Product Design

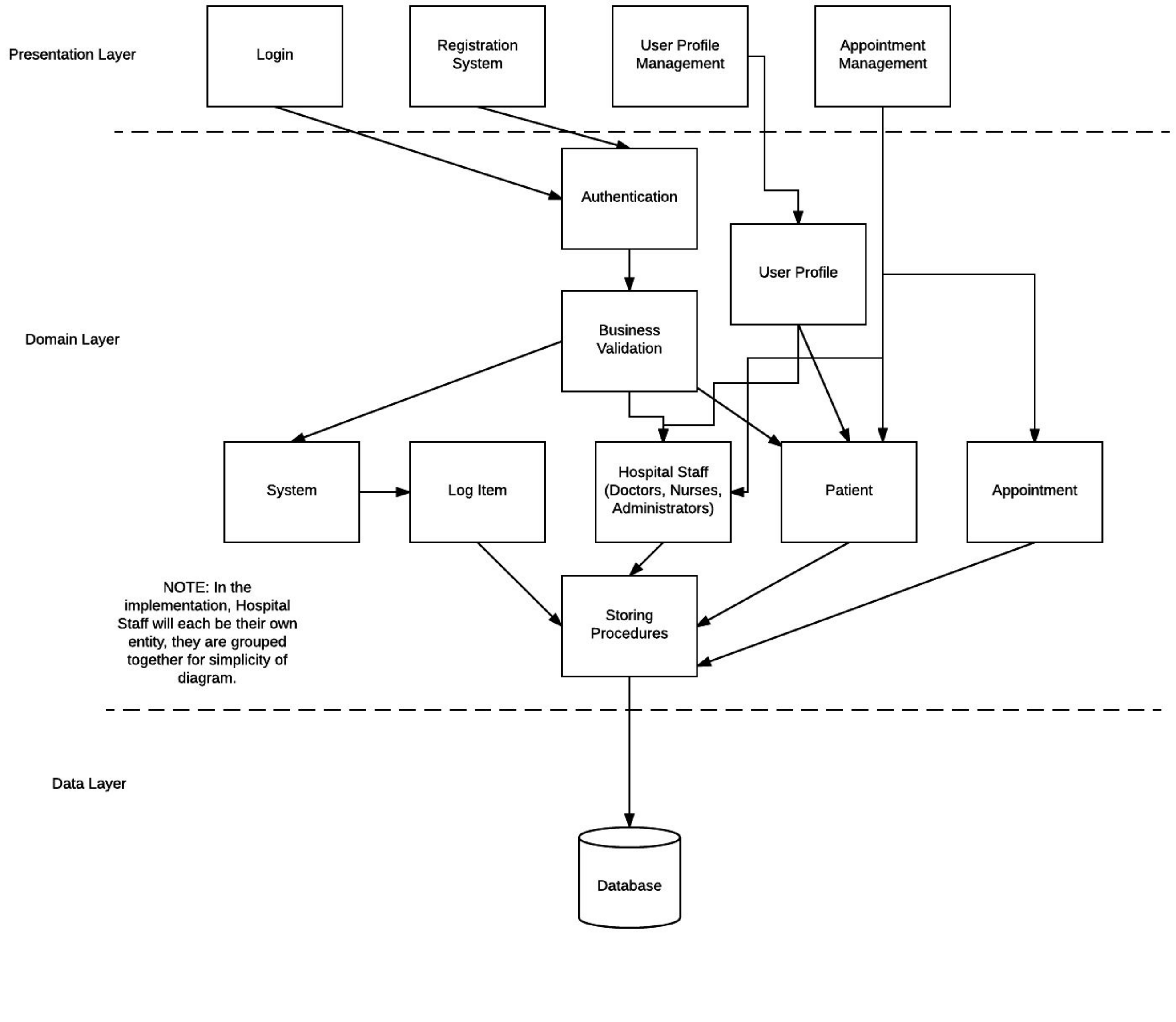
201661-12-SWEN-261-TEAM-2-**Grewp2**

Revision Number	***Revision Date***	***Summary of Changes***	***Author(s)***
0.1	*09/14/16*	*Initial revision*	*Nick Deyette, Kyler Freas, Umang Garg, Smruthi Gadenkanahalli*
0.2	*09/28/16*	*Removed the given architectural model. Modified the created one to incorporate specific elements from the HealthNet project in the domain layer.*	*Nick Deyette*
0.3	*10/01/2016*	*Updated Sequence Diagram to reflect the classes being used for R1*	*Smruthi Gadenkanahalli*
0.4	*10/02/16*	*Updated Class Diagram*	*Umang Garg*
0.5	*10/03/16*	*Updated Class Diagram*	*Umang Garg Nick Deyette*
0.6	*10/03/16*	*Added the new Design Rationale, Updated the Architectural Model to include UserProfile class*	*Nick Deyette, Kyler Freas, Umang Garg, Smruthi Gadenkanahalli*
0.7	*10/03/16*	*Modified Components and Functions table to more closely reflect model elements*	*Kyler Freas*
0.8	*10/16/16*	*Added Class Diagram for R2*	*Umang Garg*
0.9	*10/19/16*	*Updated Class Diagram for R2*	*Umang Garg*
1.0	*10/24/16*	*Updated Design Rationale*	*Nick Deyette*
1.1	*11/05/16*	*Updated the Components and Functions table and the Design Rationale*	*Nick Deyette*

