SPORTS EVENTS SYSTEM

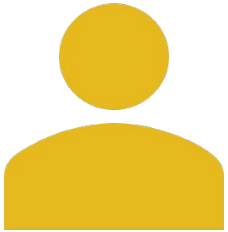
MERZ

TEAM 1

Sureshkumar Reshma [Backend]

Project Abstract

The **Sport Event System (SES)** is an application which allows users to manage various sporting events.



Users can:

Login and get authenticated.
Store player details and manage them.
Create teams and add players to the team
Display a report of the match schedule
Display the field details where the matches are conducted.
Administrators should be able to book tickets on behalf of users.

Project Goals

- Prove the application of concepts learnt and practice by applying to a project as and individual and a team.
- Understand requirements, visualize them using UML diagrams, develop the requirements and verify them by testing.
- Apply various libraries, frameworks and tools.
- Create a simple working prototype, and follow an iterative fashion to make the prototype better and apply best practices.
- Implement a loosely-coupled, highly reusable code and integrate all the modules developed from day 1 of the training.
- Develop the project in parallel with the sessions.
- Explore the docs and observe different patterns in code.
- Integrate the code written by the members of the team.

Agile Methodology

No need in long documentation and detailed specification

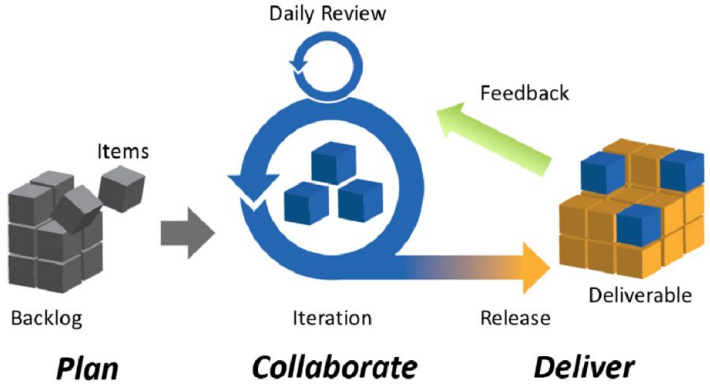
Easy adjustment to new requirements

Process visibility and high interaction between a customer and a team

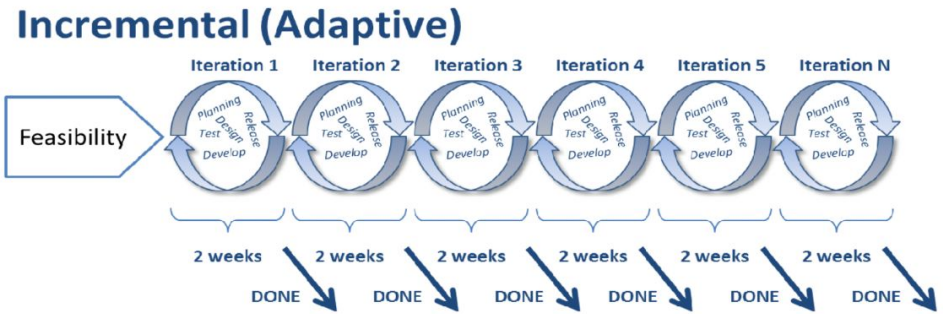
Knowledge sharing for making better decisions

Reduced project development time frames

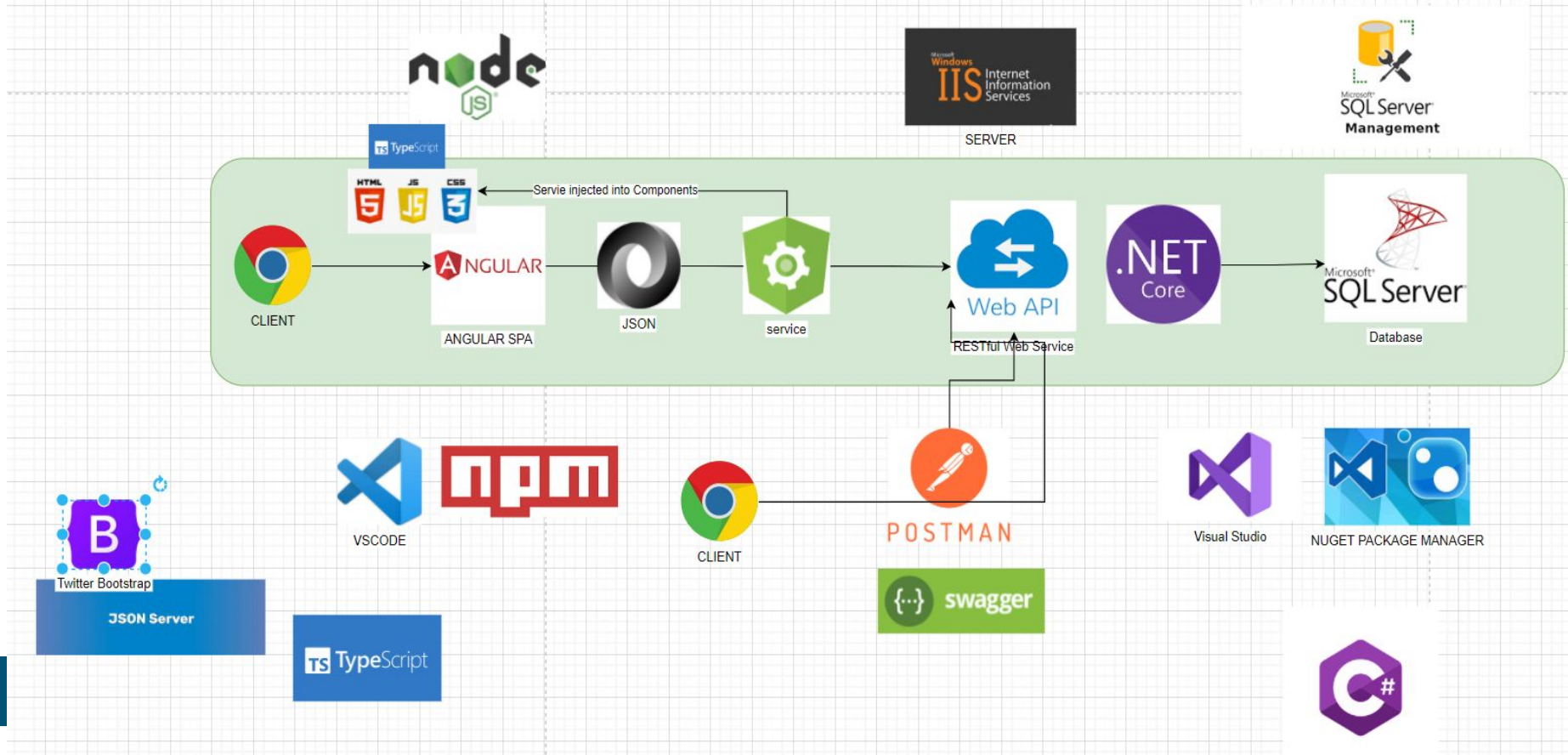
Business risks are minimized



Build The Right Product
Build It In The Right Way



Project Architecture Overview



Project Requirements

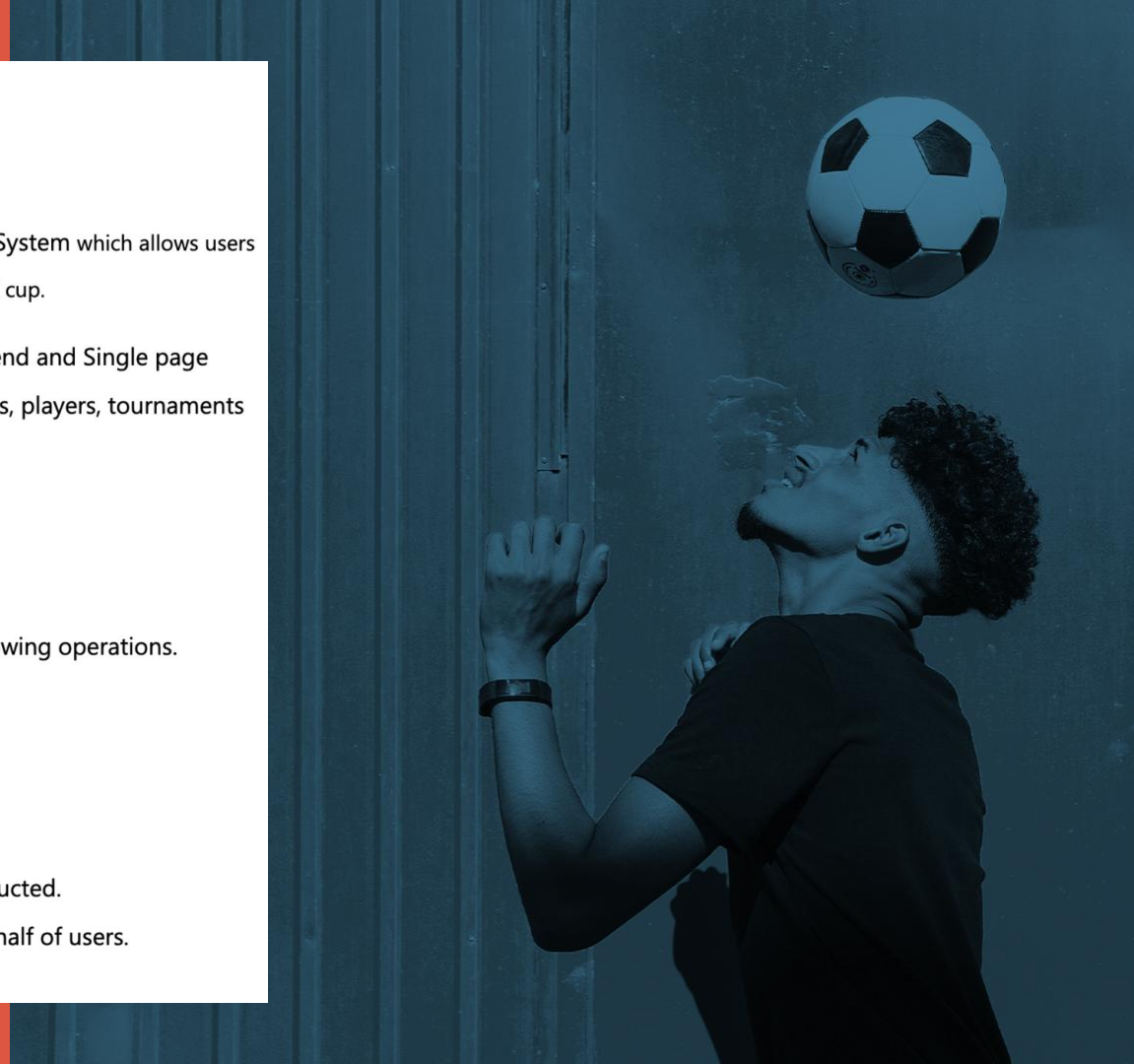
You are required to write a software solution for a Sporting Events System which allows users to manage various sporting events such as the Olympics or Football world cup.

