



SPORTS EVENTS SYSTEM

MERZ

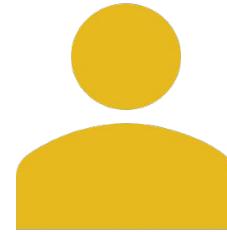
TEAM 1

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 an 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

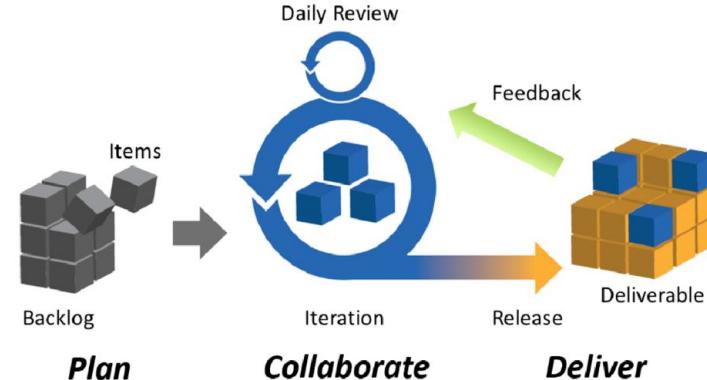
Easy adjustment to new requirements

Process visibility and high interaction between a customer and a team

Knowledge sharing for making better decisions

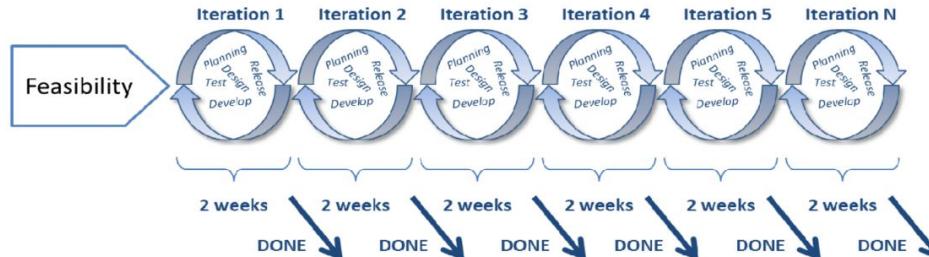
Reduced project development time frames

Business risks are minimized

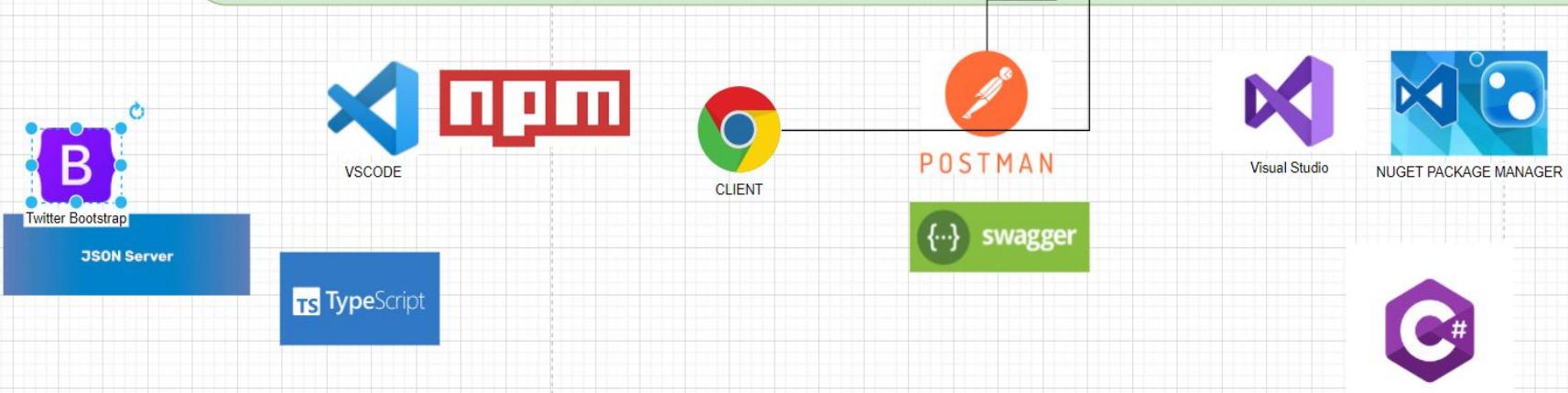
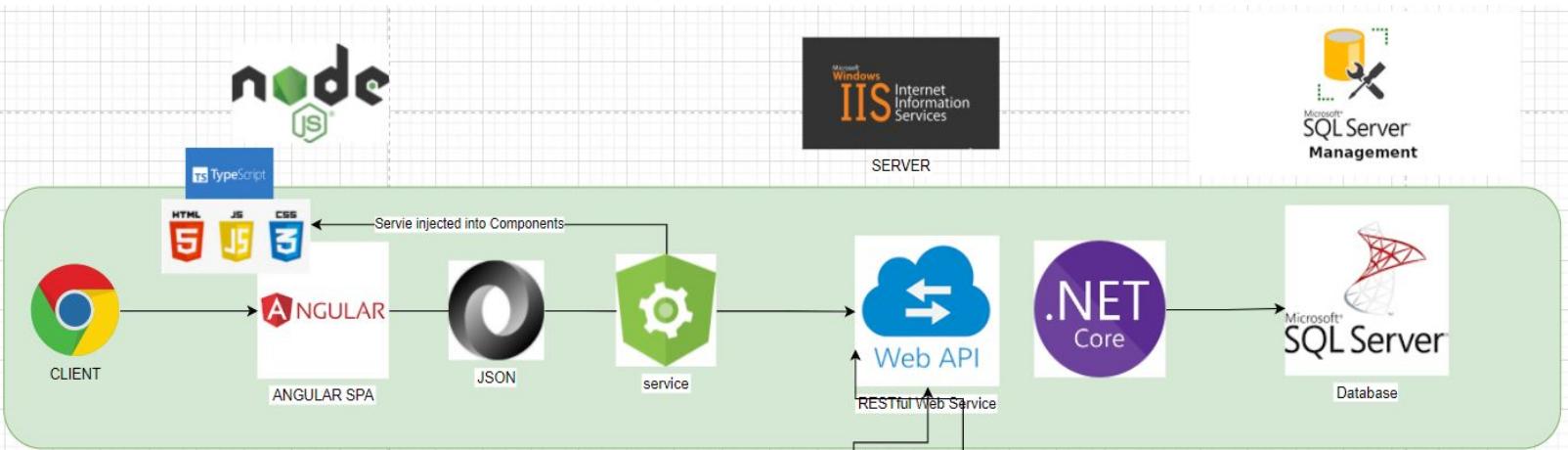


