**PCYS Scheduler**

*Team Null*

Bart Downey

Karl Lukan

Michael Voorhies

Noah Woyak

Table of Contents

[**Requirements** 3](#_Toc134112823)

[Overview 3](#_Toc134112824)

[**Functionality** 3](#_Toc134112825)

[**Team** 3](#_Toc134112826)

[Rules 3](#_Toc134112827)

[Code Guidelines 3](#_Toc134112828)

[Administration 3](#_Toc134112829)

[Configuration 3](#_Toc134112830)

[Logging 3](#_Toc134112831)

[Continuous Integrationa & Deployment 3](#_Toc134112832)

[**Technologies** 3](#_Toc134112833)

[**Architecture & Project Structure** 3](#_Toc134112834)

[Scheduler 3](#_Toc134112835)

[Scheduler.Tests 3](#_Toc134112836)

[Scheduler.Benchmarks 3](#_Toc134112837)

[**Entity Relationship Diagram** 3](#_Toc134112838)

[**Scheduler** 3](#_Toc134112839)

[**Identity** 3](#_Toc134112840)

[**Wireframes** 3](#_Toc134112841)

[Home Page 3](#_Toc134112842)

[Login Page 3](#_Toc134112843)

[Coach Dashboard 3](#_Toc134112844)

[Admin Dashboard 3](#_Toc134112845)

[Admin Schedule Field 3](#_Toc134112846)

[Team Detail 3](#_Toc134112847)

# **Requirements**

## Overview

Graphical user interface

Description automatically generatedPCYS Scheduler is an application for Portage County Youth Soccer that will allow coaches to schedule events such as games and practices. These events will take place on fields in the facility, fields are of different sizes and can be partitioned for multiple events. Attached is an overview of the facility.

The application has three types of users: public, coach and admin.  
Public users are able to see the event schedule as well as what teams will be participating.  
Coaches can schedule events on fields and can create teams in the application.  
Admins are able to manage all users, as well as modify fields. They can also schedule facility blackout times.

## **Functionality**

Public users should be able to select a team and look at a calendar of events for the current week. They should also be able to see all events in a list view.

Coaches should be able to schedule events for individual fields, these events should be denoted as a game or practice. The time selector for events should be constrained to 30-minute increments and allow for recurrence. Events should not be scheduled on top of each other on a field. Cancelling events can only be done by the coach who scheduled it.

Admins can schedule events for the whole facility, cancel/change all events, and can mark individual fields as well as the whole facility as closed. They can also manage coaches in the application and associate them with teams.

# **Team**

## Rules

* Meet in person at least every other week.
* Daily standups before 12:00pm.
  + Mandatory Tuesday-Friday, optional otherwise.
  + Done via messages in the ‘Daily Standup’ channel.
  + Format: what I did yesterday, what I’ll be doing today, potential roadblocks.
* One meeting a week.
  + Sundays at 5:00pm.

## Code Guidelines

* Code is self-documenting.
* Document public members.
* Use of .editorconfig to enforce formatting.
* Tabs instead of spaces.

# Administration

## Configuration

Parts of thr application can be configured through appsettings.json and appsettings.Development.json, the former being the configuration used in production. Currently, logging, timezone and schedule culling can be manipulated through those files.

## Logging

The application currently logs all data-access and HTTP requests. These logs can be found in src/Scheduler/bin/(Debug or Release)/net7.0/logs.txt. As alluded to above, logging can be configured, particularly when the log file is culled.

## Continuous Integrationa & Deployment

The application is configured to run GitHub actions on push to the ‘dev’ and ‘main’ branches. The former executes a continuous integration workflow which checks if the application builds and passes all tests. The latter builds the application, applys migrations to the live database, then deploys it to MyWindowsHosting. For both workflows, an email will be sent out to the repository owner if any step fails.

# **Technologies**

* ASP.NET Core MVC
* ASP.NET Identity
* Entity Framework Core
* SQL Server 2022
* xUnit
* Github
* Select2
* Moq

ASP.NET Core MVC was chosen due to the team’s familiarity with the framework, this then influenced our other choices, namely ASP.NET Identity. The application requires authentication and authorization, Identity provides this without us having to create our own.

Since we use ASP.NET Identity, it was natural for us to choose Entity Framework Core as it integrates well with it. This also allows us to develop our database rapidly as it’s an ORM.  
We decided to use SQL Server as our database since it integrated with Visual Studio and the other Microsoft technologies that we’re using.

