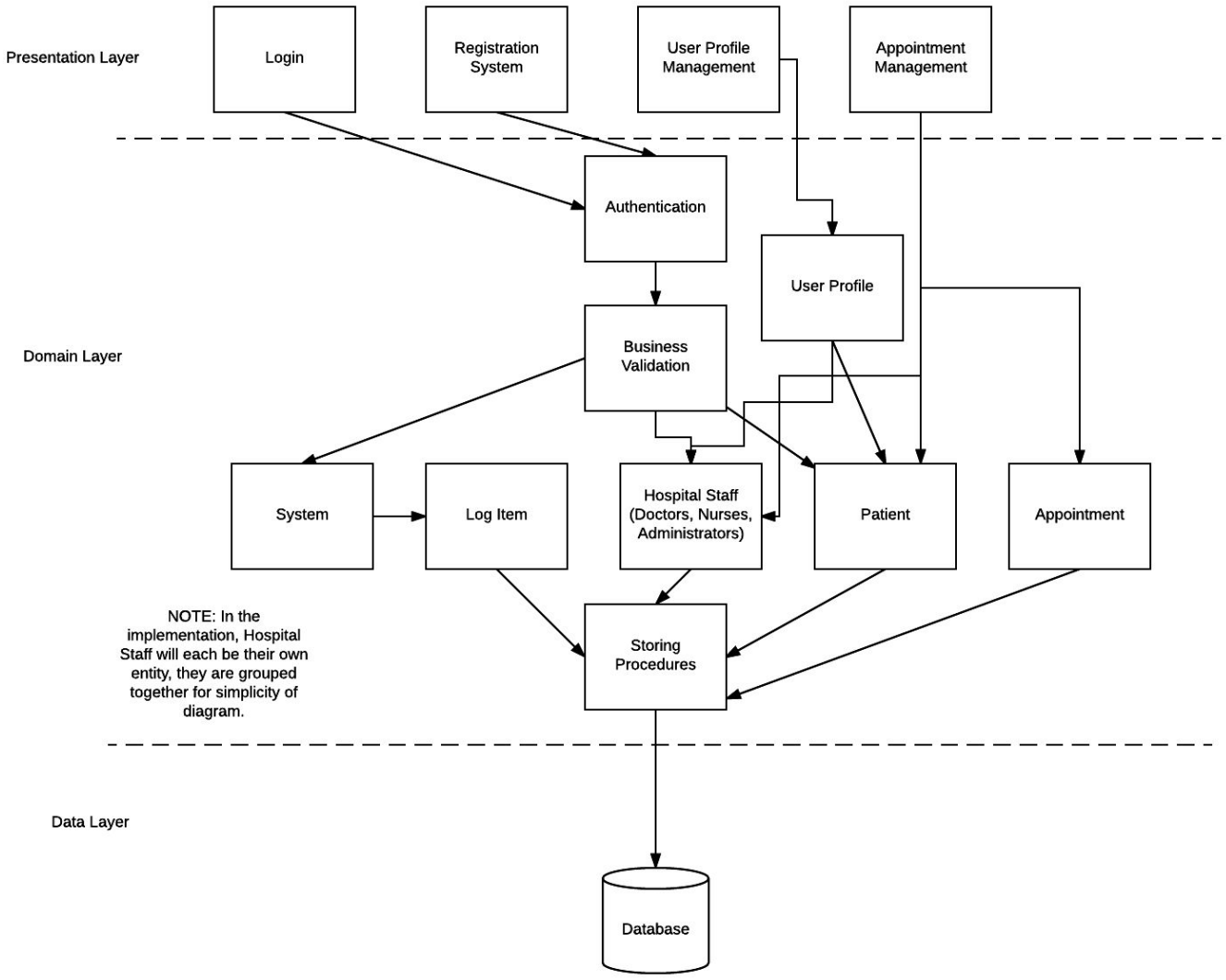
Product Design

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*

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.