Build The Right Product
Build It In The Right Way

Incremental (Adaptive)



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.

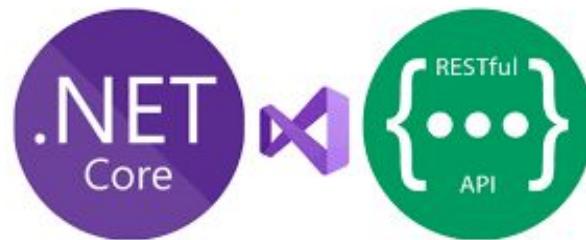
This software needs to have the following **features**:

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



Databases

Front-End {Tools}
JSON Server



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 Thinking

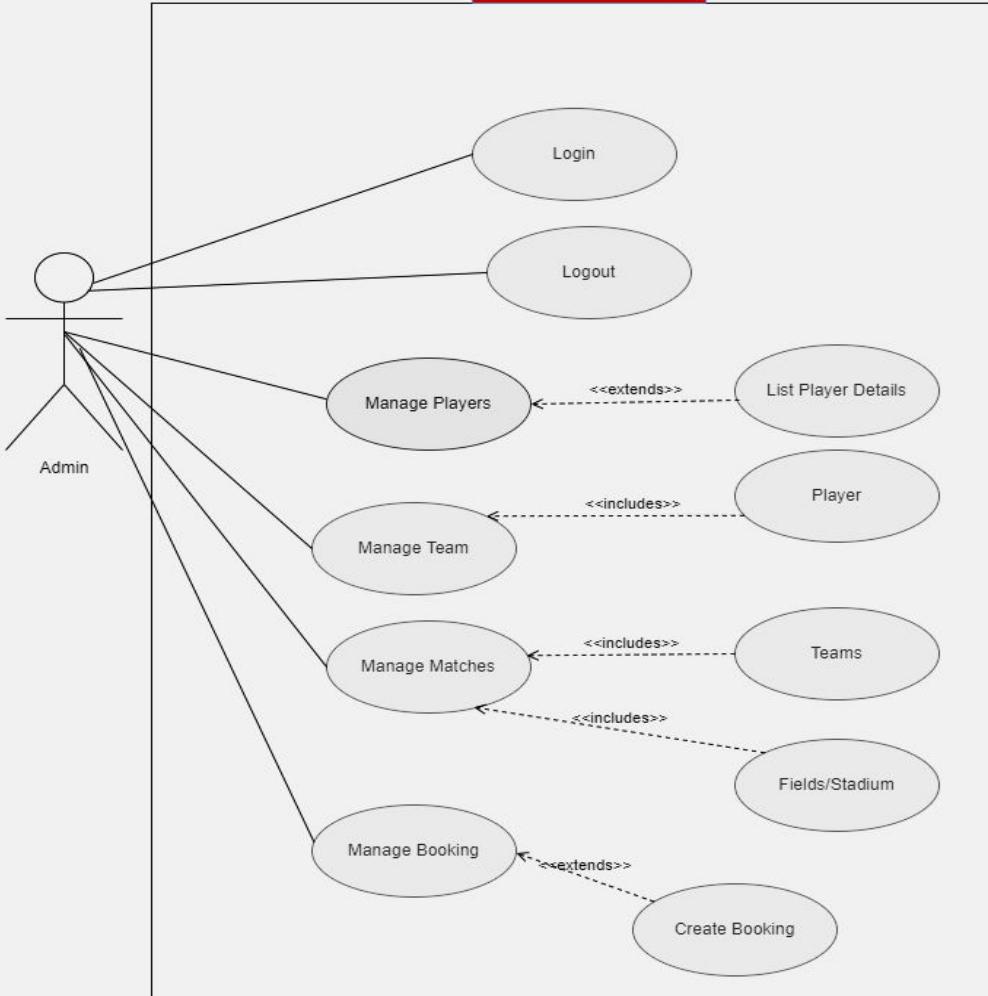
