Inpatient Therapy Scheduling System

Final Report

Team 44
UnityPoint Health - Des Moines

Mathew Wymore - Team Advisor
Alexis Cordts - Backend Developer; Team Lead
Timothy Ngo - Backend Developer
Lucas Knoll - Backend Developer
Andrew Swenson - Frontend Developer
Megan Bailey - Frontend Developer
Veronica Torres - Frontend Developer

sdmay21-44@iastate.edu
https://sdmay21-44.sd.ece.iastate.edu

Table of Contents

1 Revised Project Design

1.1 Design Evolution 2

1.2 Requirements 2

1.3 Relevant Standards 3

1.4 Engineering Constraints 3

1.5 Security Concerns and Countermeasures 4

1.6 Implementation Details 4

1.7 Testing Process and Testing Results 10

1.8 Related Products and Literature 10

2 Operation Manual 11

2.1 Step by Step Instructions 11

3 Alternative Designs 15

3.1 Initial Variants 15

3.2 Failed Variants 15

4 Other Considerations 16

4.1 Relevant Additions 16

5 Code 17

5.1 Code 17

1 Revised Project Design

1.1 Design Evolution

Our project has evolved quite consistently over the course of this class. While our design document was quite thorough and we were able to follow it most of the time, deviations were unavoidable as criteria changed and other events happened. We started out with our backend framework being a completely different tech stack that was swapped halfway through the project to ensure maintainability for the IT team we are passing this off to. We also had requirements change along the way in our weekly / bi-weekly meetings with our client. We had new features asked for, we had old features that were initially wanted were scrapped, and many other changes occurred. These changes were to be expected as the design and development processes are always changing and adapting in the software industry. Overall our core project design stayed quite intact with only minor changes and additions being the majority of our changes this semester.

1.2 Requirements

Functional Requirements

- CRUD therapists
- CRUD patients
- CRUD nurse
- CRUD admin
- CRUD location
- Add/Delete room
- Support multiple user types
    - Admin
    - Therapist
    - Nurse
- Ability to schedule multiple types of activities
    - Speech Therapy
    - Occupational Therapy
    - Physical Therapy
    - Conference time
    - Drive time
    - Rest time
    - Other
- Ability to schedule on multiple, synced schedules
    - View by room
    - View by therapist

- Support multiple locations
- Ability to print the schedule
- Prevent scheduling conflicts
- View metrics
    - How much time each therapist has spent doing patient care each week
    - How much time each patient has spent doing each type of therapy per day

Economic Requirements:

The budget is $0. This shouldn't be an issue since we don't have to purchase a cloud service.

Non Functional Requirements:

The application should be accessible by multiple users at once

1.3 Relevant Standards

- Test-driven development
- Branch-Review-Merge process
- ISO/IEC 12207 Software Lifecycle Process
- IEEE/ISO/IEC 29119-2-2013 - Software Testing
- IEEE/ISO/IEC 24748-5-2017 - Software Development Planning
- IEEE P2675 - Software Deployment

We wanted to use proper development standards when it came to designing and developing our project. We decided we wanted to use standards that properly defined stages of development such as development planning, Software testing, Software deployment as well as other basic standards to each step in the software lifecycle process. We used a Scrum approach to our development and used Branch-Merge-Review to add code to our project.

1.4 Engineering Constraints

- Visibility of two main schedules
- Allow for multiple types of users: Therapists, Nurses, admins
- Admins can manage patients, appointments, therapists, nurses, locations, and rooms
- Schedules can be printed
- Users can:
    - Login and logout
    - View relevant metrics
    - View schedule in a way that is intuitive to them

We had only a couple of real constraints in our project. We had the core functionality that was asked of us as shown above, but a majority of the design and development was up to our discretion which made the project more of a design and code process than just a coding assignment.

1.5 Security Concerns and Countermeasures

Security Requirements:

- Must adhere to HIPAA standards
- Patients can't view other patient information
- Nurses can't alter the schedule, they can only view it
- Sensitive information (passwords) must be hashed before storage

Our project is a web application and is software in its entirety meaning we don't have any real physical security needed. However, cybersecurity is more important for our project. Even though our project isn't customer-facing in any way, we are still holding sensitive user information that needs to be kept confidential.

1.6 Implementation Details

The technologies we chose are currently maintained and have ample documentation available. The tech stack we chose is very common, so a technologist working for the client should be able to maintain the application.

We chose to use React instead of Angular because the scheduling system is more component-based, which is a strength of React. Originally we had a technical stack with Java, Springboot, and a MySQL database, but after we met with Unity Points compliance team we realigned our stack with C#.NET and TSQL to ensure maintainability after the delivery date. We wanted a relational database since a lot of the data for our system is interconnected and it gives us more flexibility in how we create the system. We chose Javascript over Typescript because we had more experience with Javascript, even though Typescript has type checking built-in.

We followed Test Driven Development (Figure 1.1) and Agile methodologies (Figure 1.2). We used Test Driven Development to minimize bugs in the application, ensure high code coverage, and deliver high-quality code. We used an Agile methodology that is structured similarly to Kanban. We demonstrated our project to the client every two weeks (at the end of our sprints) and received feedback from the client at that time. We wanted to receive feedback from the client throughout the development process so we could adjust the project as needed instead of changing things last minute.