The application has to be implemented using Web API as the backend and Single page application for the front end. This application stores details of teams, players, tournaments and matches, fields and tickets.

This software needs to have the following **features:**

The administrators who handle the system should perform the following operations.

A. Login and get authenticated.
B. Store player details and manage them.
C. Create teams and add players to the team.
D. Display a report of the match schedule
E. Display the field details where the matches are conducted.
F. Administrators should be able to book tickets on behalf of users.



Frameworks


.NET
Core
RESTful
API


Entity
Framework


MySQL


Microsoft
SQL Server

Databases

Front-End {Tools}

JSON Server

Design Thinking

Design Thinking has enabled us to integrate need of people, possibilities of technology and requirements for business success.

My team and I had planned extensively through brainstorming and ideation to think of good designs for our web-design and structure of the database.

My team and I were able to provide an innovative, viable and feasible solution to real world problems and find simplicity in the problems.

We were also able to address actual requirements of the end-users and able to improve end-users' quality of experience.

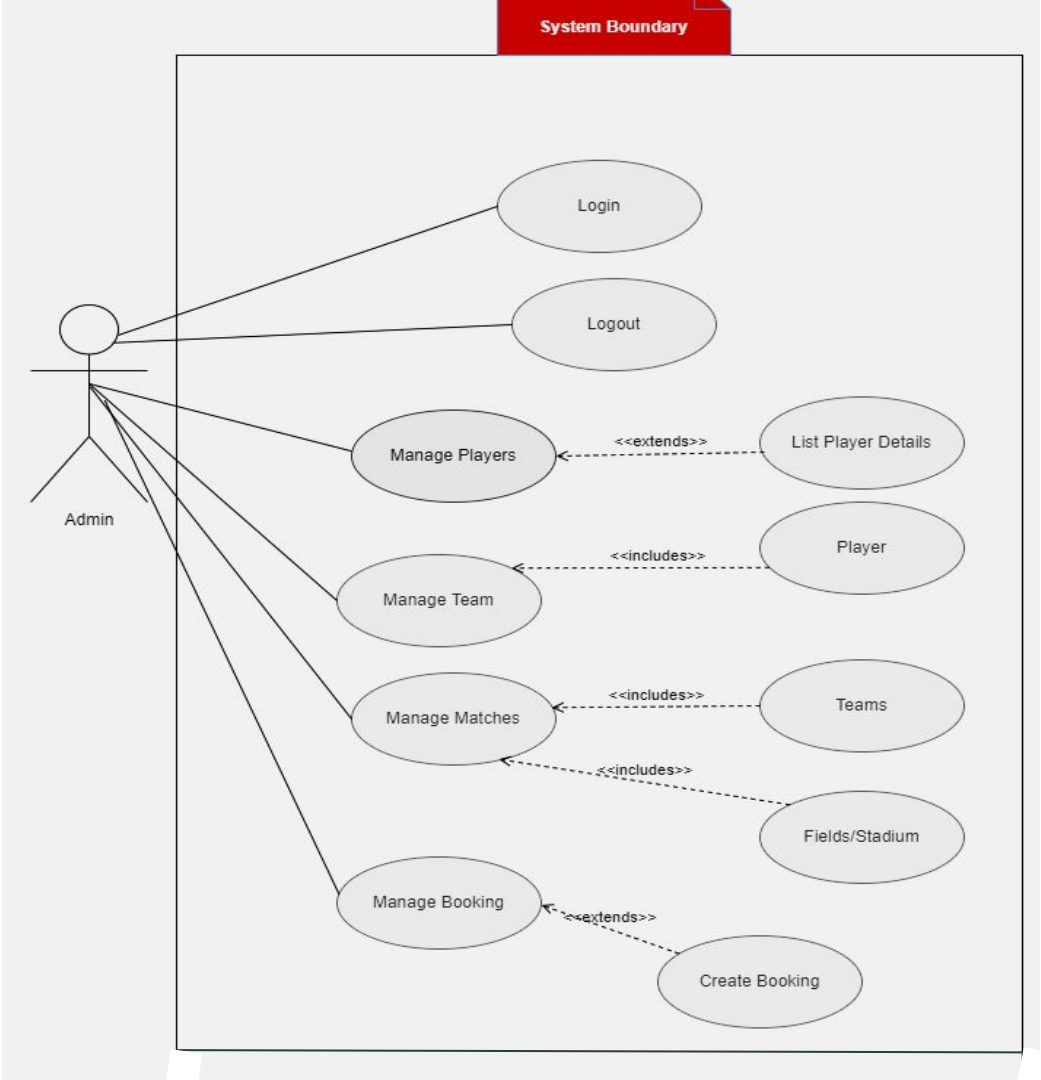
We are also able to make software designs that are more understandable, flexible, and maintainable, thanks to the Design Thinking method through analytical and intuitive thinking.

Design & Understanding Requirements

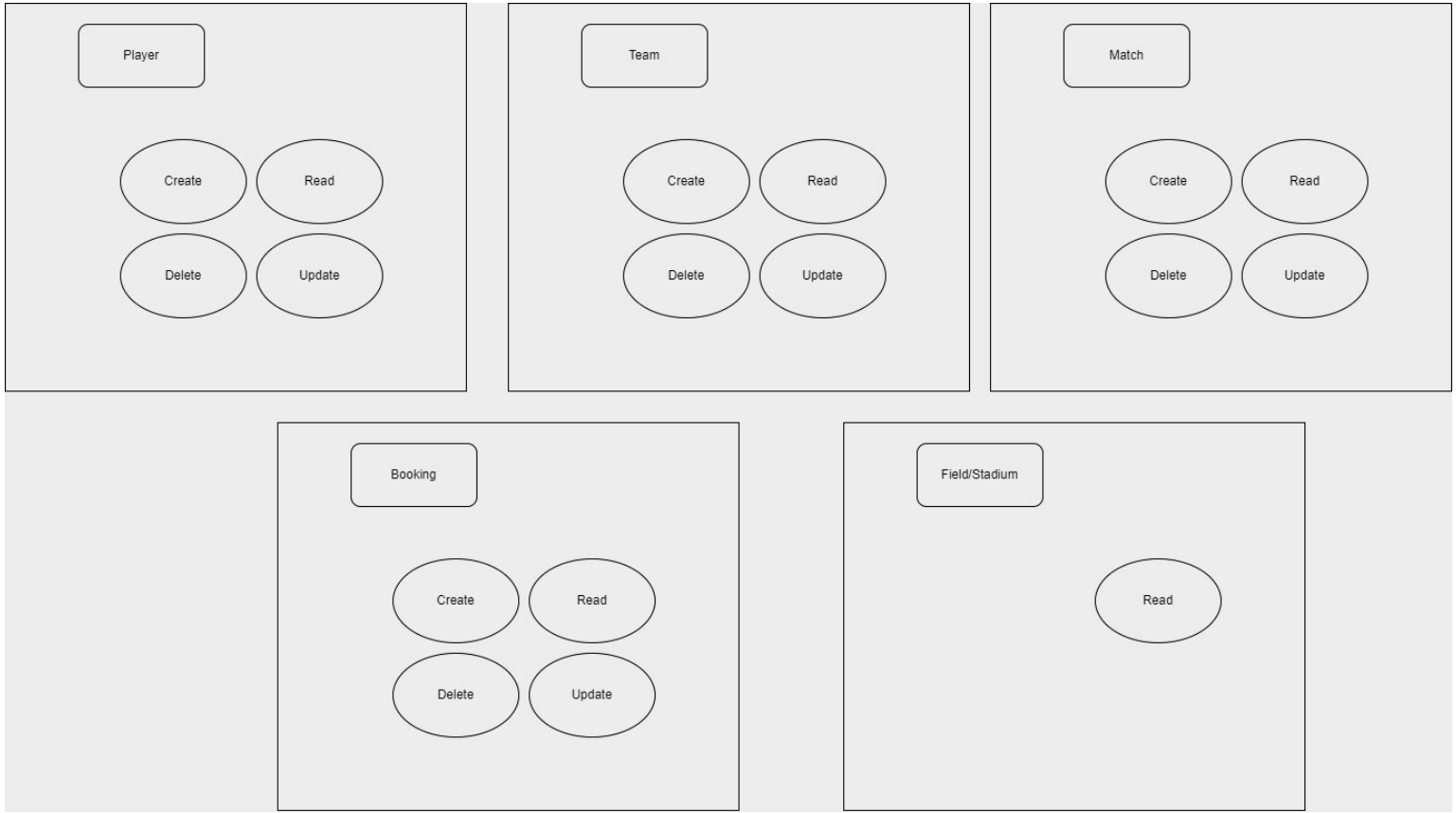
Using UML diagrams to:

- Visualise designs (before it takes place or as documentation for a project afterward)
- Understand requirements
- Bring new team members or developers switching teams up to speed quickly.
- Navigate source code.
- Plan out new features before any programming takes place.
- Communicate with technical and non-technical audiences more easily.