For unit testing we’ve gone with xUnit. Additionally, Moq is used to mock framework specific APIs.

Git is used for source control while GitHub is used to host our repository so it’s visible publicly.

# **Architecture & Project Structure**

The main application can be found in src/.

Documentation is found in docs/. You’ve probably figured this out!

The continuous integration and deployment pipelines can be found in .github/workflows/.

## Scheduler

The project is divided into four distinct layers in order to facilitate a proper flow of dependencies; these layers are (in order with the flow of dependencies): domain, application, infrastructure, and oresentation.

The domain layer contains business rules, logic, entities, and astractions that pertain to them. This layer should have no dependencies. All code pertaining to this layer is found in src/Scheduler/Domain.

The application layer acts as an interface between the consumer and the domain layer. For this project, that means it contains logging, email sending capabilities, and other services that represent a business capability. Code for thislayer is largely found in src/Scheduler/Application, and src/Scheduler/Services.

The infrastructure layer encapsulates all data-access, this could be to a database or external APIs. Code for this layer is found in src/Scheduler/Infrastructure.

The presentation layer is what the user will see, this is currently an ASP.NET Core MVC application. Code for this, as well as all other layers, is found in the Scheduler project.

## Scheduler.Tests

The application has dozens of unit tests, these are used to ensure new features work and continue to work throughout the project lifecycle. These tests can be run using the dotnet CLI, Visual Studio, or as a part of the application’s continuous integration pipeline.

## Scheduler.Benchmarks

This project is intended to provide benchmarking for performance-critical aspects of the application. Currently this only pertains to collision detection.

# **Entity Relationship Diagram**

A screenshot of a computer

Description automatically generated with low confidence  
*Additional tables are needed in order to support ASP.NET Identity.*

## **Scheduler**

**Leagues**Represents a soccer league. A league can have multiple teams inside it.

**Teams**Represents a soccer team which is apart of a league. A team will also have a coach assigned to it, UserId referencing AspNetUsers. If the user’s identifier is null, the team will be represented in the application as having a visiting coach.

**Events**Represents a scheduled event. The Games and Practices table extends it by adding additional columns, then referencing the parent Event. If the need arises, more tables can be created to represent different kinds of events. In our solution, the Event class serves as the parent to practice and game, this relationship is then represented by table-per-type mapping. An event can optionally have a recurrence pattern, represented by RecurrenceId.

**Recurrences**Represents a recurrence pattern for an event. The numeric value for the type is mapped as follows:  
0 -> Daily  
1 -> Weekly  
2 -> Monthly

**Games**A game between two teams. Game subclasses Event in the solution.

**Practices**A practice game for a team. Practice subclasses Event in the solution.

**Fields**Represents a field in the facility. A field can have many events occurring on it.

## **Identity**

**AspNetUsers**Represents a user in the application. A user can be the coach of multiple teams.

**AspNetRoles**Used to differentiate the different user groups such as coaches and admins.

**AspNetUserRoles**Represents a many-many relationship between AspNetUsers and AspNetRoles.

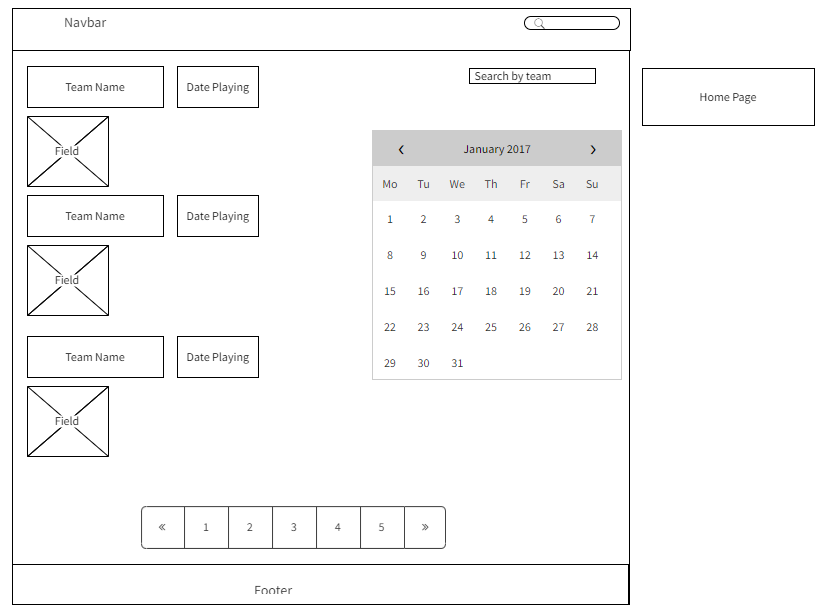
**AspNetUserClaims**Represents a claim that a user presents.