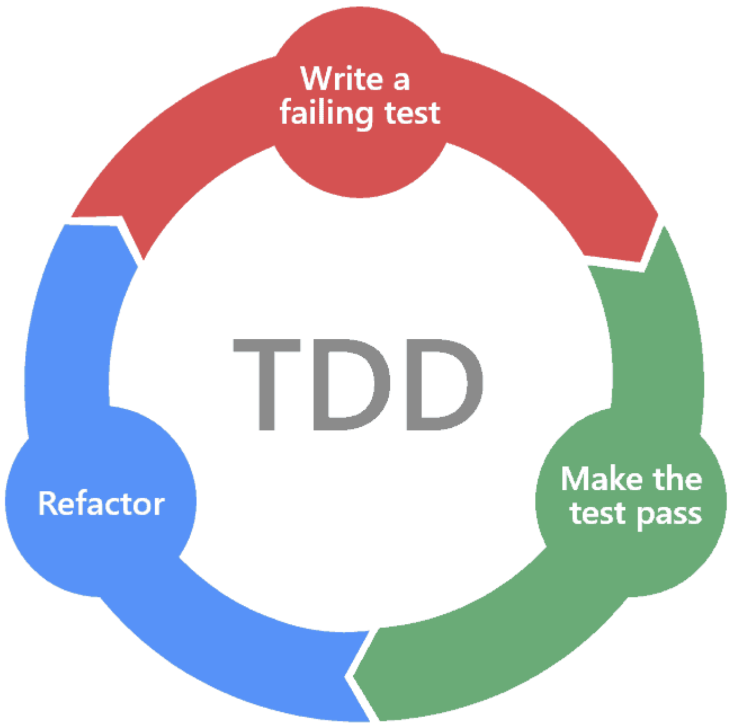


Figure 1.1 Test-Driven Development

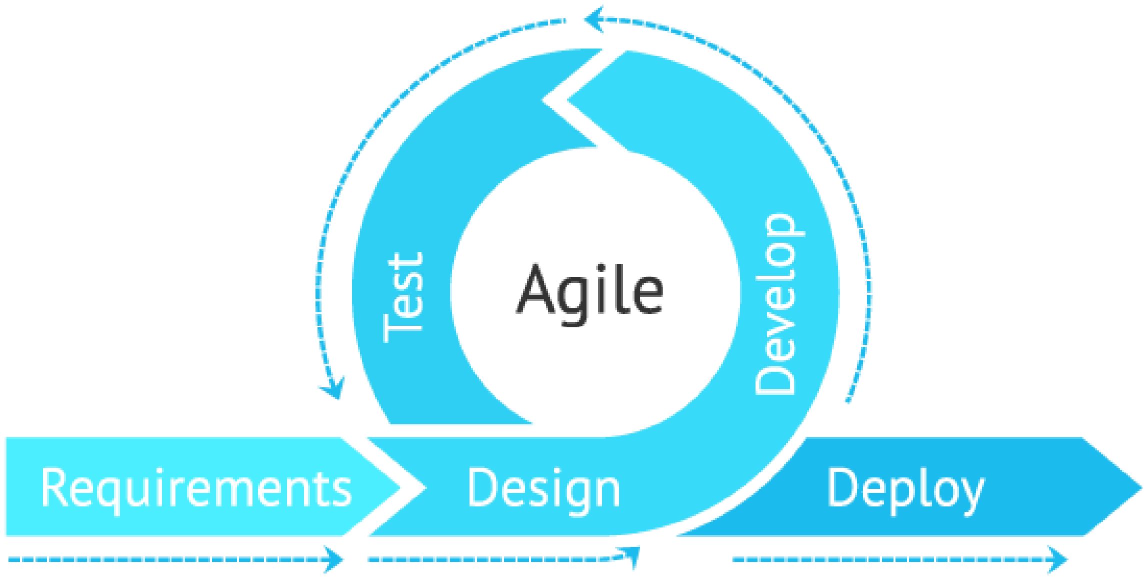


Figure 1.2 Agile Process

Below are our design diagrams. We have our system diagram which is an overview of the way our system works. Our system is really broken down into 3 major components. We have our React / Javascript framework that is the UI for our users. It is the side the client interacts with having multiple different pages for the users to access. Next, we have our MS-SQL database which stores all of our non-volatile data in a relational format. And finally, we have our ASP.NET API. The API transfers the necessary data from our database to our front end and vice versa. The API has full CRUD capabilities.

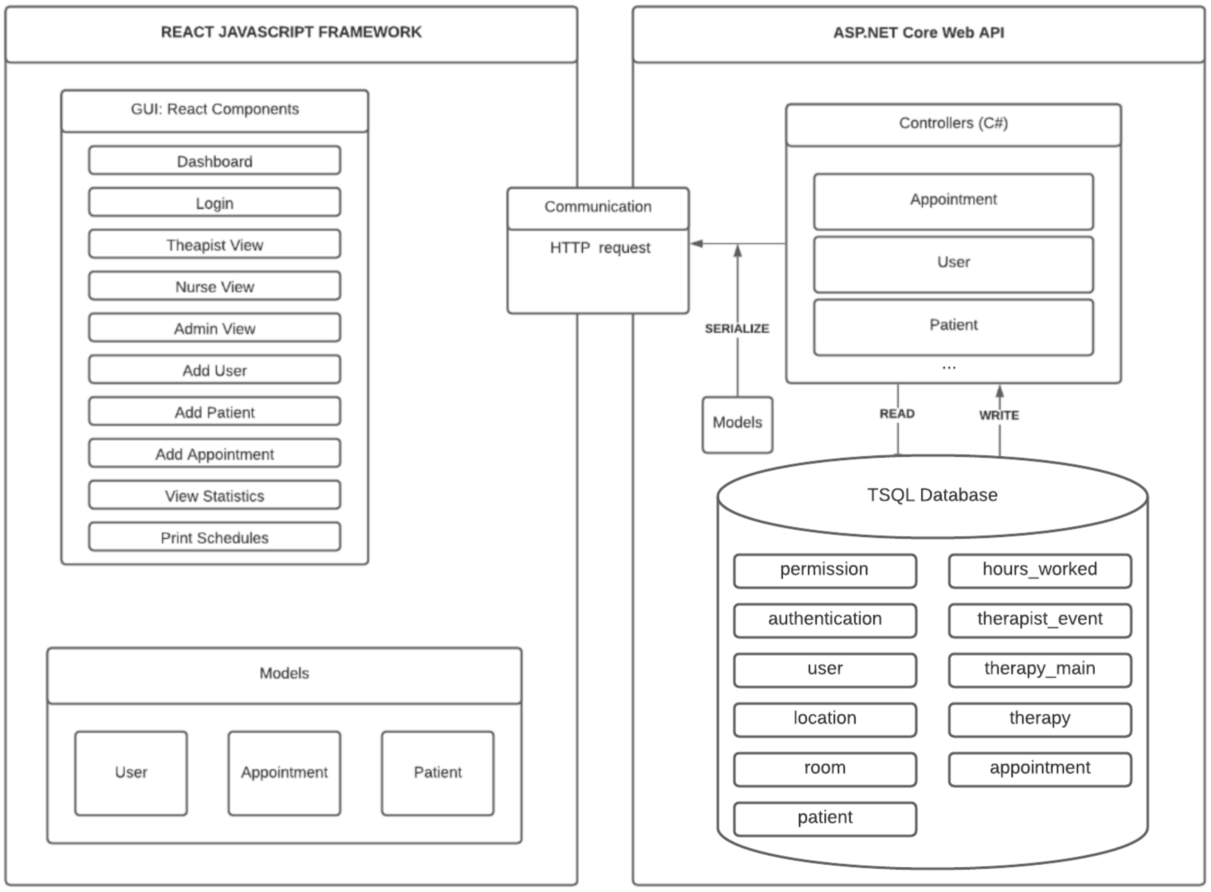


Figure 1.3 Software Architecture Overview

Next is our block diagram. Our block diagram shows the flow of a user operating our application. The user begins at the login page before it can reach the rest of the application. After a successful login, the user starts at the landing page. The landing page contains the current schedule for the week as well as other access points to the rest of the website. From the landing page, an admin can access all features of the website such as viewing metrics, creating new users, system settings, and other pages.

