SEG2105 – Introduction to Software Engineering – Fall 2023

Android Project: HAMS (20%)

Instructions

1. You will work in the same team throughout the semester
2. You will submit each deliverable as a release on GitHub
3. At the end of the semester, the team must demonstrate a completed application. For instance, one team member demonstrating a screen with one functionality while another member showing another screen with another functionality (running on a different phone), **WILL NOT be accepted**. The team must produce a single application with all the required features.

Academic Honesty

Using someone else's work and presenting it as your own is considered academic fraud. You are responsible for writing all the code for the application. However, as a software developer, it is natural to refer to online resources such as code examples or online forums. This is an important aspect of the software development process. Nevertheless, it is expected that you drastically limit the practice of directly copying code that you have not developed yourself.

If you do need to copy a significant portion of code, defined here as more than five lines, it is crucial to acknowledge its source by including a comment above the code snippet in your deliverable report. Additionally, it is essential that you thoroughly understand any code you incorporate from external sources. The instructor and/or teaching assistants may inquire about these code snippets, and you should be prepared to answer questions related to them. In fact, they are likely to focus their questions on these specific code segments.

Note about ChatGPT and other Generative Models

The course instructor acknowledges that ChatGPT and other generative language models, when used appropriately, can provide significant value to software developers. However, excessive reliance on these tools can hinder the learning process. Therefore, there are two specific scenarios in which you are permitted to use these tools:

1. To search for information about error messages or exceptions.
2. To find guidance on how to perform generic operations that involve only a few lines of code (e.g., connecting to a database).

Any usage of these models beyond these two situations is strictly prohibited. The instructor will employ experimental tools to scan your code for any signs of AI-based generation.

Project Description

In this project, you will implement the Healthcare Appointment Management System (HAMS) for a telehealth clinic. HAMS is a mobile application designed to streamline the process of healthcare appointment scheduling and management. The app support three types of users: ***Patient***, ***Doctor***, and ***Administrator***.

The ***Patient*** views available appointment slots so that they may book or cancel appointments. ***Patients*** can also rate doctors.

The ***Doctor*** manages their schedules and responds to appointment requests.

The ***Administrator*** approves the account registration requests of doctors and patients.

In the next sections, we will describe the application in detail from the perspective of each user.

Patient

To become a ***Patient***, a user submits a registration request that must be approved by the ***Administrator***. Once registered, a ***Patient*** can:

- View their upcoming appointments.
- Cancel an existing appointment.
- View their past appointments.
- Rate a ***Doctor*** following an appointment.
- Book an appointment.

Doctor

To become a ***Doctor***, a user submits a registration request that must be approved by the ***Administrator.*** Once registered, a ***Doctor*** can:

- Specify and change the shifts they would like to work.
- View their upcoming appointments.
- View their past appointments.
- Cancel appointments.
- Approve appointment requests.

The ***Doctor*** may also elect to approve all appointment requests. This way they do not have to approve individual requests. Requests are simply approved automatically.

Administrator

The ***Administrator*** is a pre-registered user. This means that the ***Administrator's*** account information (i.e., username and password) are already in the database when the system is first launched. The ***Administrator*** can approve or reject registration requests from ***Patients*** and ***Doctors***.

User Interface Design

This course does not focus on user interface design. However, students are encouraged to produce aesthetic user interfaces that are easy to use. Consider the Android Design Guidelines when designing your application: https://developer.android.com/design.

Deliverables

The project is divided into 4 incremental deliverables. Students are required to submit each deliverable by the posted deadline.

Deliverable	**Due date**
1 – GitHub and User Accounts (3%)	October 16th
2 – Mostly Administrator Features (3%)	October 30th
3 – Mostly Doctor Features (3%)	November 13th
4 – Mostly Patient Features (and Integration) (9%)	December 4th
5 – Demo (2%)	Last week of classes

The project must be carried out throughout the session and students are **strongly** encouraged to maintain a log of their project activities as such information will be needed for the final report.

Your application must be written in Java and built using Android Studio (you can use another IDE, but the TAs will only provide support for Android Studio). By the end of the semester, you must implement and submit a working application based on the specifications. Firebase or SQLite can be used for storing and retrieving the application data.

Deliverable 1 – GitHub and User Accounts

(Read the "Project Description" section carefully before you start working on this deliverable.)

You need to join our GitHub classroom, create your team, and accept this project. Please refer to "Steps to Create your Team on GitHub" for detailed instructions.