**AspNetUserLogins**Associates a user with a login.

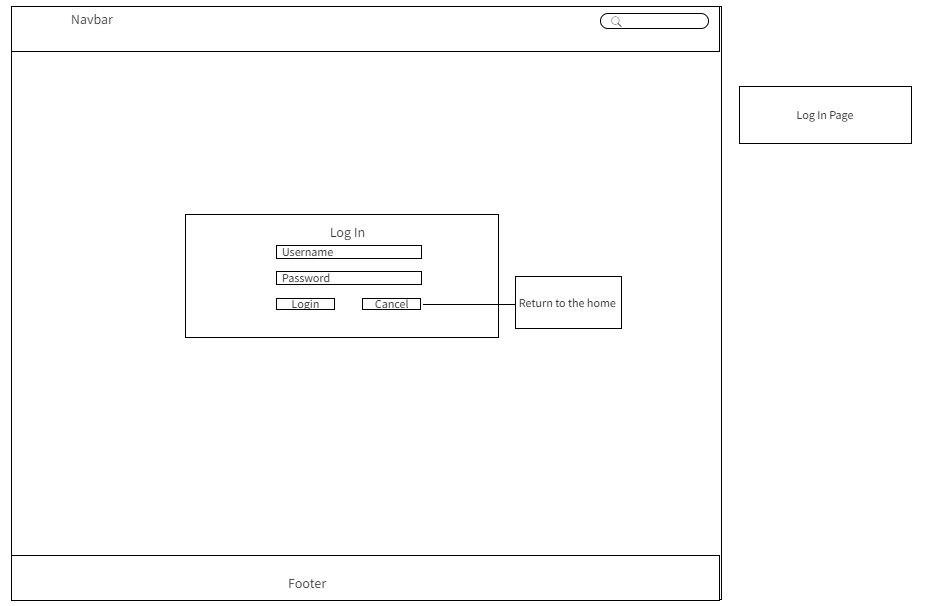
**AspNetUserTokens**Represents an authentication token for a user.