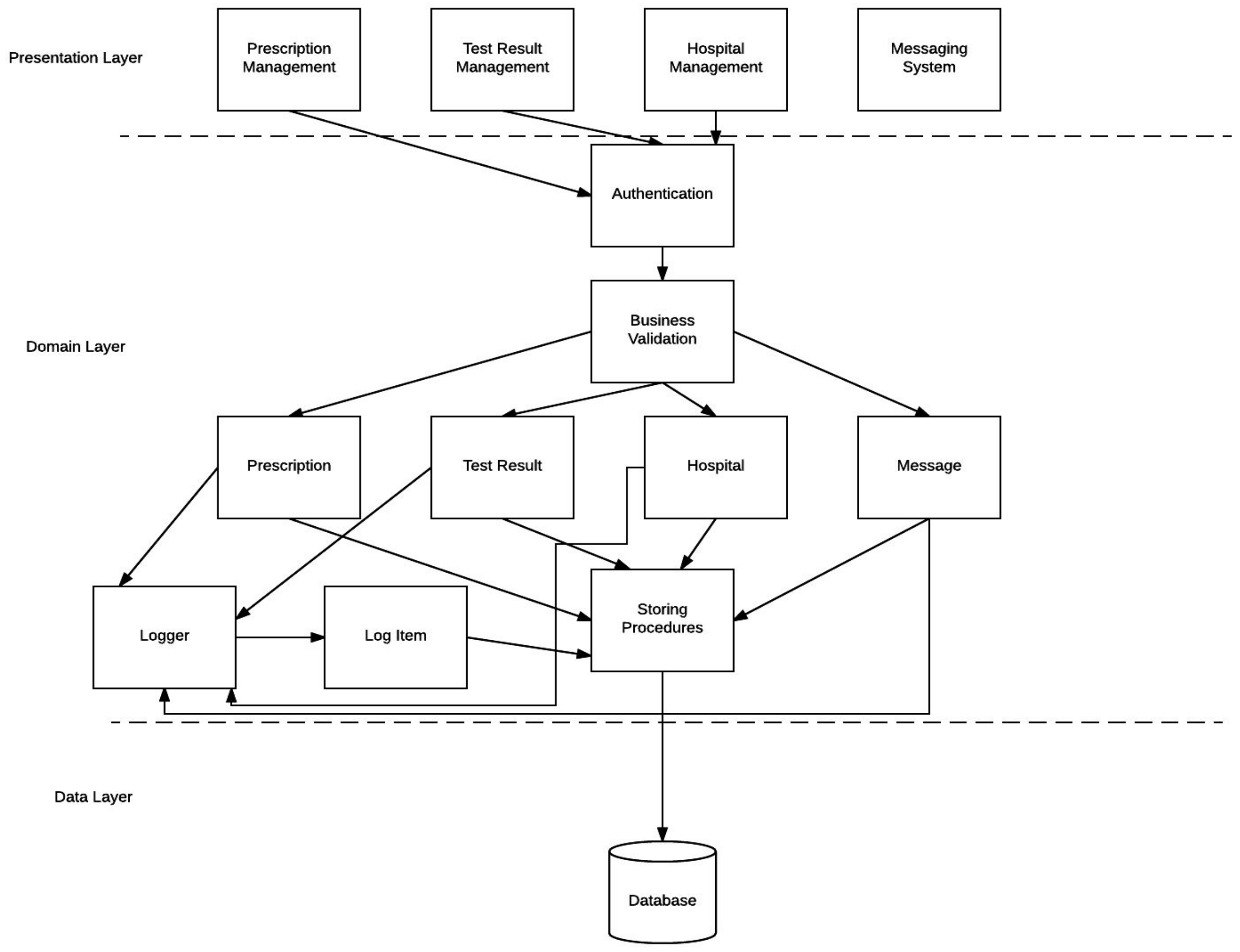
Architectural Model

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.

R1 Architectural Model



R2 Architectural Model



Components and Functions (R1/R2)