In this deliverable, you need to implement the ***Patient*** and ***Doctor*** registration forms. **For this deliverable, we will assume that registrations do not require an *Administrator*'s approval. This feature will be added in Deliverable 2.**

To register, a ***Patient*** must submit the following information:

- First name
- Last name
- Email address (which also serves as the username)
- Account password
- Phone number
- Address
- Health card number

Similarly, to register, a ***Doctor*** must submit the following information:

- First name
- Last name

- Email address (which also serves as the username)
- Account password
- Phone number
- Address
- Employee number
- Specialties (e.g., family medicine, internal medicine, pediatrics, obstetrics, and gynecology)

Note that a ***Doctor*** can have **<u>one or more</u>** specialties.

The ***Administrator*** does not need to register as they are pre-registered.

Upon registration, the user can log in. They should see a second screen with the following message, 'Welcome! You are logged in as <role> (where <role> can be ***Patient***, ***Doctor***, or ***Administrator***). The user can also log off. No additional functionality needs to be implemented at this point.

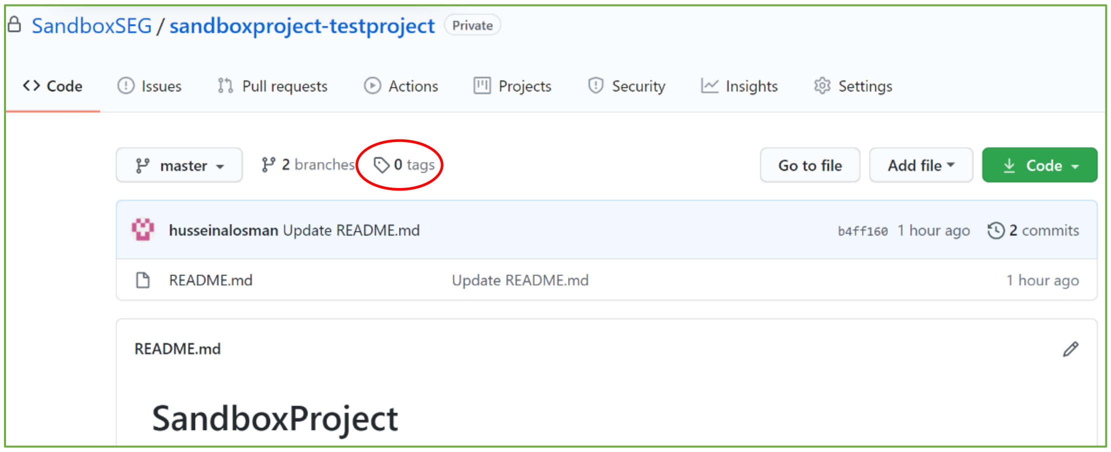
You can use Firebase or SQLite for DB support. If you do not use a DB at this point, you can simply store the information in memory (but it would be lost when you terminate the app) or on a file.

In your GitHub repository, include a PDF document that contains a UML Class diagram of your domain model. This will only include the UML classes related to deliverable 1. Hence, the UML class diagram at this stage will be exceptionally simple. *Do not include Activity classes in your UML model*. **Importantly**, in your repository, provide a README file that contains the credentials needed to sign-in as ***Administrator***.

Submission Instructions

To submit your code, you will create a release from your repository as follows:

