Electronic Scheduling Tool for Surgery Clinic

DESIGN DOCUMENT

sdmay18-26
UnityPoint
Srikanta Tirthapura
Joel May - System Admin
Repo and 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/

Revised: 2 December 2017

Table of Contents

List of Figures/Tables/Symbols/Definitions

List of Figures

Definitions

Provider:	A healthcare worker that works with UnityPoint clinic. This includes but is not limited to; surgeons, dietitians, mental health specialists, and fitness coaches.
Patient:	A person who is using the services provided by the UnityPoint weight loss clinic.
GitLab:	A service that allows for remote storage of git repositories as well as Continuous Integration support.
Schceduler:	A worker at UnityPoint clinic that schedules patients' visits.
Time Block:	A section of time between two times (i.e. 11:00 am to 1:30 pm).
EMR:	Electronic Medical Records. A system/database used to track doctors, patients, and other resources of the clinic.
Epic:	UnityPoint clinic's EMR system.
JSON:	JavaScript Object Notation.
RESTful:	Stands for Representational state transfer (REST) that allows for systems to communicate between each other. Responses on a RESTful service are comprised of standard HTML headers along with XML, HTML, or JSON in the body of the response.

1 Introduction

1.1 Acknowledgement

We would like to thank Vanessa Calderon for her assistance in resource gathering for this plan. Her and her team have provided also of valuable information to us while creating this project plan. We would also like to thank Angela Claytor, the administrator of the Weight Loss Clinic for meeting with us and explaining the processes of the clinic. Finally we thank Gretchen Wilhelmi, Derek Nohr, and Brijeet Akula from the UnityPoint Information Technology for providing us with technical details for this project.

1.2 Problem And Project Statement

The purpose of the Electronic Scheduling Tool for Surgery Clinic is to make easier for scheduling the various types of appointments. We also want to make better use of the providers and other resources as well. We are going to create a tool to aid in creating an optimal schedule.

The end product will be a web application that the clinic schedulers will use. The schedulers will be able to enter the dates, types of appointments, and providers. The system will generate a schedule that reduces the wait time for the patient and makes best use of resources. Our application will use a custom algorithm to come up with schedule suggestions. We will integrate with UnityPoint's Electronic Medical Records System to pull providers schedule, room configuration, and other information to help reduce data duplication. The algorithm will try to come up with the best schedule for all parties involved.

Currently, the schedulers spend a lot of time trying to fit appointments in. This process results in high patient wait times, ineffective use of providers, and results in provider's seeing less patients. Reducing patient wait time increases patient satisfaction and makes better use of the patient's time. By using the provider more effectively it means that more patients can be seen which is good from a business perspective.

1.3 Operational Environment

The scheduling software will be on a server and safe from any outside hazardous conditions. The server will be running Windows Server. We will be using IIS for the web server. The application will be hosted in a UnityPoint server room.

1.4 Intended Users And Uses

This product is to be designed for the receptionist at the clinic to help them schedule appointments while interacting with the patient. The receptionist should be able to generate and filter appointment times, add the desired time blocks to the schedule, and edit personnel availability. Furthermore, admins should be able to modify room availability, update the number of rooms the scheduler should consider, and manage personnel details.

With these uses in mind, in order to ensure maximum satisfaction for the receptionist we will need to design a front end that is easy to navigate and use. In order to provide maximum satisfaction for the patient, the receptionist should be able to quickly and easily convey possible time blocks that were generated and relay that information to the patient. This means our software should have short loading times and information should be displayed in a well thought out manner. Finally, this software should allow the user to add, edit, and modify data. This means we will need to take into consideration how the new information will affect the other variables and update those accordingly

to avoid scenarios like two patients sharing resources at a specific time and causing wait time for the patient.

