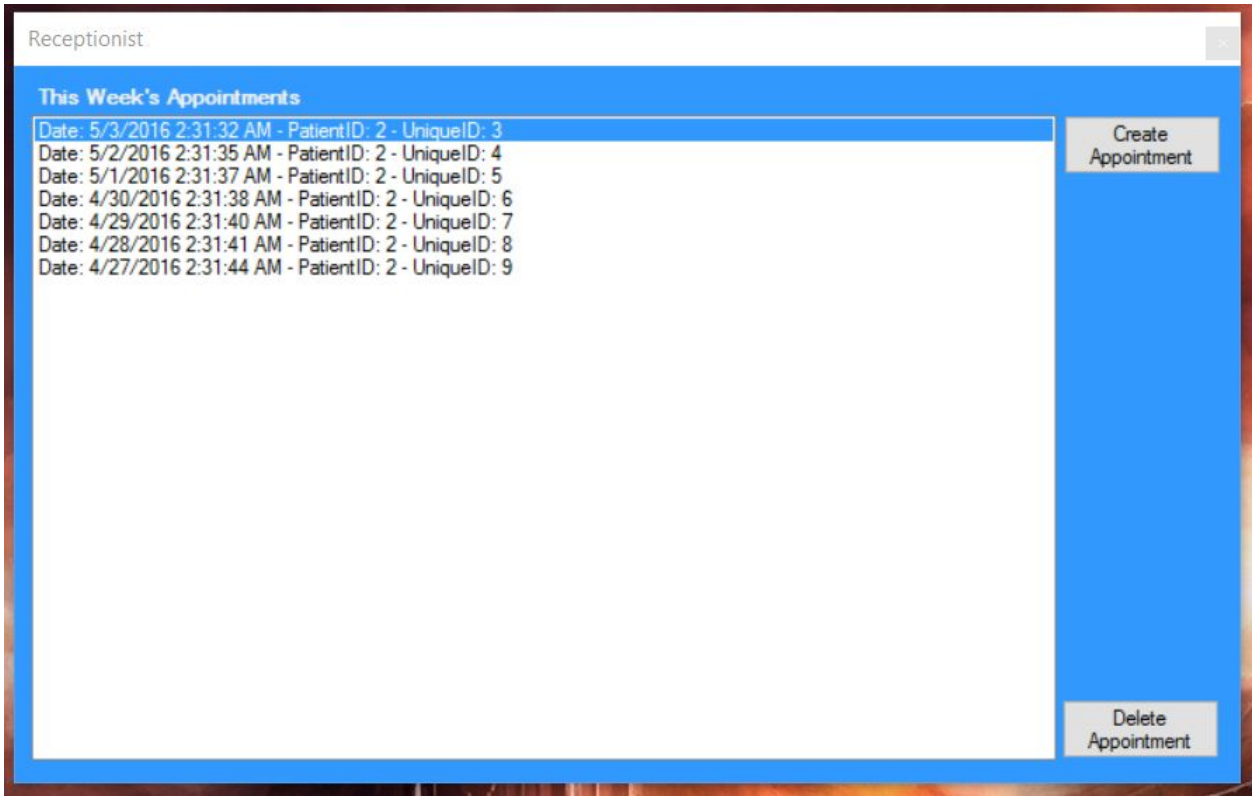
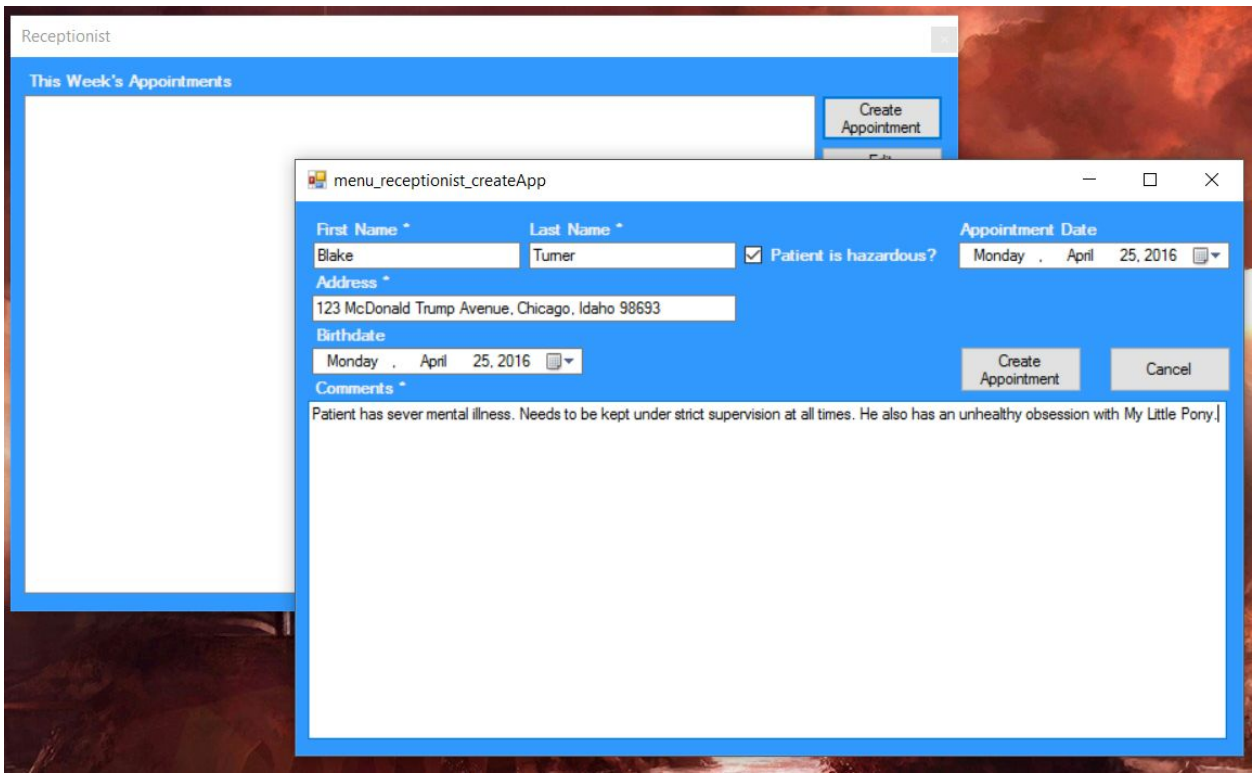
Getting Started Guide

