Setup: Create an Account object with an age parameter to test the targeted constructor.

   Action: Use the accessor method for account type to confirm that the correct account type was stored.

   Expected Result: The account type specified when creating the account is the same as the account type that the accessor method returns.

3. Test the first name accessor method for an Account that was created using the constructor that takes in an age parameter.

   Test Name: test_getFirstName_ageConstructor

   Purpose: Check that the accessor method is working for accounts created with the Account constructor that takes in an age parameter (patient accounts).

   Setup: Create an Account object with an age parameter to test the targeted constructor.

   Action: Use the accessor method for first name to confirm that the patient's first name was stored correctly.

   Expected Result: The first name specified when creating the account matches the value that the accessor method returns.

4. Test the last name accessor method for an Account that was created using the constructor that takes in an age parameter.

   Test Name: test_getLastName_ageConstructor

Purpose: Check that the accessor method is working for accounts created with the Account constructor that takes in an age parameter (patient accounts).

Setup: Create an Account object with an age parameter to test the targeted constructor.

Action: Use the accessor method for last name to confirm that the correct last name was stored.

Expected Result: The last name specified when creating the account is the same as the last name that the accessor method returns.

5. Test the age accessor method for an Account that was created using the constructor that takes in an age parameter.

   Test Name: test_getAge_ageConstructor

   Purpose: Check that the accessor method is working for accounts created with the Account constructor that takes in an age parameter (patient accounts).

   Setup: Create an Account object with an age parameter to test the targeted constructor.

   Action: Use the accessor method for age to confirm that the age was stored correctly.

   Expected Result: The age specified when creating the account is the same as the value that the accessor method returns.

6. Test the email accessor method for an Account that was created using the constructor that takes in an age parameter.

   Test Name: test_getEmail_ageConstructor

   Purpose: Check that the accessor method is working for accounts created with the Account constructor that takes in an age parameter (patient accounts).

   Setup: Create an Account object with an age parameter to test the targeted constructor.

   Action: Use the accessor method for email to confirm that the patient's email was stored correctly.

Expected Result: The email specified when creating the account matches the value that the accessor method returns.

7. Test the gender accessor method for an Account that was created using the constructor that takes in an age parameter.

   Test Name: test_getGender_ageConstructor

   Purpose: Check that the accessor method is working for accounts created with the Account constructor that takes in an age parameter (patient accounts).

   Setup: Create an Account object with an age parameter to test the targeted constructor.

   Action: Use the accessor method for gender to confirm that the patient's gender was stored correctly.

   Expected Result: The gender specified when creating the account matches the value that the accessor method returns.

8. Test the password accessor method for an Account that was created using the constructor that takes in an age parameter.

   Test Name: test_getPassword_ageConstructor

   Purpose: Check that the accessor method is working for accounts created with the Account constructor that takes in an age parameter (patient accounts).

   Setup: Create an Account object with an age parameter to test the targeted constructor.

   Action: Use the accessor method for password to confirm that the patient's password was stored correctly.

   Expected Result: The password specified when creating the account matches the value that the accessor method returns.

9. Test the Account constructor that does not take in an age parameter.

   Test Name: test_noAgeConstructorAccount

   Purpose: Check that the Account constructor that does not take in an age parameter (that we use to store admin, assistant, doctor, and nurse information) is working as we would expect it to.

Setup: Create an Account object with no age parameter to test the targeted constructor.

Action: Use the accessor methods to confirm that the correct account type, first name, last name, email, gender, and password are stored.

Expected Result: An instance of the Account class with the specified account type, first name, last name, email, gender, and password has been created.

10. Test the account type accessor method for an Account that was created using the constructor that does not take in an age parameter.

    Test Name: test_getAccountType_noAgeConstructor

    Purpose: Check that the accessor method is working for accounts created with the Account constructor that does not take in an age parameter (staff accounts).

    Setup: Create an Account object with no age parameter to test the targeted constructor.

    Action: Use the accessor method for account type to confirm that the correct account type was stored.

    Expected Result: The account type specified when creating the account is the same as the account type that the accessor method returns.

11. Test the first name accessor method for an Account that was created using the constructor that does not take in an age parameter.

    Test Name: test_getFirstName_noAgeConstructor

    Purpose: Check that the accessor method is working for accounts created with the Account constructor that does not take in an age parameter (staff accounts).

    Setup: Create an Account object with no age parameter to test the targeted constructor.

    Action: Use the accessor method for first name to confirm that the staff member's first name was stored correctly.

    Expected Result: The first name specified when creating the account matches the value that the accessor method returns.

12. Test the last name accessor method for an Account that was created using the constructor that does not take in an age parameter.

    Test Name: test_getLastName_noAgeConstructor

    Purpose: Check that the accessor method is working for accounts created with the Account constructor that does not take in an age parameter (staff accounts).

    Setup: Create an Account object with no age parameter to test the targeted constructor.

    Action: Use the accessor method for last name to confirm that the staff member's last name was stored correctly.

    Expected Result: The last name specified when creating the account is the same as the last name that the accessor method returns.

13. Test the email accessor method for an Account that was created using the constructor that does not take in an age parameter.

    Test Name: test_getEmail_noAgeConstructor

    Purpose: Check that the accessor method is working for accounts created with the Account constructor that does not take in an age parameter (staff accounts).

    Setup: Create an Account object with no age parameter to test the targeted constructor.

    Action: Use the accessor method for email to confirm that the staff member's email was stored correctly.

    Expected Result: The email specified when creating the account matches the value that the accessor method returns.

14. Test the gender accessor method for an Account that was created using the constructor that does not take in an age parameter.

    Test Name: test_getGender_noAgeConstructor

    Purpose: Check that the accessor method is working for accounts created with the Account constructor that does not take in an age parameter (staff accounts).

Setup: Create an Account object with no age parameter to test the targeted constructor.

Action: Use the accessor method for gender to confirm that the staff member's gender was stored correctly.

Expected Result: The gender specified when creating the account matches the value that the accessor method returns.

15. Test the password accessor method for an Account that was created using the constructor that does not take in an age parameter.

    Test Name: test_getPassword_noAgeConstructor

    Purpose: Check that the accessor method is working for accounts created with the Account constructor that does not take in an age parameter (staff accounts).

    Setup: Create an Account object with no age parameter to test the targeted constructor.

    Action: Use the accessor method for password to confirm that the staff member's password was stored correctly.

    Expected Result: The password specified when creating the account matches the value that the accessor method returns.

12 References

[1] Java2blog.com, "JSON.simple example – Read and write JSON," 2017. [Online]. Available: https://java2blog.com/jsonsimple-example-read-and-write-json. [Accessed: 19 March 2020]