1.5 Assumptions And Limitations

1.5.1 Assumptions

1. There will be no more than ten users actively using the system at a time.
2. The users will use the latest version of Internet Explorer (11) and Chrome (61).
3. There are a set of healthcare providers.
4. There are a set of rooms.
5. Some providers are limited to a subset of rooms.
6. There are a set of appointment types, which consist of a set of providers, durations, and optional sequencing information.
7. The set of appointment types is not rigid, and the scheduler can create a combination
8. The system will only suggest up to five available appointment times for each day based on the query.
9. Queries will be limited to searching a range of up to a month.

1.5.2 Limitations

1. The back-end will be running on-site.
2. The web front-end will only be accessible on the local network.
3. The patients will not have direct access to this system.
4. Epic must be used to obtain the room, provider, and schedule information.

1.6 Expected End Product And Deliverables

The end product is a proof of concept website that a scheduler can login and use to schedule patients within their Epic system. We plan to install and set the software up on UnityPoint's web server. We also will deliver the source code, which can be configured to be used for other clinics that rely on Epic.

We will also deliver design documents and the system architecture diagrams, in case anyone wants to modify or extend our software. The documentation will mostly be within the source code, but we will also provide API documentation for internal APIs we create.

2 Specifications And Analysis

2.1 Proposed Design

The current implementation of the scheduler is a client-server application that uses a browser to host the client (see figure 1 in Appendix). The following is a breakdown explanation of the different components in the application.

Component	Responsibility
Browser (Internet Explorer 11)	The standard web browser that UnityPoint uses for internet access, and the main browser that the application will be accessed from.
User Interface	Everything that the user can see and interact with. In this case this is realized with HTML, CSS, and JavaScript as a website.
UI Process Logic	All logic that exists in the website. This includes data validation, page flow, system API calls, etc.
API Facade	Provides a series of end points that the client can call upon while allowing for more configuration of the server logic without the client having to worry about it.
Scheduling Algorithm	The main engine for the server, this algorithm will provide all of the logic for making decisions on when we can schedule a patient (See figure 3 in Appendix).
Business Entities	These are objects that exist solely to assist the scheduling algorithm function.
Data Access	This component allows the rest of the server to communicate with Epic for both data retrieval and data storage.
Helpers and Utilities	Provides functionality to help other sections of the system have less redundant logic, and to simplify the logic in other parts of the system.
Security	This component crosses through all three layers of the system as it consists of the authentication that we will need in order to ensure security and the ability to communicate with Epic.
Communication	All communication standards are specified in this component.

This structure is to satisfy the needs of the client. Mainly that it will be able to be ran on the machines at UnityPoint clinic using Internet Explorer 11, to help the scheduler to schedule patients in a timely and easy manner. All of the scheduling should be stored in the current Epic system that is used by UnityPoint, and the data that is stored locally in the server will loaded from Epic using a file that is autogenerated by Epic on a set time interval.

2.2 Design Analysis

At this point we have diagramed a class structure for the server side application that will act as the proxy between the client and Epic (See Figure 2 in the appendix) as well as house the scheduling

algorithm that we are implementing (See Figure 3 in the appendix). This design represents the business and data layers in the overall architecture (Figure 1 in the appendix). In this server architecture the API's available for the client to call upon don't directly communicate with Epic, instead we do some processing of the requests and responses between Epic and the client to ensure that only relevant data is sent between the two. This also allows the server some flexibility on how it fulfills the requests of the client, and will allow us to easily return mock data during our testing phase.

All of that being said this architecture does have a few issues that come with it. In this architecture the list of rooms in the WeekDate object needs to initialized, but currently there is no easy way of storing room information in the Epic Database. One solution that we have thought of is to say that at a given time slot a room is occupied if a provider is in need of a room and has yet to have one assigned to them between two times. Also, there will be more processing required when using this architecture as all queries to Epic need to be converted into objects that can be used.

Finally, there are some advantages to this architecture. First off, since we are making a facade between the client requests and all other operations in the server, we have the flexibility to update parts of the server without the client needing to be adjusted. Also because we are converting data from Epic into objects instead of always using the raw JSON we will have an easier time working with the data in both the client requests and the scheduling algorithm. The ease that is being referred to is less logic required in the building of responses and the algorithm because the data is in a more workable format that has had some processing done to it.