1. In your repository, click on "tag" (see figure below)



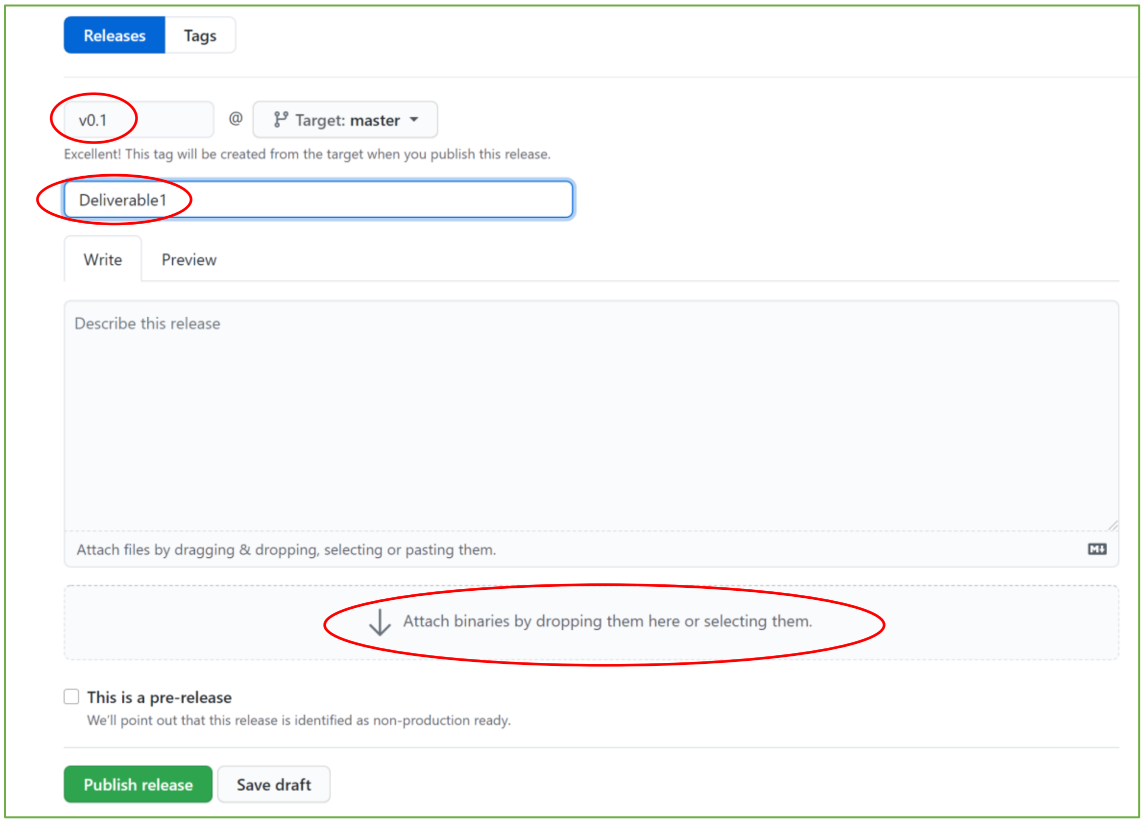
husseinalosman Update README.md b4ff160 1 hour ago 2 commits

README.md Update README.md 1 hour ago

README.md

SandboxProject

2. Click on the "Create new release" button and fill the form as follows (see figure below)
   a. For "Tag version" enter v0.1
   b. For "Release title" enter Deliverable1
   c. From the section of the form where you can attach binaries, upload the APK of your app
   d. Make sure to rename the APK after the name of your team (e.g., "Project_Group_1_debug_apk".


Releases
Tags
v0.1
@
Target: master
Excellent! This tag will be created from the target when you publish this release.
Deliverable1
Write
Preview
Describe this release
Attach files by dragging & dropping, selecting or pasting them.
Attach binaries by dropping them here or selecting them.
This is a pre-release
We'll point out that this release is identified as non-production ready.
Publish release
Save draft

Deliverable 1 Marking Scheme

Feature or Task	**% Weight (out of 100)**
The team created in GitHub classroom contains all members of the group.	10
Each member of the group has made at least **one** commit to the repository.	20
The UML Class diagram of your domain model is valid. • (-2 for each missing class) • (-2 for each incorrect generalization) • (-0.5 for each incorrect multiplicity) • (-0.5 for each missing attribute)	5
The APK is submitted.	5
A user can create a ***Patient*** or ***Doctor*** account.	15
The ***Administrator***, ***Doctor***, or ***Patient*** user can see the "welcome screen" after successful authentication. The welcome screen specifies the user role.	15
The user can log off.	10
All fields are validated. There are appropriate error messages for incorrect inputs. (-1 for each field in which the user input is not validated)	20

Optional – The group uses a DB (e.g., Firebase, SQLITE, or another similar technology).	+5 (bonus)

Deliverable 2 – Mostly Administrator Features

(Read the "Project Description" section carefully before you start working on this deliverable.)

In this deliverable, you need to implement the ***Administrator*** functionality. Moreover, if you have not done so, you should start using a DB (e.g., Firebase, SQLITE, or another similar technology).

The ***Administrator*** has an inbox to receive registration requests from ***Patients*** and ***Doctors***. Each request contains all the information the user entered in the registration form, except for the password. The ***Administrator*** can approve or reject a registration request.

The ***Administrator*** can view the list of registrations they previously rejected. They can select any request from this list and change their decision to approve the registration. However, if they approve the registration, they can no longer change their decision.

