Electronic Scheduling Tool for Surgery Clinic

Final Report

sdmay18-26
UnityPoint
Srikanta Tirthapura
System Admin - Joel May
Repo & Schedule Manager - Matthew Burket
Point of Contact & Meeting Facilitator - Ryder Schoon
System Architect & Chief Engineer - Matthew Cavalier
Meeting Scribe - Maddie Andreassen
Report Manager - Matthew Burket
Testing Manager - Luke Sternhagen

sdmay18-26@iastate.edu
https://sdmay18-26.sd.ece.iastate.edu/

Contents

Preface

Definitions

Epic	Electronic Health Record software for midsize and large medical groups, hospitals, and integrated healthcare organizations
ES5	Shorthand for ECMAScript 5, it is a scripting-language standard for JavaScript that was created in 2009
ES6	Shorthand for ECMAScript 2015, it is a scripting-language standard for JavaScript that was created in 2015, this standard is not fully enforced, but is implemented on many modern web browsers
Patient	A person who is visiting the clinic to see a provider
Provider	A medical practitioner who works at the clinic, these can range from dietitians and surgeons, to mental health workers
HIPAA	HIPAA (Health Insurance Portability and Accountability Act of 1996) is United States legislation that provides data privacy and security provisions for safeguarding medical information
Intranet	A website that is only accessible on the business's local network
API	Application Programming Interface, a definition of the protocol and parameters used/needed for programs/systems to communicate
HTML	Hypertext Markup Language
JavaScript	A scripting Language used in websites
CSS	Cascading Style Sheets, used to define how elements in a HTML page are rendered
DLL	Dynamically Linked Library
RESTful	REpresentational State Transfer (REST) is a pattern for web APIs. RESTful web applications serve a set of APIs that conform to the REST concept
JSON	Shorthand for JavaScript Object Notation, which is a data format used for transferring data between entities

Revised Project Design

Project Statement

We have developed an electronic scheduling tool as a proof of concept for the UnityPoint weight loss clinic. Our system allows for an end-user to find possible times that a patient can see multiple providers during one clinic visit.

Purpose

The purpose of this project is to build a better way for the front desk personnel at the weight loss clinic to schedule times for patients to see providers. There are two major goals for this project. First, to be able to set a maximum wait time, 10 minutes, that a patient will be waiting to see providers between each provider consultation. Second, improve how efficiently an employee that is trying to schedule appointments can find times that line up so that the patient is not waiting to see providers all day.

HIPAA

Due to this application dealing with medical records, the team needed to keep in mind HIPAA while developing the system. By design, the risk of dealing with medical records has been mitigated as no patient data is stored anywhere in the system, as well as the system should only be available as part of UnityPoint's intranet.

Implementation Details

This project implements the web application architectural pattern (see Figure 1, Appendix III). The system can be divided up into five major components (see Figure 2, Appendix III). The first component, the Server, provides web API's as well as contacting the Algorithm DLL. The second component, the Algorithm DLL, houses the logic for generating scheduling possibilities as seen in Figure 3 of Appendix III. Next there is the Models component which provides Object models that are used in the Algorithm, Server, and Database. The Database is a Microsoft SQL Server instance that holds all data needed for computation. Finally, there is the Front End, which is ran in the web browser and is what the end user interacts with.

Server

The server is written in C# using the .NET Framework, ASP.NET MVC framework, and the Entity Framework. The code is designed to run on IIS 10 with Windows Server 2016. That system requirement allows us to take advantage of new features to provide better performance.

We decided to use the .NET Framework, as it is a popular and well supported standard for applications running on Windows. From what we can tell, UnityPoint Weight Loss Clinic uses Windows in much of its business infrastructure.

ASP.NET MVC was chosen to serve the RESTful APIs, because it provides robust processing of the boilerplate parts of inputting and outputting API data. This greatly simplifies the burden on the developers and maintainers by removing the need to write code for basic input validation and input processing. The MVC framework lets us write native C# functions that take and return native data types; the framework processes, validates, and converts (serialize/deserialize JSON) the data types.

We decided to use the Entity Framework because it greatly simplifies database design. It allows for developing a set of C# classes that can then be used to populate and retrive data in the Microsoft SQL Server database. This assures that the objects used to process the data match with what is stored in the database. It also relieves the burden of designing and maintaining a database schema in addition to C# classes. An added benefit of the Entity Framework is that is mitigates any possibility of SQL injection vulnerabilities.