As we continue to move forward the architecture for the server itself may have some minor changes and tweaks due to communications with Epic, but there is a high likelihood that those would be minor additions of helper classes. That being said the overall structure of the whole application (Figure 1) won't be subject to such changes as that structure is agnostic of the requirements of Epic and thus won't be affected by it.

3 Testing And Implementation

For functional and nonfunctional testing we will need to insure that our backend, frontend, and algorithm are working correctly.

3.0 Requirements

3.0.1 Functional Requirements

3.0.1.1 Backend

Our backend is going to be running on a server and will be fetching data for the algorithm through Epic API calls.

- The backend shall call the Epic API to receive scheduling data every time a request is made from the algorithm.
- The backend shall encrypt any saved data.

3.0.1.2 Algorithm

- When the frontend requests to schedule an appointment, the algorithm shall call the backend for the updated scheduling data.
- When the frontend requests to schedule an appointment, the algorithm shall return a list of dates and time when an appointment can be scheduled.

- The algorithm shall not schedule appointments during an employee's lunch break.
- The algorithm shall not schedule appointments when an employee is not working.
- The algorithm shall not schedule appointments to close to when the employee leaves for the day.
- The algorithm shall suggest appointments during other appointment times under specific circumstances.

3.0.1.3 Frontend

- The frontend shall allow the user to search for appointments by date and time.
- The frontend shall allow the user to search for appointments by available time slot.
- The frontend shall allow the user to search for appointments by available employee.
- The frontend shall allow the user to create different kinds of appointments.

3.0.2 Nonfunctional Requirements

3.0.2.1 Backend

- The backend shall take no more than five seconds to retrieve scheduling data for the algorithm.

3.0.2.2 Algorithm

- The algorithm shall take no more than ten seconds to generate scheduling suggestions.

3.0.2.3 Frontend

- The frontend shall load in no more than one second on a powerful client computer.
- The frontend shall be easy enough to work with that someone with no experience can figure out what they need to do.

3.1 Interface Specifications

For this project we plan to use GitLab's Continuous Integration to automatically build and test our project every time a developer pushes, this will insure that every developer's code is working when it is committed. Figure 2 in the appendix shows how the backend, algorithm, and frontend interface with each other. All communications between the front and back end are done using RESTful API endpoints, to ensure ease of development and maintainability.

3.2 Hardware And Software

For testing the front end we will be using Selenium. With Selenium we will be able to automate the testing of our front end by simulating a user clicking and entering data into our websites UI.

For the backend we will need to use C# unit tests to verify that it is retrieving the scheduling data correctly and that the algorithm is functioning correctly.

3.3 Process

User Interface	The user interface will be tested using Selenium regression tests. And will be put through User Acceptance Testing to make sure that our customer understands how the interface looks.

UI Process Logic	The UI process logic will be tested using Selenium regression tests along with the User Interface.
API Facade	For this we will have to verify that the API is returning valid data. Manual testing will be done with Postman and will be tested using automation in C#.
Scheduling Algorithm DLL	We will use C# unit tests to verify that the algorithm is returning valid scheduling times.
Data Access	We will need to verify that we are receiving data from our database.
Helpers and Utilities	All of the helpers and utilities will be tested using C# unit tests.
Security	We will need to verify that all data is encrypted and that users must input valid credentials in order to use the program.

3.4 Results

So far we have not started on any testing.

4 Closing Material

4.1 Conclusion

So far we have done our requirements gathering and have developed an architecture for our project's solution. In our project we need to meet a specific list of goals for it to be acceptable. We have determined that our goals for the project are to: Maintain a list of rooms and providers, develop an efficient scheduling algorithm for the rooms and providers, and develop an interface to interact with the algorithm to schedule appointments. Through our requirements gathering we determined the best solution is to make a web application that interfaces with Epic. This solution was chosen as Epic already maintains the rooms and providers, and a web application is suited to this as the backend can query Epic when handling scheduling requests freely and the user will have a web page to interact with to submit the scheduling requests. This was chosen as the alternative of a standalone app as it would have to be installed and maintained on every individual machine.