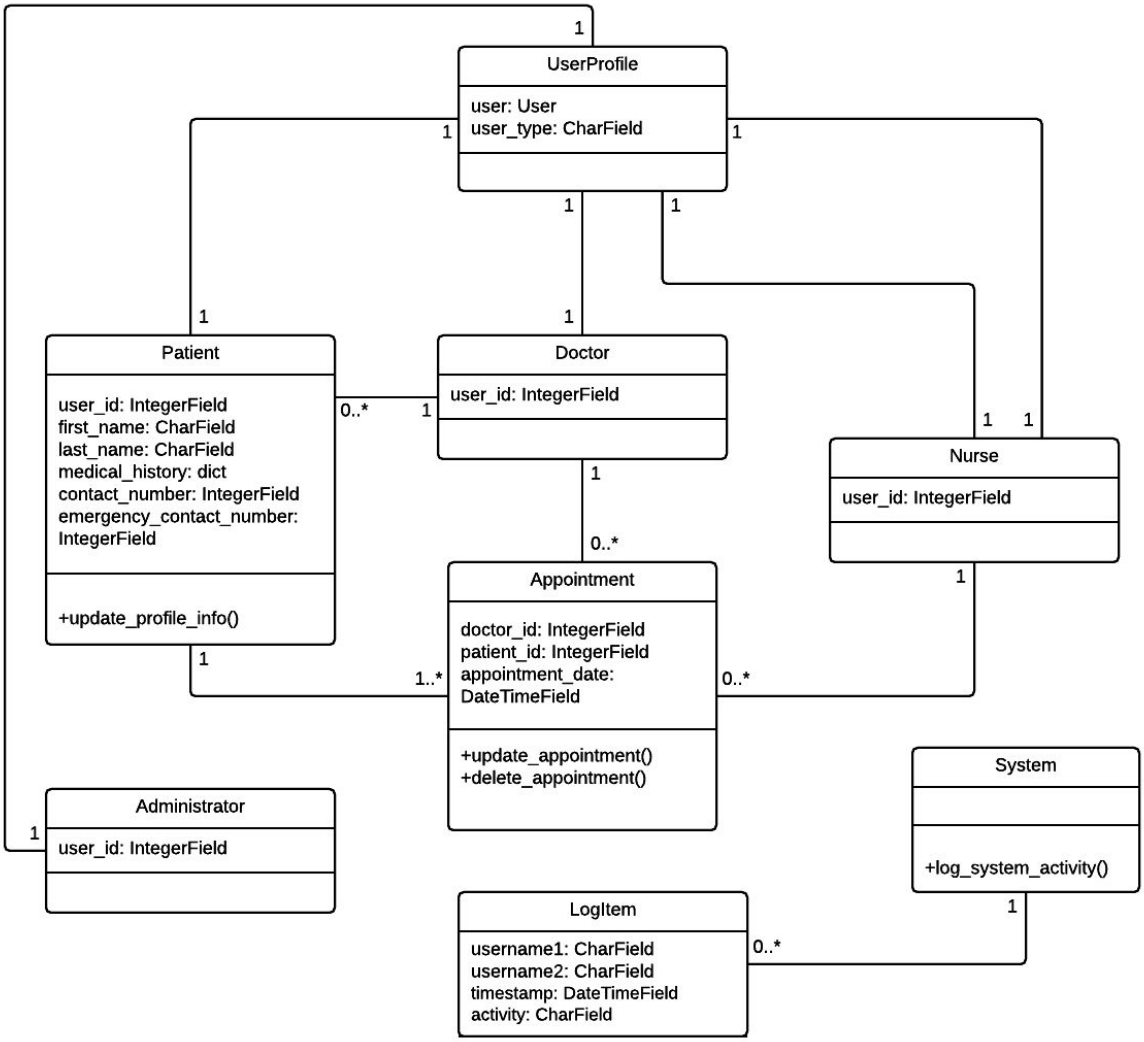

Presentation Layer
Login
Registration System
User Profile Management
Appointment Management
Authentication
User Profile
Domain Layer
Business Validation
System
Log Item
Hospital Staff (Doctors, Nurses, Administrators)
Patient
Appointment
NOTE: In the implementation, Hospital Staff will each be their own entity, they are grouped together for simplicity of diagram.
Storing Procedures
Data Layer
Database