Use Case Diagram

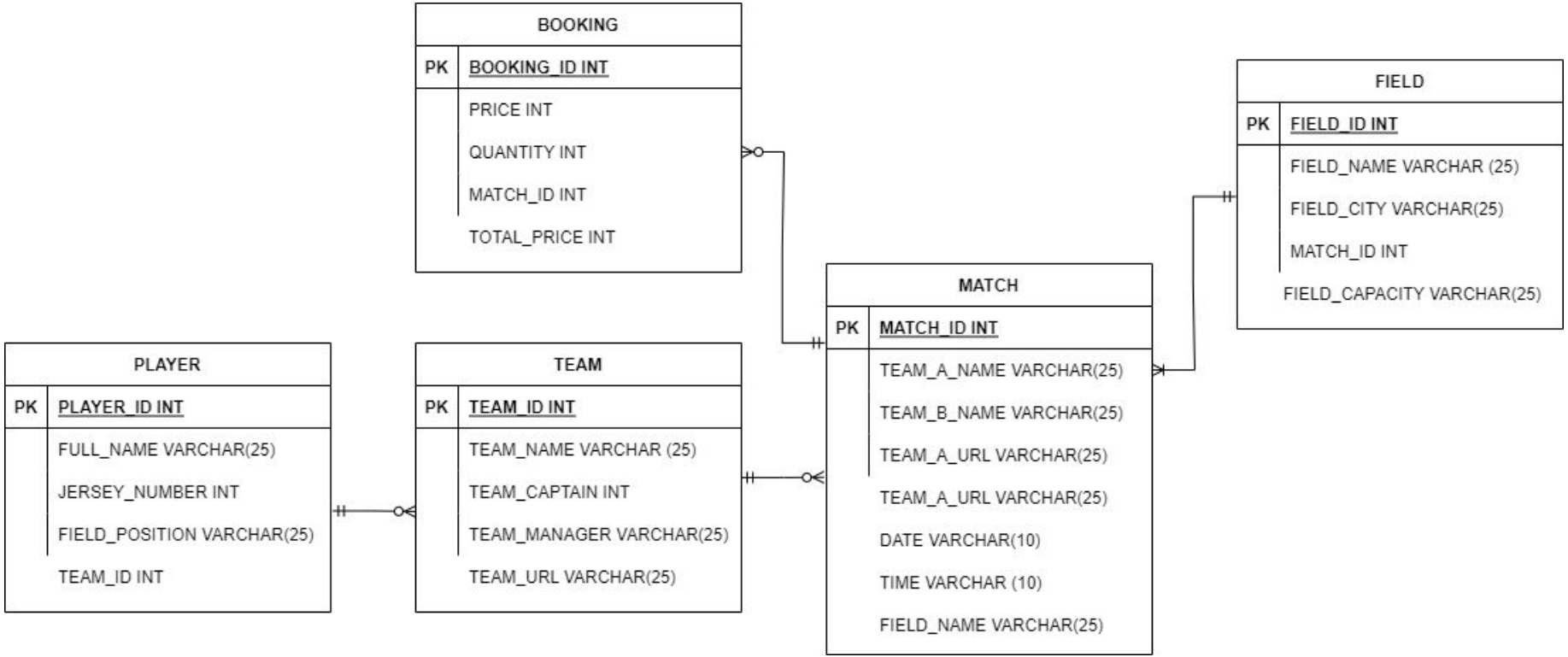


Components Diagram (CRUD)