4.2 References

At this time, we have no materials to reference.

4.3 Appendix

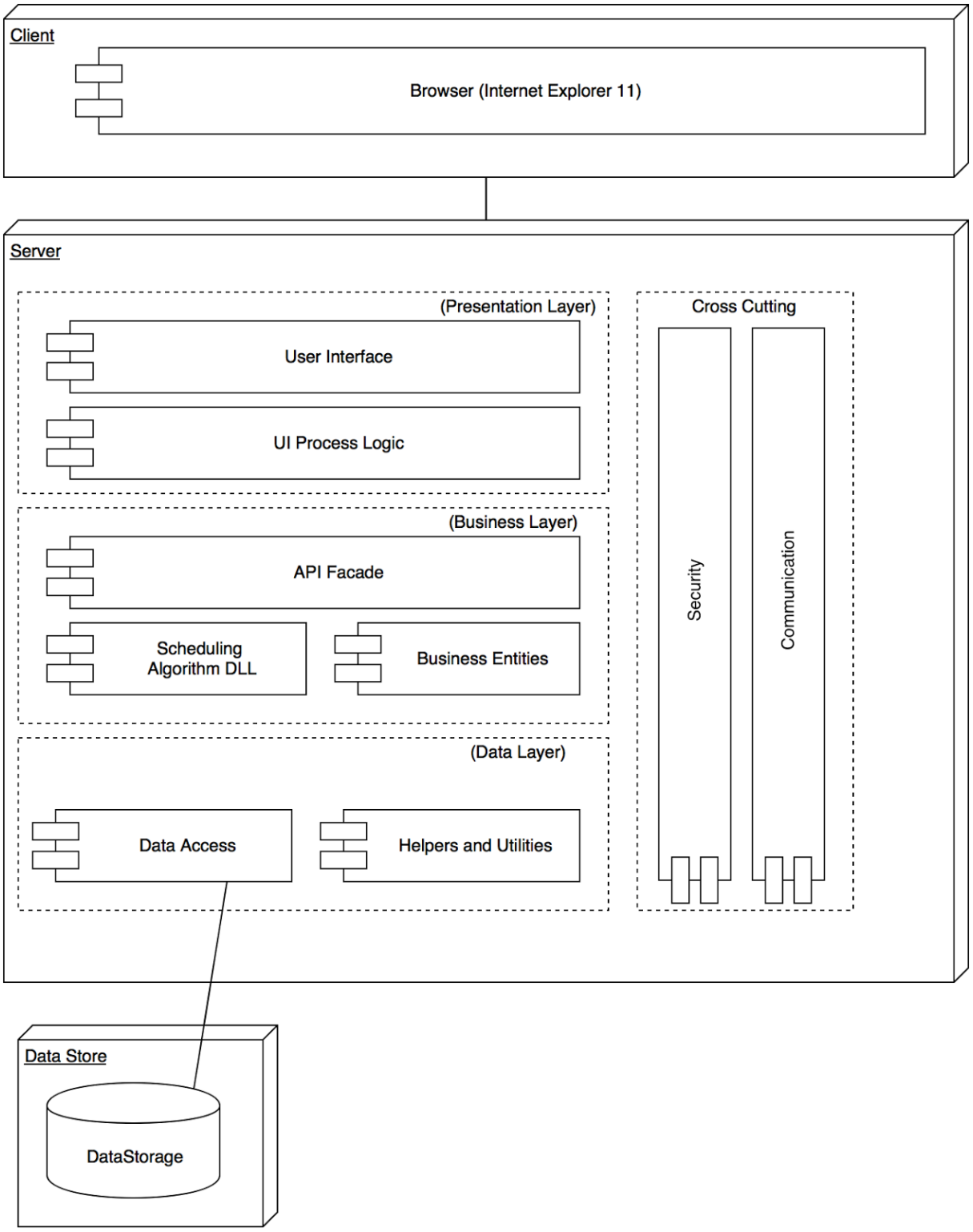

Figure 1: Component Diagram

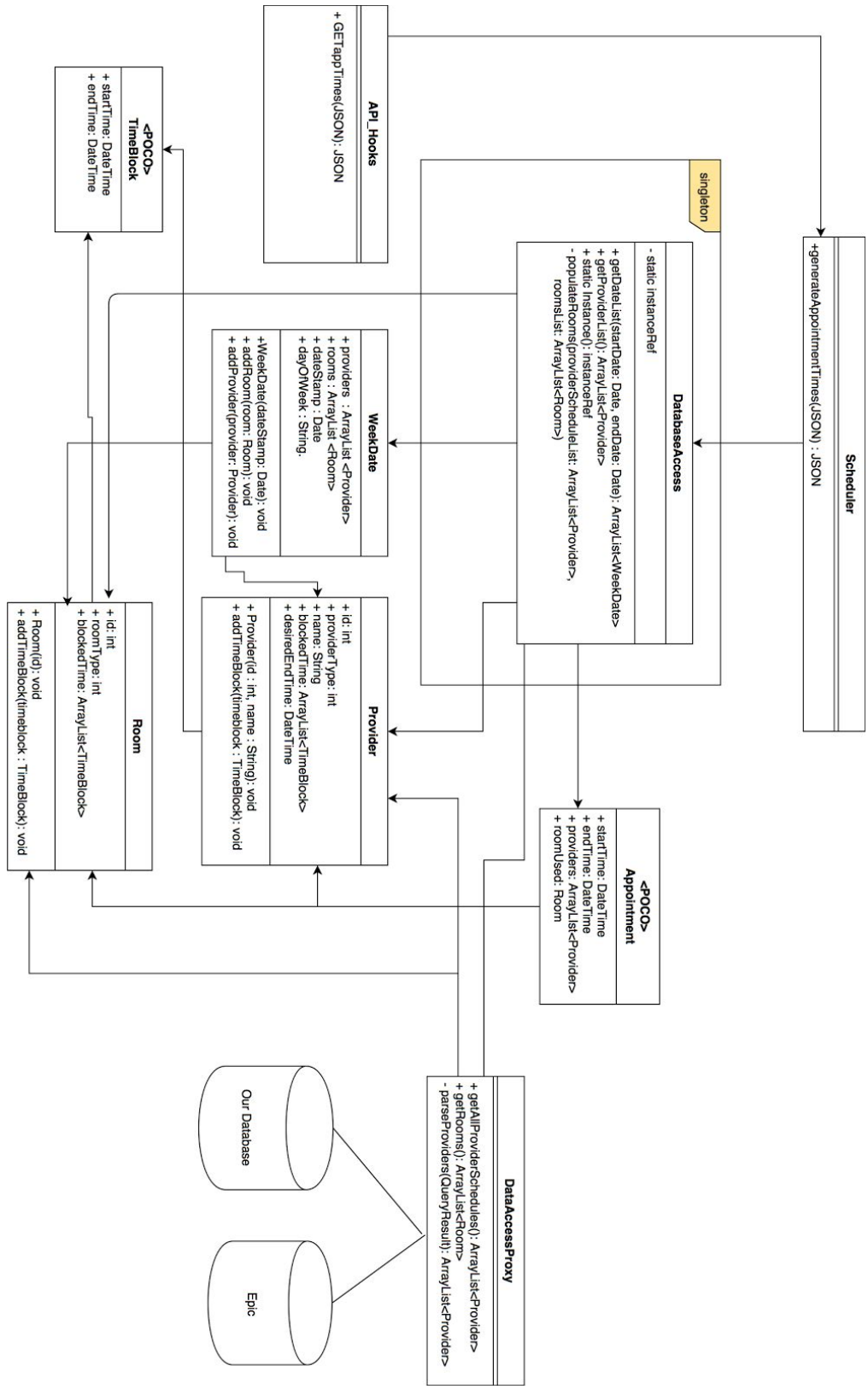

