Microservice Architecture: Emergency Room

By Shalin Singh

May 7 2022

1 Abstract

In this paper, we examine the semantics that go into building medical software using a modern microservice architecture approach. This paper includes the individual contribution done by Shalin Singh, a Computer Science student at the University of Hartford. We will look at the components done for the server side and client user interface. The application developed was a Hospital Emergency room system that manages patients, employees, and billing information. The objective of the system is to intake and manage patient information in a medical emergency. The user interface is supposed to be easily understood from a client and administrator aspect. This document will also include the requirements and deliverables requested by the client, Dr. Ingrid Russel.

2 Mapping out the first set of requirements (Deliverable 1)

For the initial set of requirements, it was complicated to understand what use cases were needed. It was easier to figure out the requirements by modeling the system. As a result, I went straight to diagramming and modeling how the data would be stored and 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 from 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, it was easier to understand what basic requirements are needed for the application. Some of the product functions we produced were employee management, patient intake service and procedure/room assignment services. 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. As for my individual contribution in the first set of requirements, I focused on what use cases would be generated off (One to Many) and (Many to Many) relationships. Some of these use cases included prescribing patient medication and assigning a patient a medical procedure. I was also responsible for structuring the initial code base and how our database, web and backend services would interact. I structured the code based on a modern day microservice architecture pattern. The pattern involved separating code into sections for the database, HTTP controllers and models for a custom ORM (Ob-

ject Relational Mapper). The client requested that object-oriented concepts should be implemented in the final product. Overall, these were my initial individual contributions for the first client deliverable.

3 Functions Generated From Modeling the System (Deliverable 2

I formulated the specific requirements by looking at functions needed for each individual entity. The first universal requirement that is needed in all software is CRUD (Create, Read, Update, Delete). For every entity I added a create function, which inserts an instance of the entity in the database. I 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. I 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.

4 Implementation And Source Code

Now I will go into the code base and how it was structured. My individual contribution was the high-level overview of how it would be structured. My team worked on copying and pasting the source code templates for each module. 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 to the services. This application does not have a large repository for business logic, so we have combined it into the controller's module. We originally planned to have a services module that would contain all the business logic. Since the business logic was not intricate, we combined the logic and did not make a separate code base. In the next sections, I will explain the individual contribution for each module of the code base. These portions will cover aspects of deliverable 3 and 4.

5 Database Repository

The first module of the code base involved the development of a database repository. The data repository contains files with functions that are responsible for database operations such as "create", "read", "update" and "delete". My job was to understand the high-level functions that would be inherited across all the required entities for the application. My team worked on the specific functions and copied the templates across all entities. Most of the functions I assigned for all entities were CRUD (create, read, update, delete), and helper methods that could query individual entities by id or some form of unique field. A key function I required for all entities was the creation of a one-to-many or, MTM (Many to Many) relationships between other entities. For example, this type of function would be used for the relationship between medications and the patient. Any entity that had multiple instances related to the patient required this type of functionality.

6 Models

For the models, I did not contribute individually to this section. That was done by the other team members. The models were formed based on the entities in the database. Most of the classes/objects that were needed were copied and pasted and replaced with the needed fields. My team's job was to take the fields from the database and implement using object-oriented concepts in python.

7 Controllers

The controller's module is meant to act as HTTP endpoints that the user can interact with. The endpoints make use of the functions from the models (ORM) and the database repository. My job was to map out the hierarchy and templates of how the endpoints would be structured. Again, most of my contribution involved creating the abstract model of how the system would be built. For the HTTP endpoints, I worked on creating the same database repository functionality in the form of a JSON web API procedure call. The database methods such as "create", "read", "update" and "delete" were implemented for all entities. The difference for the controllers is the protocols needed to be called by the user. For "Create", "Read", "Update" and "Delete", the corresponding HTTP protocols are "GET", "POST", "PUT" and "DELETE". Overall, my contribution involved creating the templates for these http procedure calls.

8 Unit Testing (Deliverable 5)

Although the requirements and deliverables are listed in sequential order, the implementation did not follow this sequence. The unit testing deliverable was assigned by the client at the end of the project. Regardless, testing was done at the start of the project for our team. The development methodology we employed was TDD (Test driven development). This is a widespread practice in industries that build software with evolving features. We did not want to write our code and find bugs closer to the final deliverable. This was my biggest specific contribution to the project. The first lines of code that I wrote were test cases for the database repository. Once my team wrote their specific sections, I wrote test cases to ensure it would work in production. Un-tested code is useless, so it is pointless to build something and then test it at the end. Every time we wrote a feature, mapped out a requirement or prototyped, we wrote corresponding test cases before the implementation. My biggest contribution

was in this deliverable, I wrote tests for every database method, ui widget and model from the ORM. I also enforced the test-driven development methodology from the start to make sure bad code did not enter the production environment.

9 Presentation and Release

For the final presentation, we put together a PowerPoint that explains all the concepts examined above. We talked about the micro serviced design pattern that was employed, we mentioned the frameworks and technology stacks that we used. We also had a full demonstration prepared for the app release. For this section of the project, I worked on the PowerPoint presentation. I was not instructed to talk on any of the slides except for the sections indicating the database repository. My overall contribution to this part was demonstrating the application and the code base. The team did an excellent job of dividing up which slides and preparing before the presentation day.

10 Conclusion)

In conclusion, these were my individual contributions to the emergency room system. Overall, I felt that the project did not strengthen my skills as a software developer. The project was very bland and did not require intricate business logic. I like to build systems that involve the use of controlling memory, audio processing and networking. This system felt like it had no potential value. There were not any new languages or frameworks learned during the semester, I had prior experience before the class. I used up and coming frameworks like Flask and Flutter that I usually use when building apps. Overall, the only challenge I faced in the project was how tedious the implementations were. If we had more time, we could have added extra features such as push notifications, state vector analytics and much more. The things I enjoyed most about working on this project were the use of reusable widgets in flutter and structuring the backend using a modern day microservice approach. The part of the project that I am most proud of was the user interface and how well it turned out. The visual appeal of a product made it look like a genuine business application. Using Flutter/Dart was the decision I am most proud of. As for potential employers, I would only feel comfortable sharing this project if experience with Flutter or Flask was needed. This project demonstrates understanding of stateful widgets in Flutter and that would be an asset to a potential employer. The project will overall be useful to have as a template for future apps that I plan to build.