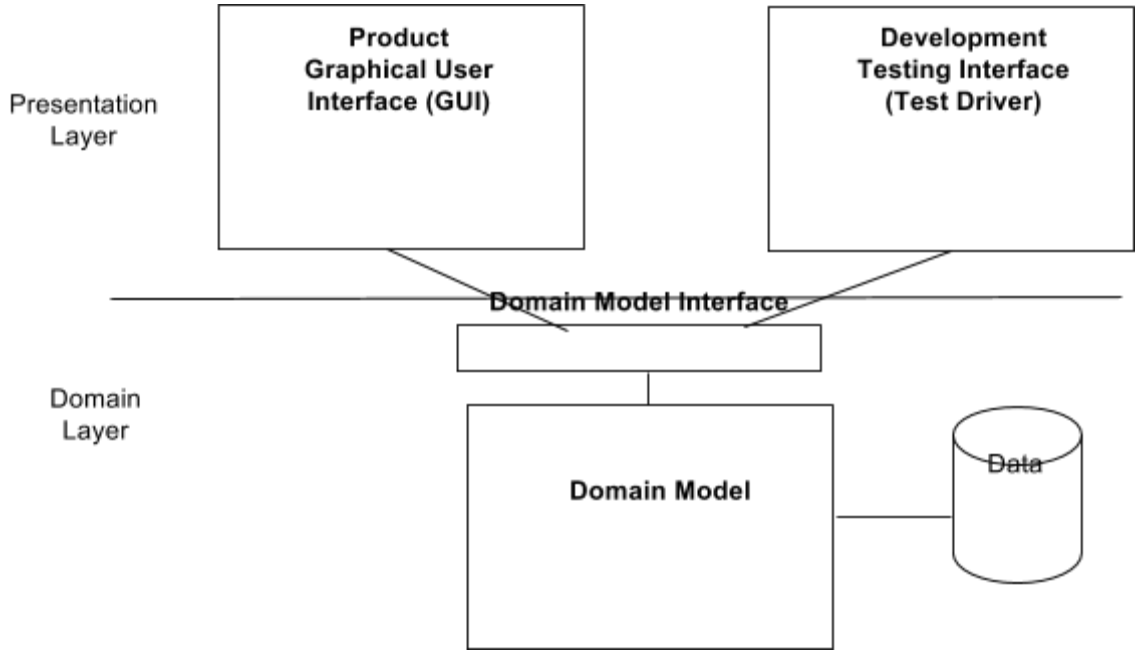
Product Design

Team: 201405-02-SWEN261-TEAM-B-ProjectCurry

Revision Number	*Revision Date*	*Summary of Changes*	*Author(s)*
1.0	2/19/2015	Initial creation	ProjectCurry
1.1	2/21/2015	Class Diagrams and Rational	ProjectCurry
1.2	2/22/2015	Sequence Diagrams	ProjectCurry
1.3	2/27/2015	Revised Class diagram and sequence diagrams	ProjectCurry
1.4	3/07/2015	Revised more sequence diagrams	ProjectCurry
1.5	3/09/2015	Verified the doc before R1 release	ProjectCurry
2.0	4/3/2015	Add additional diagrams for R2	ProjectCurry
2.1	4/11/2015	Added rationale pertaining to	ProjectCurry
2.2	5/08/2015	Added more and revised sequence diagrams	ProjectCurry

Architectural Model

This diagram represents the major subsystems of the product. Initially focus on the domain layer and its components before decomposing the user interface component. Note that a common interface allows both the GUI and a Command Line Interface to access the domain model in the same manner without regard to the type of presentation technique.

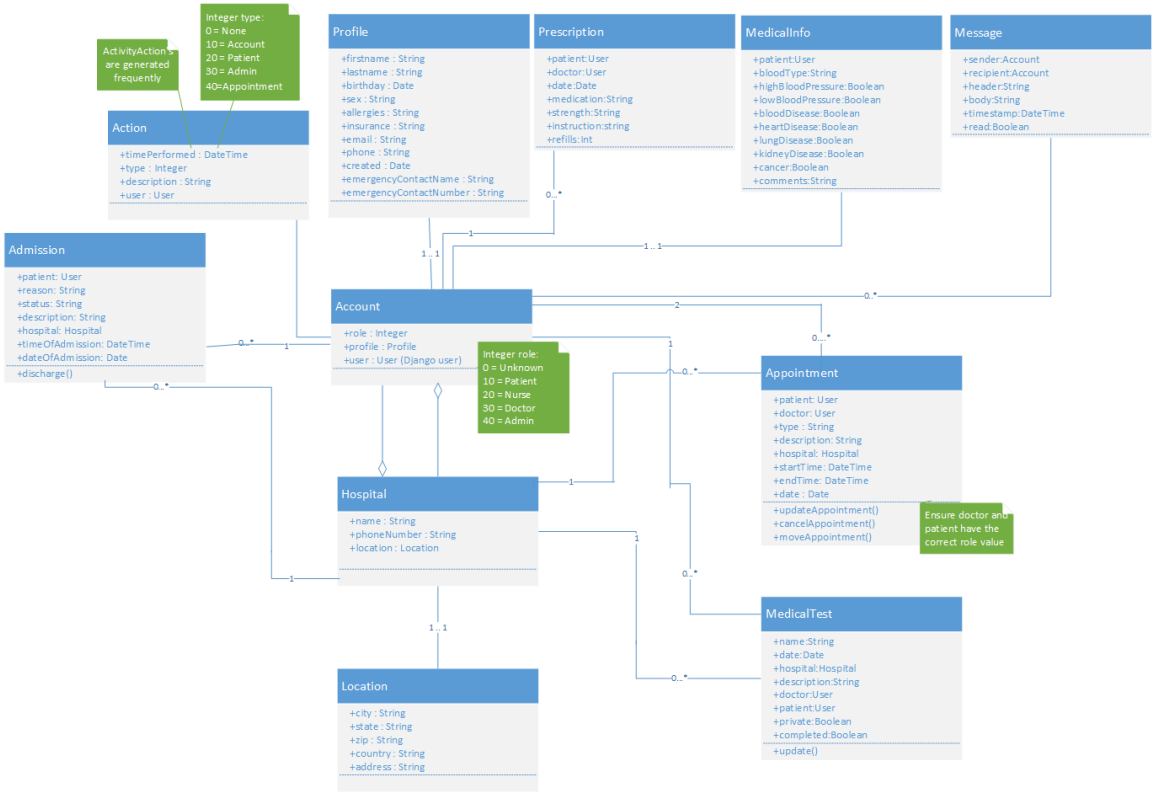


Components and Functions

<Component No. 1> Appointment	Component state: ● patient ● doctor ● status of the appointment ● description of appointment ● the hospital where the appointment will be held ● the start time for the appointment ● the end time for the appointment Component behavior ● updating an appointment. ● cancelling an appointment ● moving an appointment
<Component No. 2> Appointment Calendar	Component state: ● the user to which the list of appointments is associated with Component behavior: ● refreshing an appointment calendar ● sorting the appointments by type, user, or location
<Component No. 3> Profile	Component state: ● first name of user ● last name of user ● gender of user ● age of user ● insurance ● email ● allergies ● phone number ● date created ● emergency contract name ● emergency contract number Component behavior: ● Updating a profile.
<Component No. 4> Hospital	Component state: ● name ● location ● phone number
<Component No. 5> Location	Component state: ● city ● zip code ● state ● country ● address
<Component No. 6> Activity Logger	Component behavior: ● Activity Logger will disallow behavior if the user doesn't meat the role requirement to view the logger ● logger will sort by users ● logger will sort by type of action ● logger will sort by time of actions performed
<Component No. 7> Account	Component state: ● profile ● user ● the role which will indicate which type of user it is (doctor, nurse, etc.) Component behavior: ● the user be able to create appointments and appointment calendars ● the user will be add or remove themselves to locations (hospitals)

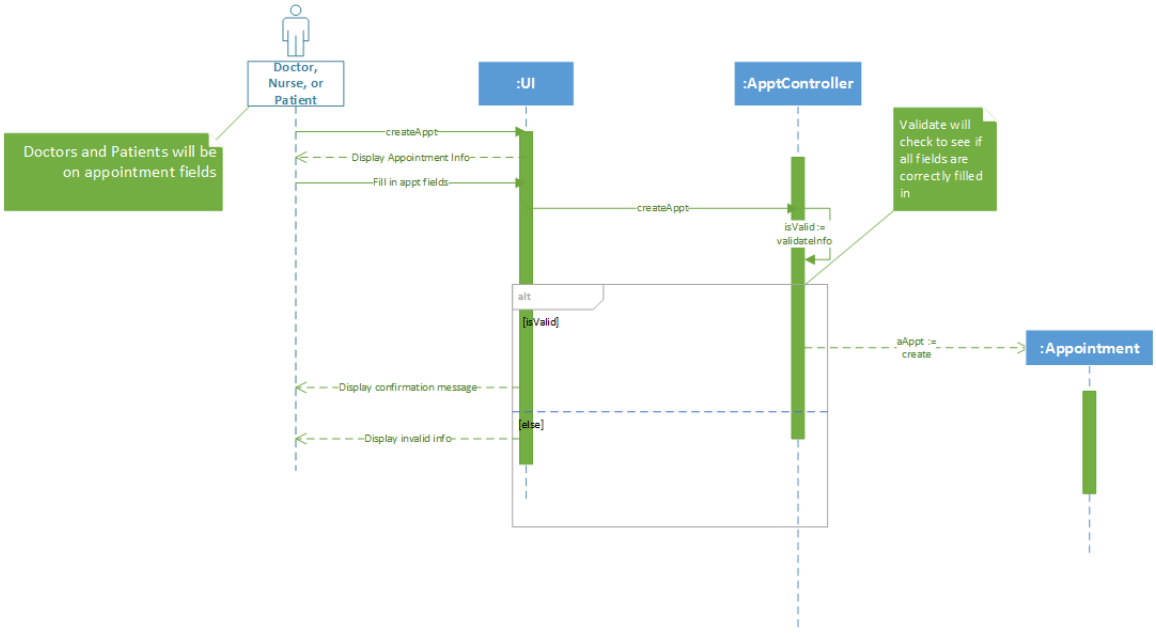
<Component No. 8> User Creator	Component behavior: ● the user creator will create profiles ● the user creator will validate profiles to ensure all fields are filled
<Component No. 9> Activity Action	Component state: ● time of action performed ● the type of action performed ● a description of the action performed ● the type of user performing the action (doctor, patient, etc.)
<Component No. 10> Admission	Component state: ● admitted patient ● reason for admission ● description of an admission ● the time a patient was admitted ● the time a patient was discharged ● hospital location of an admission Component behavior: ● Admission of a patient to a hospital ● Discharge of a patient
<Component No. 11> Medical Test	Component state: ● name ● date ● hospital ● doctor ● patient ● completed ● privacy (whether patient can view test or not) ● uploaded images Component behavior: ● updating a medical test ● viewing
<Component No. 12> Message	Component state: ● sender ● recipient ● timestamp ● status (active, deleted) ● message header ● message body ● has message been read by recipient Component behavior: ● viewing (which marks an unread message as read)

