**Functional Specification**

*Overview*:

The application I will be creating is a mobile communication software for health care providers in a hospital setting. The goals of the application would be to:

1) Help organize information on a patient. Act as an extension to an existing Electronic Health Record System

2) Allow providers to communicate/chat to each other in the application for a specific patient.

3) Allow providers with appropriate roles to order and complete procedures.

4) Give providers a ToDo list for a given patient.

In addition to the main application, I will be creating a mock EHR system that will help generate patients and their appropriate information.

***Core Requirements***

1. **Azure Web Api** – I am choosing this type for my application mostly due to the fact of scalability in the future. If I were to pursue this application outside of class, I would not want to be limited to use solely through a mobile device. In addition, I could see a benefit of having web-based single page application to have access to some of the same controllers as this application. In addition, if ever put into use this type of application would have to rely on a heavy number of other cloud services such as an Electronic health record system, perhaps “Internet of Things” health care devices, and security systems.
2. **Azure SQL** – Will store all information for my app (more information in *Application Detail* section).
   1. Patient Level Information
   2. User/Role Information
   3. Biometrics on Patient
   4. Chat history between providers
   5. Prescriptions, Procedure types and other lists that the app will need.
3. **Rest Interfaces -** There will be a controller to access each of the information above (more information in *Application Detail* section).
4. **Authentication** – Will use twitter as authentication technique to meet the final project requirements
5. **Authorization** – All rest interfaces will be authorized and in addition only accessible to certain roles. Roles will be described further in the *Application* *Details* section. The roles will be:
   1. Hospital Administrator
   2. Health Care Provider: Physician
   3. Health Care Provider: Surgeon
   4. Health Care Provider: Support Staff
6. **Azure Web Jobs** – As part of the final project, I will be creating a web service that acts like an Electronic Health Record system. This service will randomly generate patients and update their biometric status and other patient information from time to time. The activity of calling the functionality in the service will be given to web jobs.
7. **Azure File Storage** – will use Azure file storage to store imaging information for patients
8. **Secure Services** – Will use entity framework to help prevent SQL Injection attacks. In addition, will use Authorize decorator to lock down all costly resources.
9. **Universal App using C#**- I will be using this client type for the final project.
10. **Will Unit Test all Rest Services**

*Electives*

1. **Azure Notification Hubs** – Notification hubs will be used to alert providers when there is a critical change in the patients they are assigned to such as biometric information that is critical or change in patient status (re-admitted, discharged, died)
2. **Azure Websites** – Will create a small dashboard service to show usage statistics of the application as well as statistics on the mock Electronic Health Record system.
3. **Azure Search** – Will utilize this service to allow Hospital administrators to quickly go through patients, check status of patients, check all unassigned patients, check all patients in critical condition.

***Application Details***

*Assumptions*

While the following assumptions do not hold in reality, they are made in order to make the application more manageable to design and implement

* A patient can never be re-admitted to the hospital
* The scope of possible diagnoses are ~100, and the scope of possible procedures are ~20
* The only relevant biometric information is the ones stored in our *Biometrics* table.
* An admitted patient only has one thing wrong with them (only get one diagnosis code)
* A patient will only have a change in status up to 4 times a day.

*Details – Backend SQL*

1. **Provider** – represents application’s user (healthcare provider). Include twitter user id used to recognize user by their twitter login. Will include user role.
2. **ProviderPatientToDo** – will represent a list of to do items for a provider include fields for what the task is (procedure), date assigned, if it is completed, and date completed, patient it is for, (foreign key to *Provider,* foreign key to *ProcedureCode*, foreign key to *Patient*)
3. **Patient** - will represent a fake list of patients. Includes patient name, age, gender, medical status (discharged, dead, critical, stable), diagnosis, admittance date, and discharge date.
4. **Biometrics** – represents a patient’s biometrics for a particular admission. Includes blood pressure, glucose levels, Can be many biometric ratings per patient. (foreign key to *Patient*)
5. **ProviderPatient** – represents assignment of patients to users. Many to many relationship of *Provider* and *Patient*. . (foreign key to *Patient* and *Provider*)
6. **DiagnosisCode** – list of diagnosis codes possible for a patient
7. **ProcedureCode**  - list of procedure codes possible to be done and a category of what type of provider-role can complete the procedure.
8. **PatientProcedure**– represents procedures done on a patient. Many to one relationship with *Patient table* (foreign keys to *ProcedureCode* and *Patient*)
9. **PatientChatLog** – represents the provider to provider chat log for a given patient. Each row represents a message. Includes message, date of message, patient id, and provider id. Many to one relationship with *Patient* (foreign key to *Patient*)
10. **PatientImaging** – represents the imaging documents for a patient. Includes image type column. Many to one relationship with patient. (foreign key to *Patient)*

