Electronic Scheduling Tool for Surgery Clinic

PROJECT PLAN

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: 22 September 2017

Contents

Preface 2

Definitions 2

1 Introduction 3

1.1 Project Statement 3

1.2 Purpose 3

1.3 Goals 3

2 Deliverables 3

3 Design 4

3.1 Previous Work/Literature 4

3.2 Proposed System Block Diagram 4

3.3 Assessment of Proposed Methods 4

3.4 Validation 5

4 Project Requirements/Specifications 5

4.1 Functional 5

4.2 Non-functional 5

4.3 Standards 6

4.4 Test Plan 6

5 Challenges 7

6 Timeline 7

6.1 First Semester 7

6.2 Second Semester 8

7 Conclusion 9

8 References 9

9 Appendices 11

Preface

Definitions

Epic	Electronic Health Record software for midsize and large medical groups, hospitals, and integrated healthcare organizations
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
Time block	A length of time that has a starting time and and ending time
Pairing preferences	The preference of which provider is scheduled with other providers, or with a patient
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 web site that is only accessible on the business's local network

1 Introduction

1.1 Project Statement

We are developing an electronic scheduling tool for UnityPoint's weight loss clinic in West Des Moines. This needs to take the available staff and rooms and schedule them as efficiently as possible for patient visits, as well as take in a set of user-defined constraints on when and how things are scheduled.

1.2 Purpose

The purpose of this project is to develop a more efficient scheduling system that can save staff time and patient waiting time. Patient waiting time has proven to be problematic for the clinic.

1.3 Goals

We will create a web application with the following features.

- Retrieve information from Epic
- Schedule a patient time block
- Assign employees to time blocks
- Input scheduling constraints
    - These include days of the week that a patient can come in, what type of provider that they need to see, and which provider they have preference for.
- Schedule non-patient busy time for employees where they cannot be scheduled

The following are our stretch goals for our project's features, which we will implement if time allows.

- Update the information in Epic
- Set employee pairing preferences
- Set regular employee schedules
- Responsive design

2 Deliverables

We will deliver an intranet web application that connects to UnityPoint's Epic electronic medical records (EMR) system and can be used to determine available appointment times for patients. This system should be able to take in a set of constraints to change what types of appointment types should be prioritized. The proof of concept to this which will meet the bare minimum of requirements is projected to be ready for presentation early in the middle of the second semester.

3 Design

3.1 Previous Work/Literature

At the moment there do exist several products perform similar scheduling tasks. One such product is the Iowa State Course Schedule planner (see references), which allows a student to input courses and get back possible course schedules for a semester.

There are many tools available for enterprise use, however, they do not meet the unique criteria that the clinic needs. Our project is unique because the tool has to interface with Epic, and the algorithm must pair up specific doctors to specific patients and rooms. The scheduler also has to be able to schedule appointments for both providers and patients. For these reasons, those other schedulers would not quite fit their needs.

The problem at hand is also similar to the job-shop scheduling problem discussed in Com S 311, which is an optimization problem generally regarding what our client wants, a scheduler that assigns jobs to resources at particular times. Due to jobs already being scheduled and the general approach that clinics use for scheduling, our solution for this problem is to use a priority queue to sort through the appointments.

3.2 Proposed System Block Diagram

In this architecture (see appendix Figure 1), we have three systems that we are working with. The three main components are the client, the server and Epic systems. The client is how the scheduler will interact with the system. Epic systems is the already existing system that we are interfacing with that the clinic uses to store patient and provider information as well as schedules. Finally, there is the server which is where the main bulk of the logic for the system that does all of the processing and contains the algorithm that we are using to get possible appointment times.

3.3 Assessment of Proposed Methods

For this problem, there is a basic but tried and true method that we can follow when developing the system. This method is the client-server method of implementation. What this means is that we build a client (think website, desktop application) that the end user will see and interact with, and and server that is stored somewhere away from the user that handles all of the processing, storage, and communication amongst all of the client applications.

The reason that this is a strong candidate for how we implement the system is that it allows us to be flexible. We can start designing the system earlier and get the server-side processes figured out prior to planning out the end user implementation. This allows us to leave the possibility open for a website or a desktop application for the end user. This also allows us to be more confident that our system has more security as all interactions involving multiple users can happen away from a single user.

Another option that we have is to focus on building a website, that does all of the processing in the client (browser), and a server that simply delivers the page to the end user. However, this method is less than ideal as it gives the end user access to API's (see references) which could lead to security

issues. Along with that, performing computations or logic would be limited by the specific machine and the web browser that the user is interacting with.

Among these two options, we know that the server client method would be ideal for our system, and that is what we will move forward with.