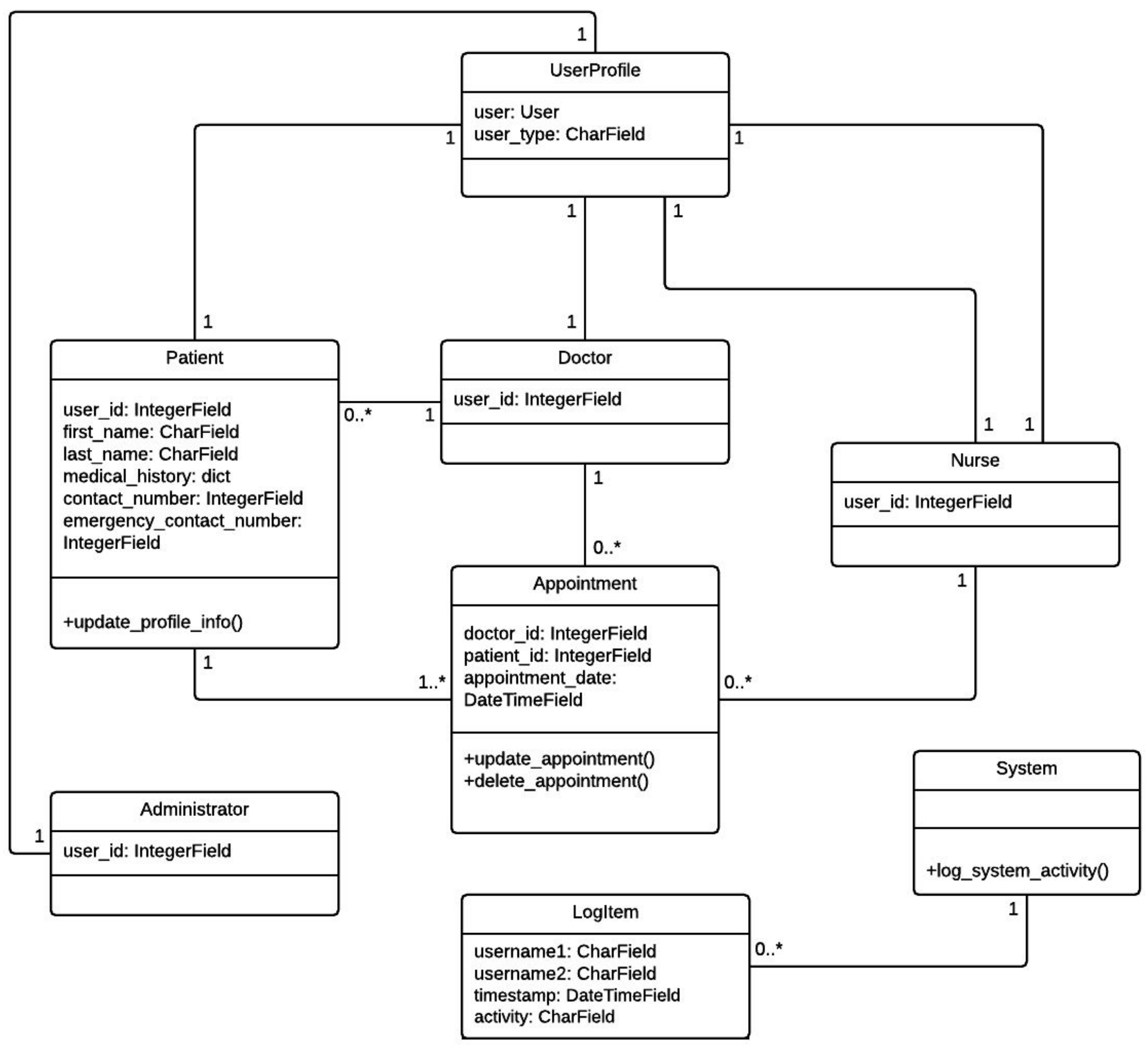
User Component	**Component state** ● Username ● Name ● Password **Component behavior** ● Gives a user the ability to: ○ Register ○ Log in * The built-in User class in Django will be imported for use. While the User class provides many more behaviors than are listed above, these are the ones which will be utilized in our design
User Profile	**Component state** ● User ● User type (i.e. Doctor, Nurse, Patient, Admin) **Component behavior** ● Allows the application to determine a given user's type
Patient	**Component state** ● User id ● First Name ● Last Name ● Contact number ● Emergency contact number ● Preferred Hospital ● Current Hospital ● Insurance ID

	• Insurance Company • Address • City • State • Zipcode • Country • Age • Weight • Height **Component behavior** • Allows the app to determine user permissions (can update own appointments) • Update_profile_info: allows patient to update their profile info • Update medical information • Admit/Discharge - adds or removes patient's Hospital • Transfer to hospital
Doctor, Nurse, Administrator	**Component state** • User id • Hospital **Component behavior(s)** • Allows the app to determine user permissions ○ Doctor can update own appointments ○ Nurse can update (but cannot delete) appointments ○ Administrator can create new users, view system logs
Appointment	**Component state** • Doctor id • Patient id • Start • End • Location **Component behavior** • Created/deleted from appointment views • Update_appointment: Allows user to make changes ○ Doctors can change patient, appointment date ○ Patients can change doctor, appointment date ○ Nurses can change patient, doctor, appointment date
Log Item	**Component state** • Username 1 • Username 2 (for account creation; specifies user that was created by admin) • Timestamp • Activity • User type 1 • User type 2

