**PCYS Scheduler**

*Team Null*

Bart Downey

Karl Lukan

Michael Voorhies

Noah Woyak

Table of Contents

[**Requirements** 3](#_Toc128673063)

[**Functionality** 3](#_Toc128673064)

[**Team Rules** 3](#_Toc128673065)

[**Code Guidelines** 4](#_Toc128673066)

[**Technologies** 4](#_Toc128673067)

[**Architecture** 4](#_Toc128673068)

[**Entity Relationship Diagram** 5](#_Toc128673069)

[**Scheduler** 5](#_Toc128673070)

[**Identity** 5](#_Toc128673071)

[**Wireframes** 5](#_Toc128673072)

# **Requirements**

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 eachother 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 is: what I did yesterday, what I’ll be doing today, potential roadblocks.
* Two meetings a week
  + Wednesdays and Sundays at 5:00pm.

# **Code Guidelines**

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

# **Technologies**

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

ASP.NET Core MVC was chosen due to the team’s familiarity with the framework, this then influenced our other choices for technologies, 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 due to it’s support for integration tests.

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

# **Architecture**

To ensure an optimal flow of dependencies, we’ve divided the solution into three distinct layers/projects.

Core contains entities, business logic and abstractions. This project has no dependencies, and every layer of the application depends on it.  
Infrastructure defines data access and contains databases and service implementations.  
Finally, the Web project serves as the application’s presentation layer.

# **Entity Relationship Diagram**

Diagram, schematic

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

## **Scheduler**

**Leagues**

**Teams**Represents a soccer team.

**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 object serves as the parent to Practice and Game, this relationship is then represented by table-per-type mapping.

**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.

**EventField**Represents the many-many relationship between Events and Fields.

## **Identity**

**AspNetUsers**Represents a user in the application.

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

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

**AspNetUserClaims**

**AspNetUserLogins**

**AspNetUserTokens**

# **Wireframes**