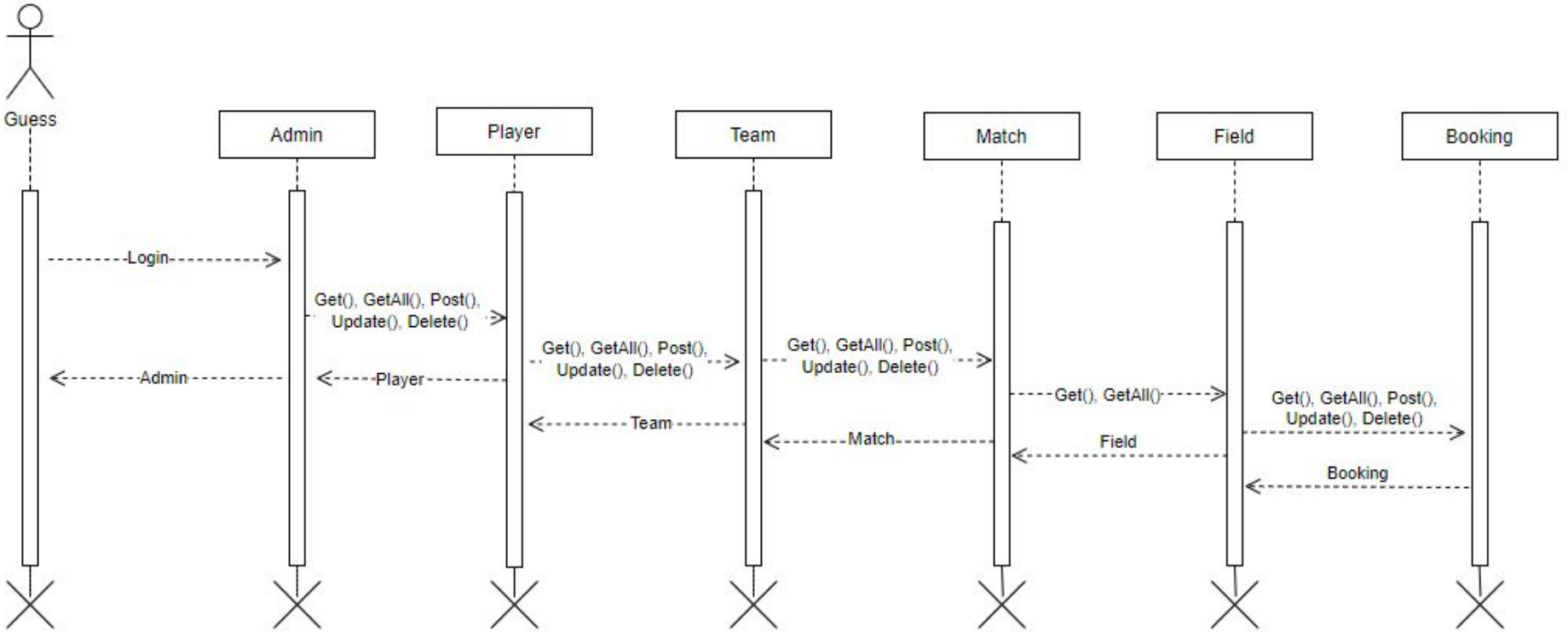


Class Diagram

Entity Relation Diagram



Sequence Diagram

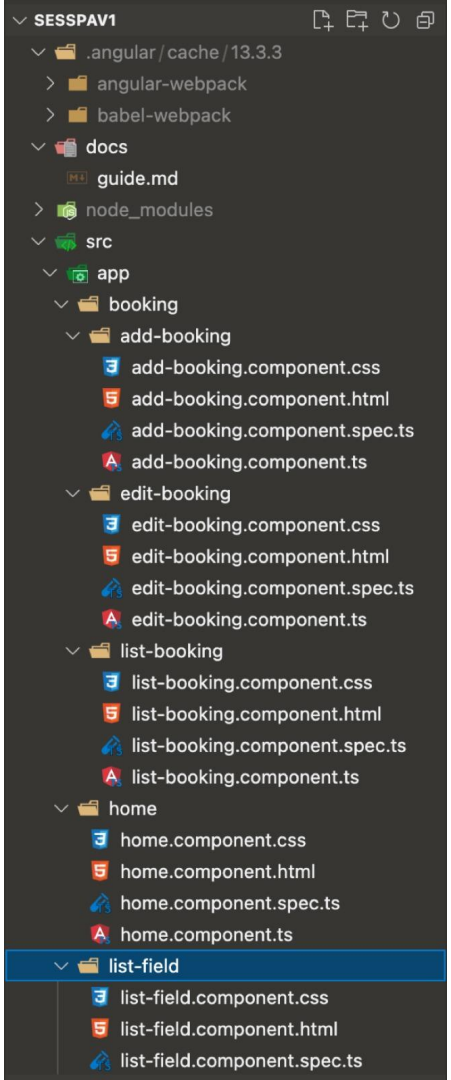


BACKEND

SESWebAPIApp Project

Layered Architecture

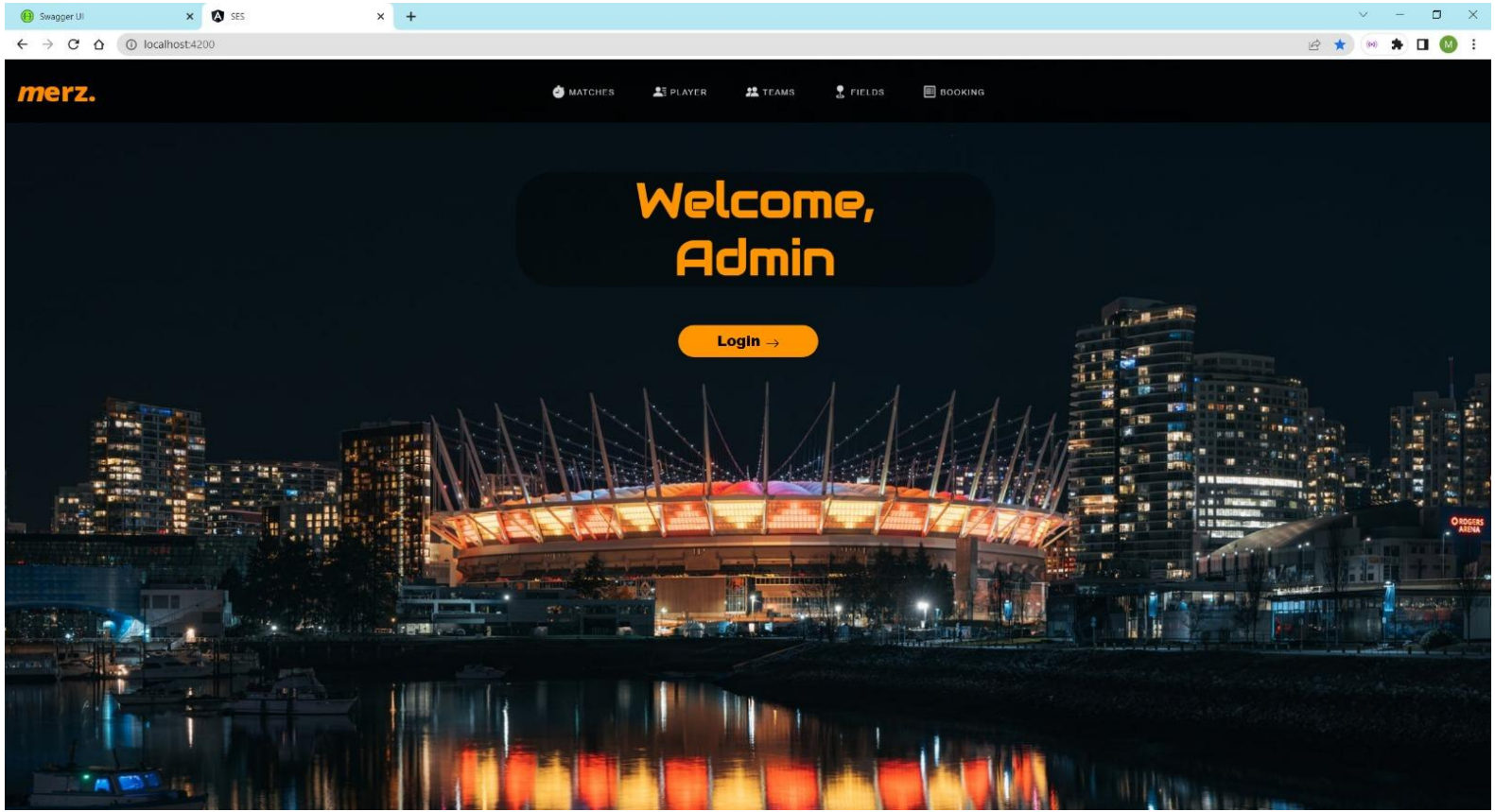
Front-End



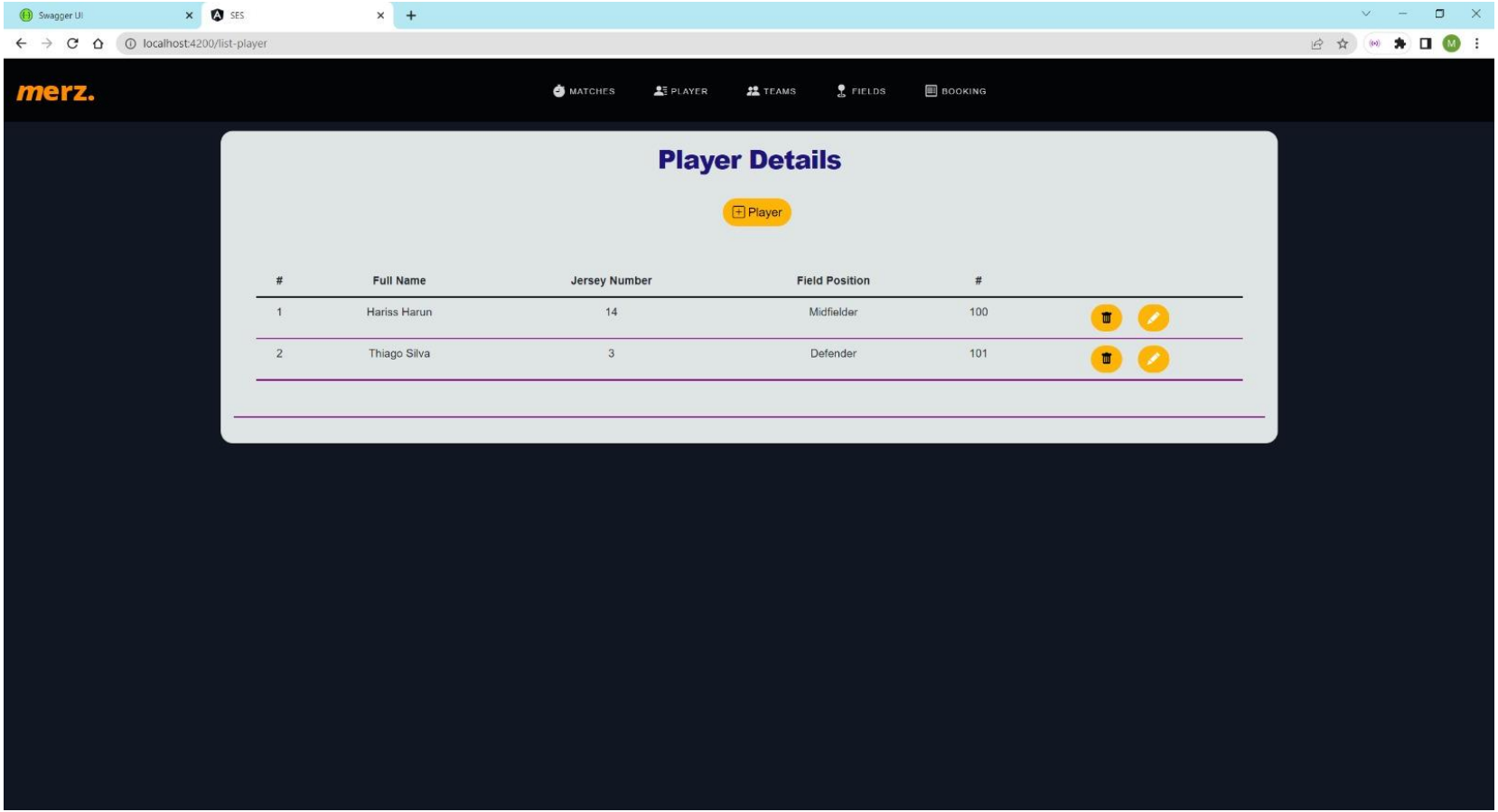
Back-End

Running App Snapshots

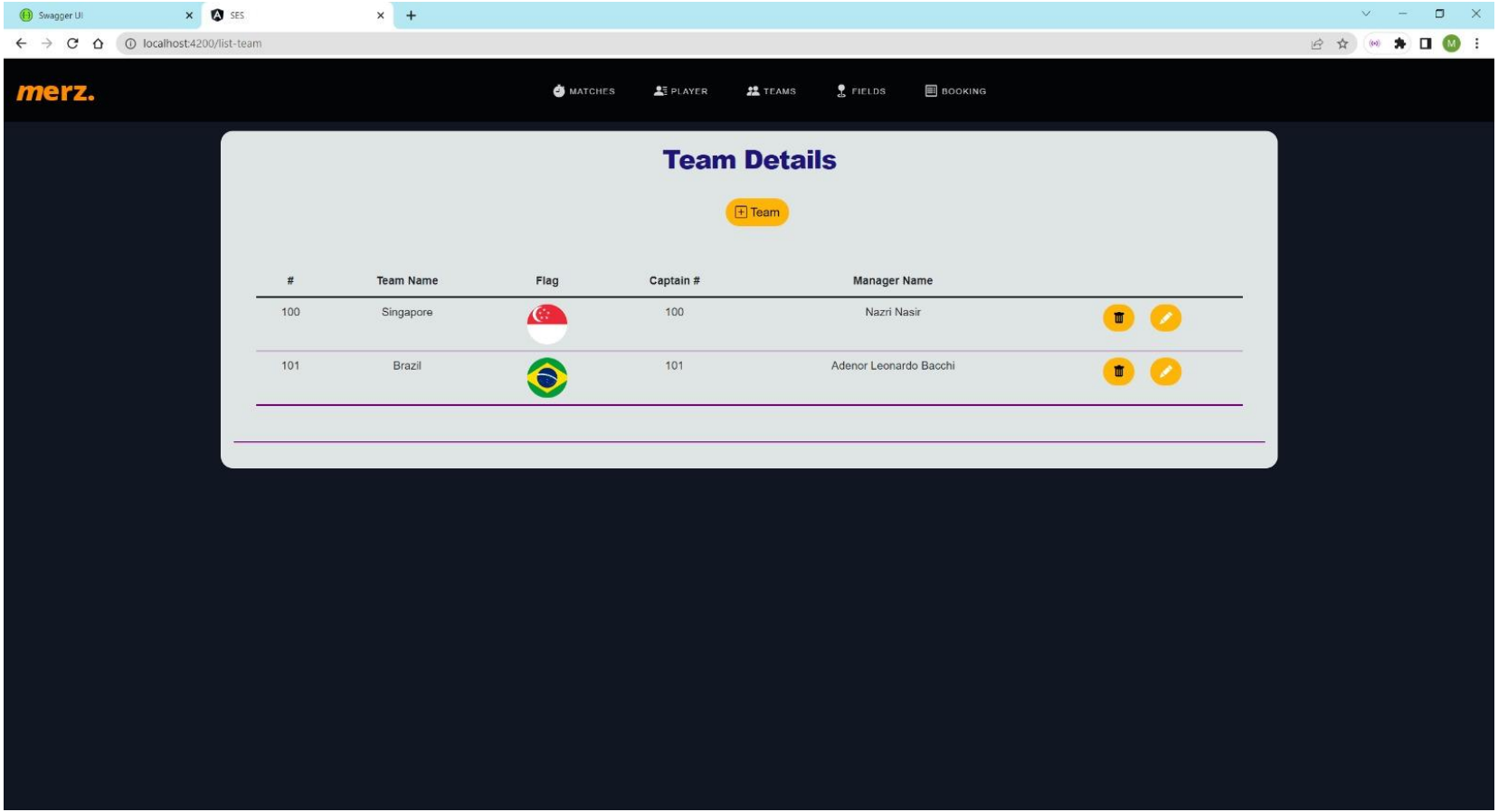
Home Page


Swagger UI
SES
localhost:4200
merz.
MATCHES
PLAYER
TEAMS
FIELDS
BOOKING
Welcome,
Admin
Login
ROGERS ARENA