Design & Understanding Requirements

Using UML diagrams to:

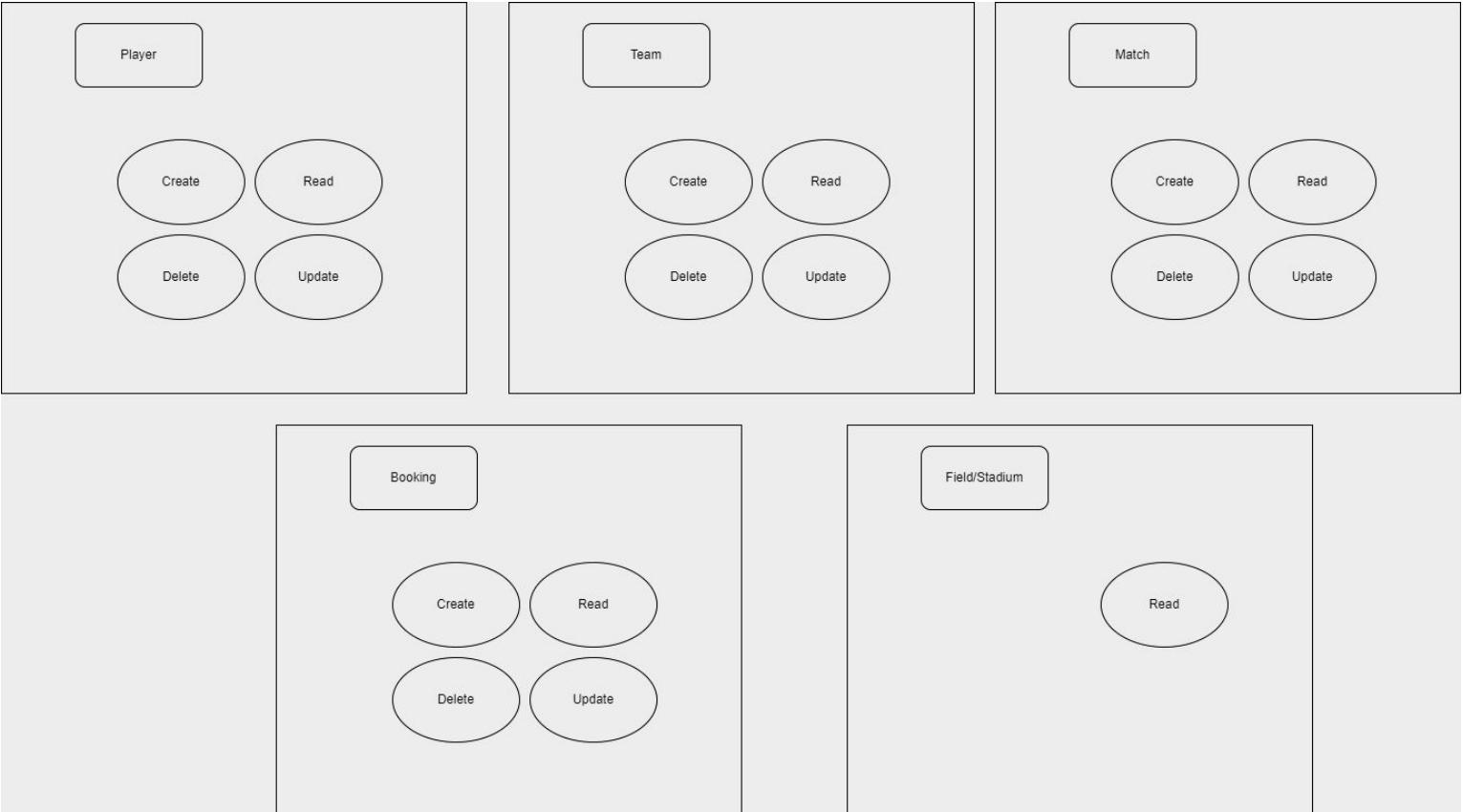
- Visualise designs (before it takes place or as documentation for a project afterward)
- Understand requirements
- Bring new team members or developers switching teams up to speed quickly.
- Navigate source code.
- Plan out new features before any programming takes place.
- Communicate with technical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