Class Diagram(s)

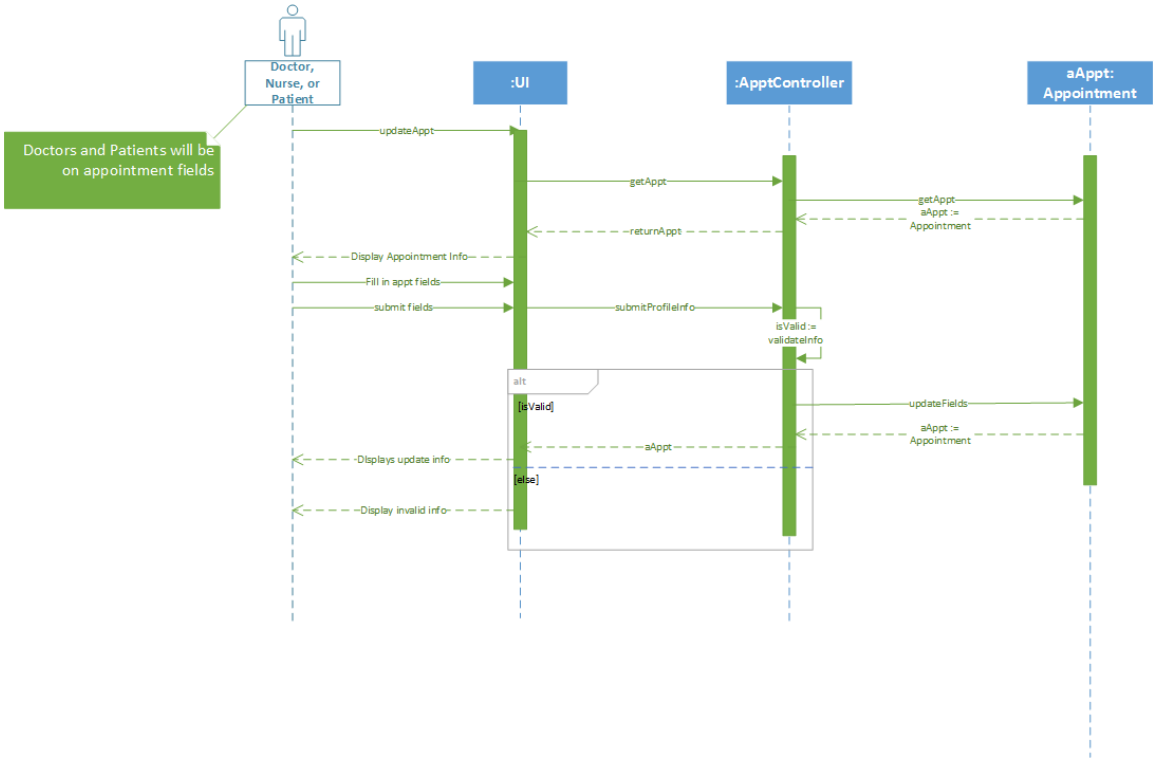


Sequence Diagram(s)