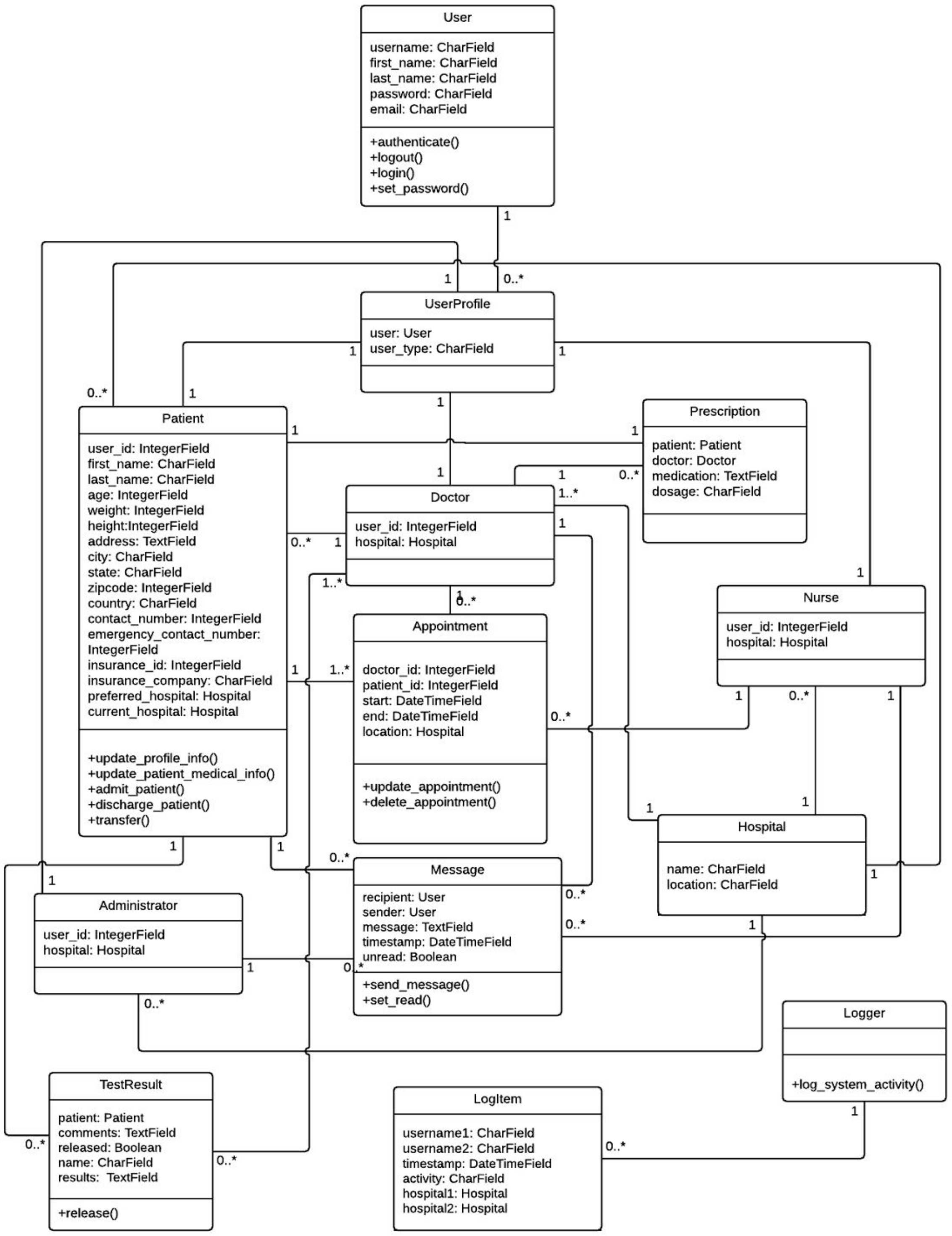
	<ul><li>Hospital 1</li><li>Hospital 2</li></ul>**Component behavior**<ul><li>Is automatically generated when<ul><li>A new user is created/registered</li><li>A user logs in or logs out</li><li>An appointment is created, updated or deleted</li><li>A user updates their profile or medical information</li><li>A patient is admitted or discharged from a hospital</li><li>A patient exports their profile information</li></ul></li></ul>
Logger	**Component state**<ul><li>Activity logs are stored as Log Items</li></ul>**Component behavior**<ul><li>Provides static methods for system/database operations<ul><li>Log_system_activity: creates a new activity log item</li></ul></li></ul>
Prescription	**Component state**<ul><li>Patient</li><li>Doctor</li><li>Medication</li><li>Dosage</li></ul>**Component behavior**<ul><li>Can be added or removed (via save() and delete())</li></ul>
Test Result	**Component state**<ul><li>Patient</li><li>Comments</li><li>Released</li><li>Name</li><li>Results</li></ul>**Component behavior**<ul><li>Release to patient</li></ul>
Hospital	**Component state**<ul><li>Name</li><li>Location</li></ul>**Component Behavior**
Message	**Component State**<ul><li>Recipient</li><li>Sender</li><li>Message</li><li>Timestamp</li><li>Unread</li></ul>**Component Behavior**

	● Send message ● Set read

Class Diagram (R1)