You must ensure that when a ***Patient*** or ***Doctor*** registers their account, they cannot gain access to the system before obtaining the approval of the ***Administrator***. When a ***Patient*** or ***Doctor*** attempt to login, one of the following occurs:

- If their registration request was approved by the ***Administrator***, they reach the welcome screen.
- If their registration request was rejected by the ***Administrator***, they get a message informing them that they cannot login as their registration was rejected. They are told that they contact the ***Administrator*** by phone to resolve the issue (you can display a made-up phone number on the screen).
- If their registration request was not processed yet by the ***Administrator***, they get a message informing them that their registration has not been approved yet.

In your GitHub repository, include a PDF document that contains a UML Class diagram of your domain model. This will only include the UML classes related to deliverables 1 and 2. *Do not include Activity classes in your UML model*. **Importantly**, in your repository, provide a README file that contains the credentials needed to sign-in as ***Administrator***.

Submission Instructions

To submit your code, create a release v0.2 in your repository. Make sure the APK file is added to your release.

Deliverable 2 Marking Scheme

Feature or Task	**% Weight (out of 100)**
The updated UML Class diagram of your domain model is valid. • (-2 for each missing class) • (-2 for each incorrect generalization) • (-0.5 for each incorrect multiplicity) • (-0.5 for each missing attribute)	10

The APK is submitted.	5
When a ***Patient*** or ***Doctor*** register, their account information is stored in the DB (along with an indicator of whether their account registration has been approved, rejected, or not processed yet).	15
The ***Administrator*** can view the list of registration requests.	5
The ***Administrator*** can view the information associated with each request (the information the user entered during registration).	5
The ***Administrator*** can approve or reject a registration request.	5
If approved, the registration request disappears from the list of registration requests.	7.5
If rejected, the registration request is added to the list of rejected registration requests.	7.5
The ***Administrator*** can view the list of previously rejected registration requests.	7.5
The ***Administrator*** can approve a previously rejected request.	7.5
The registration requests are stored in the DB	10
When a ***Patient*** or ***Doctor*** attempt to login, they are either directed to the welcome page, notified that their registration request was rejected, or informed that their registration request has not been processed yet.	15
Optional – When a user registration request is approved or rejected, they receive an e-mail and notification on their phone.	+5 (bonus)

Deliverable 3 – Mostly Doctor Features

(Read the "Project Description" section carefully before you start working on this deliverable.)

In this deliverable, you should implement most of the ***Doctor***'s functionality. The ***Doctor*** has a list of upcoming appointments and a list of past appointments. If they tap on an appointment in the list of upcoming appointments, they can see the ***Patient***'s information. If the appointment is not approved yet, they can approve it or reject it. They can also cancel a previously approved appointment.

The ***Doctor*** may also elect to approve all appointment requests automatically. This way they do not have to approve every request. Requests are simply approved by the system.

The ***Doctor*** has a list of shifts. Hence, they can view their shifts, add new shifts, or delete existing ones. To add a shift, the ***Doctor*** must specify the date, start-time, and end-time of a shift. The start-time and end-time can only be selected by 30-minute increments (e.g., 8:00, 8:30, 9:00, 9:30, etc.). The ***Doctor*** cannot specify a date that has already passed or add a shift that conflicts with an existing one.

Note that for now, the *Doctor* can delete any shift they had previously added. However, in Deliverable 4, you will implement a mechanism to prevent a *Doctor* from deleting a shift associated with a *Patient* appointment.

In your GitHub repository, include a PDF document that contains a UML Class diagram of your domain model. This will only include the UML classes related to deliverables 1, 2, and 3. *Do not include Activity* *classes in your UML model*. **Importantly**, in your repository, provide a README file that contains the credentials needed to sign-in as ***Administrator***.

Submission Instructions

To submit your code, create a release v0.3 in your repository. Make sure the APK file is added to your release.