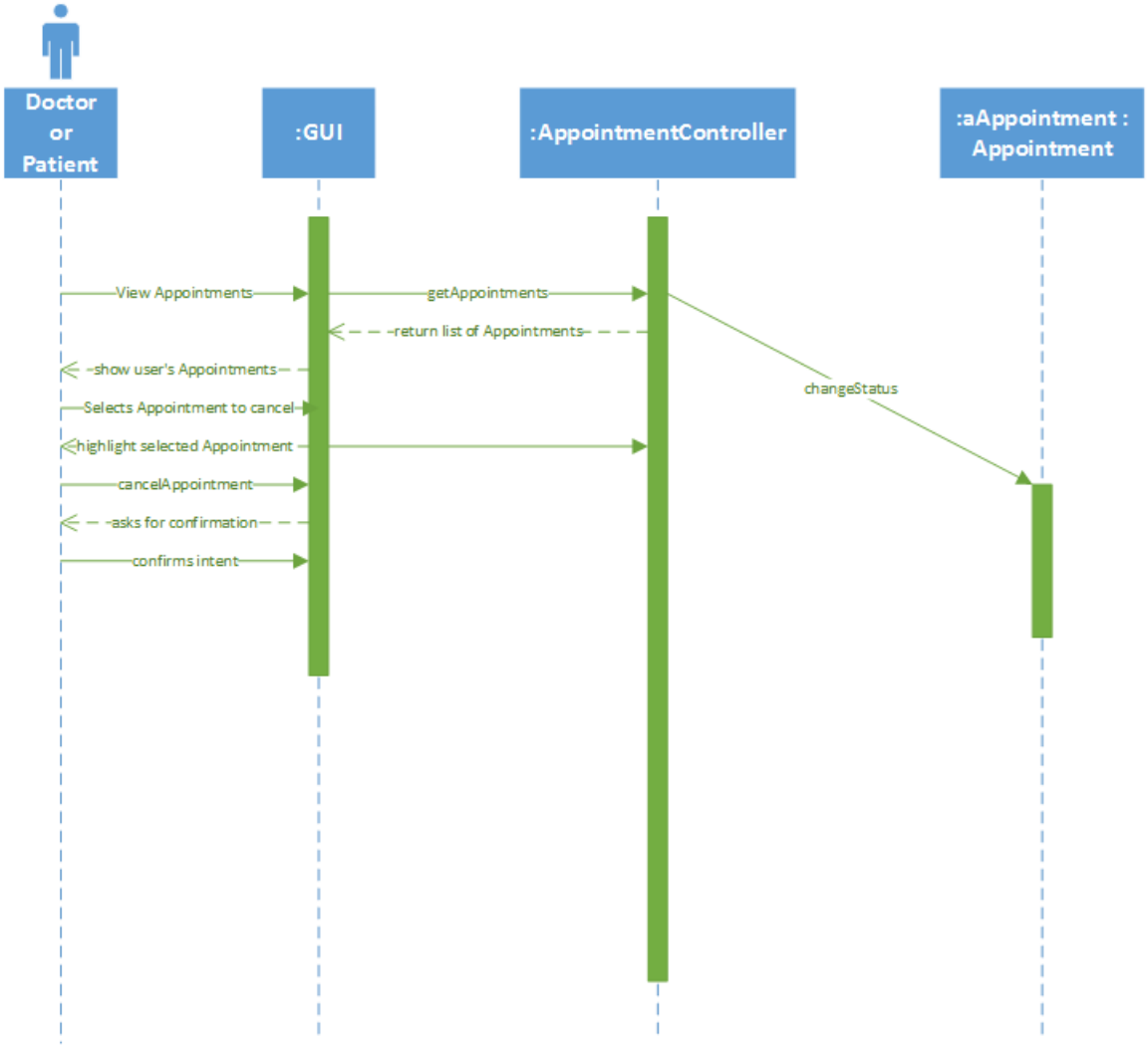
1. Create appointment



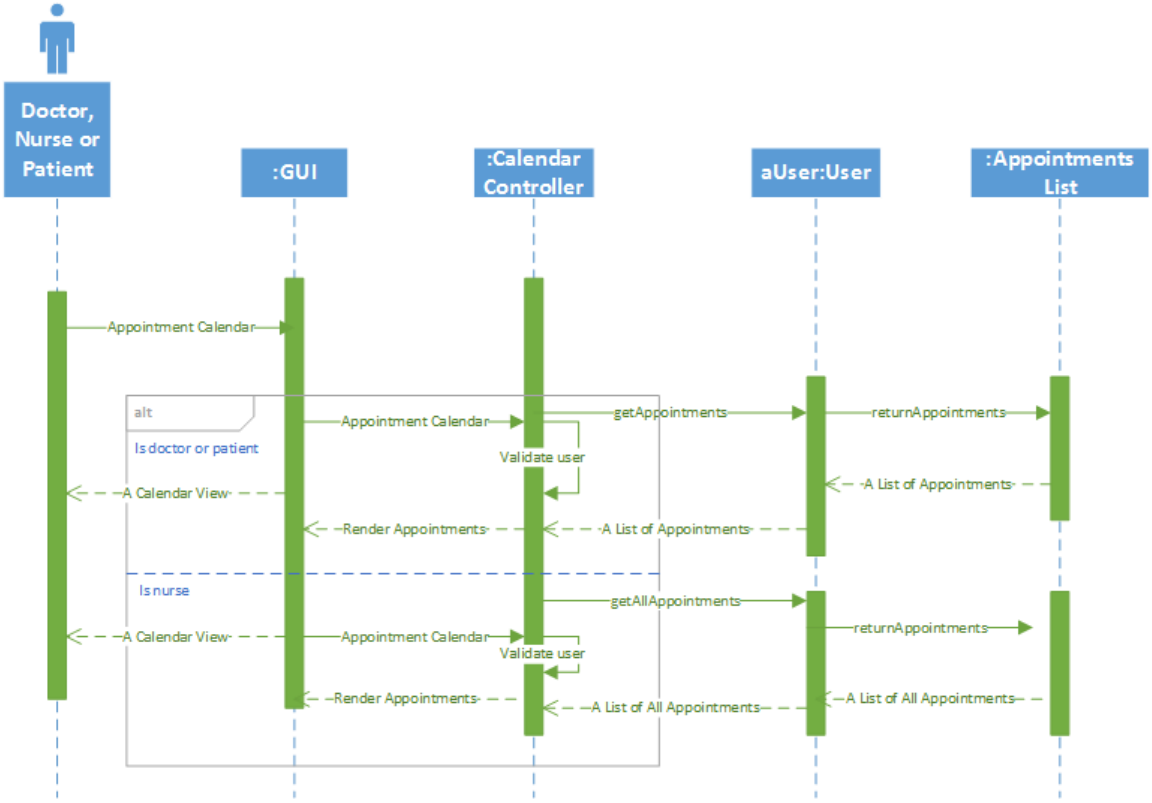
2. Update appointment



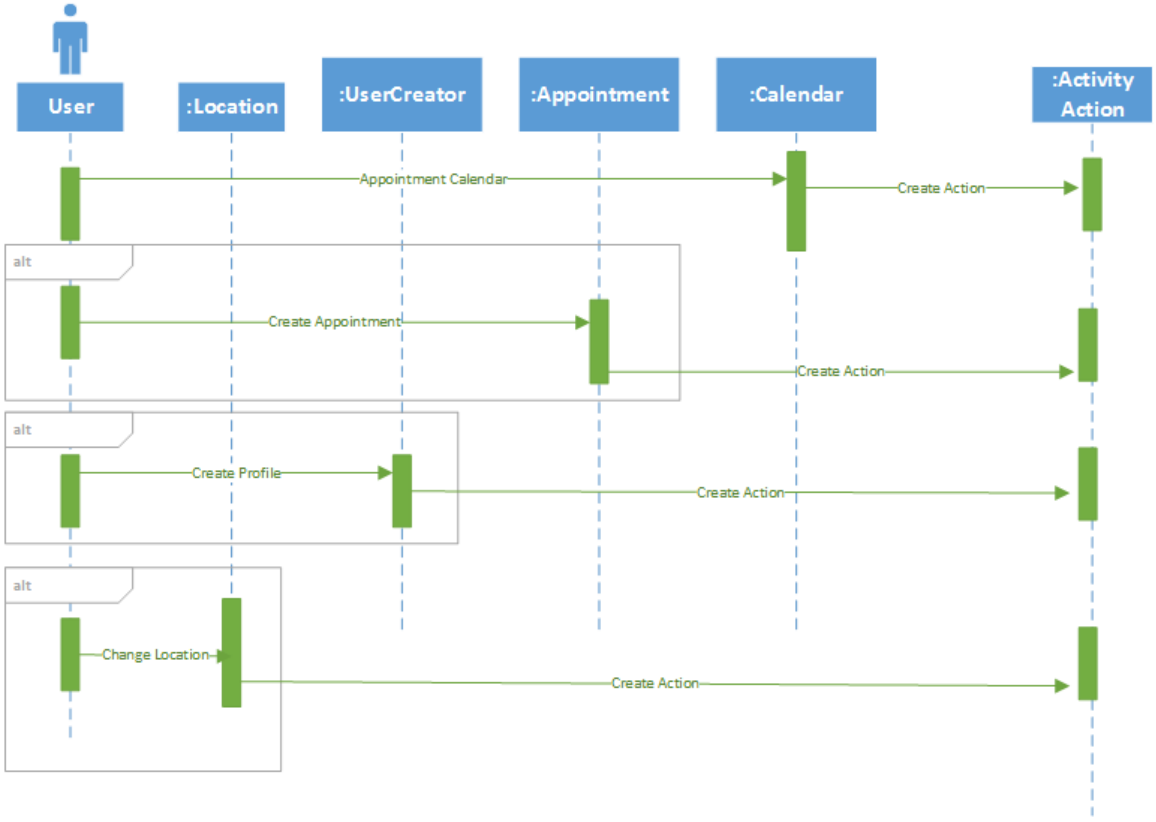
3. Cancel an appointment



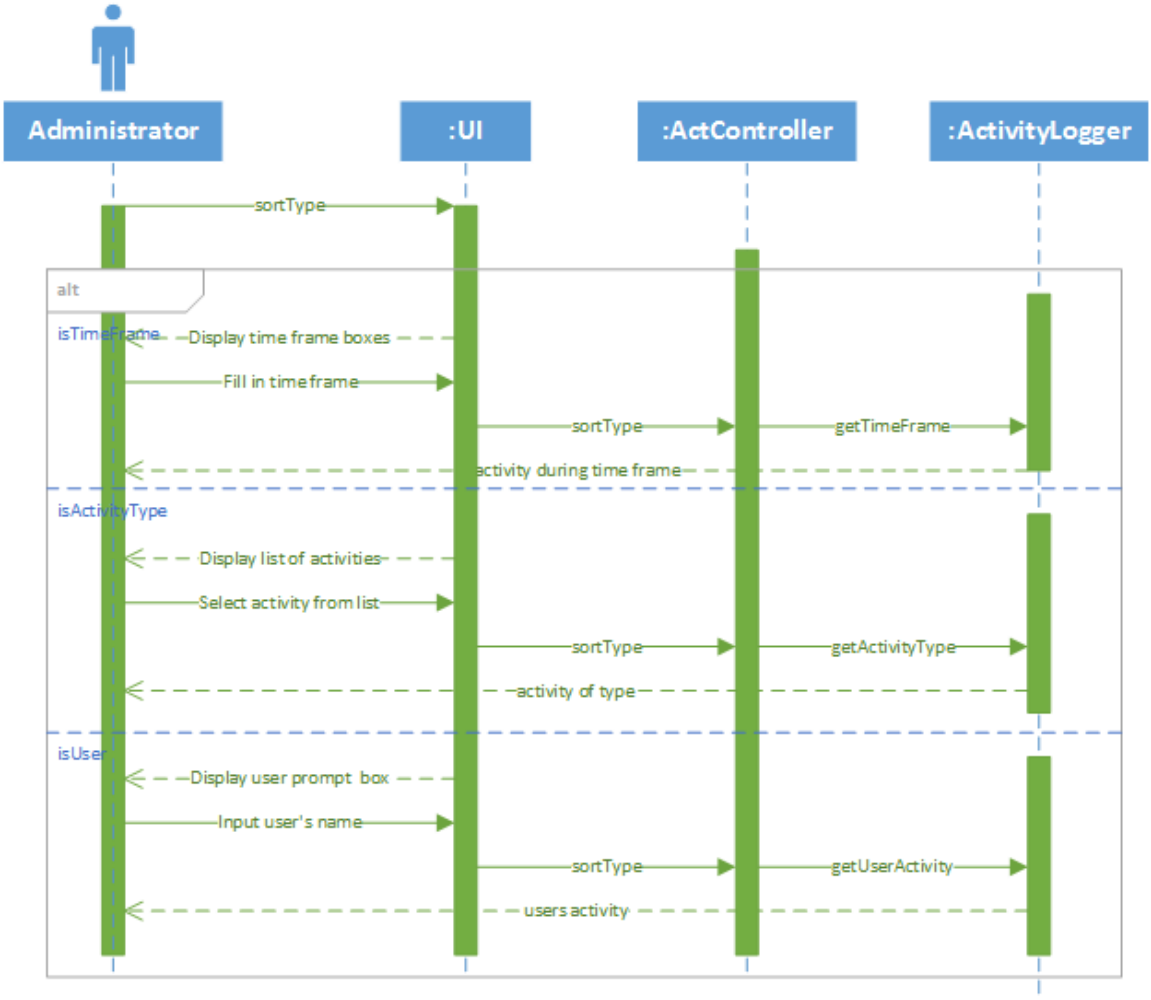
4. Appointment Calendar



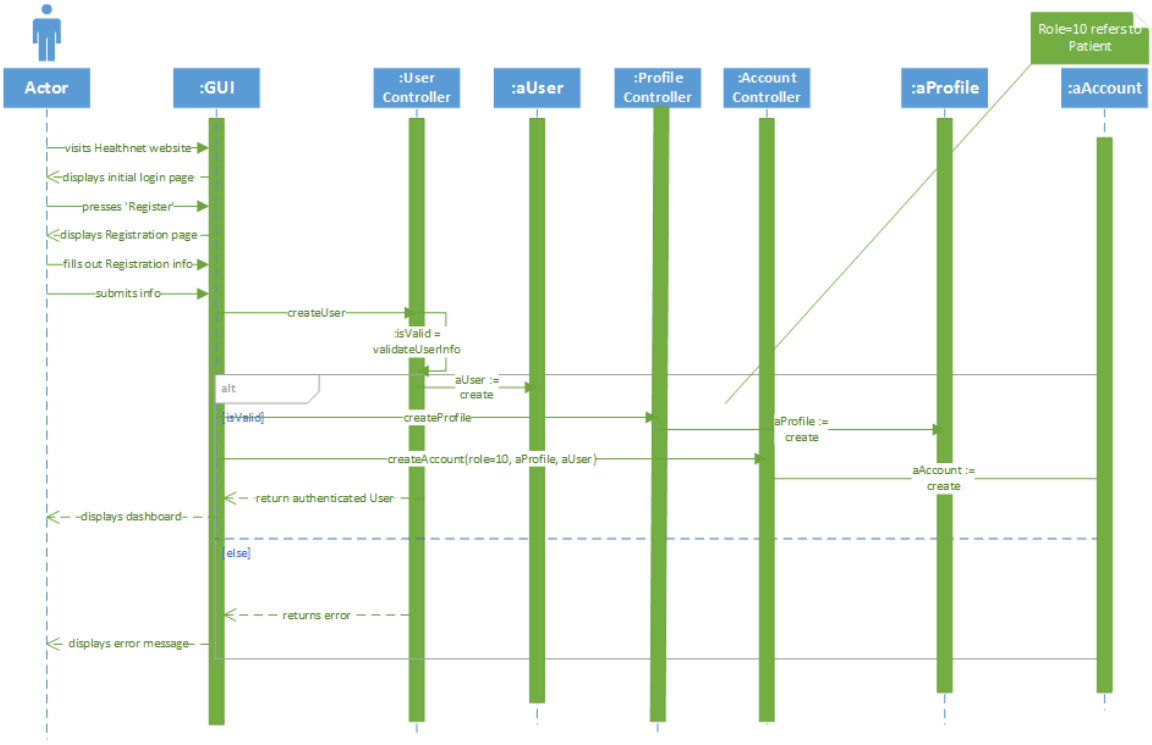
5. Activity Action



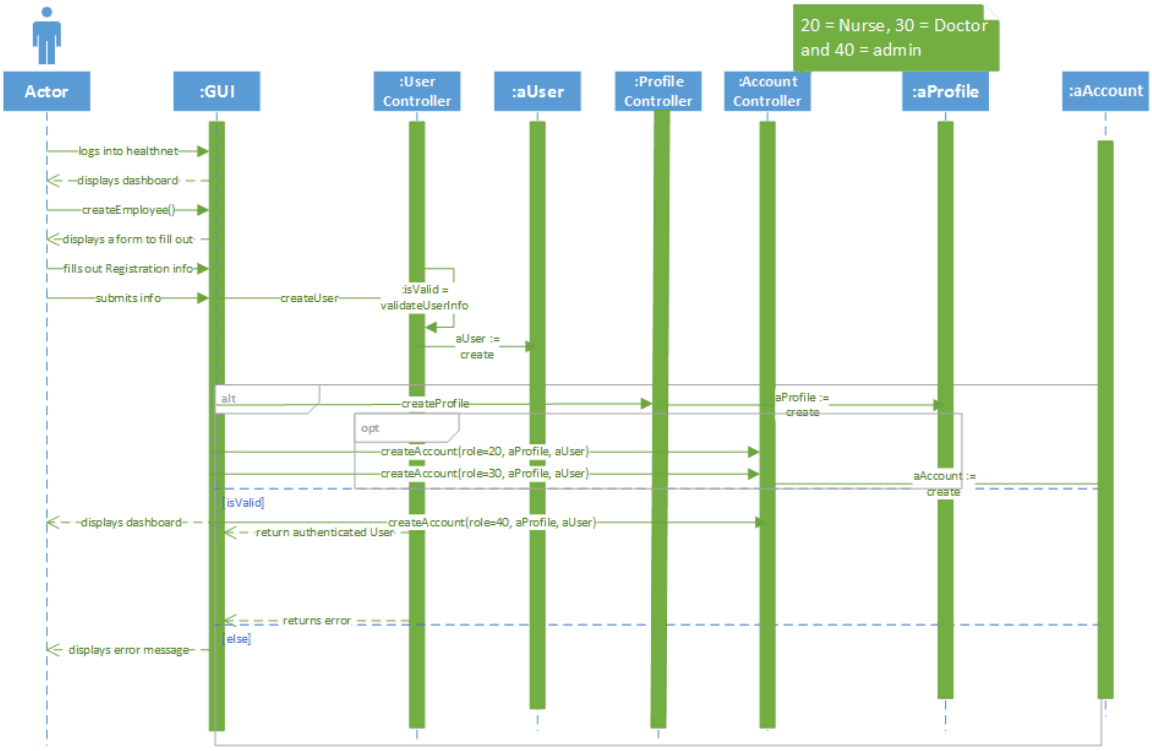
6. View Activity Log



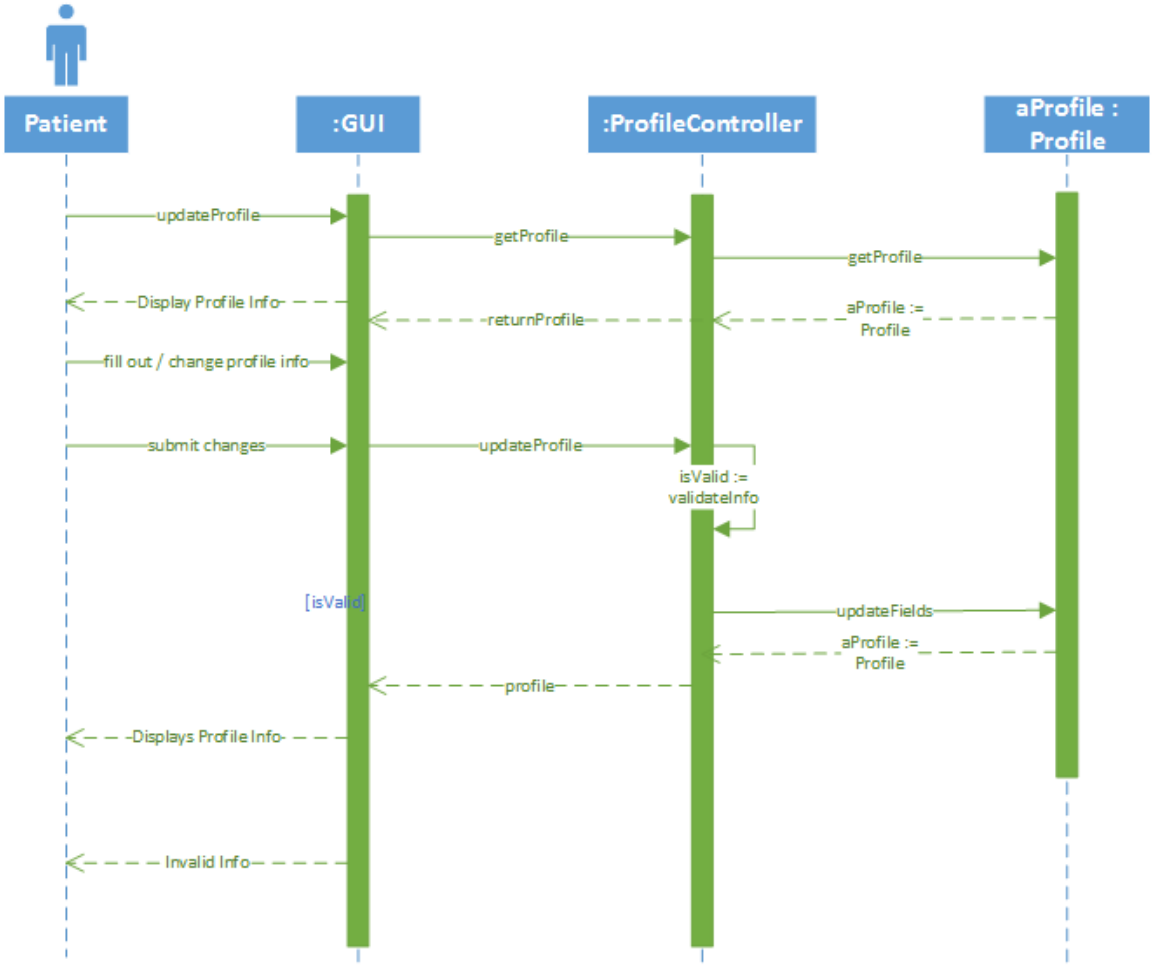
7. Patient Registration



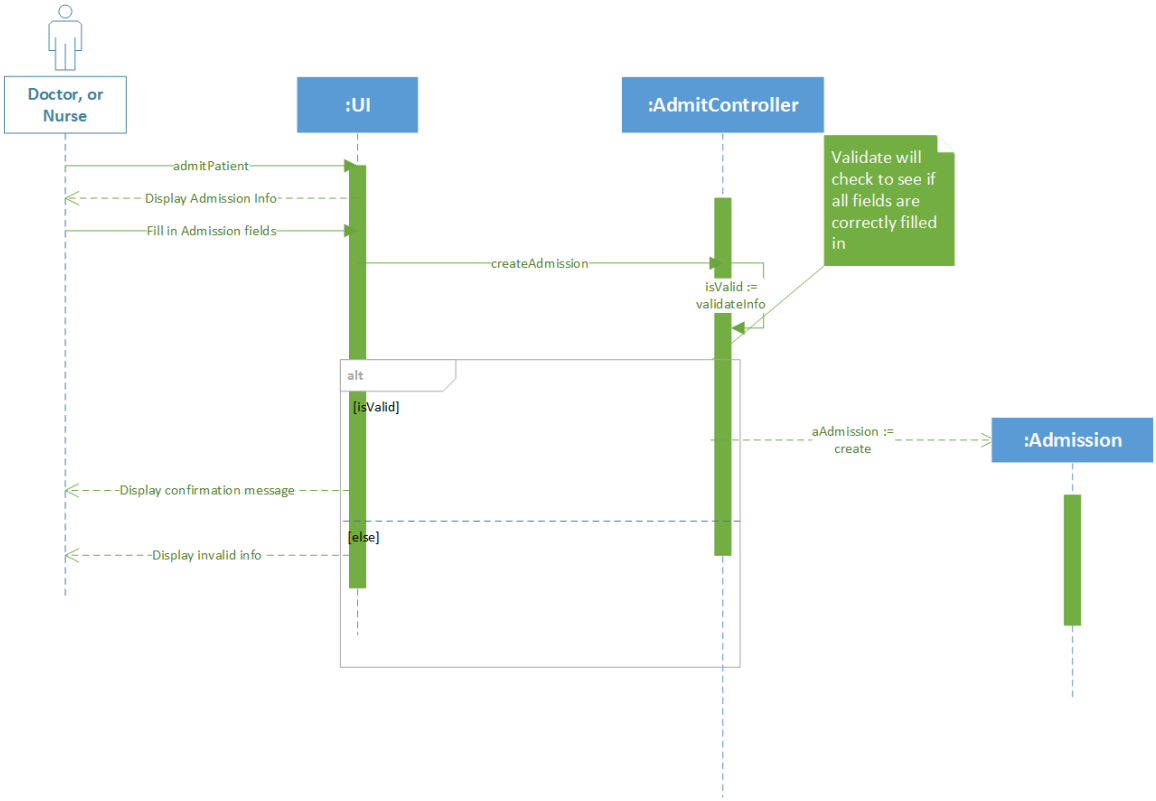
8. Admin Registration



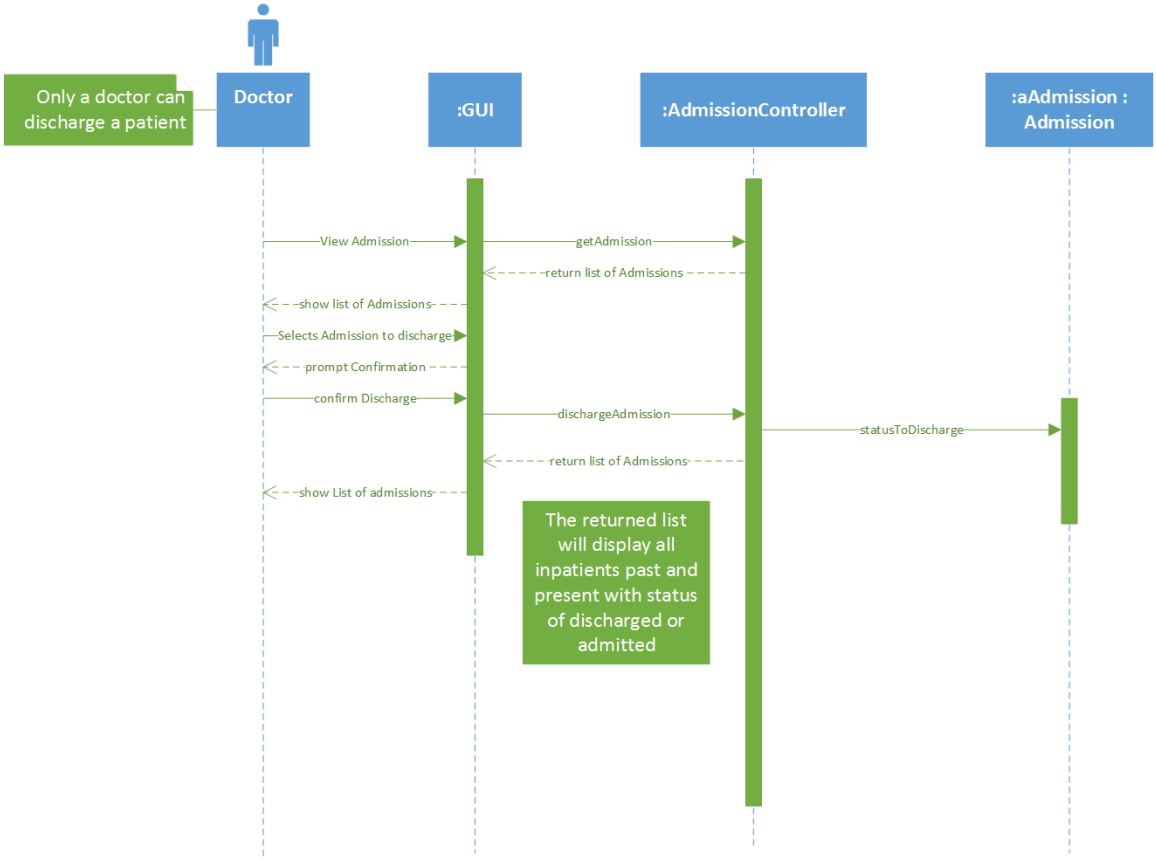
9. Update Patient Profile Information



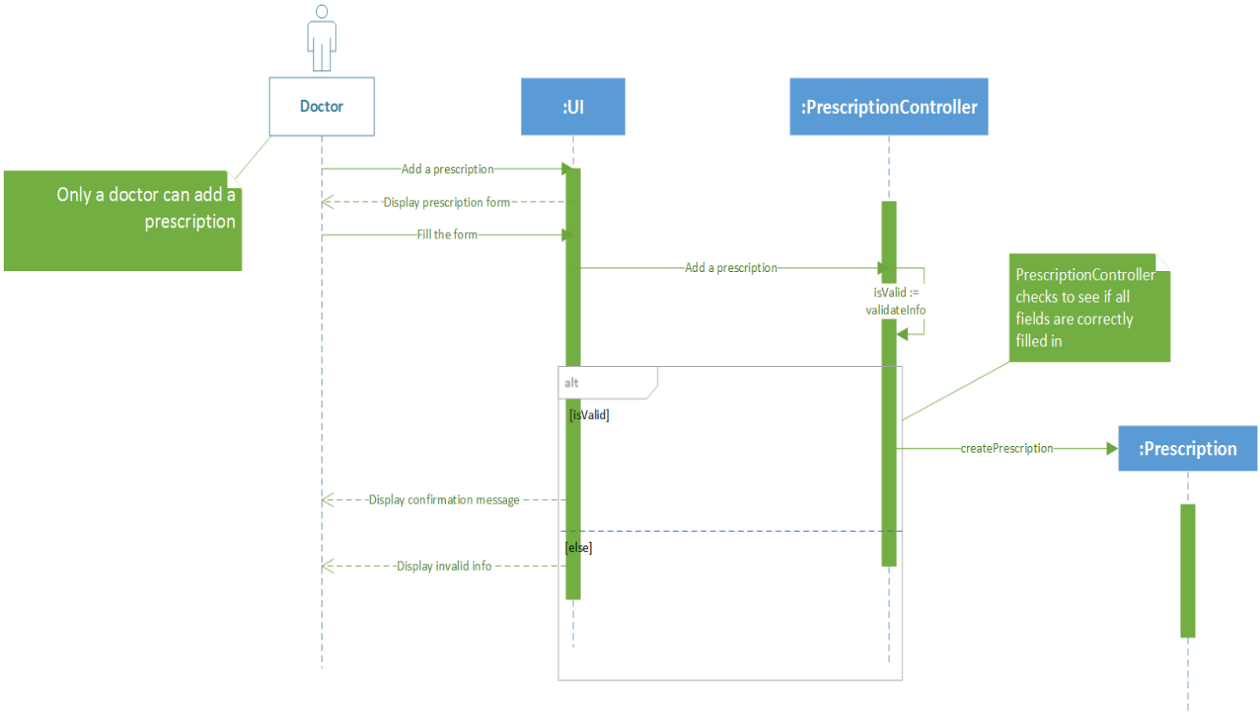
10. Admitting a Patient



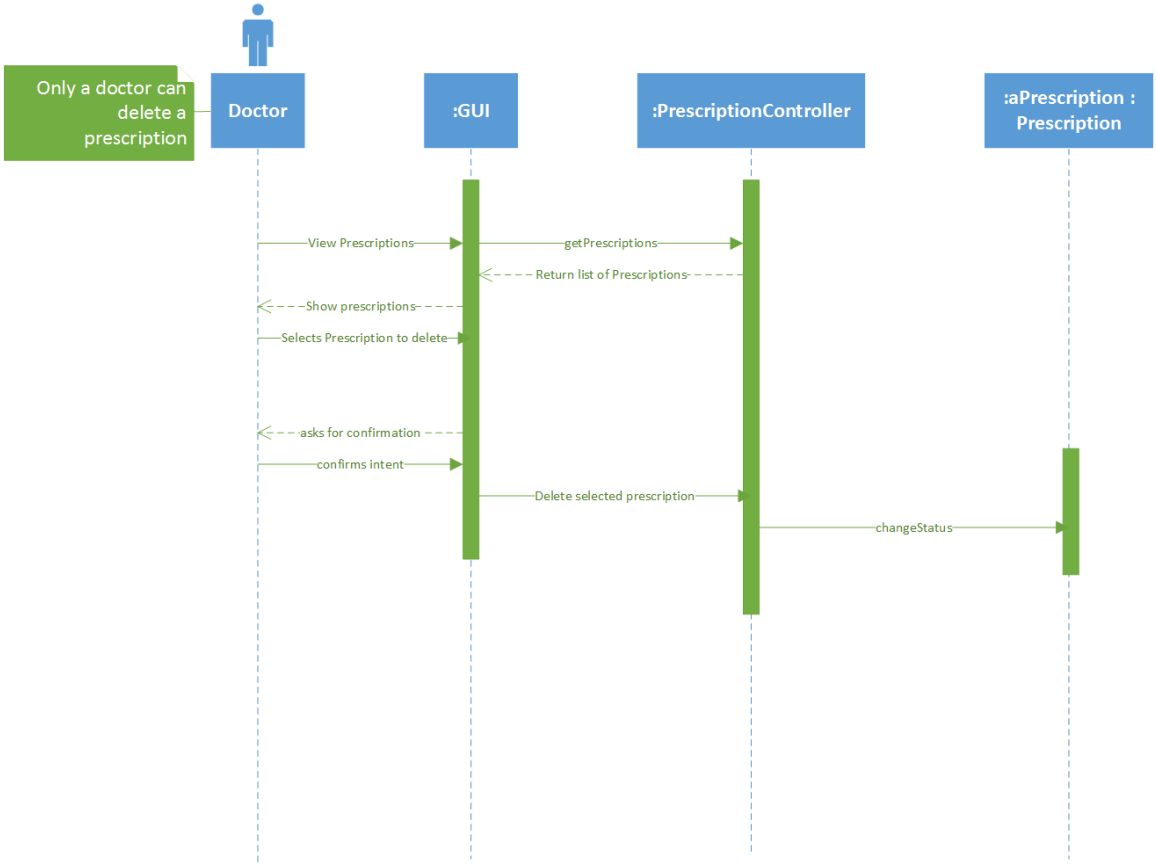
11. Discharging a Patient



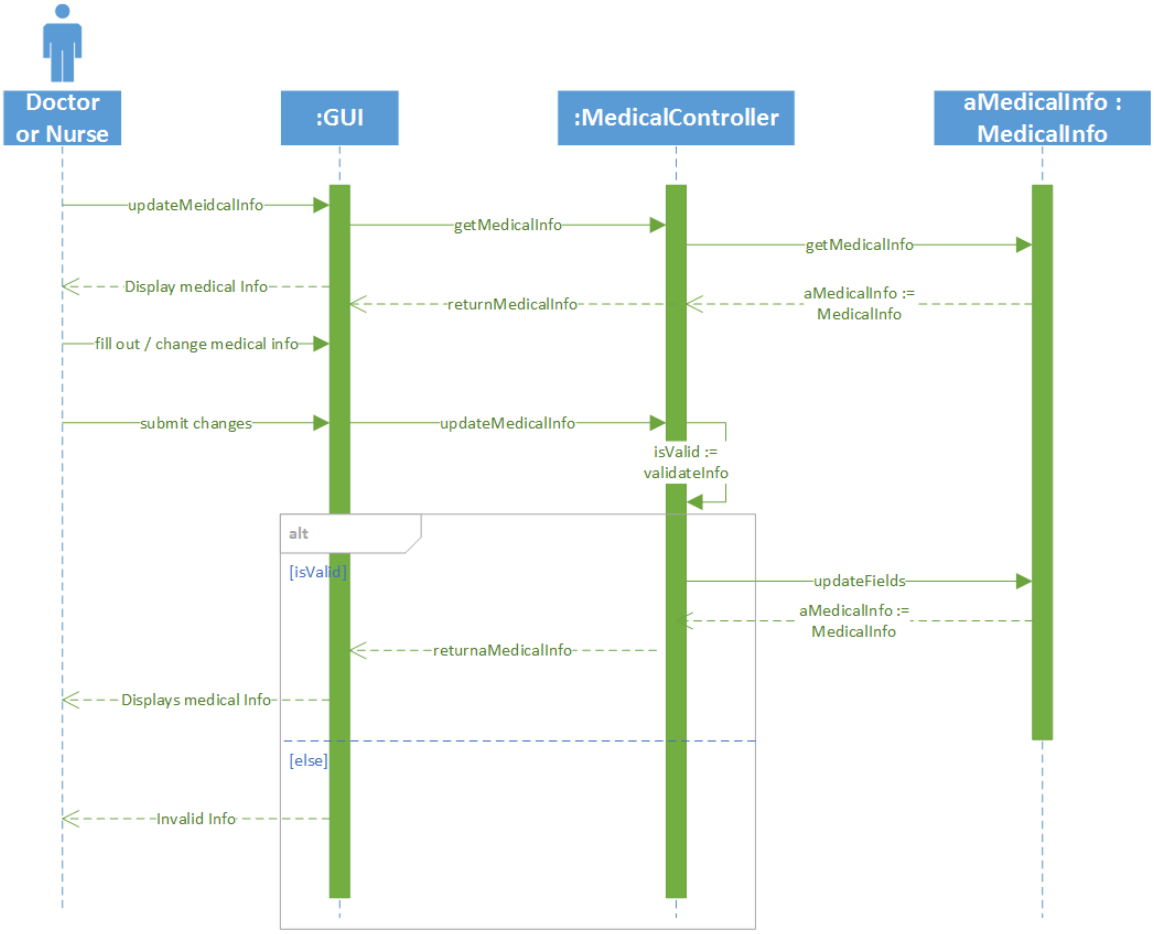
12. Add a Prescription



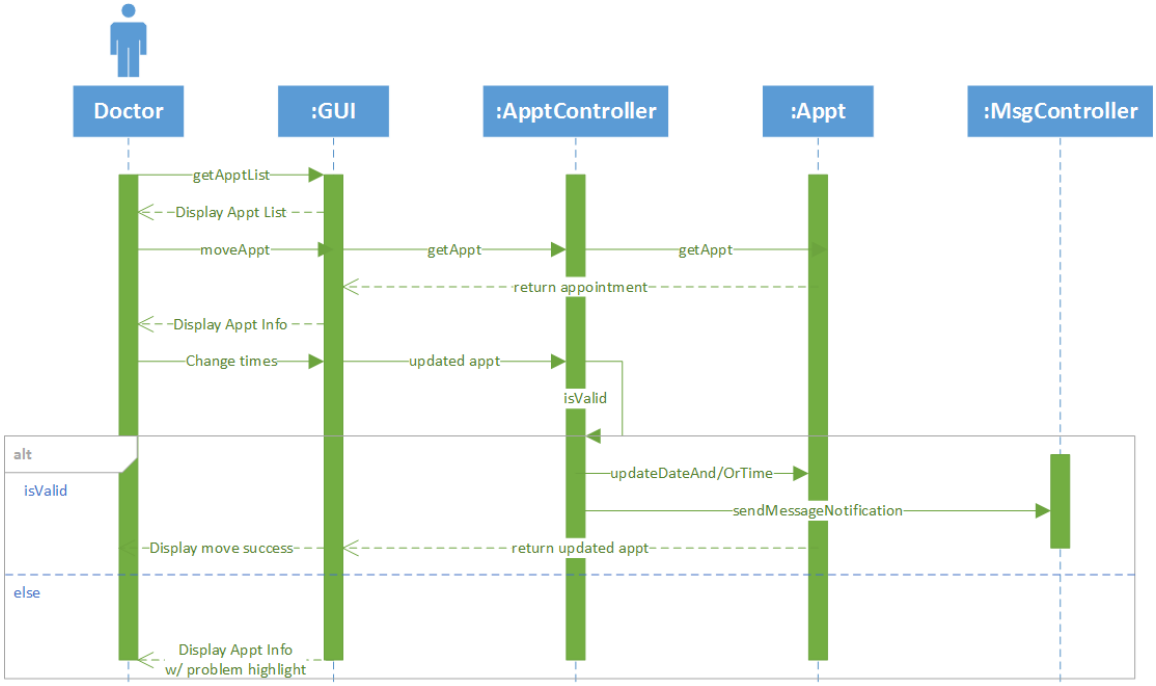
13. Remove a Prescription



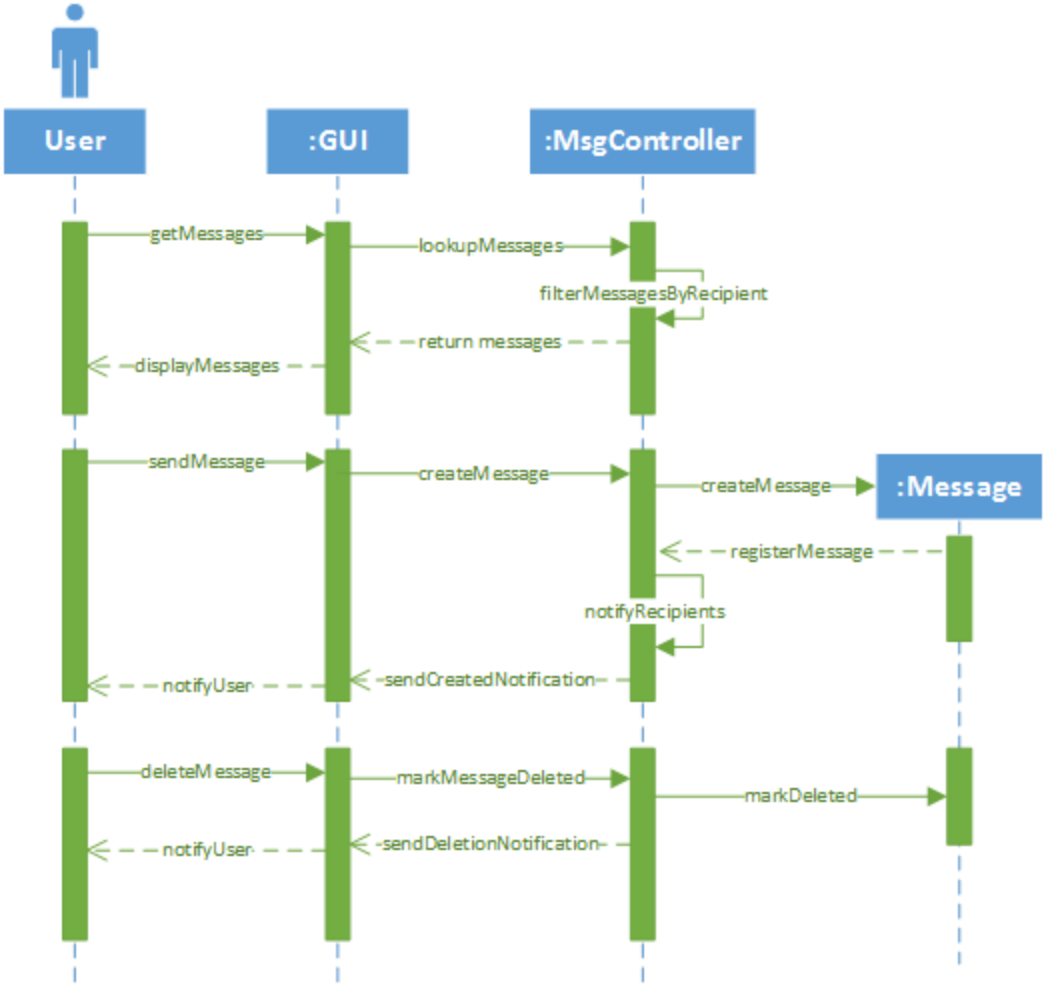
14. Updating Patient Medical Info



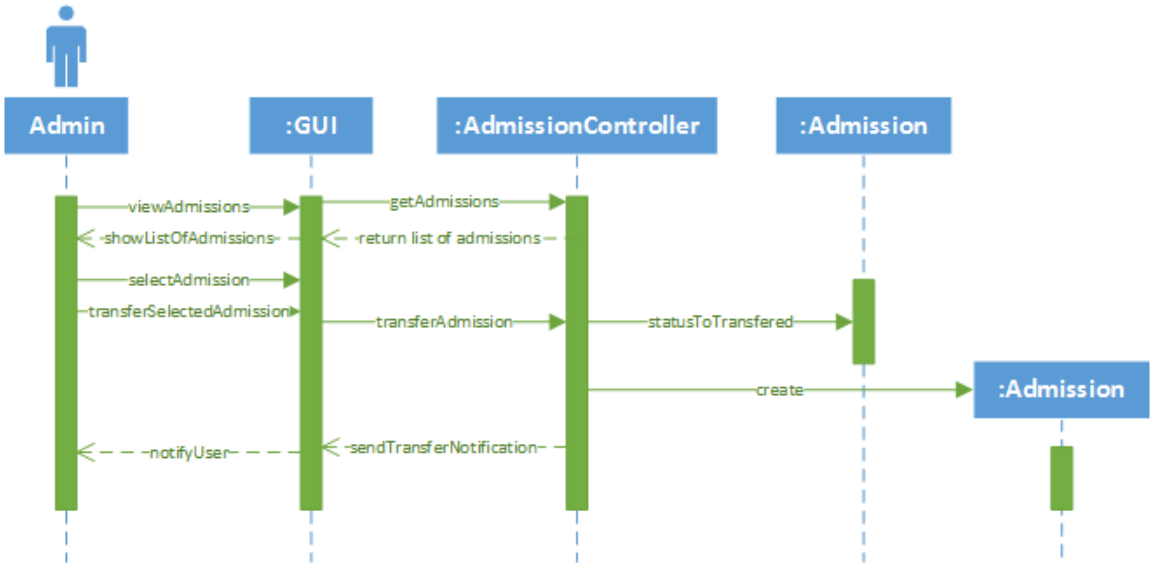
15. Move Appointment



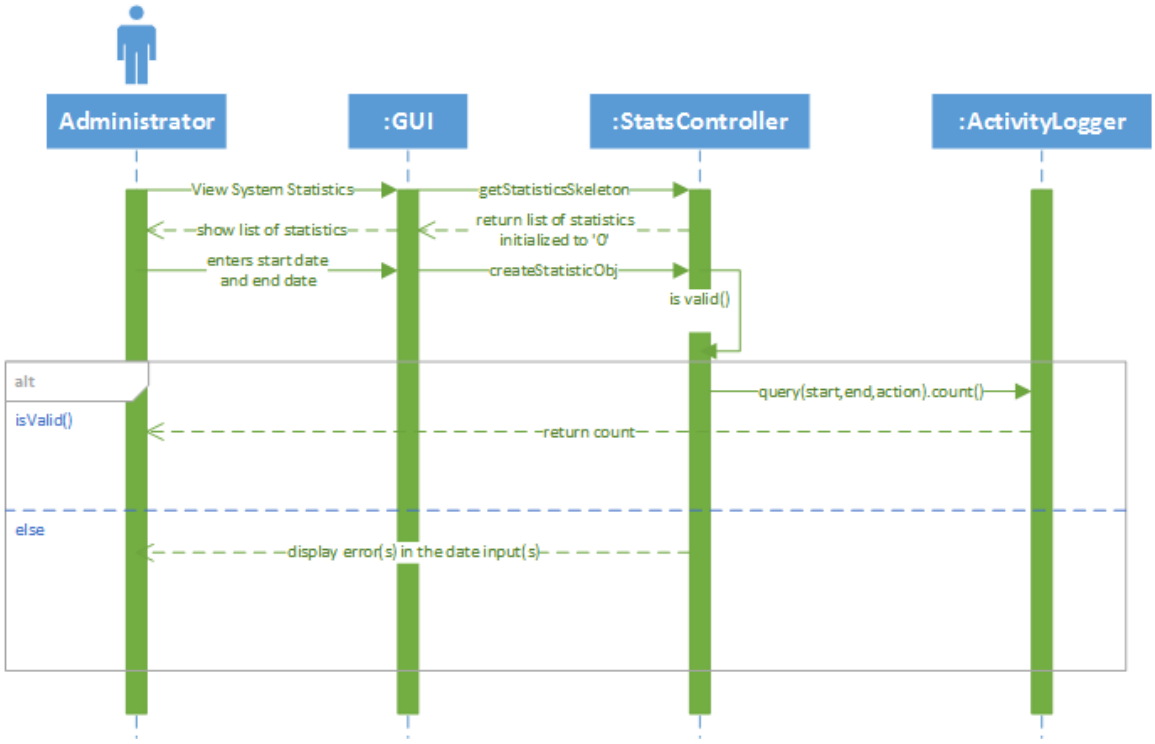
16. Send Private Message



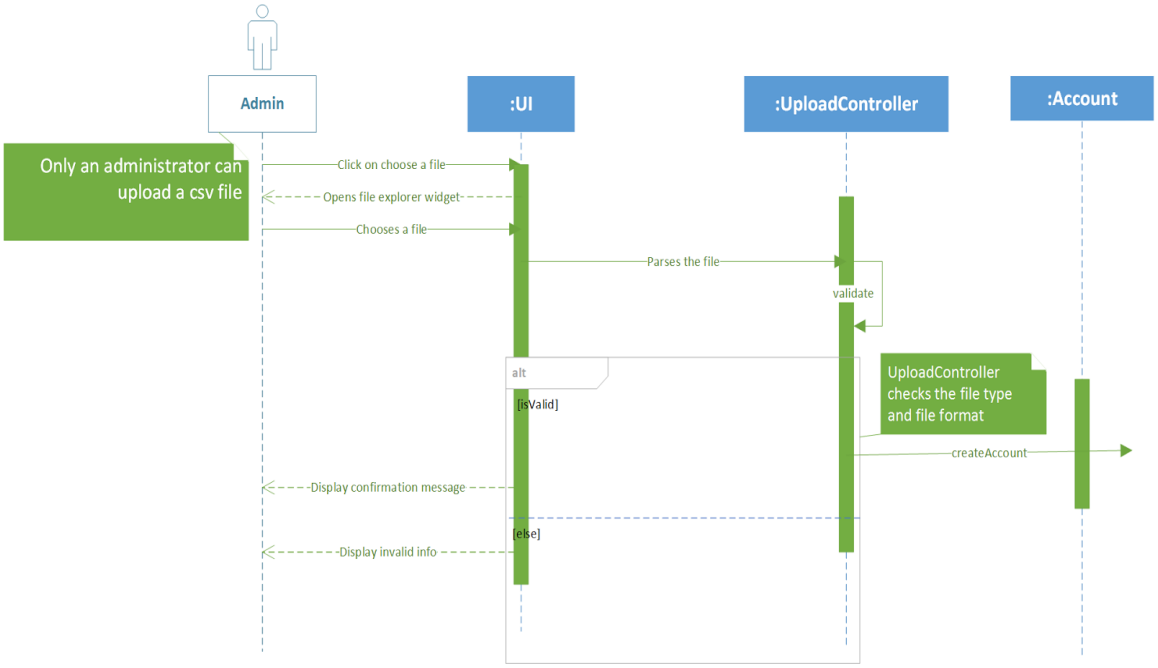
17. Transfer Patient



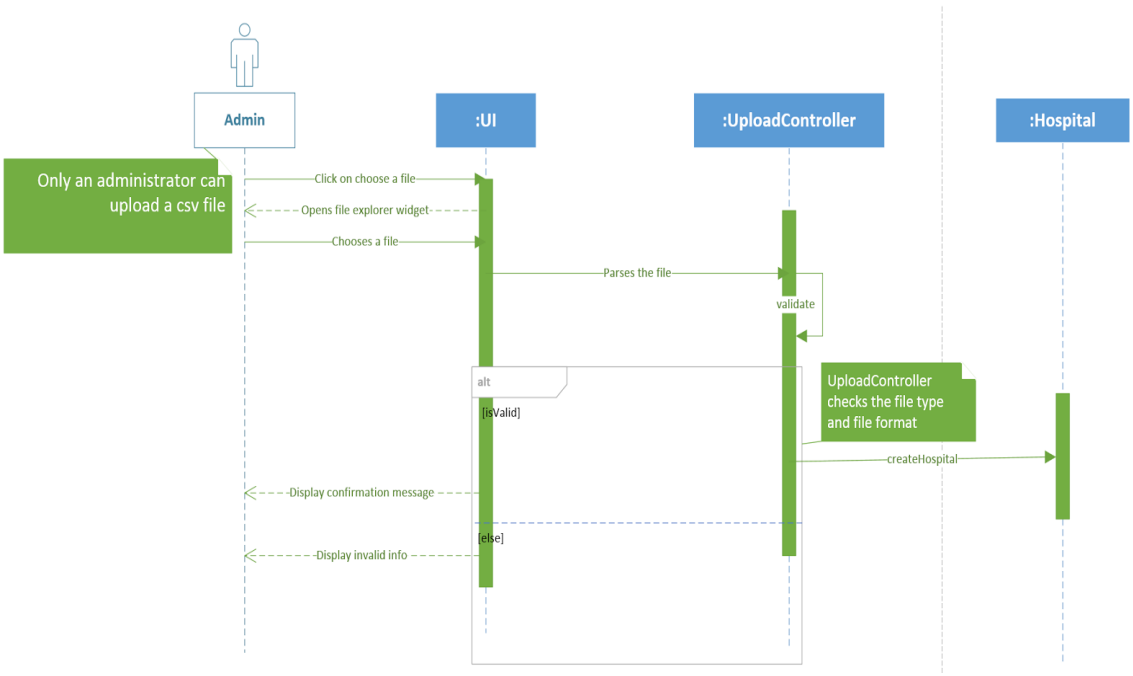
18. Viewing System Statistics



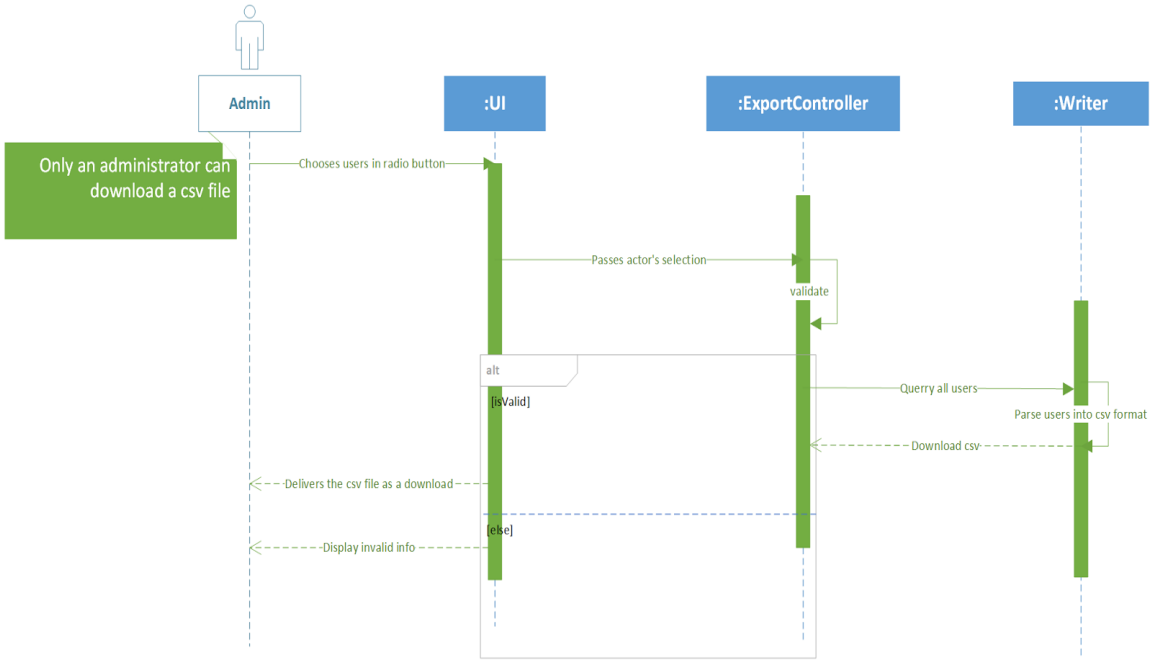
19. Import Users



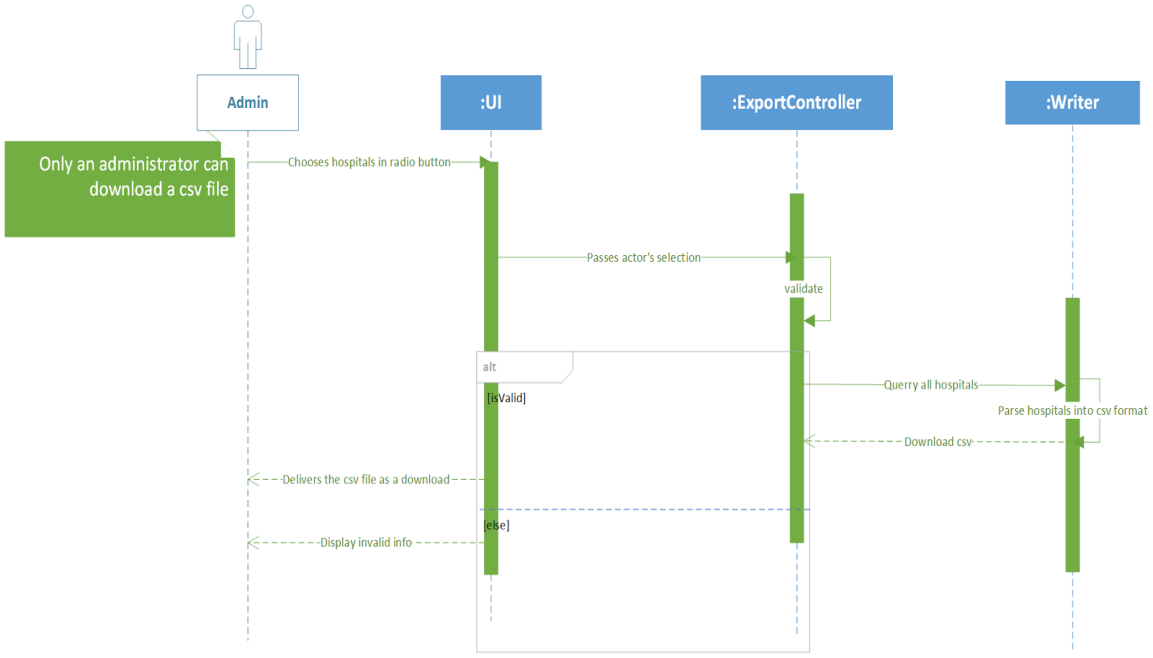