Use Case Diagram



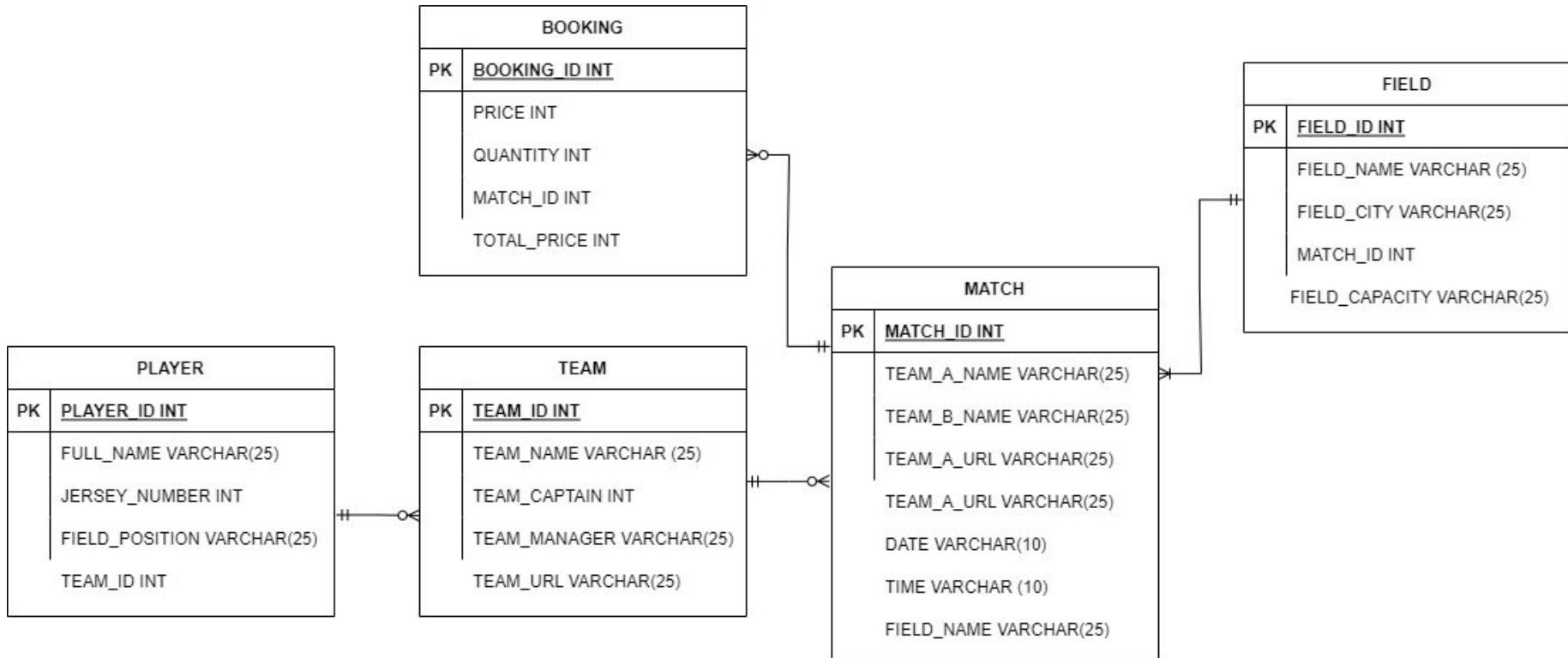
Components Diagram (CRUD)



