**INTRODUCTION**

This document will include my contribution to the CS460 Emergency Room application and feedback for the client on how they can improve on helping us deliver the best product for them. I will be going over all the tasks regarding deliverable 1,2 and 3. For deliverable 1, I will go into detail about how I devised the basic requirements needed for the project. I will also go over the architecture/planning pattern for evolving the use cases. For deliverable 2, I will go over some of the revisions that were made to the requirements based on the client's feedback. For deliverable 3, I will go over the Microservice Architecture pattern we implemented for the code base. Overall, I will be talking about all the work I contributed individually to the group.

**DELIVERABLE 1 (SOFTWARE REQUIREMENTS)**

For deliverable 1, It was hard to understand what use cases were needed, in the real world of freelancing, you do not write a massive requirements document that the client/developer cannot translate to actual tangible products. As a result, I went straight to diagramming and modeling how the data would be stored/processed for the application. I started out by understanding the entitles that would be needed in the database. The entities I started off with were Patient, Intake Patient, Medical Condition, Employee (Doctor/Nurse), Results, Permissions, Medication, and a hospital room. These entities were constructed based on the requirements of the client. I spent most of the time understanding which entitles would require a 1 to Many, Many to Many or 1 to 1 relationship. After these data relationships were established, then it was easier to understand what basic requirements are needed for the application. Some of the product functions we produced were employee management service, patient intake service and a procedure/room assignment service. All these "services" were constructed because of the DIA diagram. We formulated the user characteristics based on the employee permissions service. Some of the user characteristics were based on whether the user is a patient, nurse, administrator, Doctor, or Janitor. These permissions are necessary since a doctor and nurse have different permissions to access a patient. After the product functions and user characteristics were created, we produced specific product functions.

**SPECIFIC FUNCTIONS**

We formulated the specific requirements by looking at requirements or functions needed for each individual entity. The first universal requirement that’s needed in all software is CRUD (Create, Read, Update, Delete). For every entity we added a create function, which inserts a instance of the entity in the database. We added a “read” function which views all instances of the entity in the database. We also added “update and delete/delete all” for all entitles. All these functions are Soley responsible for populating and manipulating data in the database. Our first set of functions was for our first entity (Patient). The functions included Verifying the date of birth and calculating the BMI. The next set of functions was for the nurse (Employee). The client specifically mentioned having a function that allows the nurse to update patient notes on intake, as a result we added a function for that. We created a function for the nurse that admits/intakes the patient. The nurse also has a function to add symptoms to the patient on intake. The next set of functions were related to the Doctor (Employee). As a group, we decided that the doctor is a “parent” to the nurse and inherits all the functionality of the nurse. Therefore, the doctor can add symptoms, intake patients and update notes as well. Some additional functions that the doctor has are being able to prescribe medication, diagnose a patient with a medical condition and discharge a patient from the emergency room. Overall, these functions are what created the foundations for our use cases in Deliverable 3. Next, I will go into other requirements such as Performance, Maintainability and Architecture requirements. I will also touch on some security concerns for the application such as memory leaks and networking vulnerabilities.

**PERFORMANCE REQUIREMENTS**

**Standards Compliance**: Certain audit procedures would be followed, such as information on results of audits, financial info, how many intakes etc. It should be verified that bills are prepared properly according to visit charges of doctors, medicine, stay charges, room rent, etc. Bills should be verified with the fees/charges structure.

**Hardware Limitations**: Hardware limitations are the system is running on one CPU with a sqlite3 database. As for memory available, there is 1TB of hard drive space.

**Availability**: User/Customer information is saved in database and customer transactions are saved. Application will be able to replicate copies of the data model on disk. If any data is lost due to computer/system failure, replicated copies of SQL relational models will be saved.

**Security Requirements**: SQL server needs to have a database administrator. DB admin should have full access to all tables and entities. Users in the data model such as Nurse, Doctor, Patient need to have different permissions when viewing data. GUI Endpoints need to prevent SQL injections/string escaping. Database Repository and Business logic are separated from each other to prevent collision of application logic.