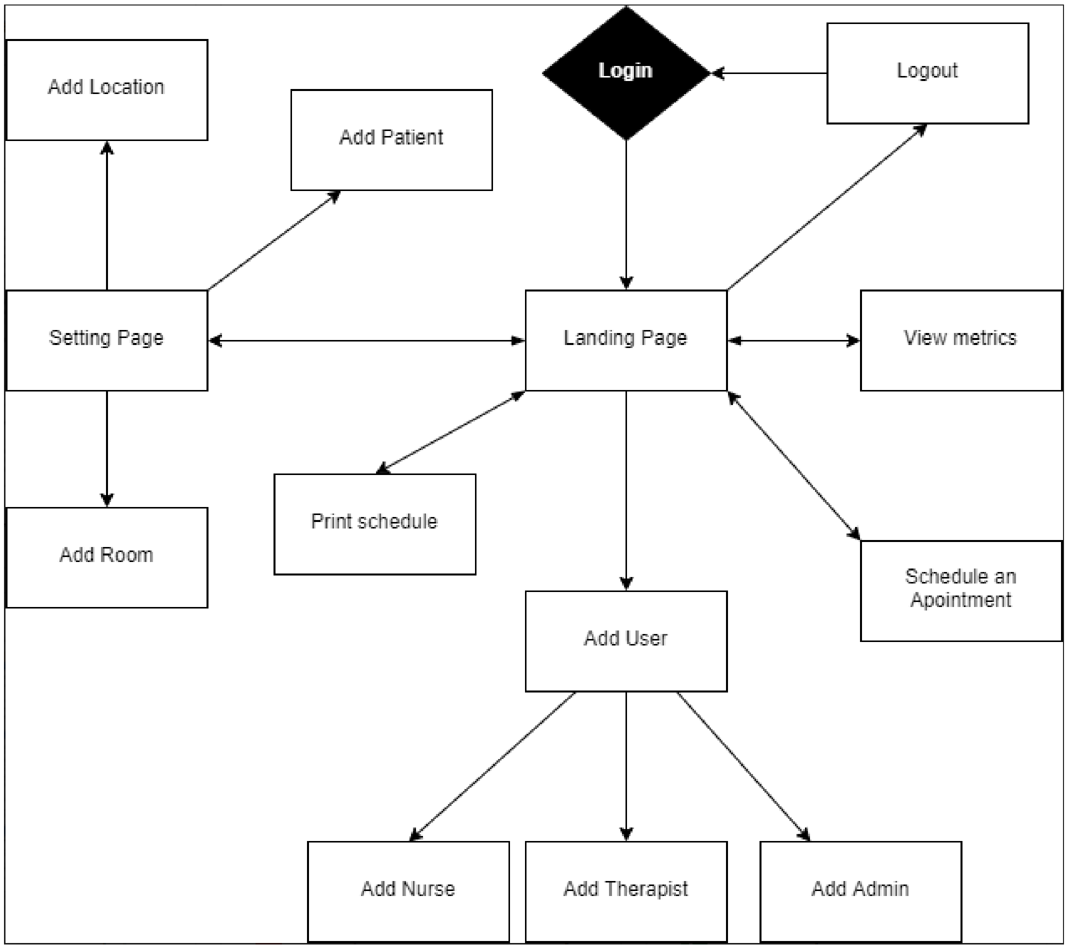


Figure 1.4 Block Diagram

Next up we have our database schema. Our database is broken down into multiple parts. We have the main tables such as patient, therapist, user, and appointment. We store all the patient, therapist, and user's basic information in their respective tables as well as all of the previous and upcoming appointments in the appointment table. We then have our extra tables such as therapy, therapy_main, permission, room, location, hours_worked, authentication which all hold smaller amounts of data that support our main tables.

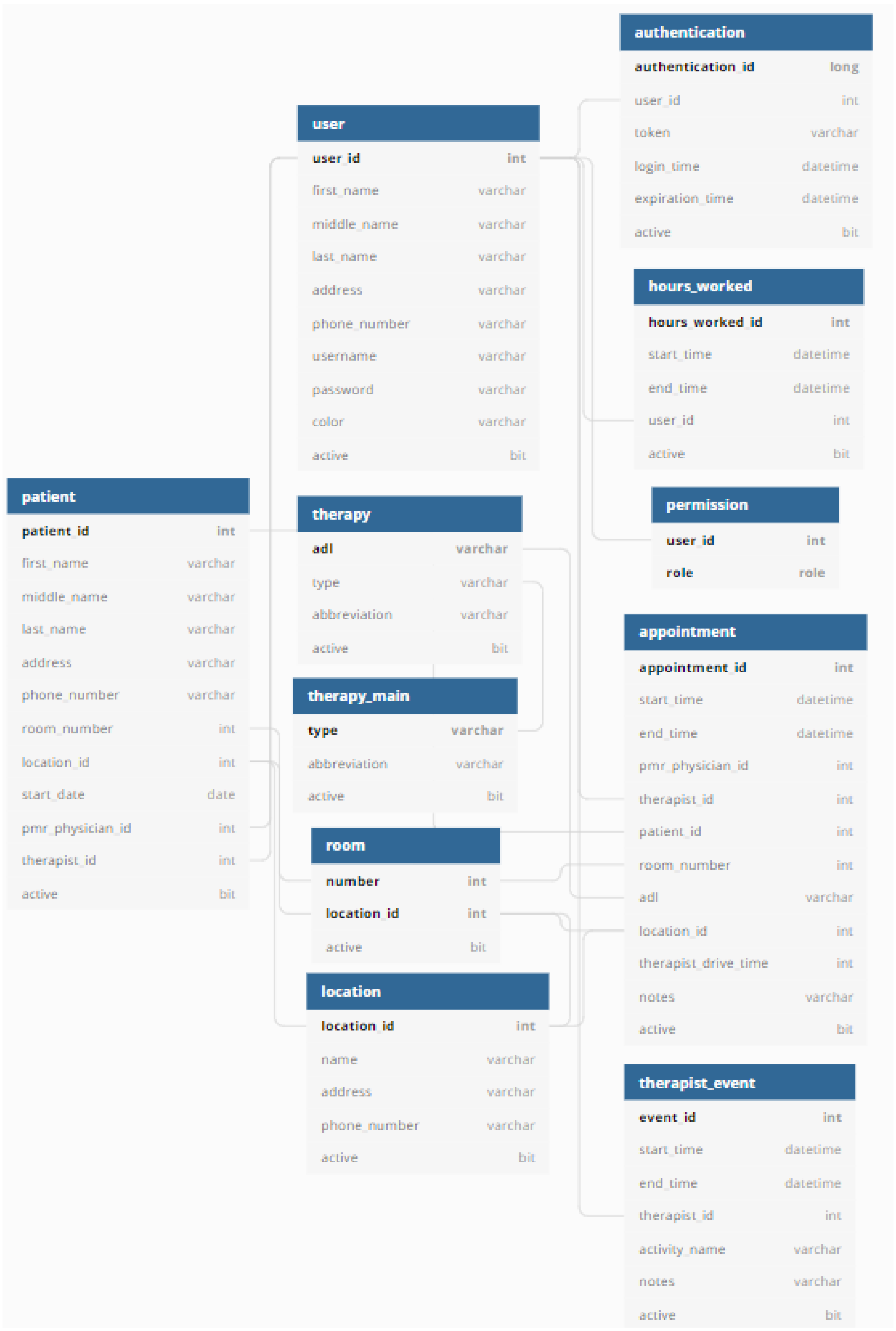


Figure 1.5 DB Schema

Next up is our Use Case Diagram. This shows all the functional capabilities of different users of our system. As you can see the admin user has full access to every feature on the application. Whereas therapists have control over creating appointments, viewing metrics, and creating patients. Finally, you have the nurse role which can only view the schedule.