Appointment Filtering Algorithm

The team decided to implement the algorithm as a C# DLL, so that multiple versions of the Algorithm could be developed and slotted into the server. The point of the algorithm is to find available appointments and return optimal appointments for the front desk personnel to use to schedule appointments. This takes into consideration room availability, personnel availability, a range of dates, all appointments in the same day, minimized wait time between appointments, patient availability, and returns the best fit appointments to the front end. (see Figure 3, Appendix III)

Front End

The Front End for the system is written using a combination of HTML, JavaScript, and CSS, and using the BlueBird.js, JQuery, and Bootstrap 4.1 frameworks. BlueBird provides functionality to guarantee that the ES6 compliant JavaScript that the team has written will work in an environment that conforms more to the ES5 standard (Internet Explorer 11). The JQuery framework is a well known and well tested framework for JavaScript that allowed for faster development of the Front End. Finally, Bootstrap 4.1, a CSS framework, was chosen as it provides nice looking UI while also being easy to modify to fit our needs.

Testing Process & Results

Server & Testing

The server's test suite consists mainly of continuous integration deployment through GitLab to a development server. This ensures all components (server, algorithm, and models) build and

deploy properly. This test occurs every time a commit is made on the master branch. If a build or deployment fails, we get an email and notification in GitLab.

In addition, there is a project of unit tests for the server. These tests are to provide manual regression testing to realize unintended externalities of changes. These tests could be integrated into the GitLab continuous integration for automated regression testing. Finally a tool called Postman was utilized to allow for testing of public API's that the server provides to ensure correct responses for given inputs. This type of testing is performed manually using Postman, but can be modified to be ran as part of the integration test suit that is used in GitLab.

Front End Testing

Testing on the front end consisted of acceptance and integration testing performed by the members of the front end team. The subteam decided that manual testing via console output and visual validation that elements were populated correctly was sufficient as there is not a whole lot of critical logic that is ran in the front end. The integration tests consisted of checking the JSON that the server would return after the server had its API established.

Algorithm Testing

The team is using unit testing to test the algorithm for correctness and optimization. We were provided what an optimal schedule looks like in order to try to get our solution as close as we can to that. We also have tests for each individual function to ensure each part of the algorithm is working the way we want it to and tests for the entirety of the filtering process. Unit testing is also used to make sure that all possible combinations of appointments for a day are generated.

Context: Related Products & Literature

Epic Software

The original plan was to integrate our project with Epic's current scheduling software to get retrieve patient, personnel, and appointment data, however, due to privacy concerns we were not given access to UnityPoint's Epic system. With this in mind, we created a work around where appointment data is parsed from a CSV downloaded from Epic's software every so often to see which appointments have already been scheduled and with what providers. Then, on the front end, an employee can enter in all personnel work schedules for our algorithm to use to filter out options.

Appendix I - “Operation Manual”

Demo

A live demo is hosted at https://sdmay18-26.ece.iastate.edu, it is available at the time of this writing, but there is no guarantee as to how long the demo will be running as it is hosted on servers provided by Iowa State University which can decided to revoke the server access at anytime after May 05, 2018.

Installation

System Requirements

The project is designed to be served on IIS 10 on Windows Server 2016 using the .NET Framework 4.6.1. A Microsoft SQL Server must be accessible from this system. We suggest installing SQL Server Express on the same server. The server code was built using Visual Studio 2017, which is the easiest way to deploy the server application to IIS.

The front end is a collection of static files, which need no compilation and can be edited with any text editor or appropriate IDE.

The website (combination of server and front end) is designed to be used with Internet Explorer 11 (but works with other modern browsers).

Server Configuration

On a Windows server that meets the requirements, configure IIS in the following way. Create a new site in IIS (or reuse an existing one); the configuration of that is left up to you to suit your needs. The site should only be served to the **intranet**, and can optionally use Windows authentication and authorization.

Add the front end files to the file system where the site is located.

Within the site, add a new “Application” with an alias of “api” from the IIS Manager. When creating the site and application, file system permissions must be set appropriately to allow the IIS application pool to read the files and the deployment user to write to the application folder. This configuration depends on how you configure your site and deployment.