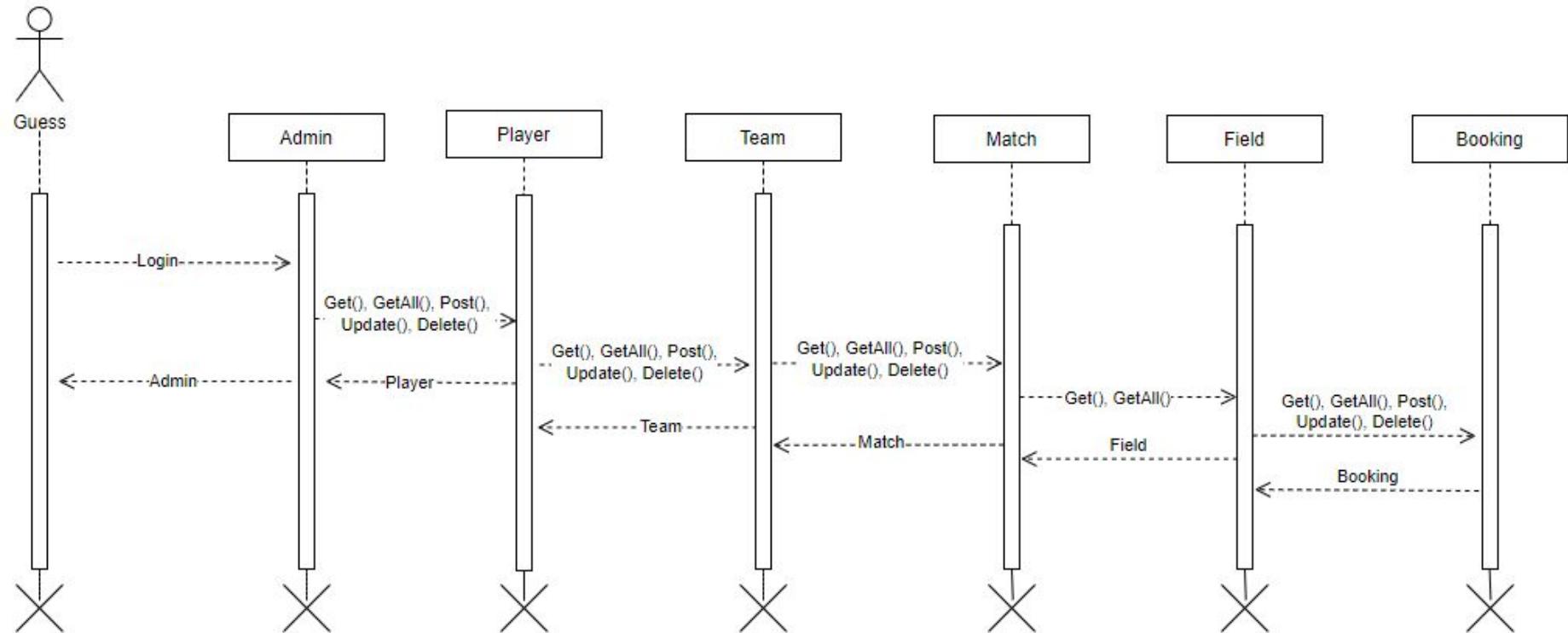
Class Diagram

Login	Player	Match
+ username: String + password: String + loginStatus: String	+ playerID: int + fullName: String + jerseyNumber: int + fieldPosition: String + teamId: number	+ matchID: int + team A Name: String + team B Name : String + team A URL : String + team B URL : String + Date : String + Time : String + Field Name : String
+ login(username, password): boolean + logout(): boolean	+ createPlayer(matchID) + deletePlayer(matchID) + editPlayer(matchID)	+ createMatch(matchID) + editMatch(matchID) + deleteMatch(matchID)
Team	Booking	Field
+teamID: int + teamName: String + teamCaptain: int + teamManager: String + teamURL: string	+ bookingID: int + price: int + quantity: int + matchID: int + totalPrice: int	+fieldID: int + fieldName: String + fieldCity: int + matchID: String + fieldCapacity: string
+ createTeam(teamID) + deleteTeam(teamID) + editTeam(teamID)	+createBooking(bookingID)	+ listField(fieldID)