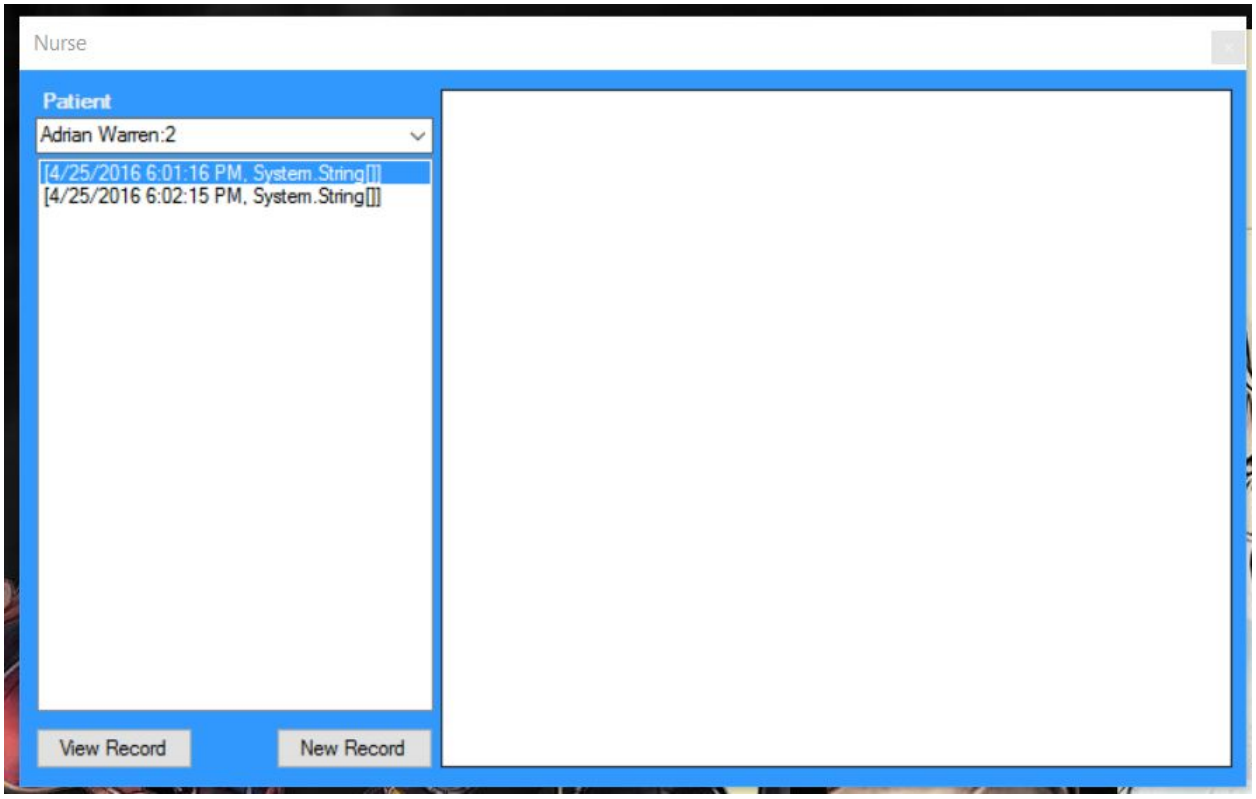
1. Download files from the github: https://github.com/tbturner09/MHC-PMS. The youtube video link will be provded in the github as well.
2. Run the MHC-PMS.exe to start the program, it will connect to the database if the an internet connection is available.
3. Enter the credentials of a particular user to sign in. There will be four different directions from the log-on screen which are based upon the profile of the user. Each individual user will have different access to the system and will be able to change or edit record information based on the roles that they have in the system.