In SQL Server, a user account must have access. In our example deployment profile in the project source code, the user must have access to .\SQLEXPRESS to create databases (dbcreator role). The user is “IIS APPPOOL\<app pool name>” using Windows authentication (which avoids the need

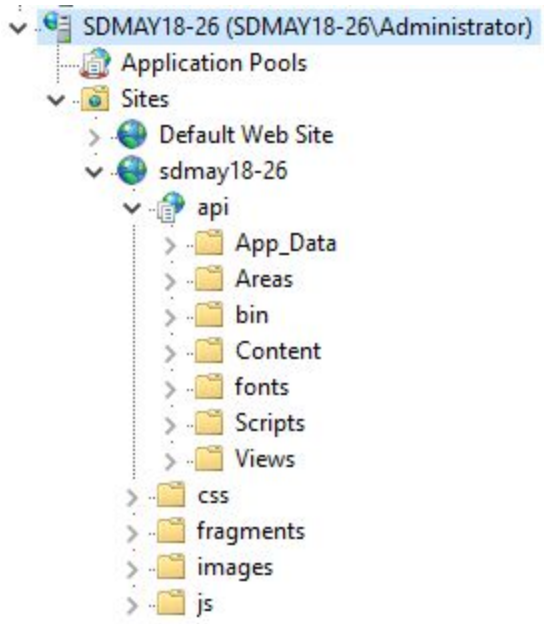
for a database connection password from the web server). The example deployment profile will create a database called “sdmay18-26”.

Open the back end source code in Visual Studio 2017. For an easy deployment, you can do this from your web server. For long-term production, this can be done with the Visual Studio Build Tools 2017 (see the deployment script in the GitLab continuous integration configuration for ideas). In Visual Studio, right click the Backend project from Solution Explorer and click Publish. If you are deploying from the web server, the publish method should be File System. You can set your database connection string, but be sure to set “MultipleActiveResultSets=true”. We have provided an example publish profile containing an appropriate connection string for a local SQL Server Express with the default instance name.

The deployment style is highly dependent on various factors. We suggest using a private GitLab server combined with a GitLab runner to deploy the application to the web server when stable changes are made. This is not a one-size-fits-all solution, so use your own discretion.

In the end, our configuration and deployment looks like this:

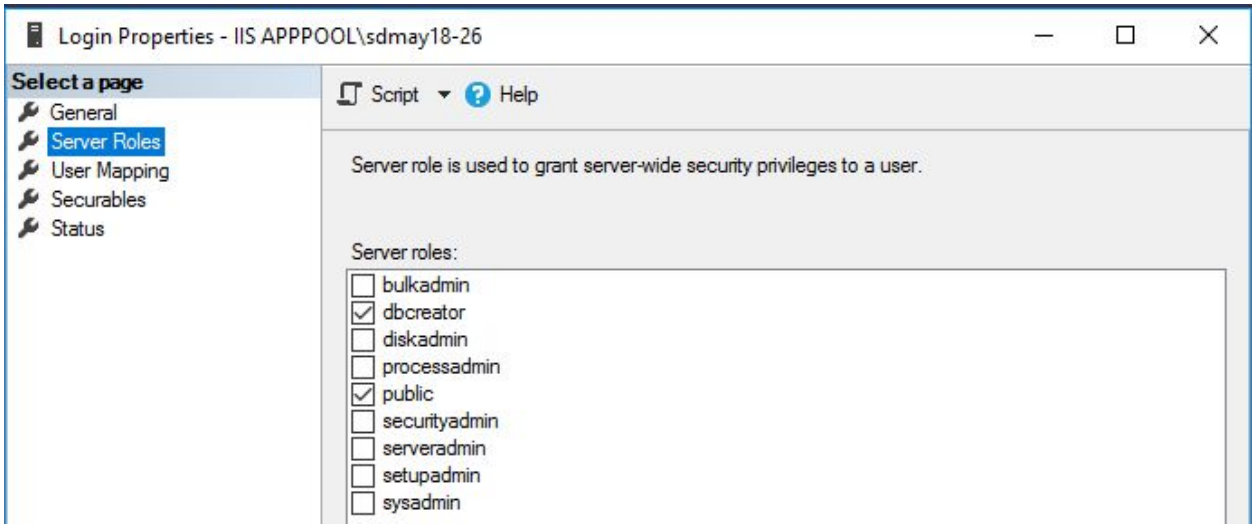
IIS:



File system permissions on both IIS site and api folders. (gitlab-runner is our user for deployment; Administrator isn’t required, but makes life easier)

Type	Principal	Access	Inherited from	Applies to
Allow	IIS_IUSRS (SDMAY18-26\IIS_IUSRS)	Read & execute	None	This folder, subfolders and files
Allow	Administrator (SDMAY18-26\Administrator)	Full control	None	This folder, subfolders and files
Allow	gitlab-runner (SDMAY18-26\gitlab-runner)	Full control	None	This folder, subfolders and files

SQL Server permissions (Security->Logins) for our IIS application pool



Our SQL connection string, models.SchedulerDBContext, is:
Data Source=.\SQLEXPRESS;Database=sdmay18-26;MultipleActiveResultSets=true;Integrated Security=True

Importing Data

Connection Strings

```
Data Source=.\SQLEXPRESS;Database=sdmay18-26;MultipleActiveResultSets=true;Integrated Security=True
```

Running The Importer

In the importer solution the importer runner project is that project that runs the importer. To run the importer ensure that the database connection is set correctly and location for the CSV file is set in Program.cs in the ImportRunner project. The CSV should have the following columns in order:

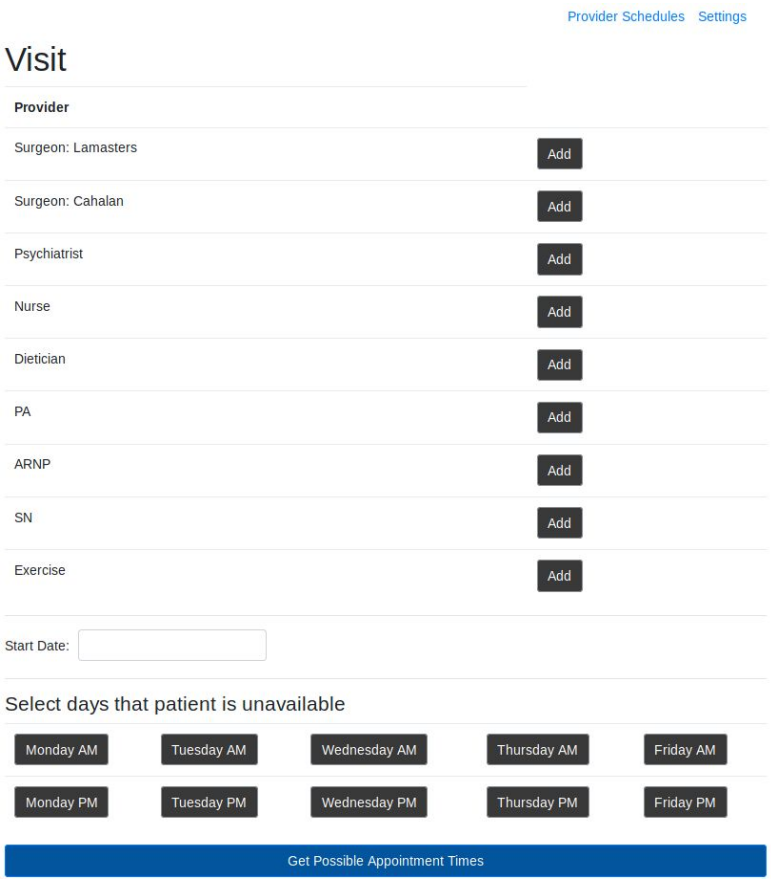
- Date
- Provider/Resource
- Appt Time
- Type
- Status
- Made Date
- Appt Length
- Speciality

Any changes in the number or order of the columns will necessitate a change in the importer.

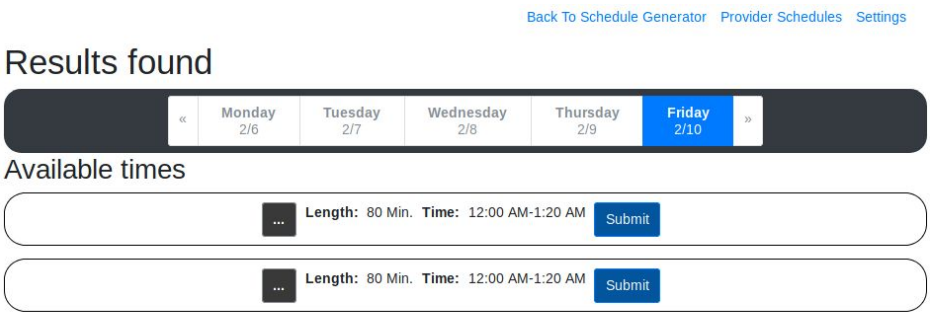
Daily Operation

There are four major pages available to the end user. For normal daily operation the user will interact with the input page and the results page as seen here:

Input Page:



Results Page:



To generate a schedule, the user needs to toggle at least one of the Add buttons that correspond with which provider the patient needs to see. Next a start date needs to be selected for when the

system should start generating schedules after. Finally, select any days that the patient is unavailable (mornings and/or afternoons) and click the Get Possible Appointment Times button.

The user will then be taken to the Results page which displays the appointment times that should have all providers available to see in succession. To view details of each possible appointment time, click the ellipsis button to give a view of the ordering for the appointment. To view a different day click on one of the other day options in the Menu containing Monday to Friday. If you click on a day and nothing happens this means that no possibilities exist for that day. To view the next or previous week click on the left or right arrows to look at the next or previous week.

Finally when an appointment slot is chosen, click the Submit button to tell the system which appointment is being used and then go enter in the data into Epic. This step helps to ensure that the data that this system is using is as up to date as possible until the next time that it is given the data from Epic.

Aside from those two pages there are two other pages that are used to set parameters for the system. The first of these two is the Provider Schedules page, which can be accessed from any page via a link at the top of the page. This page allows for entering in the two surgeon's availability for each week during a set period of time as well as enter in any times that the surgeon will be out of office.

The second of the the two auxiliary pages is the Settings page. In this page the user can set how long each provider will take for each appointment as well as other options such as maximum wait time that a patient waits between seeing each provider and the number of general purpose rooms that are available in the clinic.

Appendix II: Alternative versions

Originally, it was planned that our system would have direct communication with the Epic system that UnityPoint currently uses to store their data. After looking into the matter further, it was found that there would be no possibility of gaining access to the Epic API's needed to properly retrieve and update the data needed for the system to work properly.