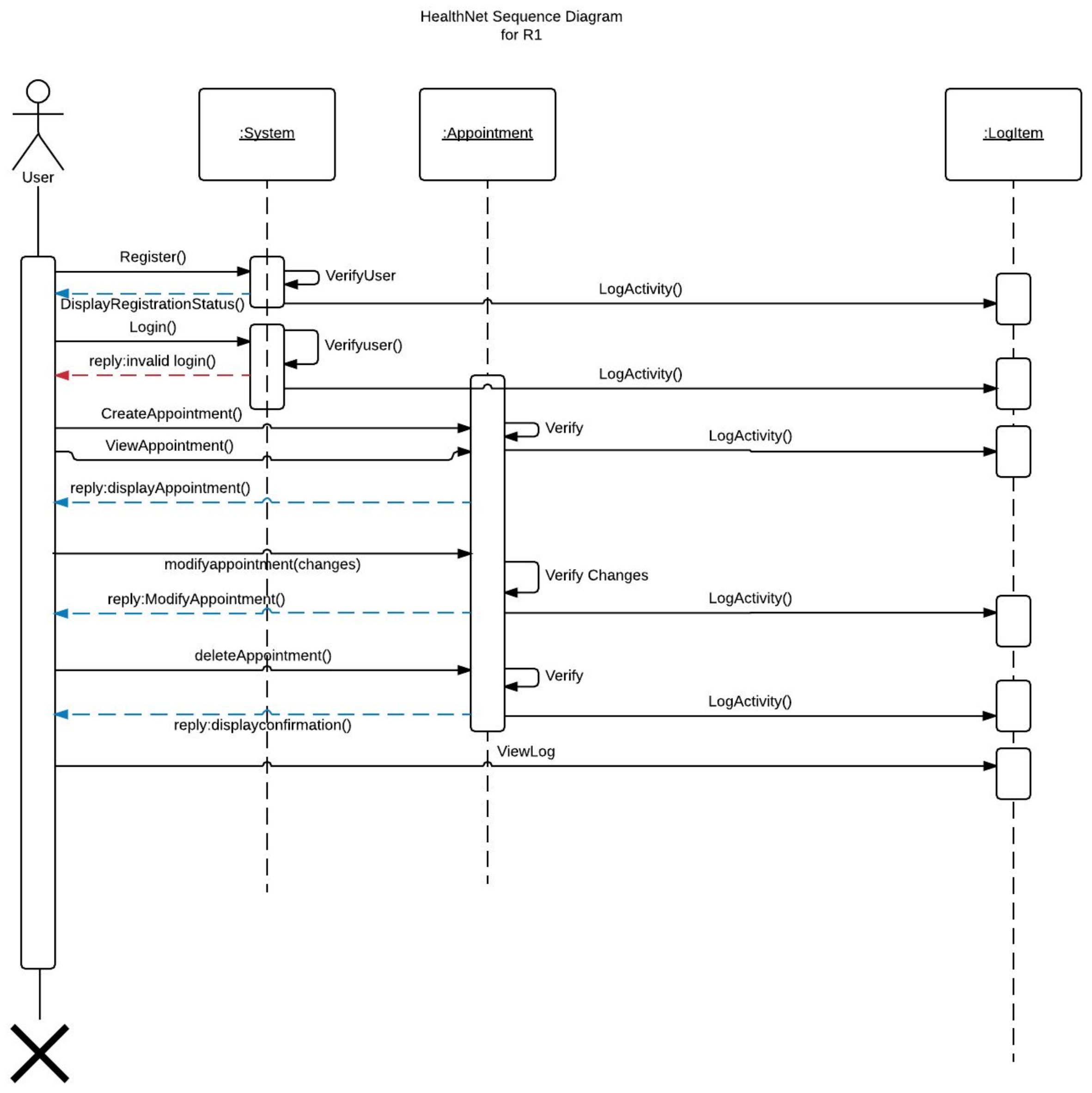


Class Diagram (R2)