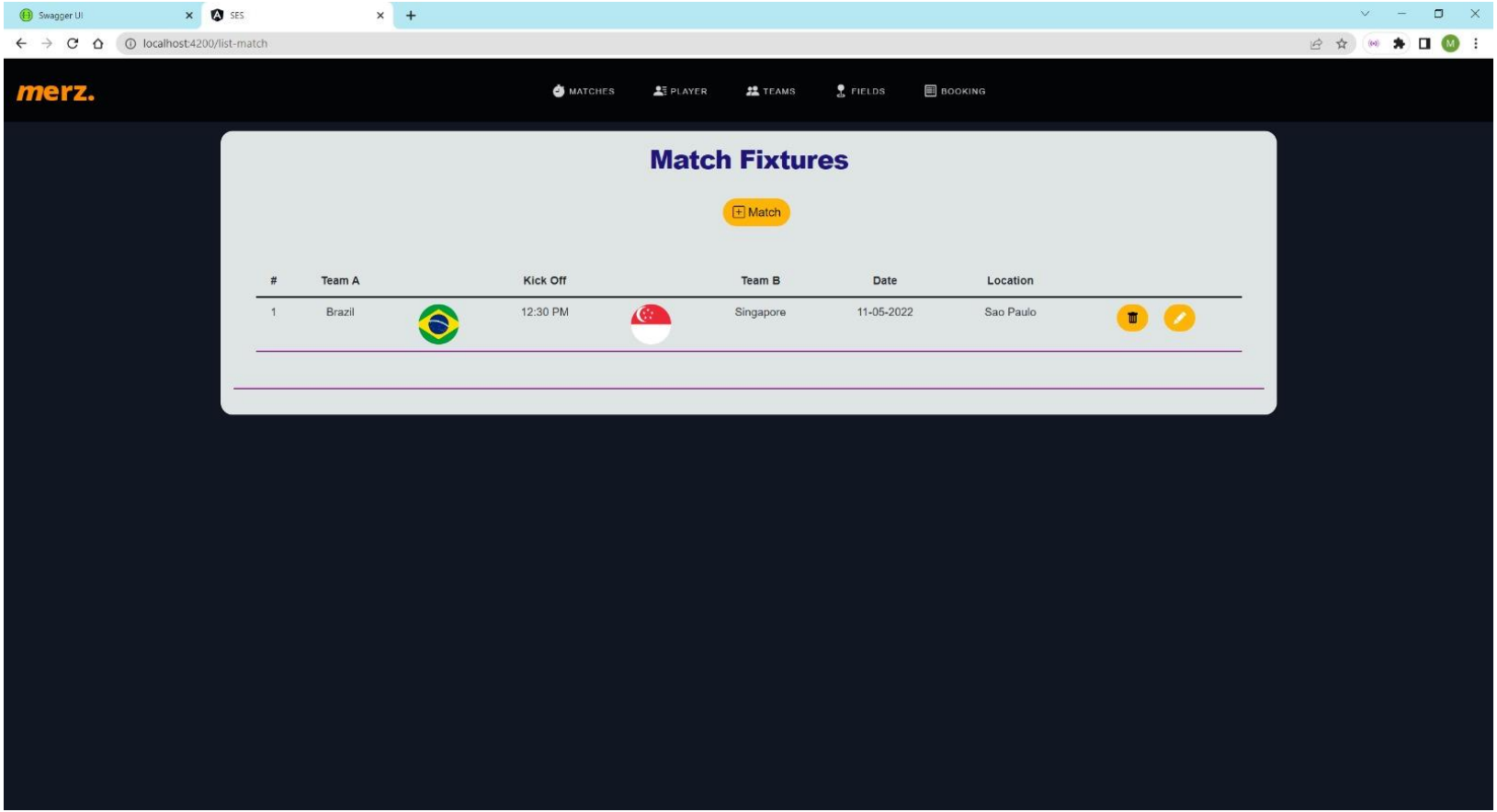
Players


Swagger UI
SES
localhost:4200/list-player
merz.
MATCHES
PLAYER
TEAMS
FIELDS
BOOKING
Player Details
Player
Full Name Jersey Number Field Position
1 Hariss Harun 14 Midfielder 100
2 Thiago Silva 3 Defender 101

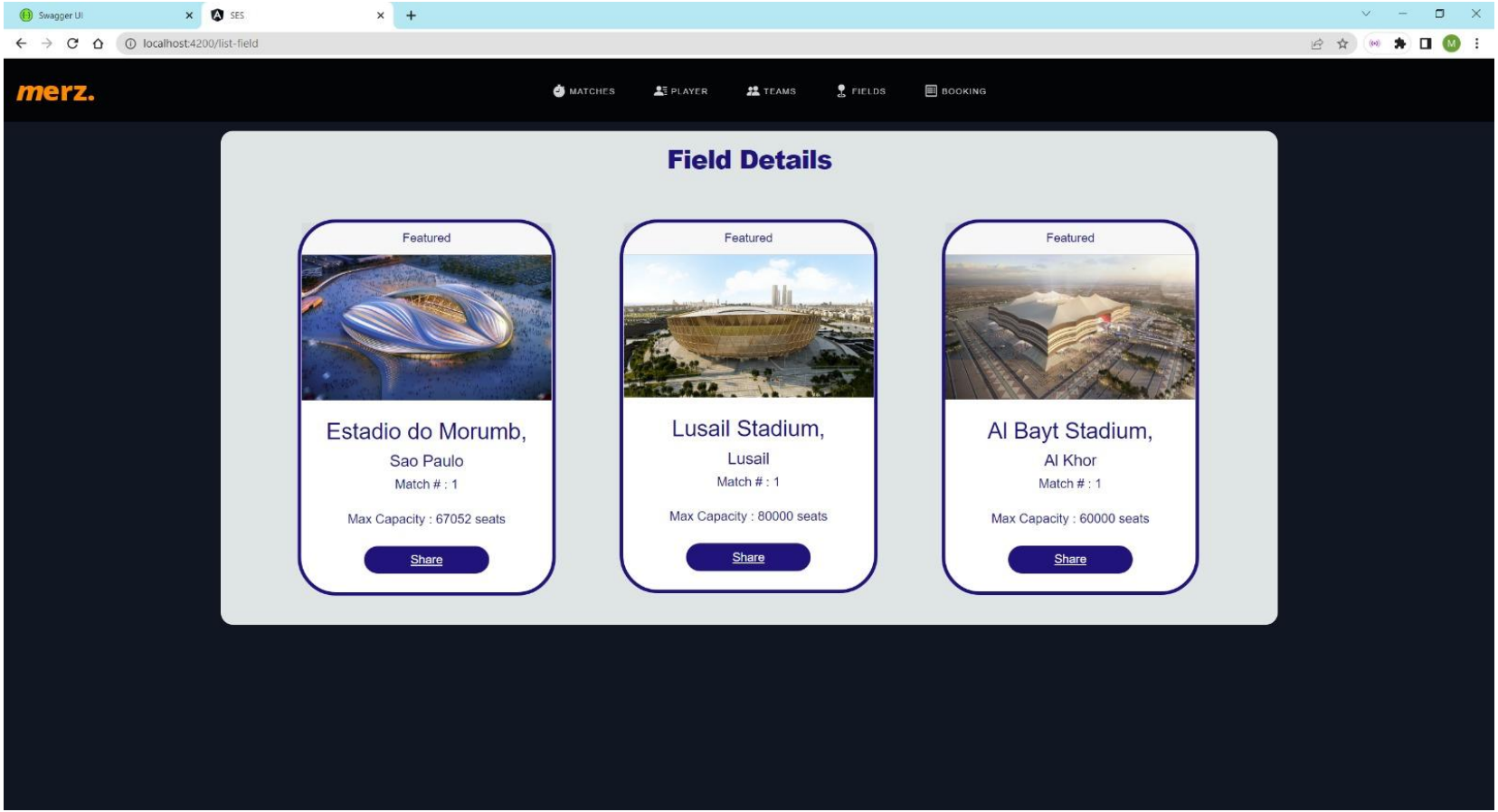
Team


Swagger UI
SES
localhost:4200/list-team
merz.
MATCHES
PLAYER
TEAMS
FIELDS
BOOKING
Team Details
Team
Team Name Flag Captain # Manager Name
100 Singapore 100 Nazri Nasir
101 Brazil 101 Adenor Leonardo Bacchi