3.4 Validation

In order to confirm a quality solution, we will implement testing procedures ranging from specific units of code all the way to manual testing. Luckily, the base problem that we are trying to solve does have a fairly clear cut requirements. The main indicator that we will look for is whether the system produces good suggestions: arranging appointments with various medical professionals as closely as possible while taking up the least time possible and avoiding down time for staff.

Our main requirements for success are the following: mitigating wait time, scheduling more appointments and helping the personnel at the front desk schedule them. One of the main ways we intend to validate these requirements is by comparing the number of appointments our scheduler schedules in a day with past appointments scheduled per day. We will also compare the amount of wait time to the past amount of wait time. All of this will be tested through a testing suite. Finally, to make sure the application is easy to use, we will reach out to both the clinic and our client for feedback on intuitiveness which will be done during our user acceptance testing.

4 Project Requirements/Specifications

4.1 Functional

Application that helps the client with scheduling - By the end of this project the client should receive an application that helps them with scheduling appointments for patients and employees.

Epic Information Retrieval - Application should be able to retrieve information from the client's current Epic scheduling software in order to suggest potential appointment times.

Scheduling Algorithm - The client would like an algorithm that can efficiently schedule multiple appointments for people with as little waiting for the patient and employee as possible. The algorithm must also be able to adapt to added or removed doctors/nurses or patients canceling an appointment.

Security - The software may contain sensitive information such as the patients or employees contact info. This data should not be accessible outside of the hospital and should be encrypted and password protected.

4.2 Non-functional

Load in under five seconds - This software needs to be able to determine the best times for a patient to schedule an appointment. It needs to be able to generate multiple options for appointments so that the patient and the scheduler can quickly determine the next appointment and get back to their work.

Flexibility - The client has requested that this software be as flexible as possible. They cannot predict their future needs so this software needs to be able to accommodate the addition or removal of rooms to schedule in, nurses and doctors being out for the day, and patients cancelling. The client suggested that this software may also be useful for other areas of the hospital so it should be applicable to those areas as well.

Usability - This software needs to be easy to use for someone who has never used the software before. For example if a substitute was ever needed for the day they should be able to learn the software as quickly as possible. At most it should take no more than five minutes to become proficient in the software with help from an experienced user.

4.3 Standards

HIPAA - This software must be HIPAA compliant. We are working with sensitive data so it needs to be secured.

JavaScript Style Guide - We will select a style guide for JavaScript when we start development.

Shared Repository Model - Developers must request a merge request before merging with master

UML Standard - We will use this standard for use case, architecture, deployment diagrams.

Continuous Delivery - As a team we should be committed to keeping Master branch bug free and ready to deploy at any time.

4.4 Test Plan

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	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.

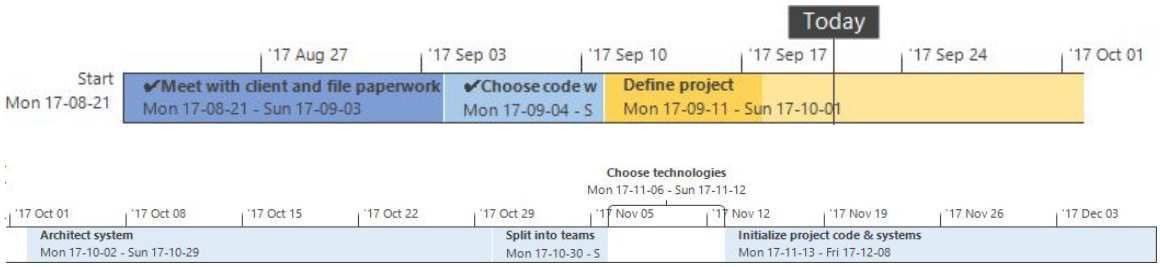
5 Challenges

A large part of our project is a scheduling algorithm. Only two of our group members have completed Com S 311, Iowa State University's undergraduate algorithms course. This could possibly cause issues with design of scheduling algorithm due the team's lack of formal knowledge of scheduling algorithms.

One of the goals of the project is to integrate with UnityPoint's Electronic Medical Records (EMR) system, Epic. Since we do not have any knowledge of Epic we do not know how much work we will need to put into integrating our code with Epic. Integrating with Epic is land of unknowns and therefore could be challenge.

6 Timeline

6.1 First Semester



Past:

- 2 weeks: Meet client and file paperwork.
- 1 week: Choose general code workflow and standards.
- 1 week: Split into teams: front-end, back-end, and algorithm.