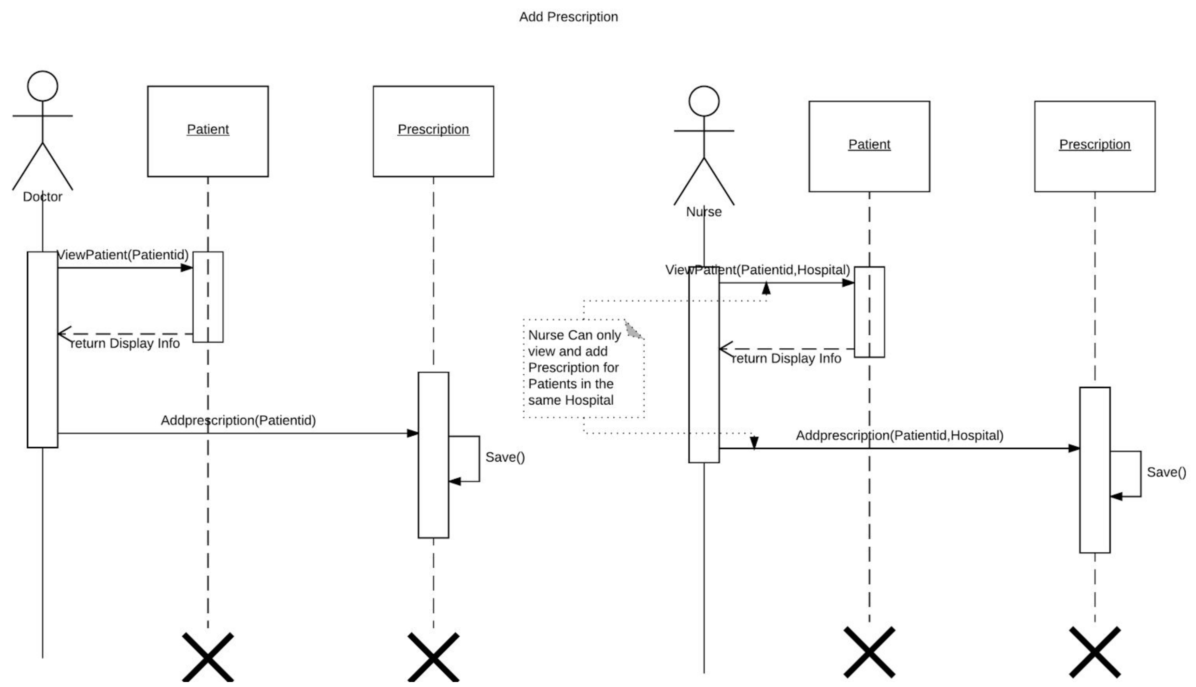
Sequence Diagram(s)