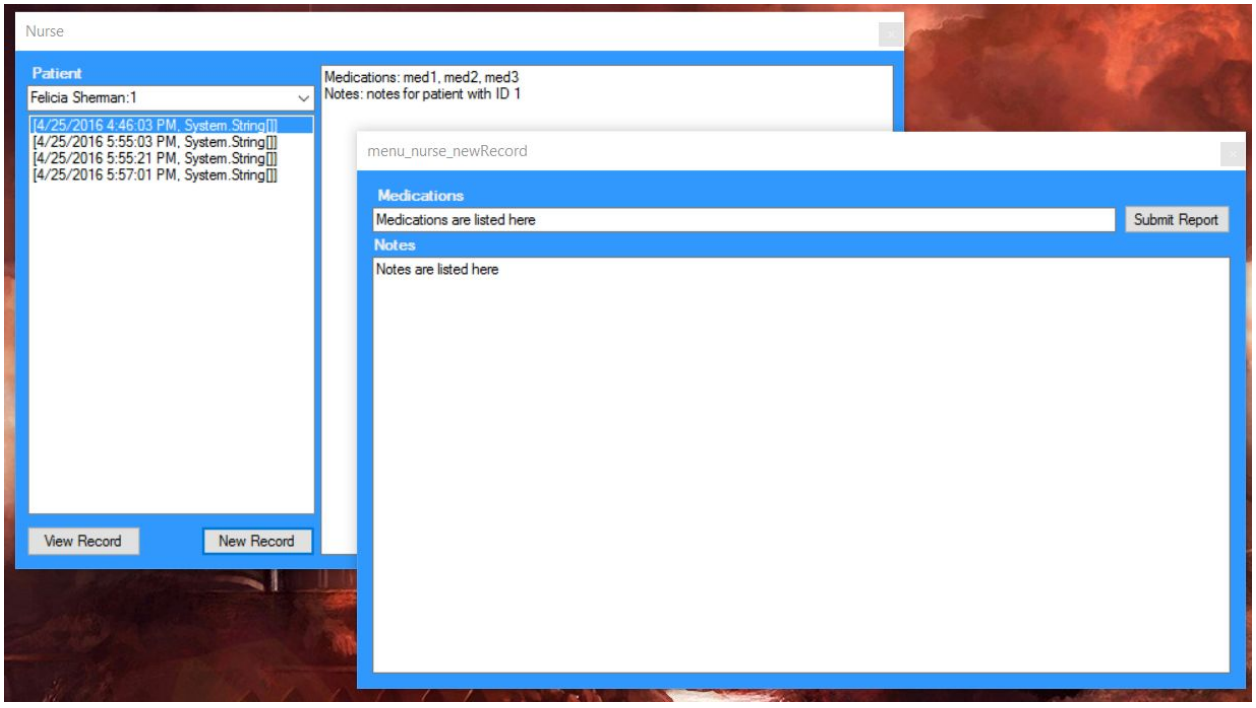
    - A receptionist can create, delete, and edit weekly appointments for patients. When adding a new appointment, the receptionist is able to enter the patient's first and last name, address, appointment date, and birthdate. The receptionist is able to enter information into the comments text box detailing information pertinent to the patient. The receptionist is able to indicate whether the patient is hazardous via a check box, and is then able to either create the appointment, or cancel the input.



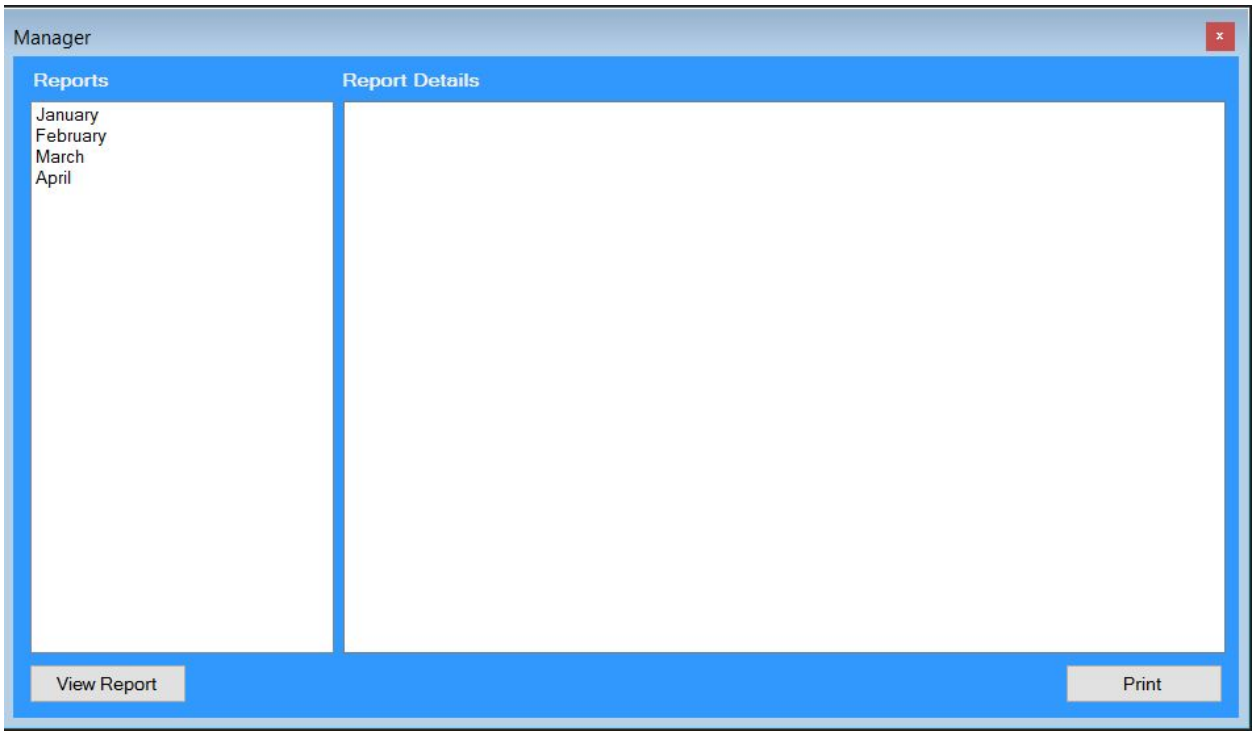


- A nurse can view appointments, create records, and view records. When creating a new record, the nurse is able to input any medications being taken by the patient, and any notes that might be relevant. After adding this information, the nurse is able to submit the report so that it may be created. After creating the record, the new record will be available for viewing, including all medication and notes that were recorded.