Appendix III: Diagrams

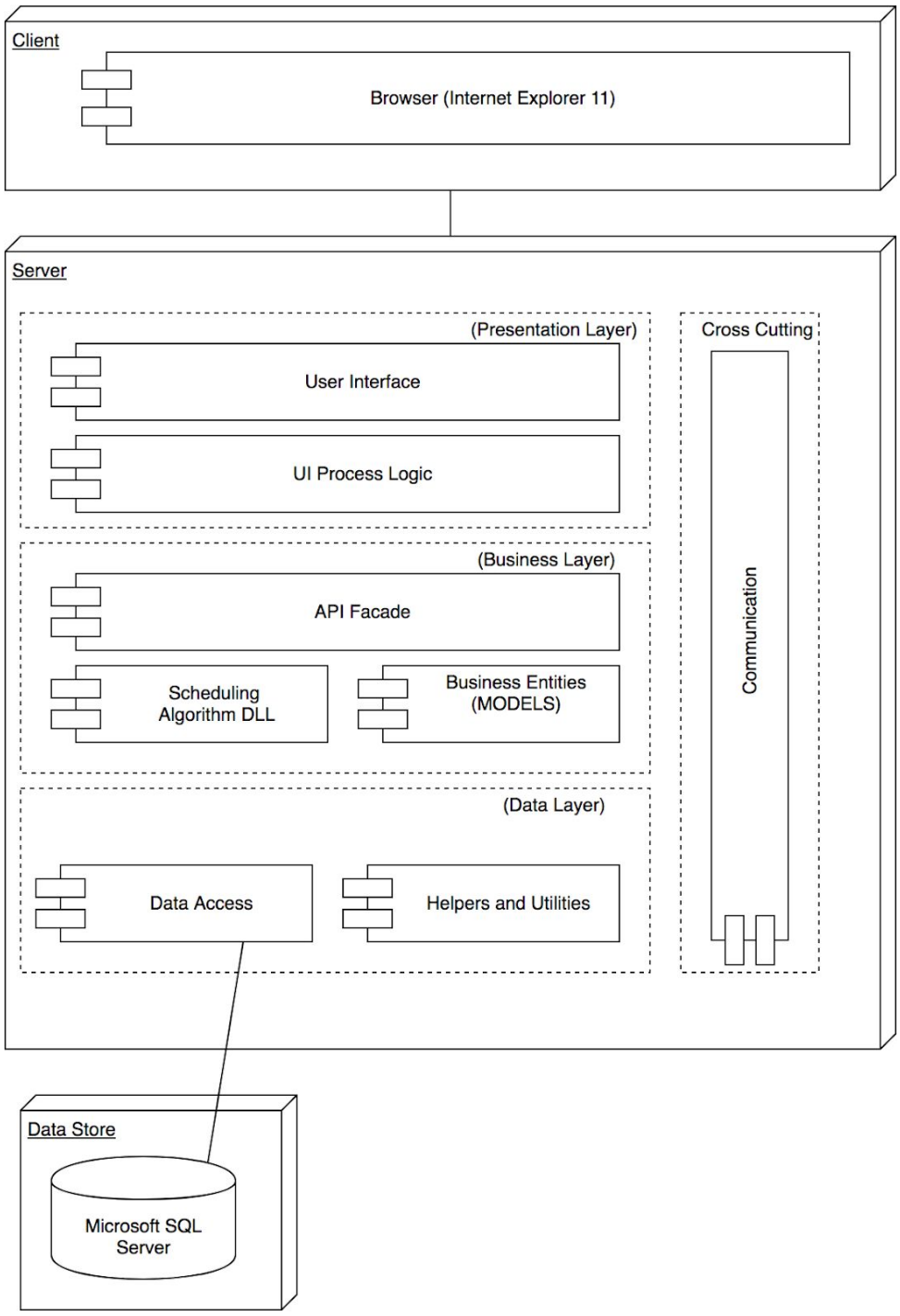


Figure 1 System Architecture Overview

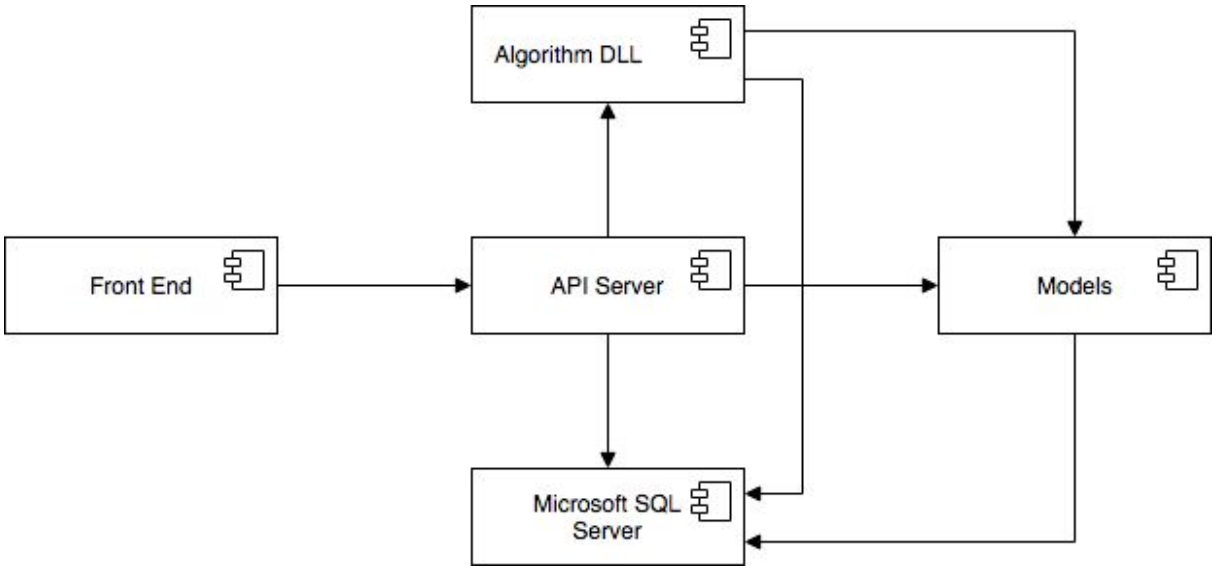


Figure 2 Component Diagram

Front End Component

This component houses all of the UI logic as well as all views that the user can see. Any interactions, such as request for appointment scheduling options generation, are sent to the API Server and the results of the request are displayed in this component.