R1 Sequence Diagram

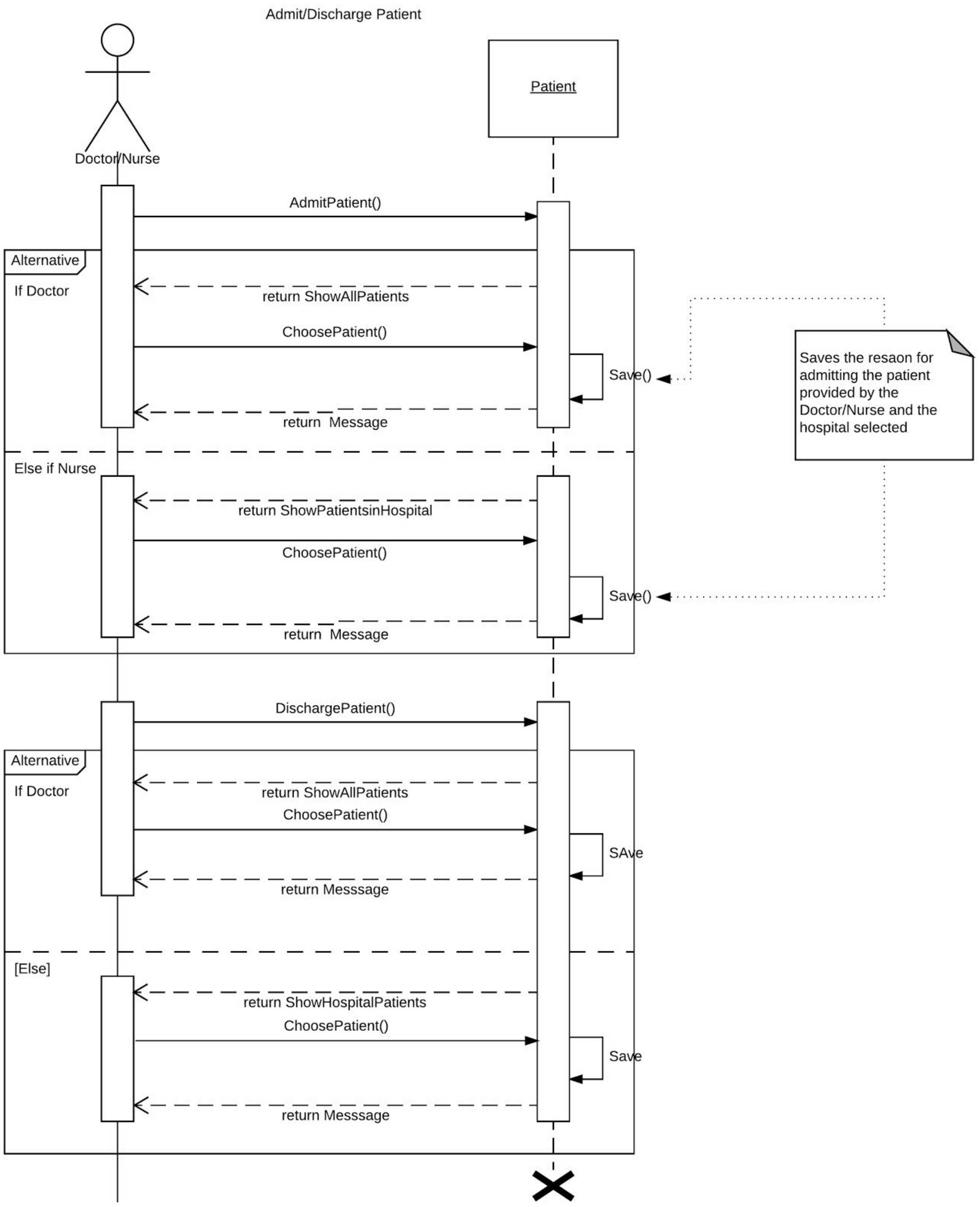


R2 Sequence Diagrams

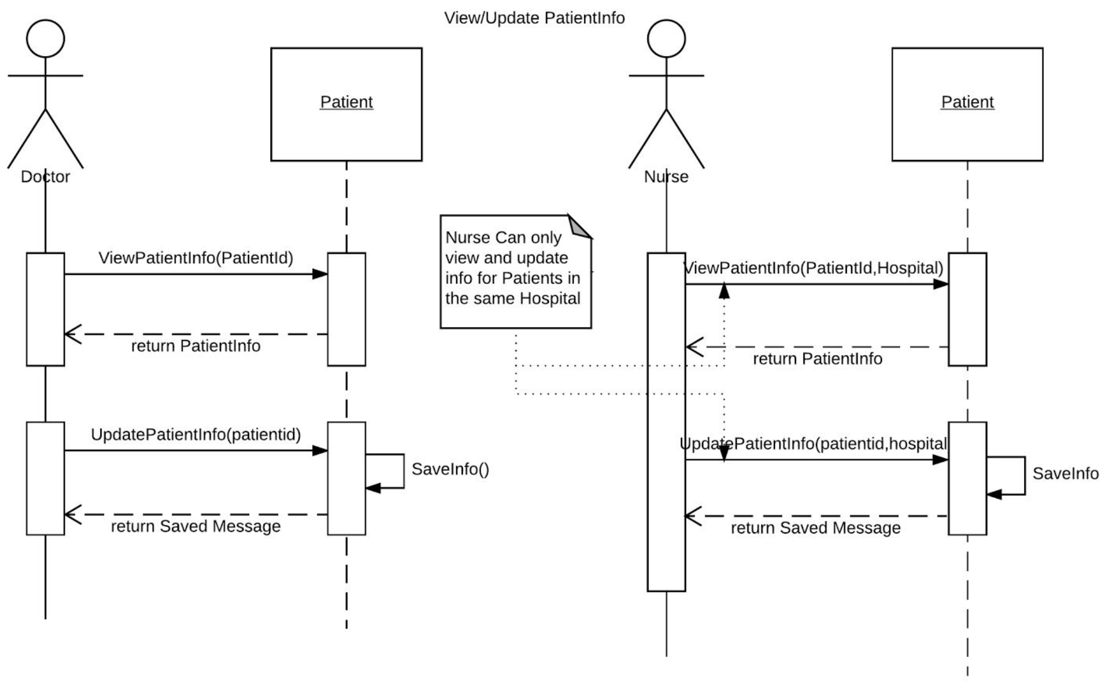
1. Add prescription


Add Prescription
Doctor
Patient
Prescription
ViewPatient(Patientid)
return Display Info
Addprescription(Patientid)
Save()
Nurse
Patient
Prescription
ViewPatient(Patientid,Hospital)
Nurse Can only view and add Prescription for Patients in the same Hospital
return Display Info
Addprescription(Patientid,Hospital)
Save()