Nurse
Patient
Adrian Warren:2
[4/25/2016 6:01:16 PM, System.String[]]
[4/25/2016 6:02:15 PM, System.String[]]
View Record
New Record


Nurse
Patient
Felicia Sherman:1
[4/25/2016 4:46:03 PM, System.String[]]
[4/25/2016 5:55:03 PM, System.String[]]
[4/25/2016 5:55:21 PM, System.String[]]
[4/25/2016 5:57:01 PM, System.String[]]
View Record
New Record
Medications: med1, med2, med3
Notes: notes for patient with ID 1
menu_nurse_newRecord
Medications
Medications are listed here
Submit Report
Notes
Notes are listed here

- A manager can view and print monthly reports that are automatically generated when the manager logs in. The manager is able to view which months have monthly reports, and once the manager chooses to view that particular report, the information within that report will be displayed.



Test Cases

Interface Test 1 - by Blake Turner
GUI - An interface test was performed to ensure that the fields and options that a user makes should not result in error. This includes testing the use of special characters in text fields, accidental use of the wrong values in entry fields, empty fields that are required to contain information, and performance of consecutive actions by users. This test revealed error causing field entries. The first thing done about this was to make sure that fields are required to have information entered into them. This will eliminate the errors thrown by having blank fields. Currently, the system may still throw errors if special characters are used, particularly from different keyboard schemes. The user interface is likely the most common cause of errors so further testing will need to be done on the interface at each iteration.

Interface Test 2 - by Brandy Brown
For the different roles in the system, we noticed that one error would mask other errors which would lead to unexpected outputs. For example, when trying to add a "dummy patient" to the database, even though the button to add the patient

seemed to be working, the patient was not actually added to the database because all of the necessary fields were not filled out before submitting the new patient form. Testing of the interface and GUI took approximately one hour, and the current status of this test is complete.

Logon - It is important to keep patient health information from being accessed by unauthorized individuals. Policies should be set by the administrators using the system, but the program could benefit from some password controls. To protect against brute force attacks on the system, the account can be temporarily locked in order to prevent persistent guessing or dictionary attacks on the logon screen.This has yet to implemented, but should be considered for future releases of the program.

Ease of use - The more simple a program is to use, the better it will be for the users. Functions should not be too complicated to use and navigation of the program should not be difficult as well. Making the program more simple can also help alleviate errors. Fields and must must be explicitly clear of what their purpose is and how they should be used. In order to make the program more simple to use, we have implemented as little interfaces as possible while maintaining a the requirements necessary for the program to function properly.

Debugging Test - by Ericka Roberts
Debugging and refactoring the code was a challenge. Some variables needed to be renamed, and magic numbers needed to be assigned variables so that the code was more easily understood and manageable. Certain statements in the code needed to be rewritten for clarity and better understanding, since a few were confusing in the first draft. A few bugs were a little difficult to work out, but since everyone in the group comes from different computer science backgrounds, there were four pairs of eyes with four different perspectives that made it easier to identify problems as they arose. Debugging took quite a long time. About three weeks were devoted to the process of refactoring, debugging, and rewriting code. The current status of the debug testing is complete.

Login Testing - by Ericka Roberts
When logging into the system, input was tested for proper usernames and passwords. Properly entered credentials should allow the system to continue as expected: the user is welcomed to their version of the home screen. Conversely, failure to enter proper credentials causes an error. The system prevents the user from getting past the login screen. Login testing took two to three hours to fully implement and test. The current status of this test is completed.