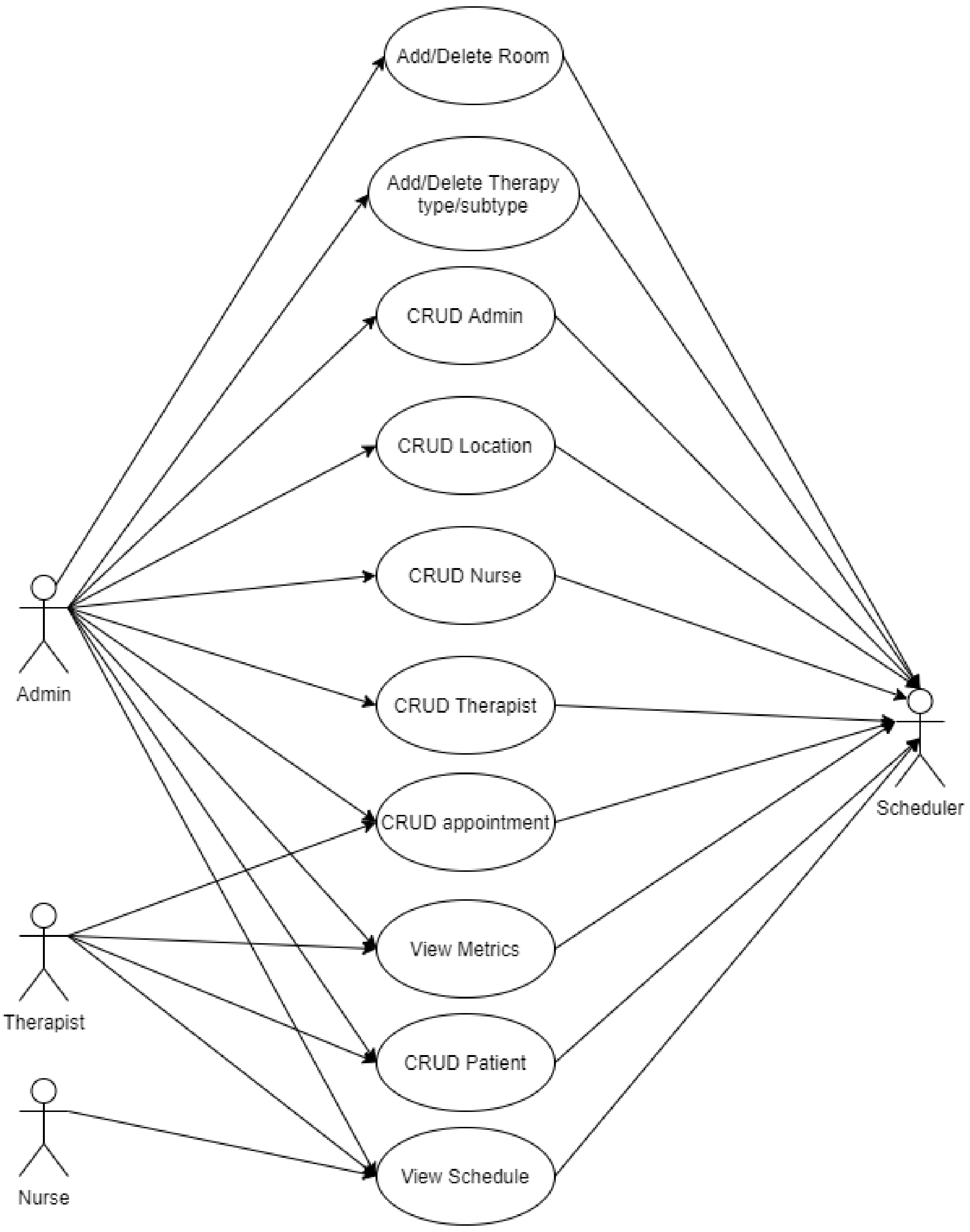


Figure 1.6 Use Case Diagram

1.7 Testing Process and Testing Results

The backend of this web application used a service-controller architecture. Controllers are used to handle requests from clients and return a result. Services are used to perform an action for a user or client, things such as creating, reading, updating, and deleting data from the database. Isolation of different responsibilities for services and controllers lends itself naturally to unit testing each service and controller. We followed a test-driven development model. Before writing any code, we would create a failing unit test, then write enough code just to pass the failing unit, refactor the code to clean it up, and finally repeat this process. Unit testing allowed us to test data validation within the service layer and acceptance/response of client requests in the controller layer. Another level of testing performed was integration testing between the service and controller layer. This ensured that the interactions between services and controllers behaved as expected. Lastly, for system-level testing, we used Postman to manually test all the endpoints and documented sample inputs and outputs. System-level testing gave us assurance that end-to-end use cases behaved as expected.

1.8 Related Products and Literature

We spent some time looking into existing scheduling applications within the healthcare industry and found Shift Admin, QGenda, and Care Cloud to be highly rated. Shift Admin has a mobile app. Our application will be viewable on mobile devices through a mobile browser. QGenda does automated scheduling. The Client doesn't want this, as they want their physicians to control their own schedules. Care Cloud has a chatbot. This feature wouldn't generate much value for The Client since their patients aren't scheduling their own appointments. Our software scheduling program while still quite similar to other related projects we still felt ours was unique enough to the client to make it better than just applying for a non-specific version.

The main difference between these products and ours is that ours will be free for The Client to use.

2 Operation Manual

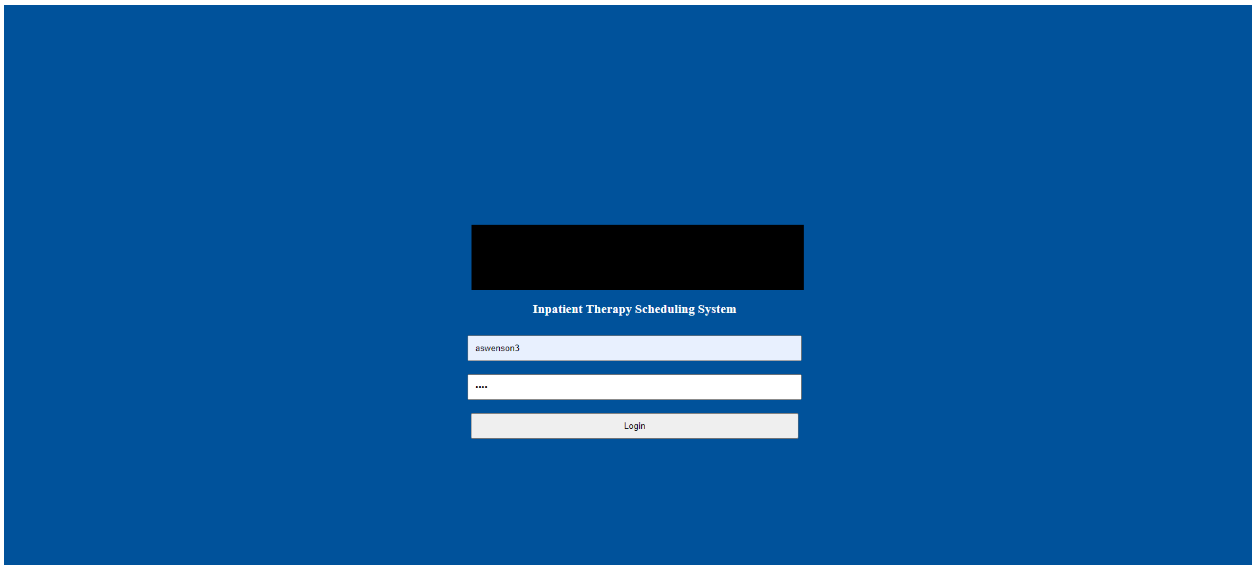
2.1 Step by Step Instructions

Setup

Download Node.js and npm: https://www.npmjs.com/get-npm
Install yarn: `npm install --global yarn`
Download the repository
In a terminal, navigate to `sdmay21-44/frontend` and run `yarn install`, then `npm start`.
The application will automatically open up in your browser.

Demo

The application will open to the schedule page. Login with an admin user.



[redacted]