Current:

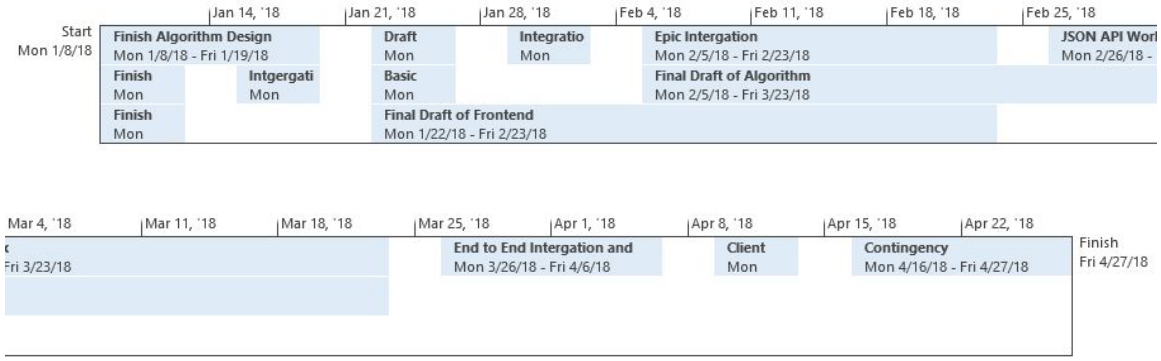
- 3 weeks: Define project and technological constraints. Get project requirements.

Future (~10 weeks):

- 4 weeks: Architect system.
- 1 week: Choose frameworks and languages/platforms. Choose style guides.
- 3 weeks: Initialize project code and systems (CI, test framework).
    - Front-end team
        - Create nonfunctional pages
    - Backend Team
        - Create end points
        - Setup Testing
        - Setup CI

- Algorithm
  - Working planning how it be implemented
  - Start small parts of the algorithm

6.2 Second Semester



- Finish Algorithm Design
  - Algorithm Team
  - Target Date: January 19
  - After this is completed we should be able to start implementing the algorithm
- Finish Front-end Prototype
  - Frontend Team
  - Target Date: January 13
  - This should give the user a good idea of how the software will look. Will not be fully functional.
- Finish integrations mock for front-end
  - Backend Team
  - Target Date: January 13
  - Since we will not have integrations with Epic and Algorithm done at this point, the backend team should create mock endpoints that both teams agree on so that both parties can work on their respective problems
- Integration of Backend and Frontend
  - Backend and Frontend Teams
  - Target End Date: February 1
  - This time will be spent working out any issues between the two teams
- Draft Algorithm output and stub methods
  - Algorithm Team
  - Target Date: February 1
  - The goal is get a good skelton done and draft mock endpoints for the backend team to hit
- Integration with Algorithm and Backend
  - Backend team and Algorithm Team

    - Target Date: February 10
    - The goal of this week is to ensure that we can communicate
- Integration with Epic
    - Backend Team
    - Target Date: February 28nd
    - During this the backend team will work on getting all information need from Epic
    - into our system.
    - Target date March 1
- Final Draft of Algorithm
    - Algorithm Team
    - Target Date: March 21
    - After this done the algorithm should be scheduling appointments.
- Final Draft of Front End
    - Frontend Team
    - Target Date: April 1
    - The front end should be most usable and ready for the customer.
- Integration with Frontend and Backend
    - Target Date: April 8, 2017
    - Backend and Frontend teams
- Improvements based on customer feedback / Contingency
    - End Date: May 4, end of the course
    - The purpose of this time is make changes based on the feedback we get from UnityPoint. Since, we will run into issues in the above timeline this time being scheduled to pad that. If we have extra time feature that we marked as stretch goals will be implemented here.

7 Conclusion

To sum things up our goal is to retrieve data from Epic and use that data to make it easier for UnityPoint Clinic to schedule patients. The primary result should be that the clinic increases the number of appointments per doctor/dietician and the secondary goal is for patient wait time to be decreased.

8 References

"Health Insurance Portability And Accountability Act of 1996." U.S. Government Publishing Office. August 1996. Accessed October 24, 2017. https://www.gpo.gov/fdsys/pkg/PLAW-104publ191/html/PLAW-104publ191.htm.

Humble, Jez. "Continuous Delivery." What is Continuous Delivery? - Continuous Delivery. Accessed October 24, 2017. https://continuousdelivery.com/.

"Schedule of Classes." Schedule of Classes: Iowa State University. Accessed October 24, 2017. http://classes.iastate.edu/home.

Seshness. "Shared Repository Model for Pull Requests and Code Review." Gist. October 23, 2012. Accessed October 24, 2017. https://gist.github.com/seshness/3943237.

"Welcome To UML Web Site!" Welcome To UML Web Site! Accessed October 24, 2017. http://www.uml.org/.

9 Appendices

Figure 1