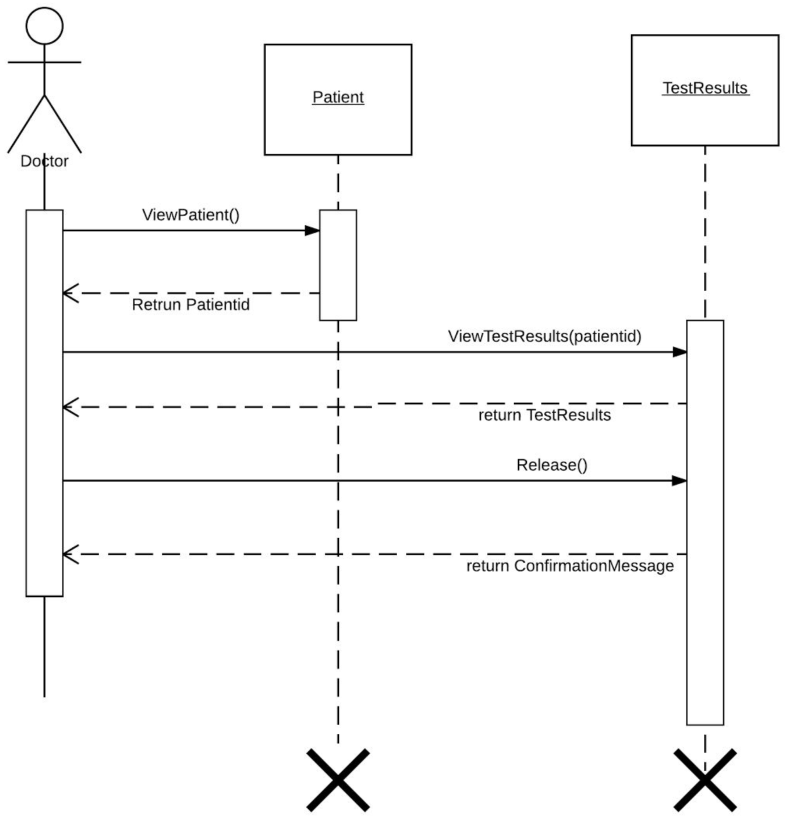
2. Admit/Discharge Patient



3. View/Update Patient


View/Update PatientInfo
Doctor
Patient
ViewPatientInfo(PatientId)
return PatientInfo
UpdatePatientInfo(patientid)
SaveInfo()
return Saved Message
Nurse Can only view and update info for Patients in the same Hospital
Nurse
Patient
ViewPatientInfo(PatientId,Hospital)
return PatientInfo
UpdatePatientInfo(patientid,hospital
SaveInfo
return Saved Message

4. Release Test Results



Design Rationale (R1/R2)

Since this is the initial revision of our design document, we have not yet had an opportunity to test the design. The first iteration will likely reveal issues with the design which will then need to be corrected. As these issues are revealed, we will make adjustments to our design, which will be recorded in this document.

As the project has continued and development is in full swing, many problems with our design became apparent. Firstly, some small changes were made, specifically changing the name of the “Appointment Calendar” class to be called “Appointment” for the sake of clarity. Next, the larger changes. Initially, we planned to implement our own User class that would store a Username and Password for the User. After doing some research into Django, we found it had a pre-written User class that handled the creation, password hashing and authentication of the User. We determined the time it would take to implement these features ourselves was not worth the short amount of time it would take to use Django’s class, so we decided to switch. Next, we decided to move around some of the methods we had for our models. Some of the methods simply did not belong where they were located, so we moved them. For example, admit_patient_to_hospital was originally in the Nurse class. We determined it was better placed in the Patient class. A few other of these moves include view_patient_medical_info being moved to Patient, view_patient_prescriptions being moved to Patient, and update_patient_medical_info moved to Patient. Next, we moved most of the “create…” method, as creating an instance of an object can be done in a view, which we felt would be easier when attempting to manage views and to keep our models from being cluttered. Ultimately, we wanted to aim for clean, concise models. The next significant change was to add a new state to Patient, Doctor, Nurse and Administrator, being user_id. This IntegerField is that id of the User object that these hospital users are linked to. This helps with navigation through the database and keeping instances of User and the hospital users linked. We then added a new class, LogItem. This class represents a single log item that will be created and logged by the System class. The LogItem has two usernames fields, which are the usernames of the users involved, a timestamp stating when the action occurred and a brief description of the activity itself. Finally, we created another new class, the UserProfile class. This class has no functionality in terms of the end user that is using the application, but rather makes development and database connections simpler. It’s attributes are an instance of the User class and a string that represents which type of user the associated User instance is. This allows us to keep track of which Users are Patients, Nurses, Doctors or Admins and helps with authentication.

For R2, we have decided to make a few changes from our R1 design. Firstly, we have added some new classes to allow for R2 functionality. These classes are Prescription, Message, Hospital and Test_Result. These classes will allow us to complete the required R2 functionality, such as manage prescriptions, sending messages, admit/discharge from hospitals and releasing test results. An important change for these classes compared to the R1 classes is that these classes are going to contain objects in their state. For example, the Test_Result class has a state for patient. This state will be the Patient object that the Test_Result belongs to. This will make querying easier. This is different from R1 because our R1 classes use a reference to an id which is connected to the object rather than the object itself. Finally, we've made some changes to our existing classes. Patient's now have state for insurance id, insurance company and their preferred hospital location. Patient also has three new methods: admit (admits patient to a hospital), discharge (discharges a patient from a hospital) and transfer (transfers a patient from one hospital to another). Doctors, Nurses and Administrators now have state for location, which is the hospital(s) they are available at. Appointments also have a state for location, which is the hospital that the appointment is scheduled for. These new states and methods will allow us to complete the functionality we have promised for R2.

After beginning the early development of R2, specifically the models and a very small part of the views, we decided we needed more changes to our models. Firstly, we added a set_read() method to the Message class, which marks the message as read. This will be useful in our message inbox view in order to determine how messages will be displayed. Next, we added new state to the Patient class: address. This state will hold the address of the patient. We also added age, weight and height state to the Patient class. Finally, we added a new method to the Patient class: update_patient_medical_info() which is going to be used for changes made to the patient's medical information. We also determined that we needed some more state in the Patient class: specifically a way to keep track of addresses and hospitals. So, we added address, city, state, zipcode and country fields to the Patient class. We also added a current_hospital field, which keeps track of the hospital the Patient is currently in and a preferred_hopsital field, which is the preferred hospital of the Patient.

The next change that was made was adding hospital1 and hospital2 state to the LogItem class. This is going to keep track of the hospitals that the users that are performing the action are at.