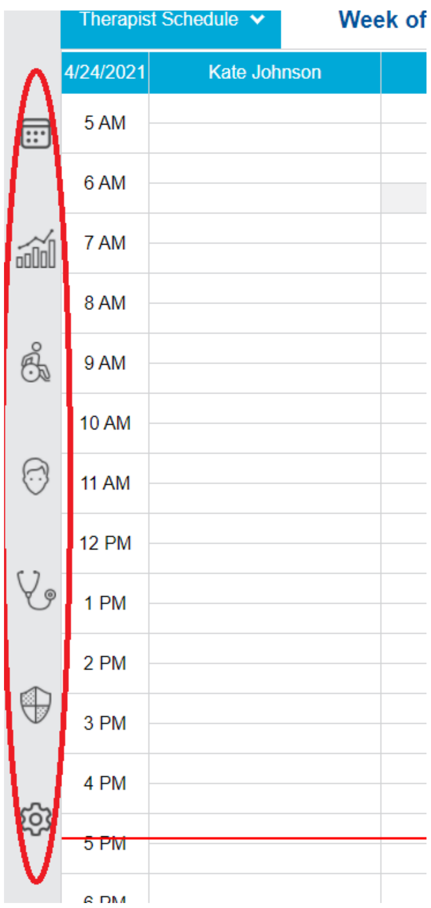
Inpatient Therapy Scheduling System

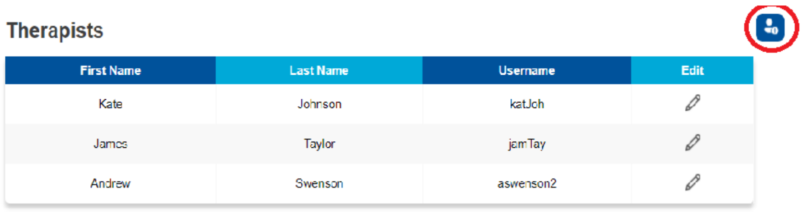
aswenson3

••••

Figure 2.1 Login Screen

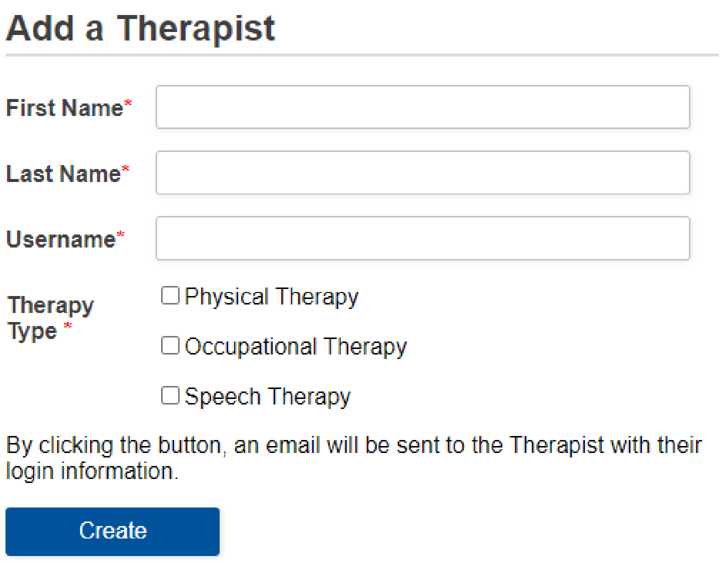
Using the left-hand navigation bar, go to the Manage Therapists page. Add a therapist or two. Repeat this process for patients.




Therapists

First Name	Last Name	Username	Edit
Kate	Johnson	katJoh	✎
James	Taylor	jamTay	✎
Andrew	Swenson	aswenson2	✎

Figure 2.3 View Therapists Screen


Add a Therapist

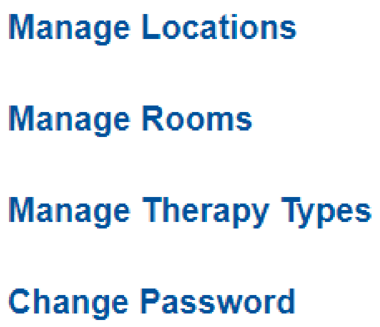
First Name*	
Last Name*	
Username*	
Therapy Type *	☐ Physical Therapy ☐ Occupational Therapy ☐ Speech Therapy

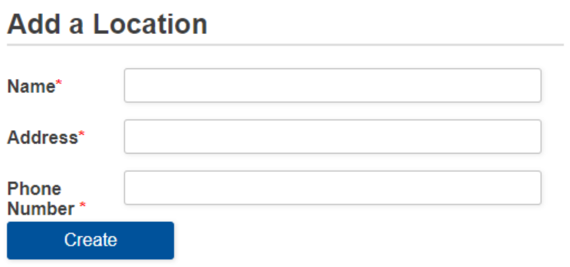
By clicking the button, an email will be sent to the Therapist with their login information.

Figure 2.2 Left Hand Navigation Bar

Figure 2.4 Add Therapist Screen

Using the left-hand navigation bar, go to the Settings page. Go to Manage Locations and add a location. Back on the settings page, go to Manage Rooms and add a room. Back on the settings page, go to Manage Therapy Types and add different types of therapies.




Add a Location

Name*	
Address*	
Phone Number *	

Figure 2.5 Settings page

Figure 2.6 Add location page

Use the Add Appointment button to create an appointment and see it appear on the calendar. You can also add an appointment by clicking on a particular calendar square and it will autofill the form with information.

Use the dropdown at the top of the page to look at different types of schedules (Room View, Therapist View, etc). Use the dropdown by the dates to change the week and use the toggle at the bottom of the schedule view to toggle between days.

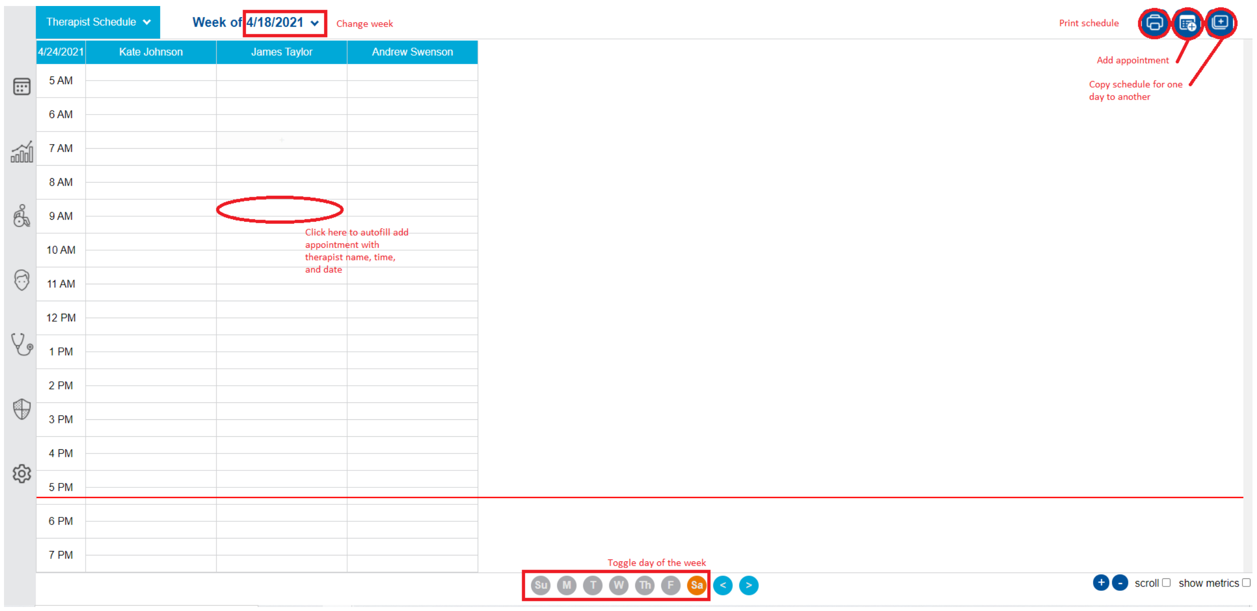
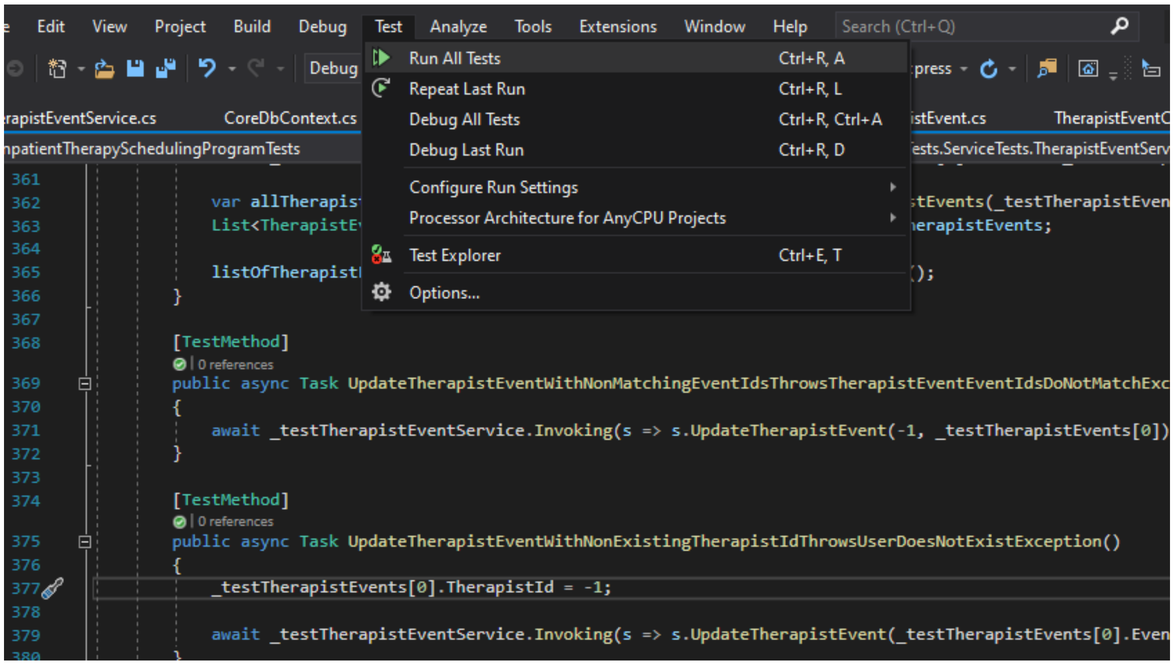