Entity Relation Diagram



Sequence Diagram

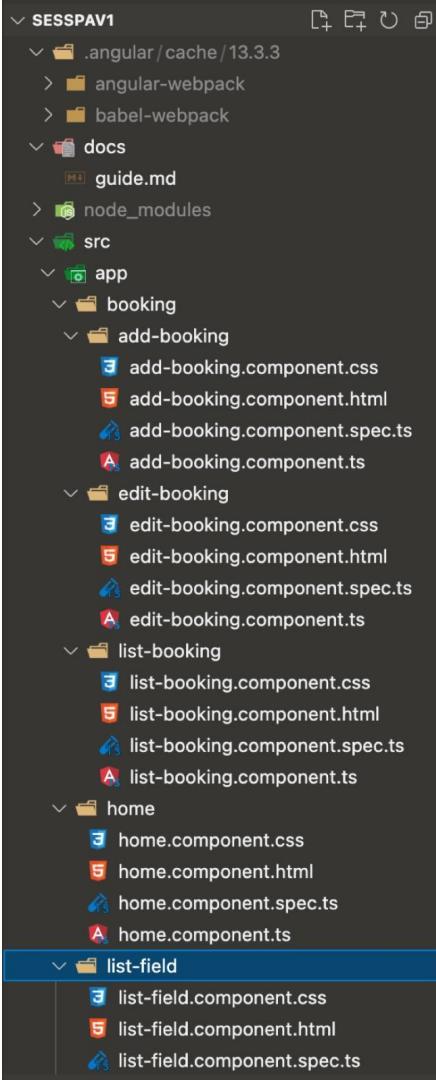


BACKEND

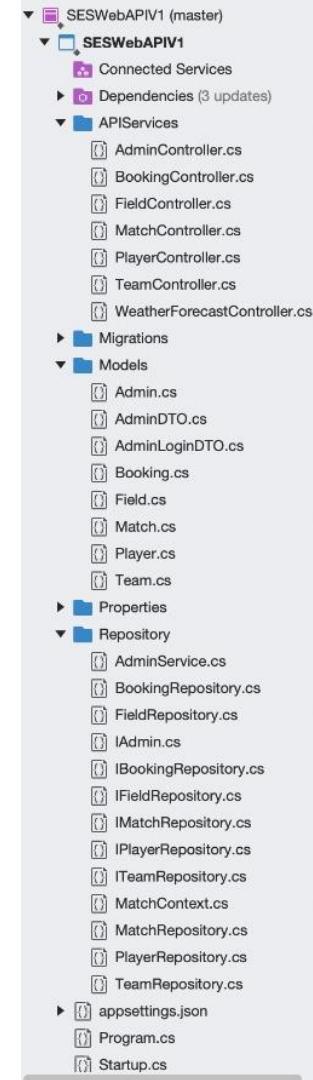
SESWebAPIApp Project

Layered

Architecture



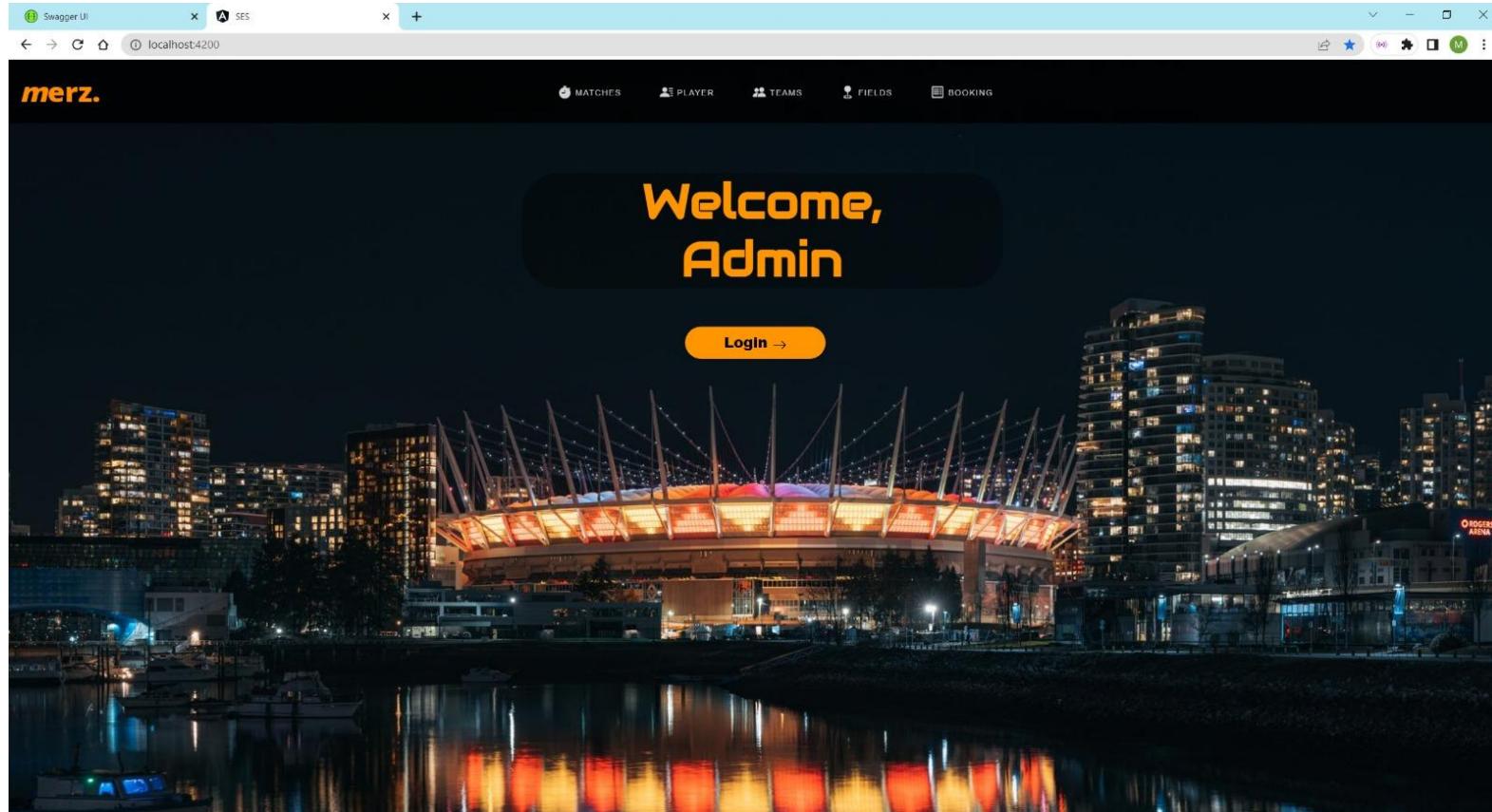
Front-End



Back-End

Running App Snapshots

Home Page



Players

The screenshot shows a web application interface for managing players. At the top, there is a navigation bar with tabs for 'MATCHES', 'PLAYER' (which is active), 'TEAMS', 'FIELDS', and 'BOOKING'. Below the navigation bar, a large card titled 'Player Details' contains a table of player information. The table has columns for '#', 'Full Name', 'Jersey Number', 'Field Position', and '#'. Two rows of data are visible:

#	Full Name	Jersey Number	Field Position	#
1	Hariss Harun	14	Midfielder	100
2	Thiago Silva	3	Defender	101

Each row has two yellow circular icons with black symbols: a trash can and a pencil.

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 X A SES +

localhost:4200/list-match

merz.

MATCHES PLAYER TEAMS FIELDS BOOKING

Match Fixtures

+ Match

#	Team A	Kick Off	Team B	Date	Location		
1	Brazil			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 Morumbi,
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

The screenshot shows a web application interface for booking. At the top, there is a navigation bar with tabs: 'Swagger UI', 'SES', and 'localhost:4200/list-booking'. Below the navigation bar, the main header features the word 'merz.' in orange. To the right of the header are five navigation links: 'MATCHES' (with a soccer ball icon), 'PLAYER' (with a person icon), 'TEAMS' (with a team icon), 'FIELDS' (with a field icon), and 'BOOKING' (with a calendar icon). The main content area has a title 'Booking Details' and a yellow button labeled '+ Booking'. Below this is a table with the following data:

#	Price	Quantity	Match Id	Total Price
1	\$100.00	2	1	\$200.00

Next to the last row are two icons: a trash can and a pencil.

Code Snapshots

Models

```
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; }
    }
}
```

```
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; }
    }
}
```

```
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; }
    }
}
```

```
using System.ComponentModel.DataAnnotations;
namespace SESWebAPIV1.Models
{
    public class Admin
    {
        [Key]
        public string Username { get; set; }
        public string Password { get; set; }
    }
}
```

```
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; }
    }
}
```

```
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; }
    }
}
```

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

```
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 ITTeamRepository<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 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 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 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://go.microsoft.com/fwlink/?LinkID=397860

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 https://go.microsoft.com/fwlink/?LinkID=397860

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]
        public async Task<ActionResult<Team>> Put(int id, Team Team)
        {
            var team = await _repository.Get(id);
            if (team != null)
            {
                _repository.Update(Team);
                return new OkResult();
            }
            else
            {
                return NotFound();
            }
        }
    }
}
```

Testing with mssql

The screenshot shows the Azure Data Studio interface with several tabs and queries running.