Deliverable 3 Marking Scheme

Feature or Task	**% Weight (out of 100)**
The updated UML Class diagram of your domain model is valid. • (-2 for each missing class) • (-2 for each incorrect generalization) • (-0.5 for each incorrect multiplicity) • (-0.5 for each missing attribute)	15
The APK is submitted.	5
The ***Doctor*** can view a list of upcoming appointments.	5
The ***Doctor*** can view the information of a ***Patient*** that requested an appointment (i.e., first name, last name, email, address, phone number, health card number).	10
The ***Doctor*** can approve or reject an appointment request.	10
The ***Doctor*** can cancel a previously approved appointment.	10
The ***Doctor*** can view a list of past appointments.	5
The ***Doctor*** may enable a setting so that all appointment requests are automatically approved by the system without further action on their part.	10
The ***Doctor*** can view the list of upcoming shifts they are working.	10
The ***Doctor*** can add a new shift by specifying the date, start-time**,** and end-time of the shift. All fields must be validated. Hence, the ***Doctor*** cannot enter a date that has already passed or a shift that conflicts with another one they had previously added.	10
The ***Doctor*** can delete an existing shift.	10
Optional – The group integrates with CircleCI to see the automated builds and test unit execution.	+5 (bonus)

Deliverable 4 – Mostly Patient Features (and Integration)

In this deliverable, you should implement the ***Patient***'s functionality. The ***Patient*** can view a list of upcoming appointments. They can tap on an upcoming appointment from the list and opt to cancel it. Cancellation are only possible is the appointment is not scheduled to start in the next 60 minutes. Otherwise, it is too late for cancellation. The ***Patient*** can also view their past appointments and tap on a past appointment to rate the doctor (1 to 5 starts).

Patients can search for appointments based on medical specialty (e.g., family medicine, internal medicine, pediatrics, obstetrics, and gynecology). Hence, to view available appointment time slots, they must first specify the desired specialty. Each appointment time slot represents a 30-minute period. After finding a suitable time slot, a ***Patient*** can tap on it to book the appointment. The booked appointment will then be added to the ***Patient***'s list of upcoming appointments. Once a time slot is booked, it will no longer be listed when other ***Patients*** look for appointments.

As opposed to Deliverable 3, in this deliverable, you must ensure that a ***Doctor*** cannot delete a shift if it is associated with one or more ***Patient*** appointments. If a ***Doctor*** tries to delete such a shift, the system should display a message informing the ***Doctor*** that they must first cancel all appointments linked to this shift before they can delete it.

Finally, you must include at least **4 unit test cases** (simple local tests) to your code. There is no need to include instrumentation or use Espresso Tests (UI).

In your GitHub repository, include a PDF document that contains your final report (see the marking scheme for information on what to include in the document). Moreover, in your repository, provide a README file that contains the credentials needed to sign-in as ***Administrator***.

Submission Instructions

To submit your code, create a release v0.4 in your repository. Make sure the APK file is added to your release.

Deliverable 4 Marking Scheme

Feature or Task	**% Weight (out of 100)**
The updated UML Class diagram of your domain model is valid. • (-2 for each missing class) • (-2 for each incorrect generalization) • (-0.5 for each incorrect multiplicity) • (-0.5 for each missing attribute)	10
The APK is submitted.	5
The 4 Unit test cases (simple local tests) are implemented. There is no need to include instrumentation or Espresso Tests (UI).	10
The final report is submitted and includes: • Title page (2.5 points) • Short Introduction (2.5 points) • Updated UML class diagram • Table specifying the contributions of team members for each deliverable (10 points) • All the screenshots of your app (10 points) • Lessons learned (5 points)	30
The application supports the requirements of Deliverables 1, 2, and 3 (In case these deliverables were not previously completed, they still need to be implemented for this deliverable)	20
The ***Patient*** can view a list of upcoming appointments.	5

The ***Patient*** can cancel an upcoming appointment. Cancellations should only be possible if the appointment is not scheduled to start in the next 60 minutes.	5
The ***Patient*** can view their past appointments.	5
The ***Patient*** can rate a ***Doctor*** with whom they previously had an appointment.	5
The *Patient* can search for appointments by specifying a medical specialty and selecting a time slot from the available ones. An appointment is a 30-minute time slot.	5
Once booked, the appointment appears in the ***Patient***'s list of upcoming appointments.	5
A booked time slot is not listed when ***Patients*** look for appointments.	5
The ***Doctor*** cannot delete a shift if it is associated with one or more ***Patient*** appointments.	5
Optional – Rather than searching for appointments by specifying a medical specialty, the ***Patient*** can opt to search for a ***Doctor*** by name (i.e., first name, last name, or both). If the search yields only one ***Doctor***, then that ***Doctor***'s available time slots are displayed. If the search returns multiple ***Doctors***, then the user can tap on a specific ***Doctor***'s name to view their available time slots. The ***Patient*** can then book a time slot.	+5 (bonus)