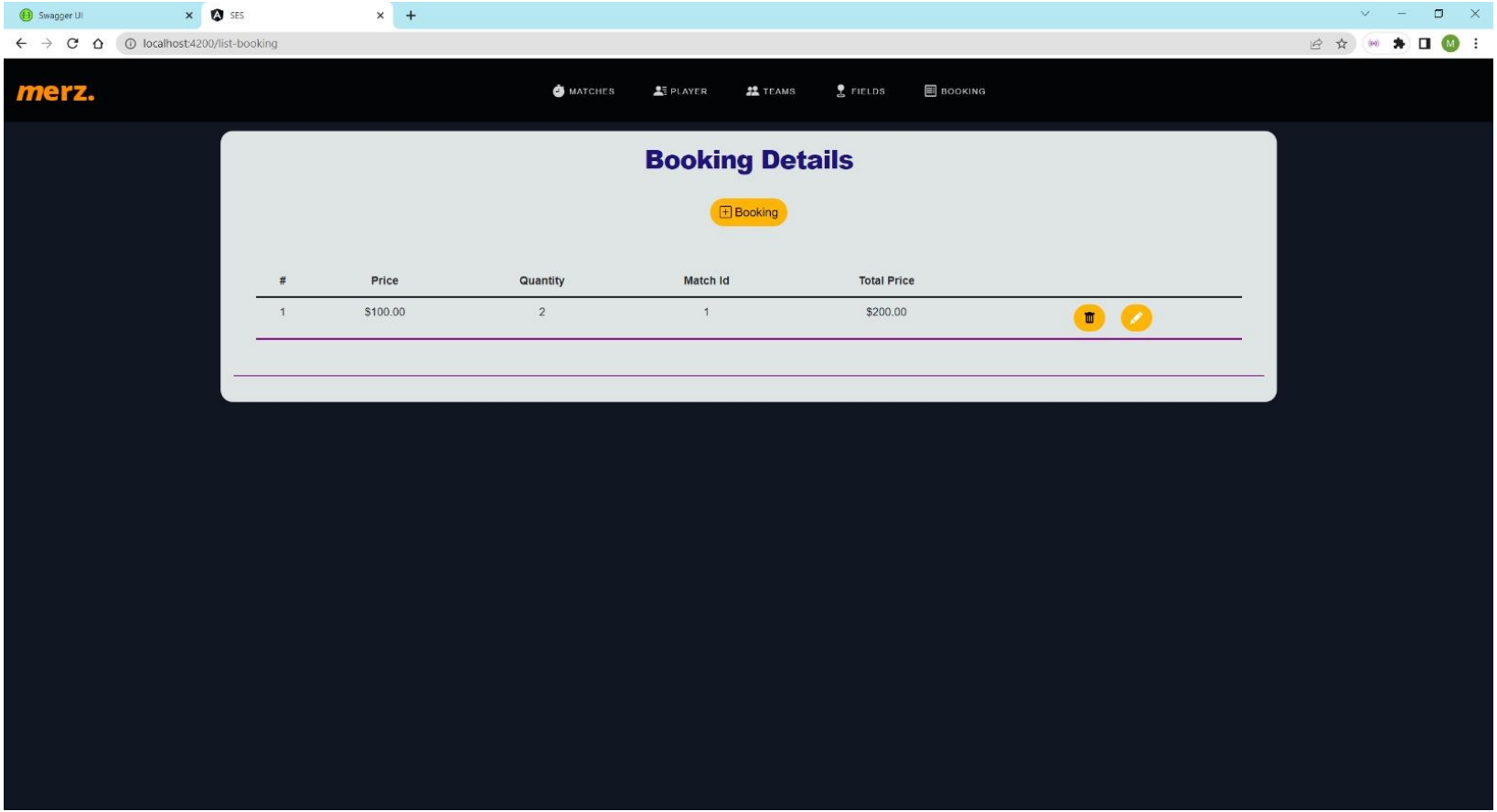
Matches


Swagger UI
SES
localhost:4200/list-match
merz.
MATCHES
PLAYER
TEAMS
FIELDS
BOOKING
Match Fixtures
Match
Team A Kick Off Team B Date Location
1 Brazil 12:30 PM Singapore 11-05-2022 Sao Paulo

Field


Swagger UI
SES
localhost:4200/list-field
merz.
MATCHES
PLAYER
TEAMS
FIELDS
BOOKING
Field Details
Featured
Estadio do Morumb,
Sao Paulo
Match # : 1
Max Capacity : 67052 seats
Share
Featured
Lusail Stadium,
Lusail
Match # : 1
Max Capacity : 80000 seats
Share
Featured
Al Bayt Stadium,
Al Khor
Match # : 1
Max Capacity : 60000 seats
Share

Booking


Swagger UI
SES
localhost:4200/list-booking
merz.
MATCHES
PLAYER
TEAMS
FIELDS
BOOKING
Booking Details
Booking
Price Quantity Match Id Total Price
1 $100.00 2 1 $200.00

Code Snapshots

Models

```cs
using System.ComponentModel.DataAnnotations;

namespace SESWebAPIV1.Models
{
    public class Player
    {
        [Key]
        public int Id { get; set; }

        public string FullName { get; set; }

        public int JerseyNumber { get; set; }

        public string FieldPosition { get; set; }

        public int TeamId { get; set; }
    }
}
```

```cs
using System.ComponentModel.DataAnnotations;

namespace SESWebAPIV1.Models
{
    public class Field
    {
        [Key]
        public int FieldId { get; set; }

        public string FieldName { get; set; }

        public string FieldCity { get; set; }

        public int MatchId { get; set; }

        public int FieldCapacity { get; set; }

        public string FieldURL { get; set; }
    }
}
```

```cs
using System.ComponentModel.DataAnnotations;

namespace SESWebAPIV1.Models
{
    public class Booking
    {
        [Key]
        public int BookingId { get; set; }

        public int Price { get; set; }

        public int Quantity { get; set; }

        public int MatchId { get; set; }

        public int TotalPrice { get; set; }
    }
}
```

```cs
using System.ComponentModel.DataAnnotations;

namespace SESWebAPIV1.Models
{
    public class Admin
    {
        [Key]
        public string Username { get; set; }

        public string Password { get; set; }
    }
}
```

```cs
using System.ComponentModel.DataAnnotations;

namespace SESWebAPIV1.Models
{
    public class Team
    {
        [Key]
        public int TeamId { get; set; }

        public string TeamName { get; set; }

        public int TeamCaptain { get; set; }

        public string TeamManager { get; set; }

        public string TeamURL { get; set; }
    }
}
```

```cs
using System.ComponentModel.DataAnnotations;

namespace SESWebAPIV1.Models
{
    public class Match
    {
        [Key]
        public int MatchId { get; set; }

        public string TeamA { get; set; }

        public string TeamAURL { get; set; }

        public string TeamB { get; set; }

        public string TeamBURL { get; set; }

        public string Date { get; set; }

        public string Time { get; set; }

        public string Location { get; set; }
    }
}
```

```cs
namespace SESWebAPIV1.Models
{
    public class AdminDTO
    {
        public string Username { get; set; }

        public string Password { get; set; }
    }
}
```

```cs
namespace SESWebAPIV1.Models
{
    public class AdminLoginDTO
    {
        public string Username { get; set; }

        public string Password { get; set; }
    }
}
```

Repository

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.Extensions.Logging;
using SESWebAPIV1.Models;
using SESWebAPIV1.APIServices;

namespace SESWebAPIV1.Repository
{
    public class PlayerRepository : IPlayerRepository<int, Player>
    {
        private readonly MatchContext _context;
        private readonly ILogger<MatchRepository> _logger;

        public PlayerRepository(MatchContext context, ILogger<MatchRepository> logger)
        {
            _context = context;
            _logger = logger;
        }

        public async Task<Player> Add(Player item)
        {
            try
            {
                _context.Players.Add(item);
                await _context.SaveChangesAsync();
                return item;
            }
            catch (Exception exception)
            {
                _logger.LogError(exception.Message);
                _logger.LogError("----" + DateTime.Now.ToString());
            }
            return null;
        }

        public async Task<Player> Delete(int key)
        {
            try
            {
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.Extensions.Logging;
using SESWebAPIV1.Models;

namespace SESWebAPIV1.Repository
{
    public class TeamRepository : ITeamRepository<int, Team>
    {
        private readonly MatchContext _context;
        private readonly ILogger<MatchRepository> _logger;

        public TeamRepository(MatchContext context, ILogger<MatchRepository> logger)
        {
            _context = context;
            _logger = logger;
        }

        public async Task<Team> Add(Team item)
        {
            try
            {
                _context.Teams.Add(item);
                await _context.SaveChangesAsync();
                return item;
            }
            catch (Exception exception)
            {
                _logger.LogError(exception.Message);
                _logger.LogError("----" + DateTime.Now.ToString());
            }
            return null;
        }

        public async Task<Team> Delete(int key)
        {
            try
            {
                var item = await Get(key);
                _context.Teams.Remove(item);
```

To be continued…

Interfaces

```
using System.Collections.Generic;
using System.Threading.Tasks;

namespace SESWebAPIV1.Repository
{
    public interface IPlayerRepository<N, P>
    {
        Task<P> Get(N key);

        Task<P> Update(P item);

        Task<P> Delete(N key);

        Task<P> Add(P item);

        Task<ICollection<P>> GetAll();
    }
}
```

```
using System.Collections.Generic;
using System.Threading.Tasks;

namespace SESWebAPIV1.Repository
{
    public interface IMatchRepository<K, T>
    {
        Task<T> Get(K key);

        Task<T> Update(T item);

        Task<T> Delete(K key);

        Task<T> Add(T item);

        Task<ICollection<T>> GetAll();
    }
}
```

```
using SESWebAPIV1.Models;

namespace SESWebAPIV1.Repository
{
    public interface IAdmin
    {
        AdminLoginDTO Login(AdminLoginDTO admin);
    }
}
```

```
using System.Collections.Generic;
using System.Threading.Tasks;

namespace SESWebAPIV1.Repository
{
    public interface ITeamRepository<N, P>
    {
        Task<P> Get(N key);

        Task<P> Update(P item);

        Task<P> Delete(N key);

        Task<P> Add(P item);

        Task<ICollection<P>> GetAll();
    }
}
```

```
using System.Collections.Generic;
using System.Threading.Tasks;

namespace SESWebAPIV1.Repository
{
    public interface IFieldRepository<F, T>
    {
        Task<T> Get(F key);