*Details – Roles*

Non-User

Super User – this user has the ability (and only the ability) to generate new fake patients from the mock electronic health record system. This user is not meant to be a “real” role in the system, and therefore cannot access any other part of the application other than using their screen to “ask” for new fake patients

Application Users

1. Hospital Administrator – receives new patients assigned to them and must assign to an existing physician. If physician orders a surgery procedure, must assign to surgeon. Can view their assigned patients chat logs. Receive all of their assigned patient’s alert notifications.
2. Physician – can participate in their assigned patients chat logs. Receive all of their assigned patient’s alert notifications. Can complete any “physician” type procedures. Can “order” any procedure. Can discharge patients. Can upload patient imaging. Can only have 5 patients assigned at a time.
3. Surgeon - can participate in their assigned patients chat logs. Receive all of their assigned patient’s alert notifications. Can complete any “surgery” type procedures. Cannot order any procedures. Can only have 5 patients assigned at a time.
4. Support Staff – can view and complete all outstanding “support” type procedures of all patients. Do not participate in chat log, can view chat log. Can upload patient imaging. Cannot order any procedures.

*Details – Business Logic*

Mock Electronic Health Record System

In reality, this application would hook into an already existing electronic health record and requires a patient pool that has changing health status. Therefore, part of this application will include the ability to generate fake patients along with updating their biometrics during their hospital stay. The functions of this system are:

1. Per Super User request, generate a number of patients AND assign them to a hospital administrator (split the assignment evenly to all current users in the *User* table who are hospital administrators).
2. Every hour the system will “roll the dice” to determine if a patient’s biometrics and medical status will be changed. Changes biometrics based on a random percent difference from last biometrics. Based on biometrics will change medical status to stable/critical/dead. Will only change each patient up to 4 times a day. Will only change patients that have not been discharged.
3. After every “critical” procedure or surgery completed by a provider, a patient’s medical status will randomly change.

Rest API

1. Patient – all calls require authenticated user
   1. Create Fake Patients
      1. POST Call with number of patients to generate
      2. Only accessed by Super User Role
   2. Get Assigned Patients
      1. GET call with userId parameter
      2. Only accessed by hospital administrator, physician, and surgeon
      3. Returns patient names and medical status where medical status is NOT dead or discharged
   3. Assign Patient to Provider
      1. Put call with patientId as parameter and patient object with an assigned physician/surgeon
      2. Only accessed by hospital administrators
   4. Discharge Patient
      1. PUT call with patient id and patient object
      2. Only accessed by physician
      3. Updates patient with discharge as status code
2. Biometric – all calls required authenticated user
   1. Get Biometric Data for a patient
      1. GET Call with patient id parameter
      2. Only accessed by physician and surgeon
      3. Returns all of patients biometric data
3. User– all calls require authenticated user
   1. Get User Information
      1. GET Call
      2. Returns user role
4. Chat – all calls require authenticated user
   1. Get Message by Patient
      1. GET Call with patient id parameter
      2. Surgeons, physicians, and support staff can access
      3. Returns messages and messenger ordered from earliest to most recent
   2. Add Message in patient chat log
      1. POST call with message object and patient id
5. Procedure/Diagnosis
   1. Get items
      1. GET Call
      2. returns a list of procedure/diagnosis items used for drop downs in the interface
6. ToDo – all calls require authenticated user
   1. Get ToDo list by user
      1. GET call with userid
      2. Returns all To Do items for a user
   2. Get ToDo list by Patient
      1. GET call with patientId
      2. Only accessed by users that have been assigned to the patient
      3. Returns all To Do items assigned to a patient
   3. Complete a ToDo item
      1. PUT call with ToDo object
      2. Only completed by appropriate role
      3. Change isCompleted property on ToDo Object to true
   4. Create ToDo Item
      1. POST call with procedure code
      2. Creates a ToDo object with procedure code and assigns to user depending on a few things
         1. Physician can directly assign to another provider already assigned to the patient.
         2. Physician can choose to not assign to anyone in which case if the procedure requires a physician/surgeon, the procedure is assigned to a hospital administrator in order to triage the procedure to a different provider.
   5. Assign/Re-assign a ToDo Item
      1. PUT call with ToDo object
      2. Only be done by physician or hospital administrator
7. Document– all calls require authenticated user
   1. Create Document
      1. POST call with patientid and file object
   2. Get Document List
      1. GET call with patientid as parameter
      2. Returns all documents for a given patient
   3. Get Document
      1. Get call with documentId
      2. Downloads file
8. Dashboard
   1. Summary Statistics
      1. This will probably represent several different routes, but ultimately will return interesting aggregates of data that will be return to the Azure website

*Front End*

Universal App

Azure Web Site