Figure 2: Class Diagram

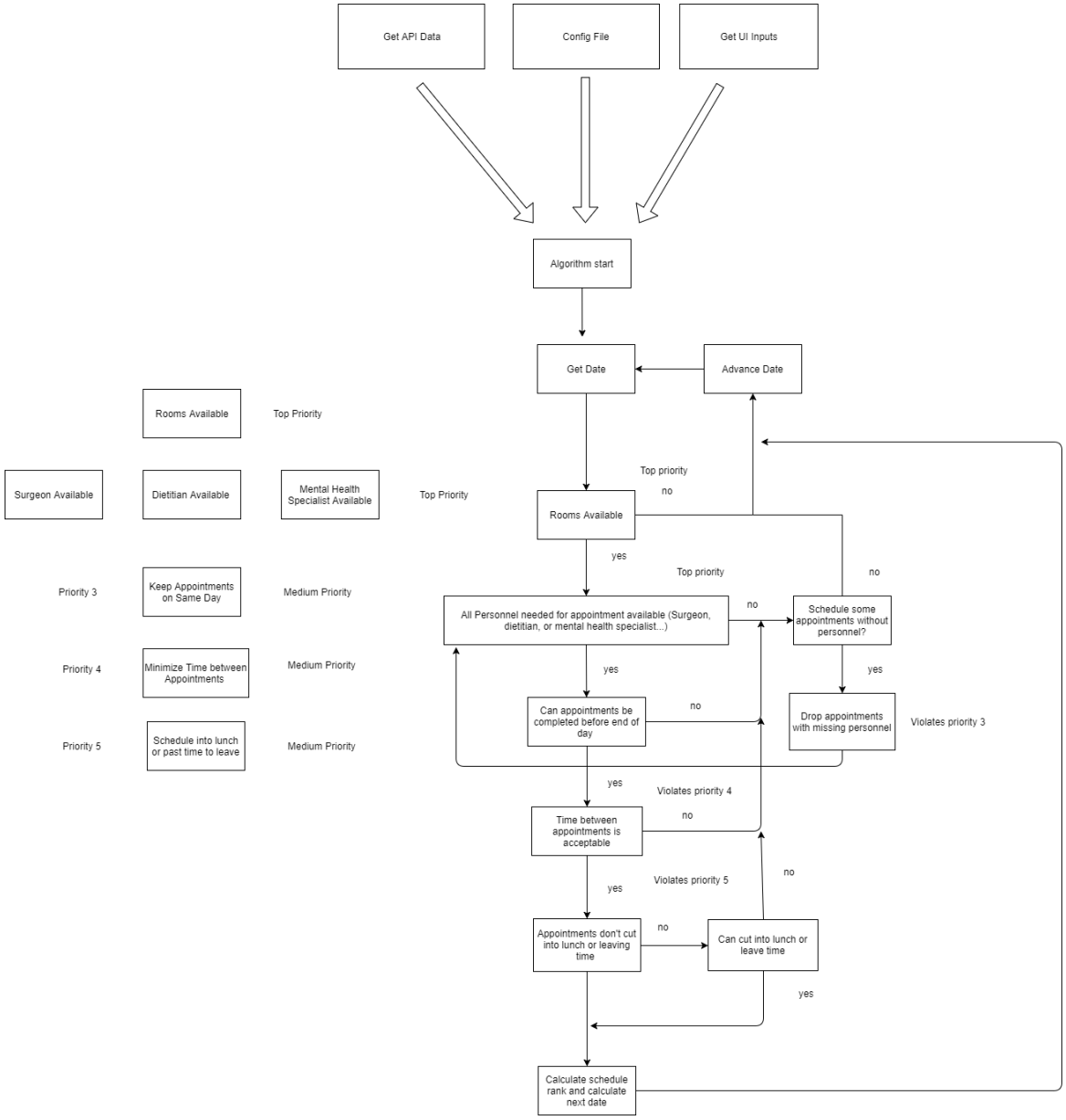

Figure 3: Algorithm Process Diagram