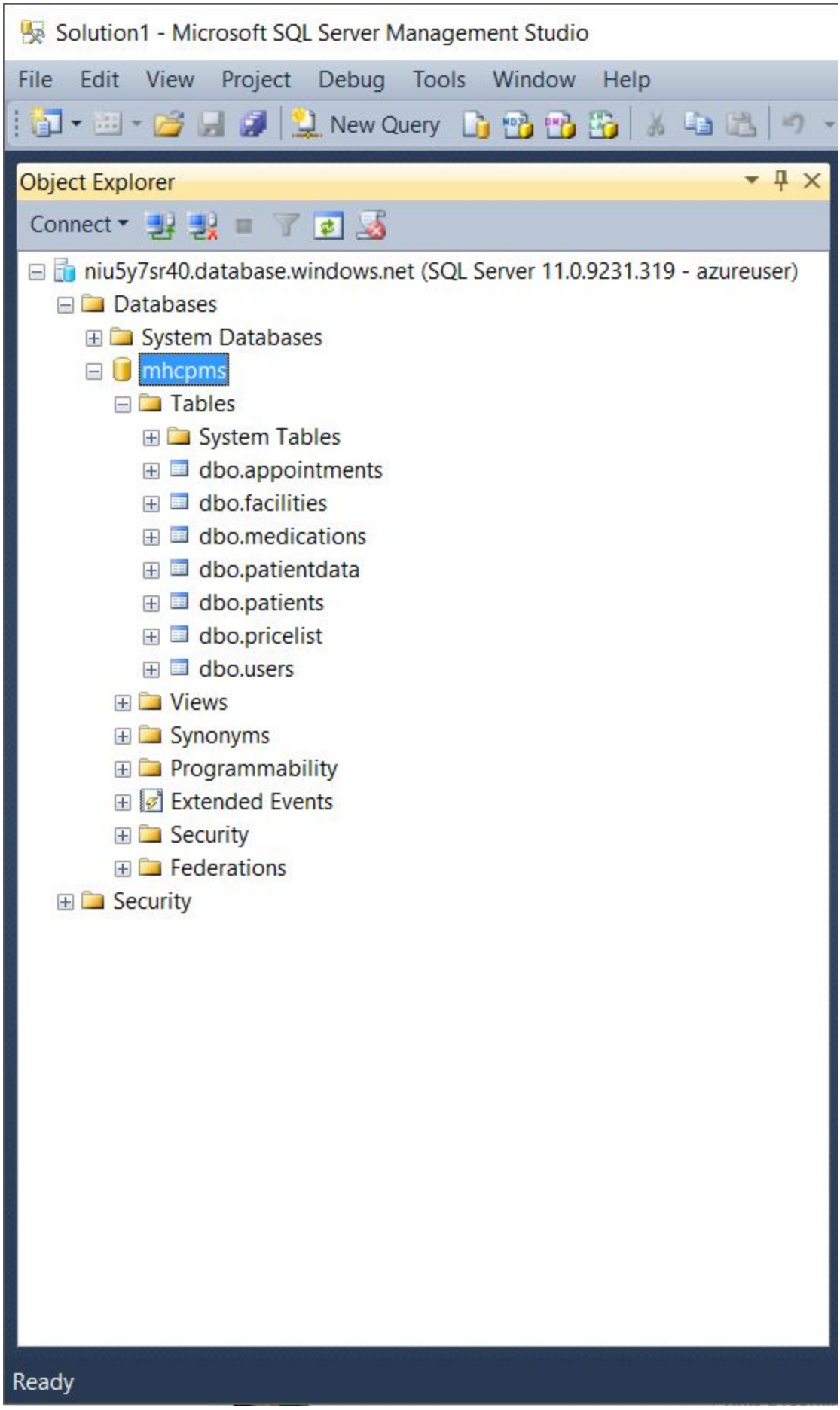
Other Tests

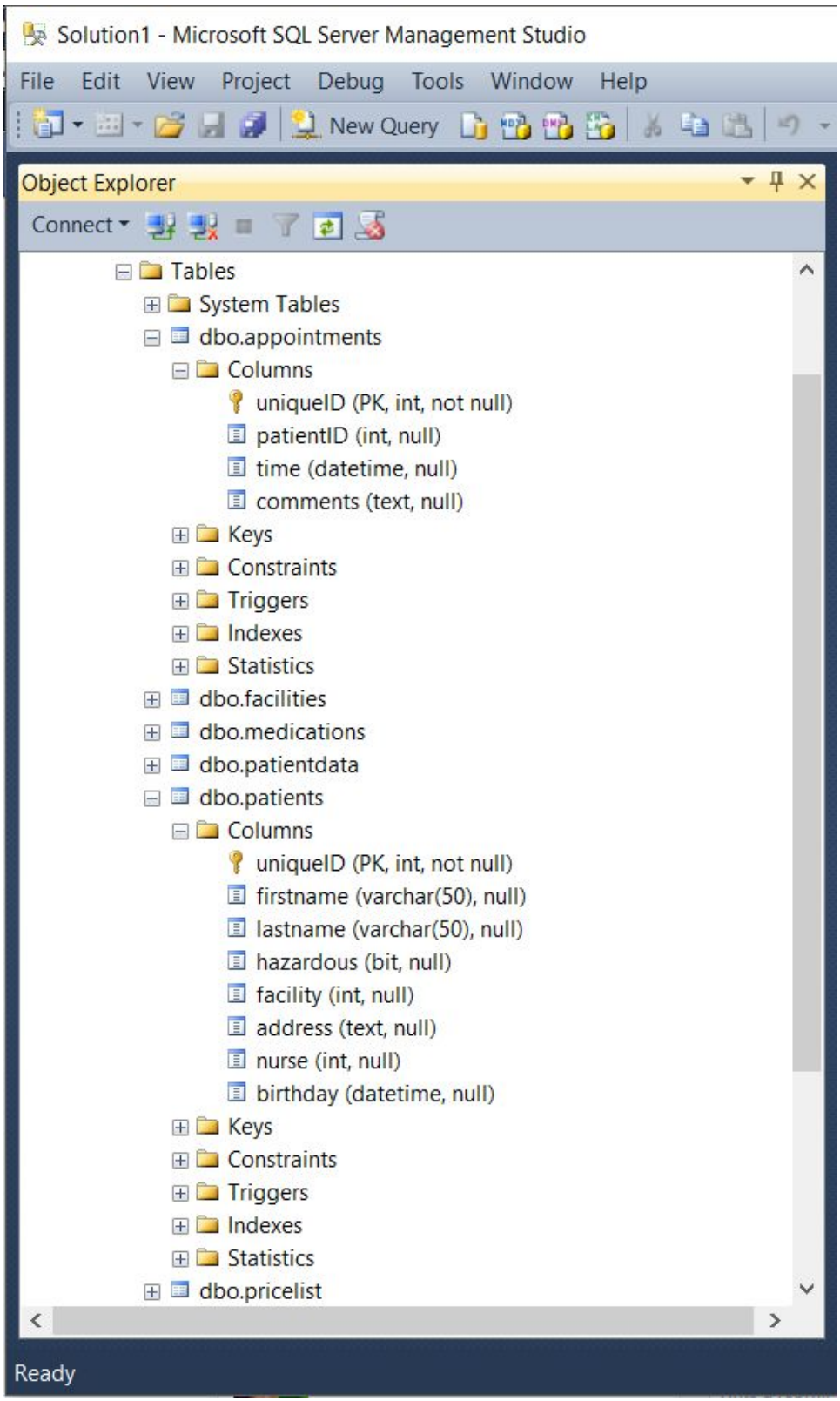
The test if the interface was mentioned in the previous paragraphs, however, there are more specific test that were performed on the program addressed below:

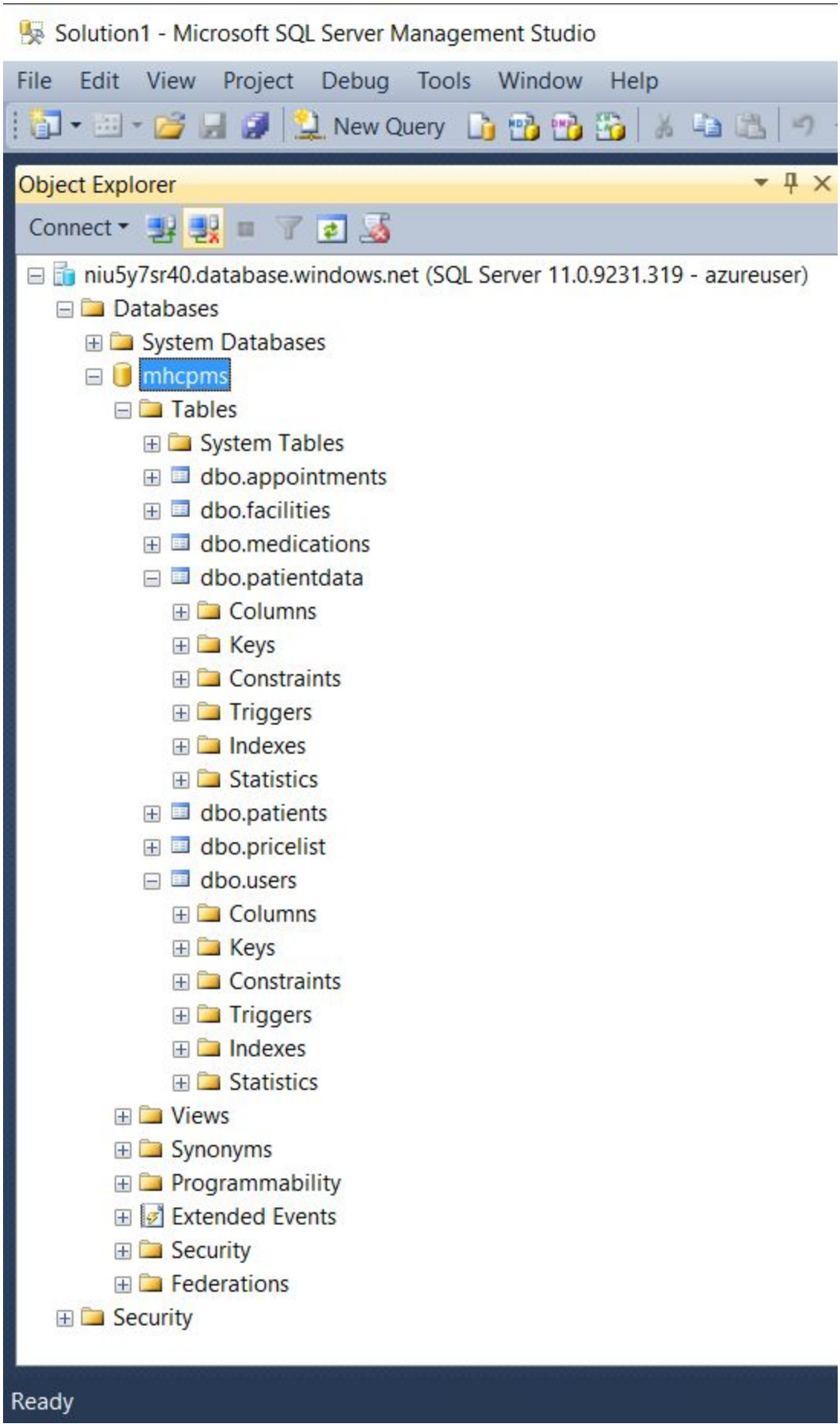
1. Performed a performance test, logging in multiple times while leaving the system operation for a long time. We found that the system would increase in the resources it would use at each login because of hidden panels that were created from each log-in. This will be patched in the next iteration.

2. Tested input of uncommon characters and symbols. Found that some objects from the date/time calendar could cause errors in the system if used improperly. It was removed from the system in place of a () for further testing.

3. Local SQLite database was removed in exchange for an Azure cloud based SQL database. This took some trial and error experimentations with our system, SQLite and cloud services, and we came to the conclusion that resources can be easily scalable and elastic in this case. Below are three screenshots of the database and how it links in with the system.