**External Interface Requirements**

**User Interfaces**: User Interface: The product will look and function like a form where you can add, edit, and delete information. The interface will be password protected so only users with the right privileges can access certain parts of the form. For example, the doctor can look at all the information collected by all levels of users while the registration nurse would only have access to the registration information on a patient.

**User Interfaces**: User Interface: The product will look and function like a form where you can add, edit, and delete information. The interface will be password protected so only users with the right privileges can access certain parts of the form. For example, the doctor can look at all the information collected by all levels of users while the registration nurse would only have access to the registration information on a patient.

**Security Requirements**

SQL server needs to have a database administrator. DB admin should have full access to all tables and entities.

Users in the data model such as Nurse, Doctor, Patient need to have different permissions when viewing data.

GUI Endpoints need to prevent SQL injections/string escaping.

Database Repository and Business logic are separated from each other to prevent collision of application logic.

## **Maintainability**

Down below is a list of things the developers will be responsible for fixing if bugs/issues come up.

Database Repository: All the code responsible for interacting with the database

Services: Employee Management Service, Patient Intake and Room assignment. All the logic and code needed to provide data to the GUI.

GUI: PYQT user interface. Any issues that occur with the GUI and layout/style of application will be subject to review if any errors come up.

# **Application Architecture**

This is a model of how the database functions will be separated from the business logic of the application. The figure shows a library designated for business logic and the data repository. The database repository will use sqlite3, with the ER diagram implemented.

The object relational mapper is just a way to serialize database entitles into classes/objects.

**Deliverable 2**

It is important to note that the client mentioned that more “specific functionality” was needed for the application when “ALL of the ABOVE” was already implemented. All the specific use cases were done BEFORE deliverable 2 because of our Diagram. The client mentioned that we needed specific functions for the Nurse and Doctor when they were already there on our GitHub repository. Personally, I am disappointed by this because I worked hard on my use cases, as did my group and the client did not read them. I know the client did not read deliverable 1 because we made NO changes to deliverable 2 and it was graded the same. Remember that we want to deliver the best product to the client. If they are not taking the time to read the requirements and plans, we have set in place, it is going to be hard to deliver on that commitment.

**Deliverable 3 (Implementation and Code and use cases)**

Now I will go into the code base and it how it was structured. In short, we used a modern-day micro service architecture pattern. This means that all the components of our system can be deployed on different machines etc. We separated the application into 4 modules. One module was dedicated to all database functions for sqlite3. Our second module was the “Controllers “or Json rest Api web endpoints. Our third module was a custom ORM with all our database entities implemented as classes. This module was made so that we could implement the required object-oriented concepts the client wanted. If the requirements were to expand for “business logic,” we would have a module dedicated for the services. This application does not have a large repository for business logic, so we have combined it into the controller's module.

**Database Repository**

The database repository has (CRUD) Create, Read, Update, delete functions written for the sqlite3 API in python. These are the specific functions discussed in deliverable 1. The database repository also contains 1 to M, M to M and 1 to 1 relationship for functions such as “Prescribe medication to patient”, “Diagnose medical condition”, “Assign procedure to room”, “Assign employee to patient”, “Assign employee permission”, “Check availability for rooms”, all the use cases we submitted for deliverable 1,2 and 3. As far as my individual contribution for the use cases, I worked on “Prescribe medication to patient” and “Assign procedure to room”. My group did the other use cases

**Controllers**

The controller's module is just the HTTP web Json endpoints using the Flask rest Api framework. All these endpoints allow the user to use our database repository from the API. The API will also be used for our “Flutter” front end to communicate and retrieve and send data back and forth.

**Front End (Flutter)**

For now, we have decided to use a front-end framework called “Flutter” which allowed for cross platform development. Our application will work on mobile, web and desktop. This has not been finished yet due to no confirmation of whether our API satisfies the clients' requirements. Once we confirm that the API meets the requirements, we will have an agile methodology in place to start creating widgets and Api calls.

This concludes the overall document of the emergency room application (Criticare).

**GITHUB:** https://github.com/shalinsingh818/CS460EmergencyHospitalDB