**Software Design Description**

**for AI Chatbot and Scheduler**

**Version 1.0 approved**

**Prepared by:**

Devin Roering

Bikash Timalsina

Toua Yang

Chandler Matilla

Dylan Wahlstrom

**Course Project**

SE340 Software Design

Fall, 2019

10-4-2019

# Table of Contents

1.0 INTRODUCTION

1.1 Purpose

1.2 Scope

1.3 Glossary

1.4 Reference Material

1.5 Overview

2.0 Deployment Diagram

# 3.0 Architectural Design

# 4.0 Data Structure Design

# 5.0 Use Case Realizations

# 6.0 User Interface Design

# 7.0 Help System Design

# Index

**List of Figures**

# 1.0. Introduction

## 1.1. Purpose

Our goal is to create a Chatbot that utilizes cloud infrastructure and visualization services to answer FAQ, schedule appointments with physicians, and pay outstanding bills. A chatbot is an automated program that answers questions and completes similar tasks through a text box or through links in order to help move through a website and find the information you need with the help of artificial intelligence. This would be a good way for clients to get their information while keeping the physicians free. For our project, we will be focusing on setting up the project to work for clinics. This document will display the way in which the system is to be architected. It contains graphs and diagrams that will assist with user scenarios and class structure.

## 1.2. Scope

The scope of this product entails the utilization and integration of an AI chatbot, a visualization of a human that physically and verbally responds to input based on the AI’s response to a question, and the necessary APIs to implement a system that can schedule appointments, pay and view current bills, and answer frequently asked questions.

The goal of this process is to free up staff and allow them to more effectively do their job, while the Chatbot handles support, scheduling, and billing in a streamlined and professional way. The AI must be professional, meaning the dictionary used will omit certain forms of language that could be perceived as rude or abrupt. If patients feel uncomfortable with the system, they will not be inclined to use the service again.

This product shall not diagnose, handle the distribution of confidential information, or control prescriptions. The only user information that should be accessible to the program is the users account information on file, and any medical flags or alerts, such as heart conditions or diabetes information. This is only to assist the physicians that the program puts clients in touch with at diagnosing by allowing them to view all necessary information easily. A Link to this information would be sent to physicians with appointment times as they are scheduled to allow for prep time to be completed more efficiently. This will speed up wait times and improve the speed in which physicians can assist the public.

## 1.3. Glossary

Chatbot- a program that receives user input and displays a response based on that input.

AI- Artificial intelligence, a way to create simulated intelligence in a computer program, making the

responses it gives relevant to the questions asked in the chatbot.

Visual Studio- A Microsoft program that allows code to be edited and assembled, as well as compiled.

Unity- Visualization Software used in the creation of animation. It is easily integrated into Visual Studio.

Github- An online service that will allow us to publish projects that can then be downloaded by users.

## 1.4. References

< List here any references to other documents cited anywhere in this document including references to related project documents, especially the SRS. Add references here when other project documents are created. This is usually the only Bibliography in the document. >

AI Chatbot and Scheduler SRS document will be referenced inside of this document. It contains more design elements.

## 1.5. Overview of Document

This document will show the design strategies and ideas for the creation of the AI Chatbot and Scheduler including the ways it will interact with users and the physicians. The remaining sections of this document will discuss the different diagrams associated with the module.

The Deployment diagram will provide a physical look of the system with devices and processors indicated. The Architectural Design will show the entities that collaborate in the program to perform a task. It will also describe the abstract specification and interface design of the project through a top-level diagram.

The Data Structure Design will look at and the different data structures and their implementation. The Use case section will show diagrams detailing the user stories within the module. The UI will discuss the look of the module and how it will function for users. Finally, the Help System design will describe how the users can receive support, as well as the system response to errors and crashes.

## 2.0. Deployment Diagram

< The deployment diagram provides a physical look at the system with each processor and device indicated. This provides a background for the rest of the document as no software component can straddle two physical locations. Each physical location will have its own software unit and units in different physical locations will collaborate to provide the services that logically seem to be straddling the units. >

## 3.0. Architectural Design

< A software system is a set of communicating entities that collaborate to perform a task. The Architectural Design shows these entities, their relationships and the relationship to the actors in the system. This top level is a diagram where each entity has a name, a (proto-) type, an abstract specification and an interface design. The abstract specification is a description of its purpose, its functionality, its attributes (including dependency on other entities) and the constraints under which it must operate. It also describes resources, that is, any elements used by the entity which are external to the design such as physical devices (e.g., printers), software services (e.g., math libraries) and processing resources (e.g., buffers). The interface design is the list of the services that it provides to clients. These services are methods (procedures and functions), each carefully documented using a pre-condition/post-condition formalism.

Each entity in turn may provide its services by having an internal architectural design with its own set of subordinate entities. These entities may be called sub-systems, components, modules or classes. The decomposition of a higher-level entity into subordinate entities must be explicit. The algorithm that shows how each method of the larger entity is performed by these components must be explicit. Any data stored in an entity must be explicitly described (see Data Structure Design below).

Note that while the abstract specification (architecture) and the interface (detailed) design are usually developed separately in an iterative approach, they are combined for documentation. >

# 4.0. Data Structure Design

< If not already covered above, design in detail and specify the data structures to be used in the implementation. If these include databases, define the table structure of all databases including full field descriptions and all relations. Graphical languages are appropriate. Note that a database is an object and may have been fully described in the previous section. If the material is covered under Architectural Design, this section is omitted. >

# 5.0 Use Case Realizations

< For each use case in the Requirements Specification there must be a use case realization here. That is, there must be a sequence of events using the design objects that will perform all of the operations promised in the SRS. It is possible that there will be several use case realizations here to show all the alternative and exception paths required. The relationship between use case specifications (full descriptions in the SRS) and use case realizations must be bi-directional (referenced from SRS to here and from here to SRS) and explicit. Each use case realization must also be cross-referenced to the use case test in the test design. This relationship between use case specifications and use case test must be bi-directional and explicit. >

# 6.0 User Interface Design

< Refer to User Interface material in the SRS and supplement with any design considerations not mentioned there. You should discuss the expected effectiveness of your design. >

# 7.0 Help System Design

< Describe the **structure** of the help system and how it is to be accessed. Will it be context sensitive? Will there be a system reference capability? >

# Index

< generate here >