20. Import Hospitals



21. Export Users



22. Export Hospitals



Design Rationale

Rationale Id	1
Initial Idea	Initially we decided on an abstract User class that Doctor, Nurse, Administrator, and Patient would inherit from. From an object oriented point of view, this would be the most organized situation.
Problem	In our research into Django, we found that in version 1.6, multiple inheritance in models can result in possible data loss without warning.
Solution	We decided to elevate the functionality of SQL and try to work around the inheritance problem using foreign keys. We will have a basic profile model that holds all the standard information such as name, age, and contact info. Then we will have a user model which hold the user's role (Patient, Nurse, Doctor, Admin) as well as a foreign key to the profile. Having this layout will allow a lot of flexibility as we can add additional foreign keys to the user model in the future such as medical records or chat history.

Rationale Id	2
Initial Idea	We considered having some form of the Patient, Nurse, Doctor, and Admin class extend from the user class.
Problem	After trying to come up with ideas on how we would link to a user's profile page, we realized it would be difficult to go from the user object to say a doctor object.
Solution	Instead, the user will have a integer representing their 'rank' so to speak. A patient will have a rank 10, nurse with 20, doctor with 30, and admin with 40. This allows for a clear cut permission system that has room to expand for new types of users. This also means will will only have one object representing a user instead of 4 different objects.

Rationale Id	3
Initial Idea	To have a list of locations attached to a profile in the form of strings.

Problem	Having a list in SQL appeared to be problematic. There was a workaround by serializing the list when saving it into the database and deserializing it when retrieving it, but this proved to be too much work.
Solution	Elevating the fact that SQL is a relational database, we decided to make a separate model for locations and therefore a separate table. The location model contains a user and a hospital and there will be a zero to many relationship of locations users. Is a user is part of two hospitals, there were be two entries in the location table for that user.

Rationale Id	4
Initial Idea	Initially, appointments had one start and one end time.
Problem	After meeting with our client, we needed to have the ability to show different times. We needed the original time and a modified time in case the appointment was moved.