        Task<ICollection<T>> GetAll();
    }
}
```

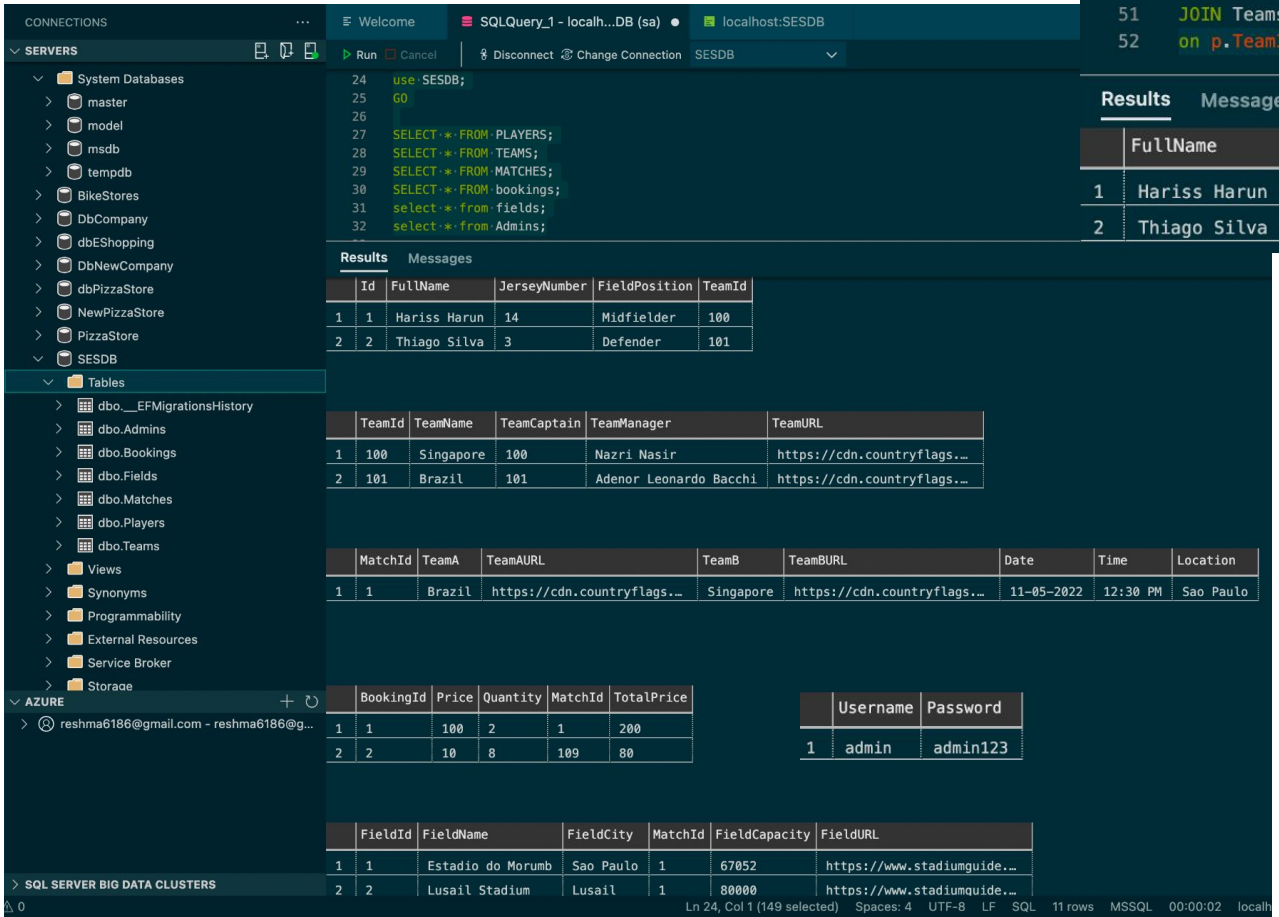
```
using System.Collections.Generic;
using System.Threading.Tasks;

namespace SESWebAPIV1.Repository
{
    public interface IBookingRepository<K, T>
    {
        Task<T> Get(K key);

        Task<T> Update(T item);

        Task<T> Delete(K key);

        Task<T> Add(T item);

        Task<ICollection<T>> GetAll();
    }
}
```

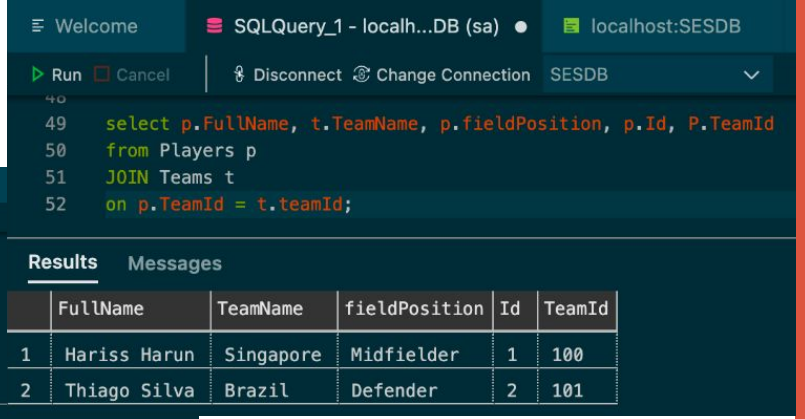
Controllers

```
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using SESWebAPIV1.Models;
using SESWebAPIV1.Repository;

// For more information on enabling Web API for empty projects, visit https:/

namespace SESWebAPIV1.APIServices
{
    // save it for front end
    [Route("api/[controller]")]
    [ApiController]
    public class PlayerController : ControllerBase
    {
        private readonly IPlayerRepository<int, Player> _repository;

        public PlayerController(IPlayerRepository<int, Player> Repository)
        {
            _repository = Repository;
        }

        [HttpGet]
        public async Task<ActionResult<List<Player>>> GetAll()
        {
            var players = await _repository.GetAll();
            return players.ToList();
        }
        [HttpPost]
        public async Task<ActionResult<Player>> Post(Player player)
        {
            return await _repository.Add(player);
        }
        [HttpGet]
        [Route("GetPlayerByID/{id}")]
        public async Task<ActionResult<Player>> Get(int id)
        {
            return await _repository.Get(id);
        }
```

```
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using SESWebAPIV1.Models;
using SESWebAPIV1.Repository;

// For more information on enabling Web API for empty projects, visit h

namespace SESWebAPIV1.APIServices
{
    [Route("api/[controller]")]
    [ApiController]
    public class TeamController : Controller
    {
        private readonly ITeamRepository<int, Team> _repository;

        public TeamController(ITeamRepository<int, Team> Repository)
        {
            _repository = Repository;
        }

        [HttpGet]
        public async Task<ActionResult<List<Team>>> GetAll()
        {
            var teams = await _repository.GetAll();
            return teams.ToList();
        }
        [HttpPost]
        public async Task<ActionResult<Team>> Post(Team Team)
        {
            return await _repository.Add(Team);
        }
        [HttpGet]
        [Route("GetTeamById/{id}")]
        public async Task<ActionResult<Team>> Get(int id)
        {
            return await _repository.Get(id);
        }

        [HttpPut]
```

Testing with mssql





Due to Mac OS, I had to install Docker & Azure Data Studio to access mssql server

Testing Web APIs

Testing with Postman

Testing with Swagger

CRUD Operations

Player

GET	/api/Player
POST	/api/Player
PUT	/api/Player
DELETE	/api/Player
GET	/api/Player/GetPlayerByID/{id}

Team

GET	/api/Team
POST	/api/Team
PUT	/api/Team
DELETE	/api/Team
GET	/api/Team/GetTeamById/{id}

WeatherForecast

GET	/WeatherForecast

Field

GET	/api/Field
GET	/api/Field/GetFieldByID/{id}

Match

GET	/api/Match
POST	/api/Match
PUT	/api/Match

Match

GET	/api/Match
POST	/api/Match
PUT	/api/Match
DELETE	/api/Match
GET	/api/Match/GetMatchByID/{id}

Admin

POST	/api/Admin/Login

Booking

GET	/api/Booking
POST	/api/Booking
PUT	/api/Booking
DELETE	/api/Booking
GET	/api/Booking/GetBookingByID/{id}

Testing with Swagger

Responses

Curl

```
curl -X GET "http://localhost:5000/api/Player" -H  "accept: text/plain"
```

Request URL

```
http://localhost:5000/api/Player
```

Server response

Code	Details
200	Response body

```
[
  {
    "id": 1,
    "fullName": "Hariss Harun",
    "jerseyNumber": 14,
    "fieldPosition": "Midfielder",
    "teamId": 100
  },
  {
    "id": 2,
    "fullName": "Thiago Silva",
    "jerseyNumber": 3,
    "fieldPosition": "Defender",
    "teamId": 101
  }
]
```

Response headers

```
content-length: 188
content-type: application/json; charset=utf-8
date: Tue19 Apr 2022 03:40:09 GMT
server: Kestrel
```

Responses

Responses

Curl

```
curl -X POST "http://localhost:5000/api/Booking" -H  "accept: text/plain" -H
{\"bookingId\":0,\"price\":10,\"quantity\":8,\"matchId\":109,\"totalPrice\":0
```

Request URL

```
http://localhost:5000/api/Booking
```

Server response

Code	Details
200	Response body