Figure 2.7 Main Dashboard/schedule view for admins

You can also add other admin users on the Manage Admins page, which you navigate to using the left-hand side navigation bar.

Test

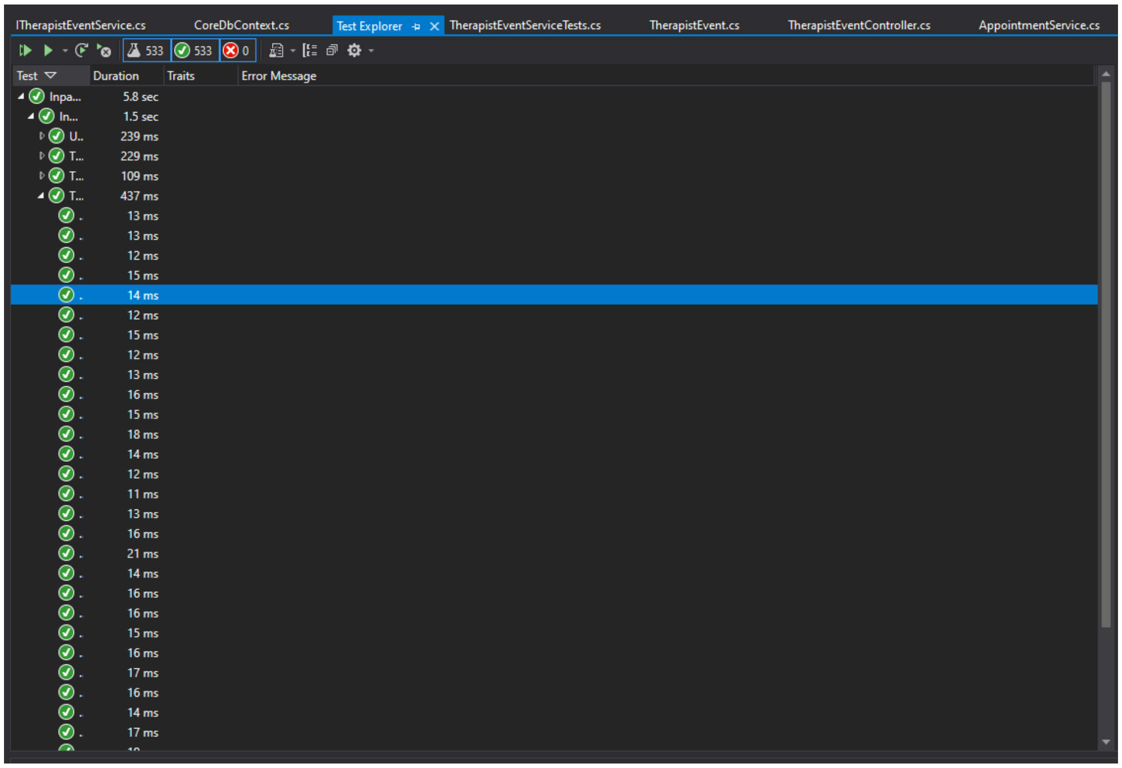
To run the backend end tests, select the Test tab in the navigation bar of VS Code (Download VS Code here: https://code.visualstudio.com/download). This should give a drop-down with an option called Run All Tests. Select Run All Tests. A pop up called Test Explorer should appear that runs all tests and shows the results of the test runs.



```
        var allTherapis                                       tEvents(_testTherapistEven
        List<TherapistE                                       herapistEvents;

        listOfTherapist                                       );
    }

    [TestMethod]
    0 references
    public async Task UpdateTherapistEventWithNonMatchingEventIdsThrowsTherapistEventEventIdsDoNotMatchExc
    {
        await _testTherapistEventService.Invoking(s => s.UpdateTherapistEvent(-1, _testTherapistEvents[0])
    }

    [TestMethod]
    0 references
    public async Task UpdateTherapistEventWithNonExistingTherapistIdThrowsUserDoesNotExistException()
    {
        _testTherapistEvents[0].TherapistId = -1;

        await _testTherapistEventService.Invoking(s => s.UpdateTherapistEvent(_testTherapistEvents[0].Even
    }
```

Figure 2.8 Test Dropdown

A pop up called Test Explorer should appear that runs all tests and shows the results of the test runs.


ITherapistEventService.cs
CoreDbContext.cs
Test Explorer
TherapistEventServiceTests.cs
TherapistEvent.cs
TherapistEventController.cs
AppointmentService.cs
533
533
0

```
Test        Duration   Traits   Error Message
Inpa...     5.8 sec
 In...      1.5 sec
  U..       239 ms
  T...      229 ms
  T...      109 ms
  T...      437 ms
   .        13 ms
   .        13 ms
   .        12 ms
   .        15 ms
   .        14 ms
   .        12 ms
   .        15 ms
   .        12 ms
   .        13 ms
   .        16 ms
   .        15 ms
   .        18 ms
   .        14 ms
   .        12 ms
   .        11 ms
   .        13 ms
   .        16 ms
   .        21 ms
   .        14 ms
   .        16 ms
   .        16 ms
   .        15 ms
   .        16 ms
   .        17 ms
   .        16 ms
   .        14 ms
   .        17 ms
```

Figure 2.9 Completed Tests

3 Alternative Designs

3.1 Initial Variants

During our design process, we considered two different options. The first was creating a web application. A web application solves almost all of the client's issues; visibility, usability, increase in features, and many other issues as well. Our second idea was to create a desktop application. This idea would also solve our user's problem but we felt it was less flexible than the web application and would be harder to implement. We decided a web application would allow for greater visibility for therapists to use on tablets and mobile devices.

3.2 Failed Variants

Originally we had decided to use a backend technical stack that used Java, Springboot, and a MySQL database. We got semi-far using this structure before meeting the Unity Points compliance team. We found out their entire tech team only used the C# framework and a java application would be quite an annoyance to upkeep. So we decided to realign our stack with C#.NET and TSQL to ensure maintainability after the delivery date.