Solution	The solution is to have two start times, one for the doctor and another for the patient. When the doctor pushes an appointment back, the start time on his end is changed, but the start time on the patent's end remains the same.

Rationale Id	5
Initial Idea	Initially, we wanted to have the hospital be an attribute
Problem	After deciding that we will have a location table that will have foreign keys to the user, it would be cumbersome to have the hospital be an attribute as we would need to be able to reference the hospital frequently.
Solution	We decided for the hospital to be its own model instead because this way, we can simply put a foreign key in the location table that links to a hospital. Additionally, we can add new fields to hospital without interfering with other models.

Rationale Id	6
Initial Idea	We wanted to have the tools to create, update, move, and cancel an appointment be in the user class.
Problem	While creating the class diagrams, we saw that this left a lot of the functionally for the appointment model split between two models.
Solution	We settled on having the user be left with a createAppointment method for convenience and pushing the rest of the methods to the appointment class.

Rationale Id	7
Initial Idea	Initially for Prescription class, we decided to have an attribute of type integer called dosage to instruct the patient how many milligrams of medication to take daily.
Problem	The patient had to do some calculation to figure out the amount of tablets or capsules he/she needed to take daily. It was not user friendly and intuitive for the patient.
Solution	After a group discussion and some research, we decided to have an attribute named instruction of type string, instead of dosage, for the doctor to input the instruction.

Rationale Id	8
Initial Idea	Initially we thought the admission model would have the attributes for date and time be accessible at the time the form was filled out, meaning the user could pick anytime for which the patient was admitted.
Problem	The doctor/nurse can pick a time which is grossly off from the time that the patient was admitted in actuality
Solution	After a meeting Dr.House (customer) he explained that he would rather have them be automatic as in the the time that the form was filled out would be the time and date thus not accessible by the user.

Rationale Id	9
Initial Idea	Initially for the discharge of a patient we thought we would delete the selected patient from the inpatient list.
Problem	If the patient was deleted from the inpatient it would be harder to log and would not give the doctor/nurse the opportunity to view old inpatient or if a patient was recently discharged.
Solution	After a meeting with our customer we realized it would be better not to delete the selected patient from the list but rather give them a status for whether they have been discharged or are still admitted in the hospital

Rationale Id	10
Initial Idea	Initially we thought that the sidebar in the UI would encapsulate tabs for both the listing of appointments/admissions and creation of them.
Problem	The UI would look very cluttered and may distract the user. We want to remember to keep the memory load low for the user.
Solution	After a meeting with our customer we realized it was time to change around our UI. Lists and creations became a joined tab thus making it easier to navigate and decreasing memory load for the users.

Rationale Id	11
Initial Idea	Initially we thought that the Admissions model would only need a reason for admissions meaning no need for a field for description.
Problem	Reasons can vary and may be easier for doctors/nurses to have a drop down menu of reasons for admission for faster submission of admissions.
Solution	We decided that we would have a field for the reason of admission. These admissions were taken from a statistical analysis done across the the U.S. for the most common reasons of hospital admissions. Also an "other" option was added just incase a reason in the list was not sufficient. The admission will also be supported by a description pertaining to the admission but not required like a reason will be.