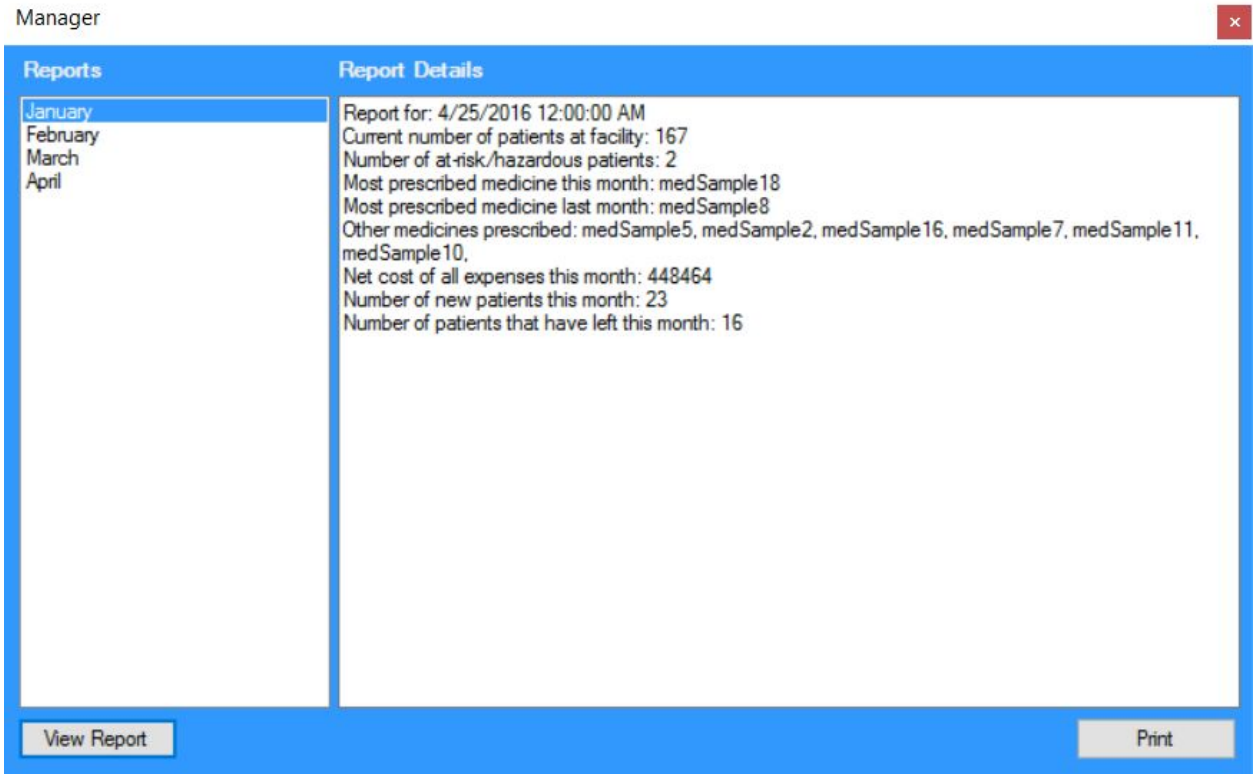

Solution1 - Microsoft SQL Server Management Studio
File Edit View Project Debug Tools Window Help
New Query
Object Explorer
Connect
niu5y7sr40.database.windows.net (SQL Server 11.0.9231.319 - azureuser)
Databases
System Databases
mhcpms
Tables
System Tables
dbo.appointments
dbo.facilities
dbo.medications
dbo.patientdata
dbo.patients
dbo.pricelist
dbo.users
Views
Synonyms
Programmability
Extended Events
Security
Federations
Security
Ready


Solution1 - Microsoft SQL Server Management Studio
File Edit View Project Debug Tools Window Help
New Query
Object Explorer
Connect
Tables
System Tables
dbo.appointments
Columns
uniqueID (PK, int, not null)
patientID (int, null)
time (datetime, null)
comments (text, null)
Keys
Constraints
Triggers
Indexes
Statistics
dbo.facilities
dbo.medications
dbo.patientdata
dbo.patients
Columns
uniqueID (PK, int, not null)
firstname (varchar(50), null)
lastname (varchar(50), null)
hazardous (bit, null)
facility (int, null)
address (text, null)
nurse (int, null)
birthday (datetime, null)
Keys
Constraints
Triggers
Indexes
Statistics
dbo.pricelist
Ready


Solution1 - Microsoft SQL Server Management Studio
File Edit View Project Debug Tools Window Help
New Query
Object Explorer
Connect
niu5y7sr40.database.windows.net (SQL Server 11.0.9231.319 - azureuser)
Databases
System Databases
mhcpms
Tables
System Tables
dbo.appointments
dbo.facilities
dbo.medications
dbo.patientdata
Columns
Keys
Constraints
Triggers
Indexes
Statistics
dbo.patients
dbo.pricelist
dbo.users
Columns
Keys
Constraints
Triggers
Indexes
Statistics
Views
Synonyms
Programmability
Extended Events
Security
Federations
Security
Ready

4. When entering data into the system, a system test that was done was trying to submit the new patient form without any text or characters in any of the text boxes and seeing how the system would react with this case of implementation. If the user in the system tries to create a patient without entering all necessary text fields, the system will reject the attempt and alert the user with an error message. If the user creates a patient with the proper documentation, then the attempt will be successful and be added to the database.