4 Other Considerations

4.1 Relevant Additions

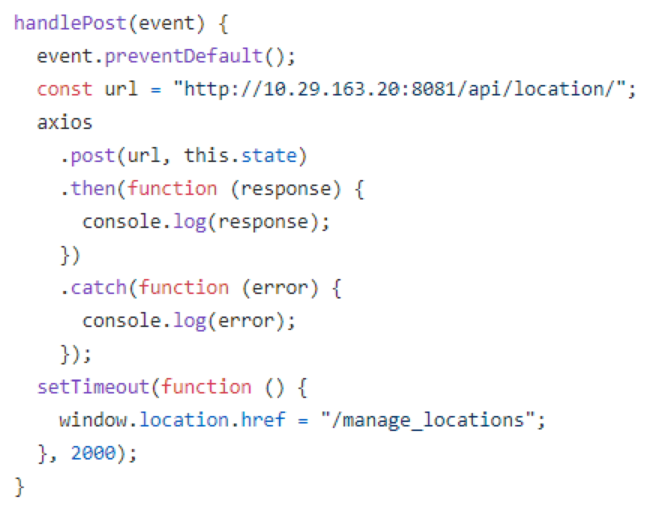
Overall this whole project was a learning experience. Interacting with a client instead of writing code for an assignment is an entirely different approach and experience. Using proper development standards and truly planning out this project in its entirety before writing code was another new experience. Using the branch-review-merge process was helpful to maintain the quality of our code, but we were also pleasantly surprised that it was useful in spreading implementation information throughout our team.

The backend team completely swapping over tech stacks in the middle of the project is another great learning opportunity, because in professional development events like that happen all the time. And the ability to adapt to changes quickly and effectively will be a great tool to have moving forward.

5 Code

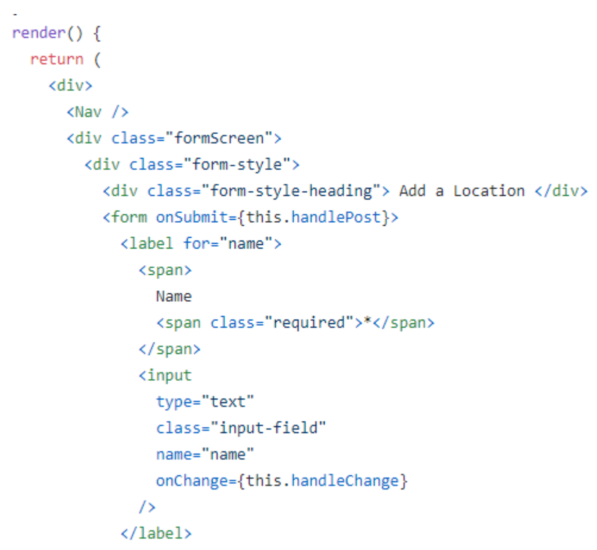
5.1 Code

The frontend uses Axios for network requests:



```
handlePost(event) {
  event.preventDefault();
  const url = "http://10.29.163.20:8081/api/location/";
  axios
    .post(url, this.state)
    .then(function (response) {
      console.log(response);
    })
    .catch(function (error) {
      console.log(error);
    });
  setTimeout(function () {
    window.location.href = "/manage_locations";
  }, 2000);
}
```

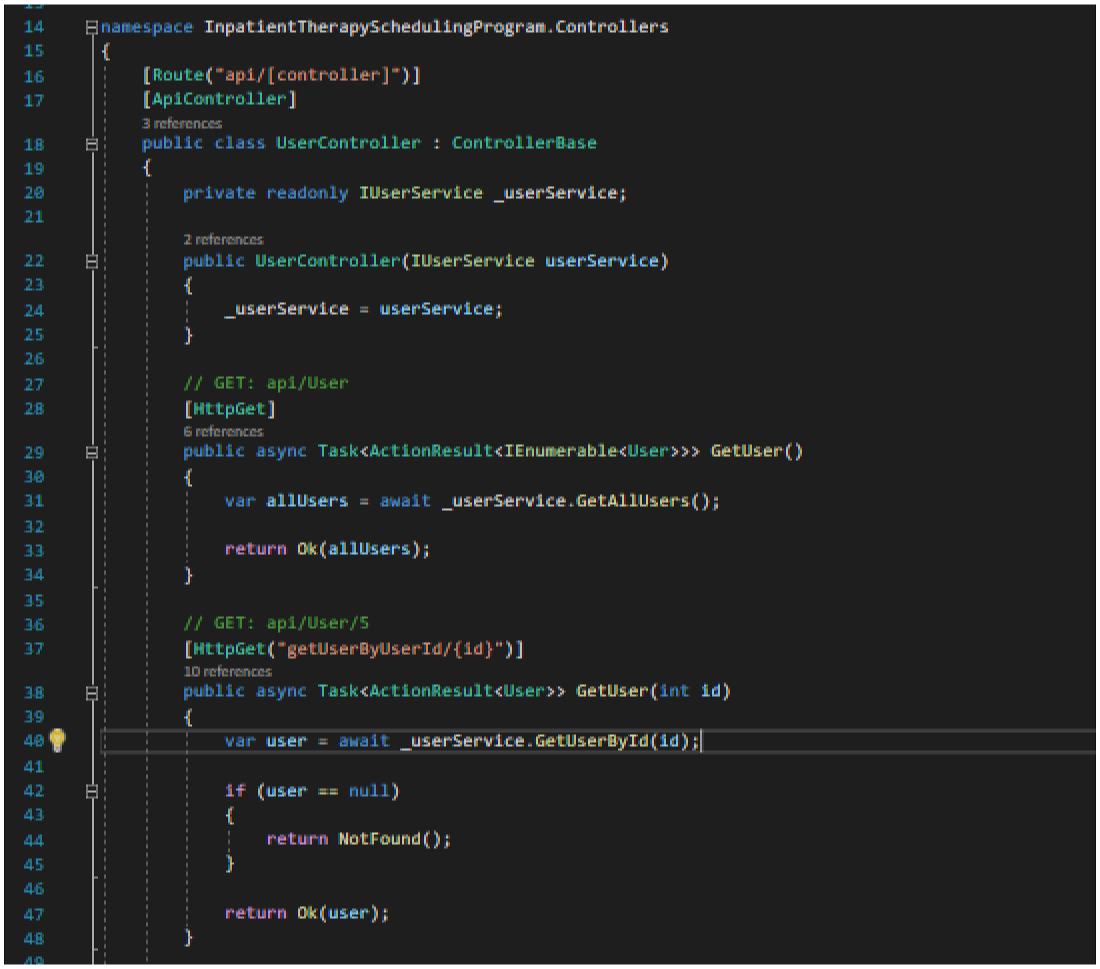
The frontend uses React to render the UI:



```
render() {
  return (
    <div>
      <Nav />
      <div class="formScreen">
        <div class="form-style">
          <div class="form-style-heading"> Add a Location </div>
          <form onSubmit={this.handlePost}>
            <label for="name">
              <span>
                Name
                <span class="required">*</span>
              </span>
              <input
                type="text"
                class="input-field"
                name="name"
                onChange={this.handleChange}
              />
            </label>
```

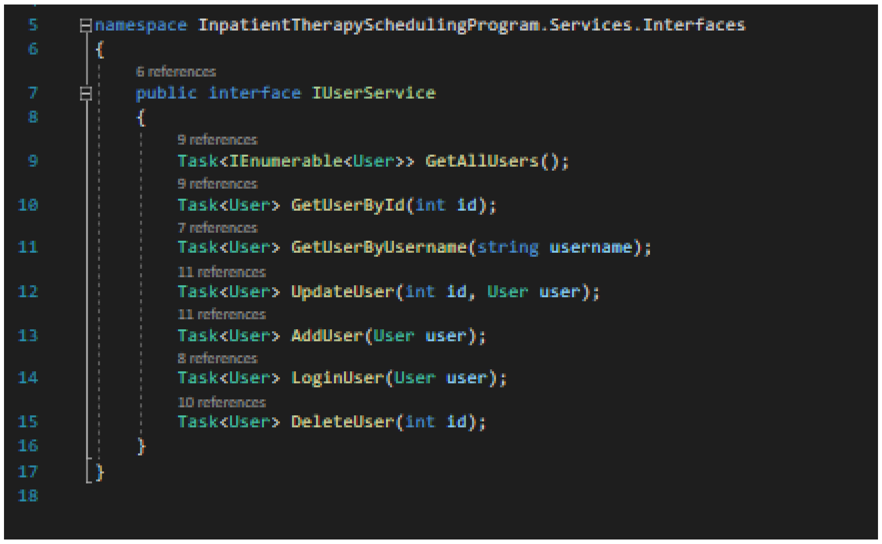
The backend is an API setup with a set of the controller, model, services, and test:

UserController Example:



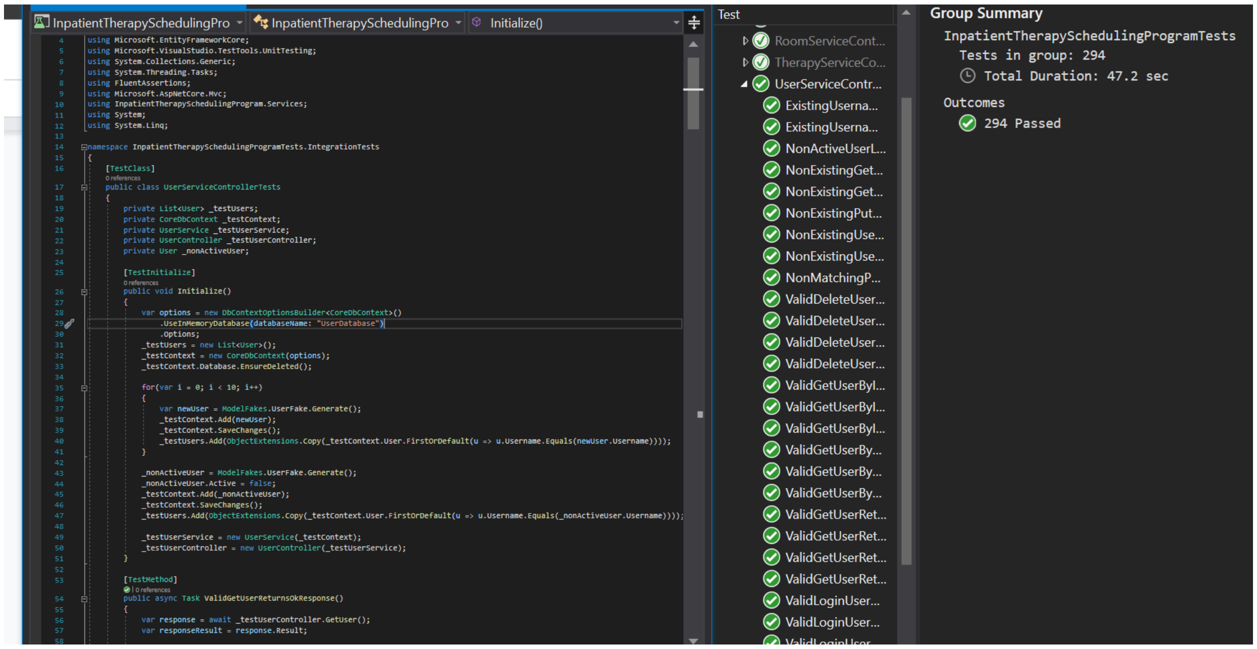
```
namespace InpatientTherapySchedulingProgram.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    3 references
    public class UserController : ControllerBase
    {
        private readonly IUserService _userService;

        2 references
        public UserController(IUserService userService)
        {
            _userService = userService;
        }

        // GET: api/User
        [HttpGet]
        6 references
        public async Task<ActionResult<IEnumerable<User>>> GetUser()
        {
            var allUsers = await _userService.GetAllUsers();

            return Ok(allUsers);
        }

        // GET: api/User/5
        [HttpGet("getUserByUserId/{id}")]
        10 references
        public async Task<ActionResult<User>> GetUser(int id)
        {
            var user = await _userService.GetUserById(id);

            if (user == null)
            {
                return NotFound();
            }

            return Ok(user);
        }
```

User Service Interface Example:



```
namespace InpatientTherapySchedulingProgram.Services.Interfaces
{
    6 references
    public interface IUserService
    {
        9 references
        Task<IEnumerable<User>> GetAllUsers();
        9 references
        Task<User> GetUserById(int id);
        7 references
        Task<User> GetUserByUsername(string username);
        11 references
        Task<User> UpdateUser(int id, User user);
        11 references
        Task<User> AddUser(User user);
        8 references
        Task<User> LoginUser(User user);
        10 references
        Task<User> DeleteUser(int id);
    }
}
```

User Test Example:



```
using Microsoft.EntityFrameworkCore;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using System.Collections.Generic;
using System.Threading.Tasks;
using FluentAssertions;
using Microsoft.AspNetCore.Mvc;
using InpatientTherapySchedulingProgram.Services;
using System;
using System.Linq;

namespace InpatientTherapySchedulingProgramTests.IntegrationTests
{
    [TestClass]
    0 references
    public class UserServiceControllerTests
    {
        private List<User> _testUsers;
        private CoreDbContext _testContext;
        private UserService _testUserService;
        private UserController _testUserController;
        private User _nonActiveUser;

        [TestInitialize]
        0 references
        public void Initialize()
        {
            var options = new DbContextOptionsBuilder<CoreDbContext>()
                .UseInMemoryDatabase(databaseName: "UserDatabase")
                .Options;
            _testUsers = new List<User>();
            _testContext = new CoreDbContext(options);
            _testContext.Database.EnsureDeleted();

            for(var i = 0; i < 10; i++)
            {
                var newUser = ModelFakes.UserFake.Generate();
                _testContext.Add(newUser);
                _testContext.SaveChanges();
                _testUsers.Add(ObjectExtensions.Copy(_testContext.User.FirstOrDefault(u => u.Username.Equals(newUser.Username))));
            }

            _nonActiveUser = ModelFakes.UserFake.Generate();
            _nonActiveUser.Active = false;
            _testContext.Add(_nonActiveUser);
            _testContext.SaveChanges();
            _testUsers.Add(ObjectExtensions.Copy(_testContext.User.FirstOrDefault(u => u.Username.Equals(_nonActiveUser.Username))));

            _testUserService = new UserService(_testContext);
            _testUserController = new UserController(_testUserService);
        }

        [TestMethod]
        ✓ | 0 references
        public async Task ValidGetUserReturnsOkResponse()
        {
            var response = await _testUserController.GetUser();
            var responseResult = response.Result;
```

```
▷ ✓ RoomServiceCont...
▷ ✓ TherapyServiceCo...
◢ ✓ UserServiceContr...
    ✓ ExistingUserna...
    ✓ ExistingUserna...
    ✓ NonActiveUserL...
    ✓ NonExistingGet...
    ✓ NonExistingGet...
    ✓ NonExistingPut...
    ✓ NonExistingUse...
    ✓ NonExistingUse...
    ✓ NonMatchingP...
    ✓ ValidDeleteUser...
    ✓ ValidDeleteUser...
    ✓ ValidDeleteUser...
    ✓ ValidDeleteUser...
    ✓ ValidGetUserByI...
    ✓ ValidGetUserByI...
    ✓ ValidGetUserByI...
    ✓ ValidGetUserBy...
    ✓ ValidGetUserBy...
    ✓ ValidGetUserBy...
    ✓ ValidGetUserRet...
    ✓ ValidGetUserRet...
    ✓ ValidGetUserRet...
    ✓ ValidGetUserRet...
    ✓ ValidLoginUser...
    ✓ ValidLoginUser...
```

Group Summary

```
InpatientTherapySchedulingProgramTests
  Tests in group: 294
  Total Duration: 47.2 sec

Outcomes
  ✓ 294 Passed
```