```
{
  "bookingId": 2,
  "price": 10,
  "quantity": 8,
  "matchId": 109,
  "totalPrice": 80
}
```

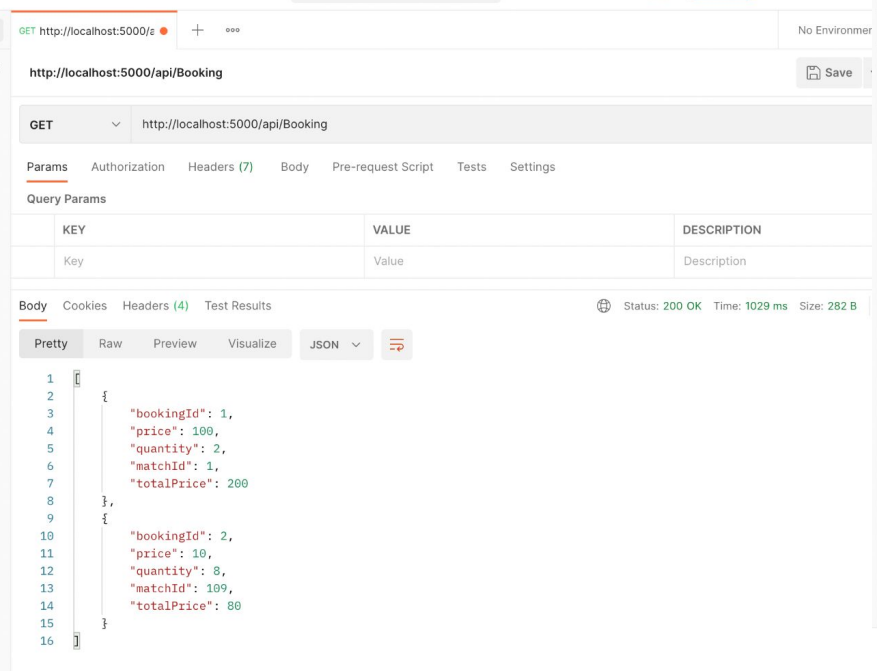
Response headers

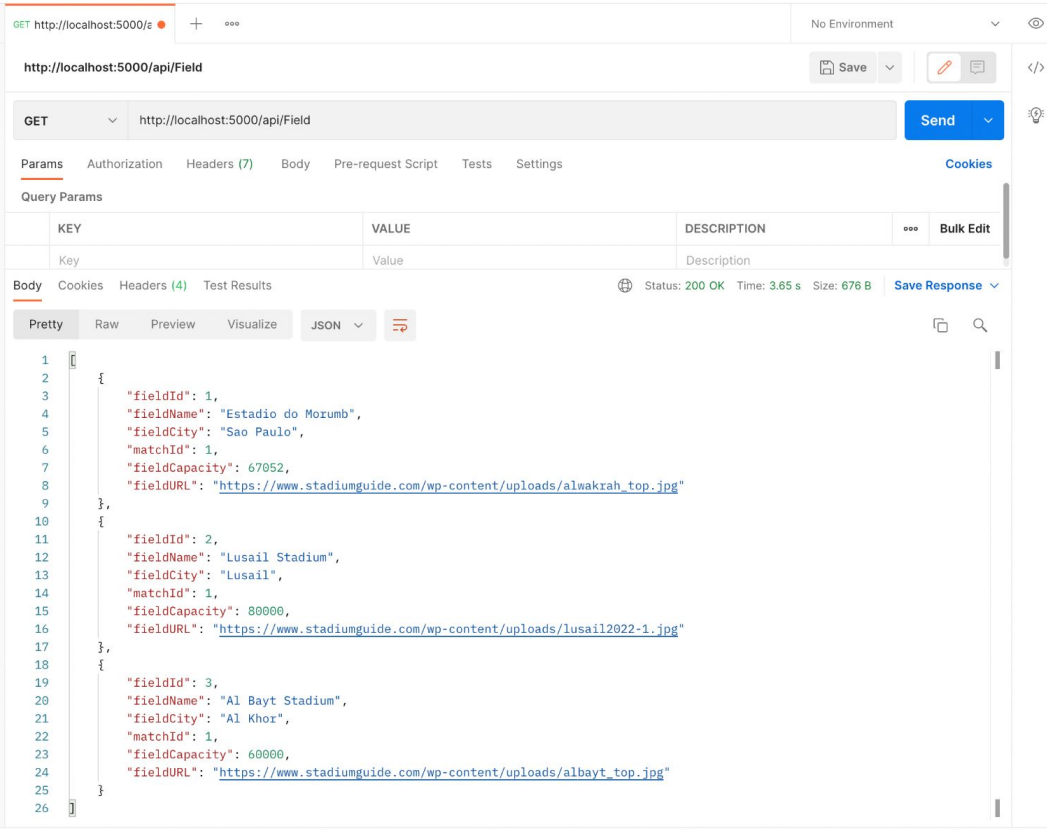
```
content-length: 69
content-type: application/json; charset=utf-8
date: Tue19 Apr 2022 03:43:58 GMT
server: Kestrel
```

Responses

Code	Description
200	Success

Testing with Postman





Agile Activates

As part of the back-end team, we were agile and learnt the flows of the frontend.

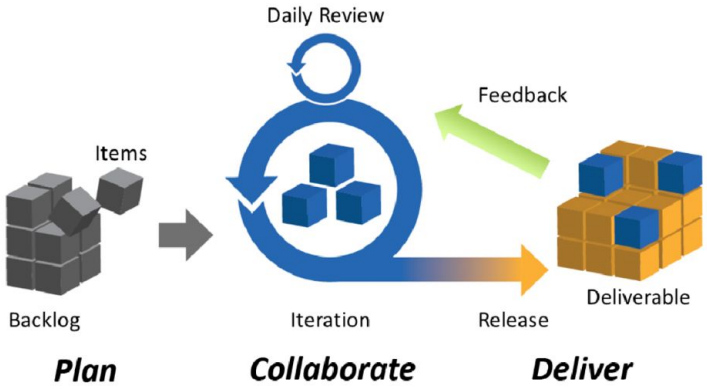
We actively learnt and applied our knowledge in whatever way we can in our project.

We followed TDD where we have fixed time but scope varies and we make mistakes intentionally so that we can rectify and refine the scopes of our project

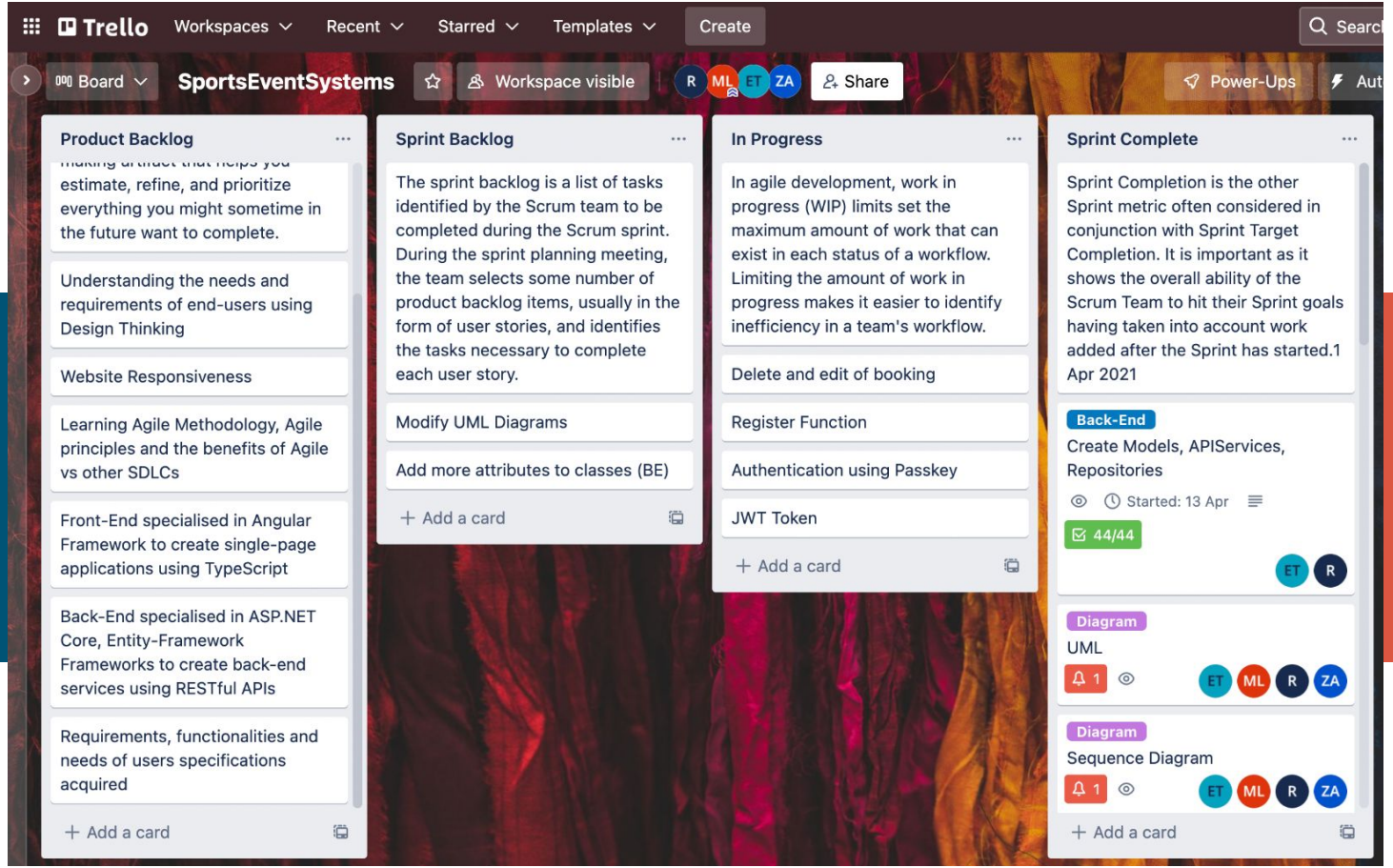
We also implemented a version 2 for back-end to experiment and implement extra features such as the addition of foreign keys, JWT and authentication, which is intended for the next iteration.

We also followed Agile Principles and implemented Design Thinking.

We also updated the Trello board.



Trello Board



Learning & Experiences

► Learning to build a small application in a business context from the scratch was fulfilling and help us further understands the concepts taught during the trainings.
► The pace was well adjusted to accommodate new learners.
► Each topic was taught in detail with ample examples.
► There were enough exercises to practice the things taught.
► The details on configurations and set-ups were also emphasize well.
► The instructor was able to address our individual concerns and guided the participants in troubleshooting various issues.
► The instructor kept track of our progress and consistently get our feedbacks to adjust his pace and gave us more explanations.
► The testing topics were helpful and emphasize the importance of it in development.

To conclude, the training was accommodating and enjoyable with the instructor taking measures to ensure that we are all engaged and able to take away learning bits regardless of our prior experiences.

Thank you!

MERZ