Components and Functions

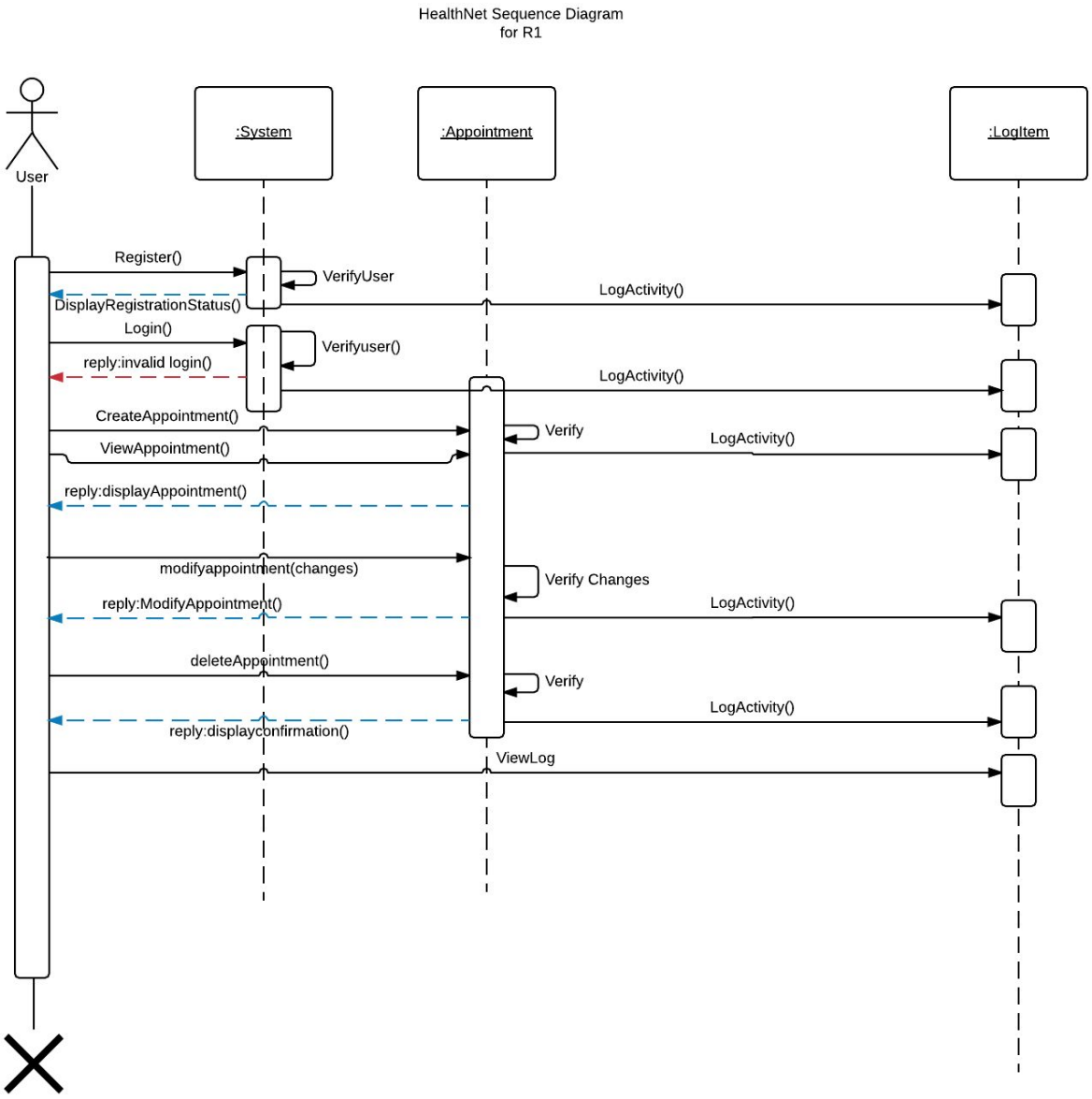
User Component	**Component state** ● Username ● Name ● Password **Component behavior** ● Gives a user the ability to: ○ Register ○ Log in * The built-in User class in Django will be imported for use. While the User class provides many more behaviors than are listed above, these are the ones which will 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 ● Name ● Contact number ● Emergency contact number **Component behavior** ● Allows the app to determine user permissions (can update own appointments) ● Update_profile_info: allows patient to update their contact info
Doctor, Nurse, Administrator	**Component state** ● User id **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 ● Appointment date **Component behavior** ● 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 **Component behavior** ● Is automatically generated when ○ A new user is created/registered ○ A user logs in ○ An appointment is created, updated or deleted
System	**Component behavior** ● Provides static methods for system/database operations ○ Log_system_activity: creates a new activity log item

Class Diagram(s)



Sequence Diagram(s)



Design Rationale

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.