API Server Component

This component serves the web pages and files that the Front End uses as well as providing API's that the Front End can consume to perform operations on the system. Requests either trigger a call to the Algorithm DLL or the Microsoft SQL Server.

Algorithm DLL Component

This DLL is used by the API Server to generate possible appointment times and combinations for the API server.

Microsoft SQL Server Component

This is the Database where all settings and scheduling information is housed.

Models Component

This component provides Object definitions that are used in the API Server, Algorithm DLL, and the Microsoft SQL Server components.

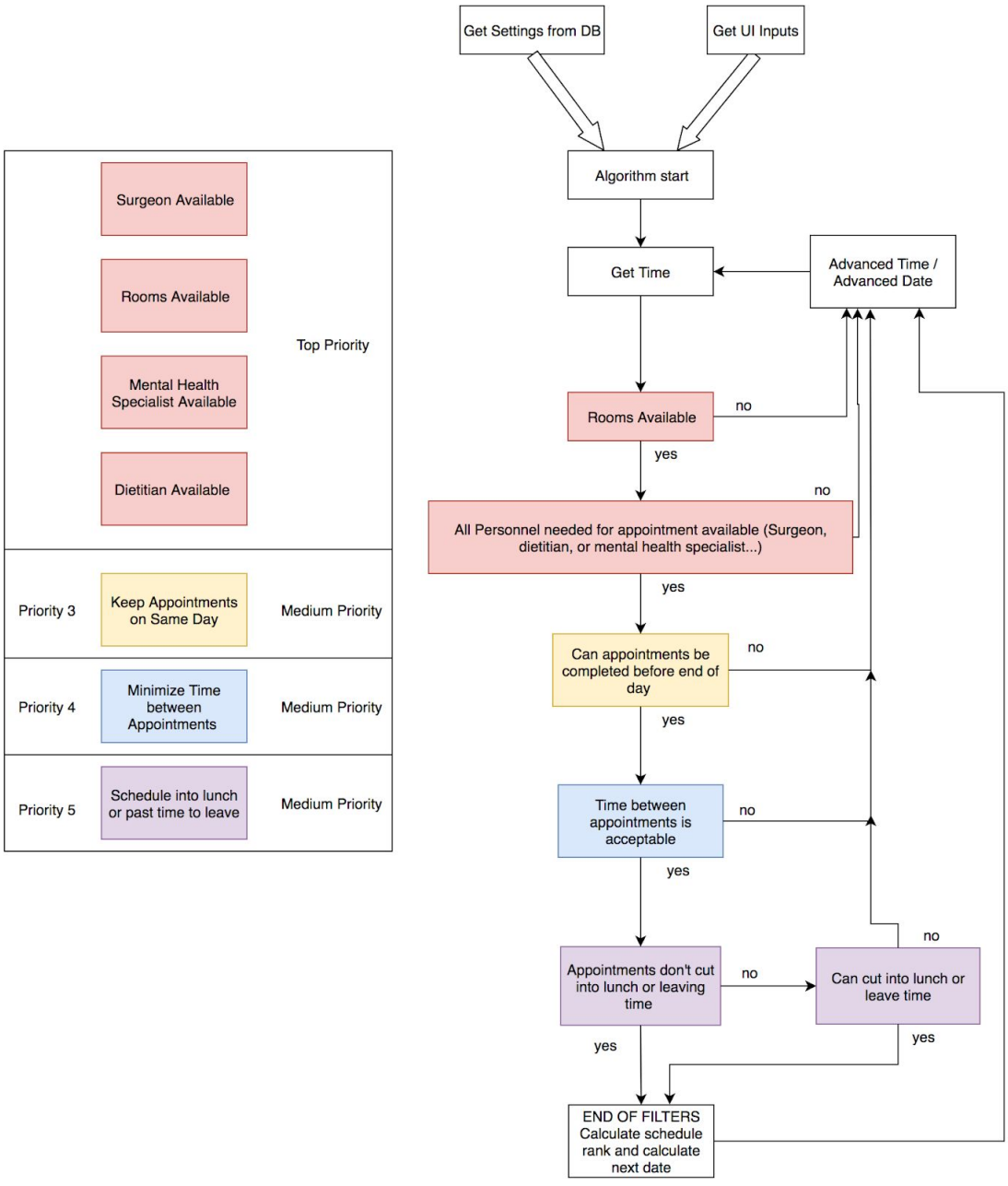


Figure 3 Algorithm Logic