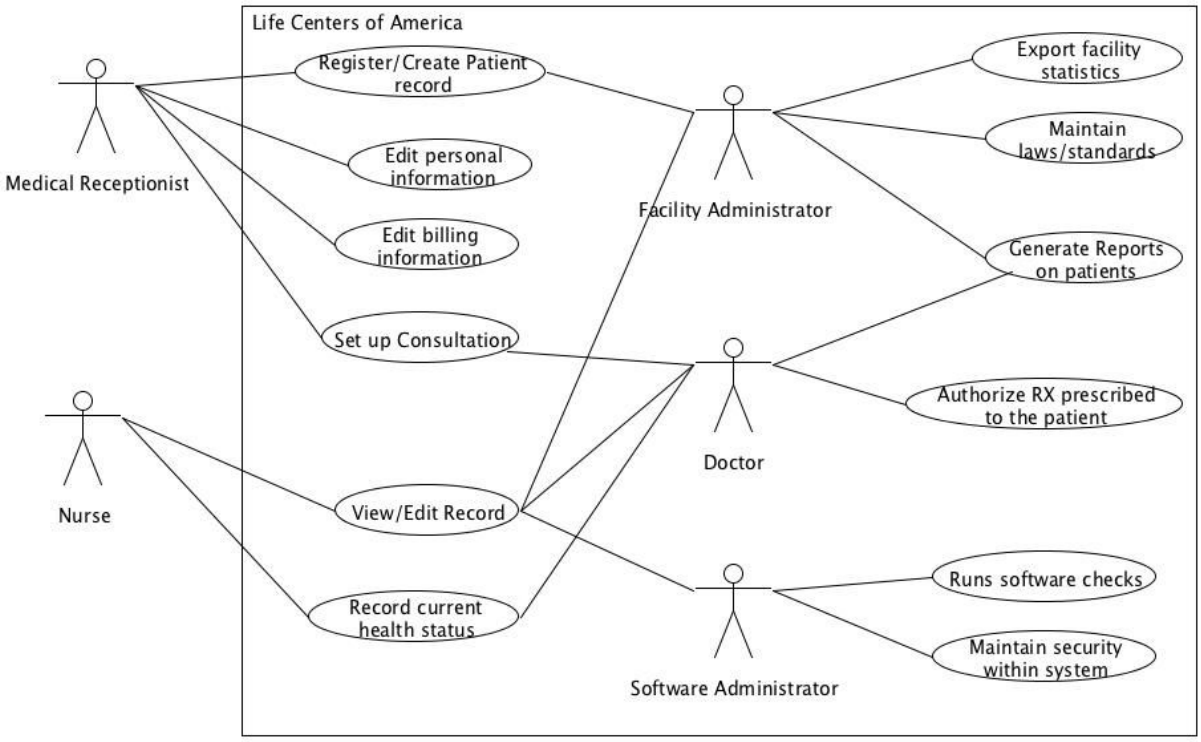
5. An integration test that was done on the system was the manager monthly report that is to be automatically generated when the managerial user logs into the system. Based on the information that is stored into the system, the manager report is auto-generated every time the manager logs on. A specific test that was done was to log in as a manager role and have the report generated. The next step in the test was to add a few more patients into the database and run another manager report by logging in as a manager. Based on this test, we were able to see a slight change in the monthly report information based on new statistics and details that was added into the system in between report generations. A change that will be made in the future will be to alert the user when a new month has arrived so that the information collected will restart over for the month.



6. Performed a shared memory interface test where different subsystems are tested to share the same block of memory AKA the azure database. Data that is placed in the memory by one subsystem is tested and retrieved by another system. An example of this would be clinicians updating medicine information into the system and then the system using that information to export it into a medical report that is available to managers.

Briefly describe all **system/user acceptance tests** that you did on the system, the test results, and any changes that you made to the system as a result. The use-cases developed to identify system interactions can be used as a basis for system testing. Each use case usually involves several system components so testing the use case forces these interactions to occur. The sequence diagrams associated with the use case documents the components and interactions that are being tested. (30 points)

1. When running the inspections on the system, we took into consideration the requirements specifications, software architecture, UML design models, database schemas, and the program itself to examine the system prototype. Specifically with UML diagrams and looking at the use case diagram that was created in previous deliverables, we decided to go from having five users that was originally created to having three in the system. The system has the functionalities to add other roles in the system but for straightforward reasons, we decided to only implement three roles and they are receptionist, clinicians, and managers. The managerial role has the same roles as the facility administrator. The roles have changed a little over time to mesh in with the system but in the future if we plan to create more user roles in the system, the jobs will be evenly distributed between the actors to model the use case diagram more closely. Please refer to the diagram below:



2. While running user acceptance tests on the system, we decided to refer back to the sequence diagrams that were created in previous deliverables to follow through on the ease of use with receptionists entering in patient information or medical records into the system. The system still follows through with the flow of the diagram. The receptionist creates a record, the nurse edits the record with notes and changes in medication, and the manager generates the monthly report. Based on the sequence diagram that was created previously, these actions deem this software ready to be accepted and deployed in the customer environment. The changes that need to be made in the future is adding more roles and responsibilities into the system in case a physician wants to have different access from a clinician, or if a facility administrator has a different job description than a manager's role in the system.
3. Another user acceptance test on the system falls under planned acceptance testing where we planned on time and establishing a testing schedule by planning an order in which tests were to be ran. We defined risks to the testing process that could have possibly occurred which were system crashes and inadequate performance issues and we came up with ways this could be mitigated by troubleshooting and debugging the system as these risks occurred.
4. As far as user run acceptance tests are concerned, the plan for the future is for customers to test this system in an environment where the software could potentially be used though this could be impractical. There could possibly be some time spent to train the users on how this system would work, but the

idea is for newcomers to use the system to see how ease of use plays into new user experiences.

5. Another plan for the future would be a user reject/accept system where we would meet with potential customers to see if the system should be accepted or rejected according to their standards. If the system is not good enough for use, then we plan to further plan and fix the identified problems.

Contributions to Deliverable 3

1. **Patrick Coffelt** – Creation and implementation of the MHS-PMS database and programming of the key requirements.
2. **Blake Turner** – Coding of the GUI and testing of the interfaces.
3. **Ericka Roberts** – Documentation and Design of the system
4. **Brandy Brown** – Documentation and Testing of the system