# **Wireframes**

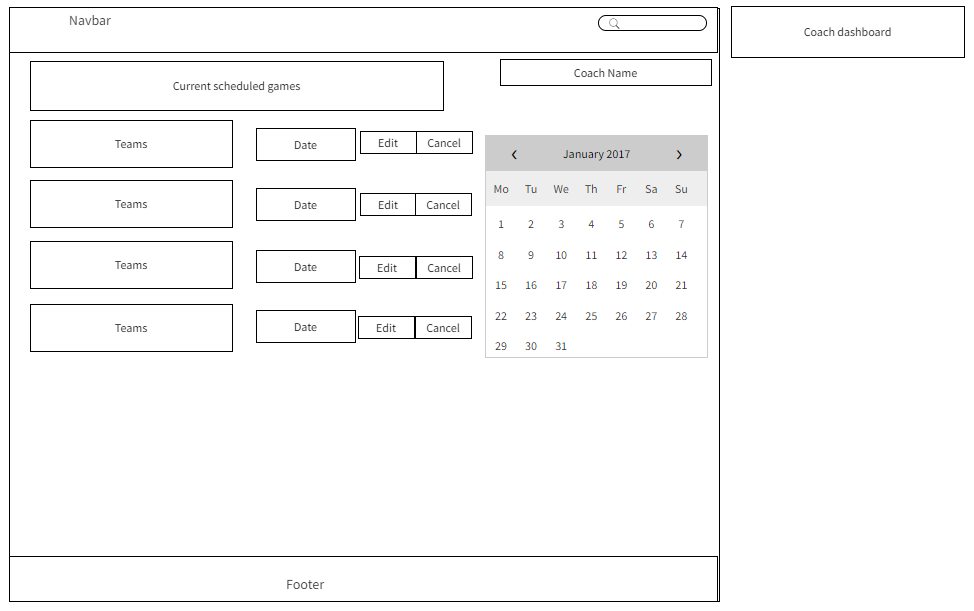
## Home Page



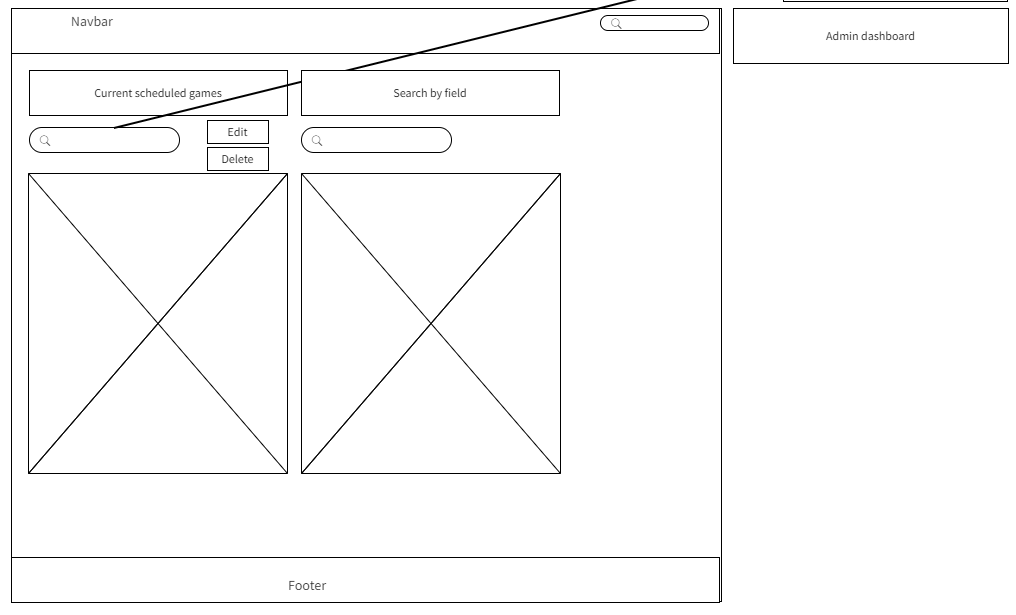
## Login Page



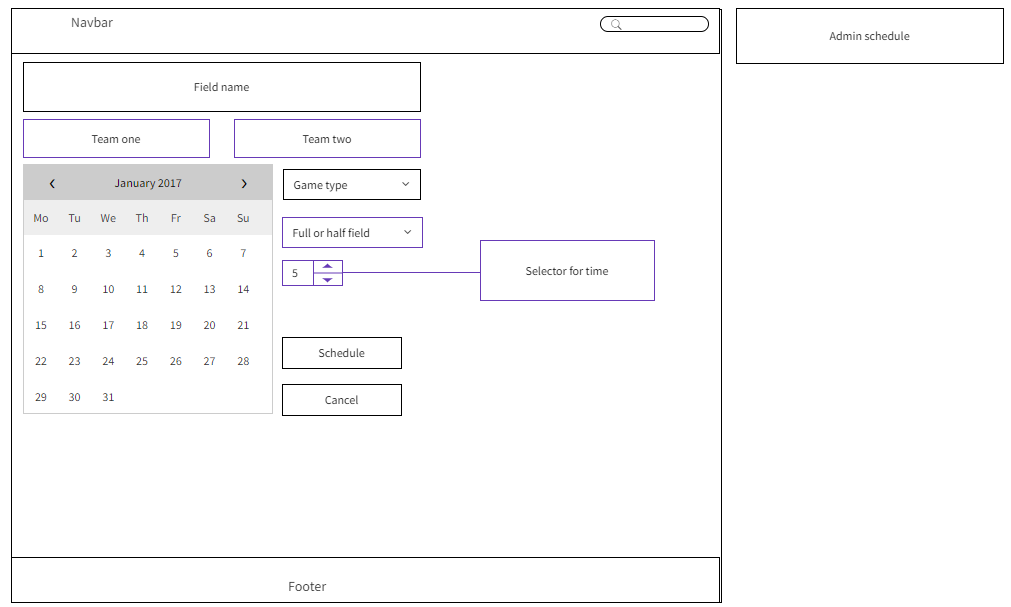
## Coach Dashboard



## Admin Dashboard



## Admin Schedule Field



## Team Detail