CONNEXIONS tab:

- Servers: System Databases, master, model, msdb, tempdb, BikeStores, DbCompany, dbEShopping, DbNewCompany, dbpizzastore, NewPizzaStore, PizzaStore, SESDB.
- Tables: dbo._EFMigrationsHistory, dbo.Admins, dbo.Bookings, dbo.Fields, dbo.Matches, dbo.Players, dbo.Teams, Views, Synonyms, Programmability, External Resources, Service Broker, Storage.

SQLQuery_1 - localh...DB (sa) tab:

```
use SESDB;
GO
SELECT * FROM PLAYERS;
SELECT * FROM TEAMS;
SELECT * FROM MATCHES;
SELECT * FROM bookings;
select * from fields;
select * from Admins;
```

localhost:SESDB tab:

```
24 use SESDB;
25 GO
26
27 SELECT * FROM PLAYERS;
28 SELECT * FROM TEAMS;
29 SELECT * FROM MATCHES;
30 SELECT * FROM bookings;
31 select * from fields;
32 select * from Admins;
```

Results tab:

	Id	FullName	JerseyNumber	FieldPosition	TeamId
1	1	Hariss Harun	14	Midfielder	100
2	2	Thiago Silva	3	Defender	101

Messages tab:

	TeamId	TeamName	TeamCaptain	TeamManager	TeamURL
1	100	Singapore	100	Nazri Nasir	https://cdn.countryflags.com
2	101	Brazil	101	Adenor Leonardo Bacchi	https://cdn.countryflags.com

AZURE tab:

	BookingId	Price	Quantity	MatchId	TotalPrice
1	1	100	2	1	200
2	2	10	8	109	80

	Username	Password
1	admin	admin123

SQL SERVER BIG DATA CLUSTERS tab:

	FieldId	FieldName	FieldCity	MatchId	FieldCapacity	FieldURL
1	1	Estadio do Morumbi	Sao Paulo	1	67052	https://www.stadiumguide.com
2	2	Lusail Stadium	Lusail	1	80000	https://www.stadiumguide.com

Welcome SQLQuery_1 - localh...DB (sa) ● localhost:SESDB

Run Cancel Disconnect Change Connection SESDB

```
49 select p.FullName, t.TeamName, p.fieldPosition, p.Id, P.TeamId
50 from Players p
51 JOIN Teams t
52 on p.TeamId = t.teamId;
```

Results Messages

	FullName	TeamName	fieldPosition	Id	TeamId
1	Hariss Harun	Singapore	Midfielder	1	100
2	Thiago Silva	Brazil	Defender	2	101

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

The screenshot shows the Postman application interface with two requests listed in the sidebar.

Request 1: GET http://localhost:5000/api/Booking

- Method: GET
- URL: http://localhost:5000/api/Booking
- Params tab is selected.
- Query Params table:

KEY	VALUE	DESCRIPTION
Key	Value	Description

- Body tab is selected.
- JSON response (Pretty):

```
1 [
2   {
3     "bookingId": 1,
4     "price": 100,
5     "quantity": 2,
6     "matchId": 1,
7     "totalPrice": 200
8   },
9   {
10     "bookingId": 2,
11     "price": 10,
12     "quantity": 8,
13     "matchId": 109,
14     "totalPrice": 80
15   }
16 ]
```

Request 2: GET http://localhost:5000/api/Field

- Method: GET
- URL: http://localhost:5000/api/Field
- Params tab is selected.
- Query Params table:

KEY	VALUE	DESCRIPTION
Key	Value	Description

- Body tab is selected.
- JSON response (Pretty):

```
1 [
2   {
3     "fieldId": 1,
4     "fieldName": "Estadio do Morumb",
5     "fieldCity": "Sao Paulo",
6     "matchId": 1,
7     "fieldCapacity": 67052,
8     "fieldURL": "https://www.stadiumguide.com/wp-content/uploads/alwakrah_top.jpg"
9   },
10  {
11    "fieldId": 2,
12    "fieldName": "Lusail Stadium",
13    "fieldCity": "Lusail",
14    "matchId": 1,
15    "fieldCapacity": 80000,
16    "fieldURL": "https://www.stadiumguide.com/wp-content/uploads/lusail2022-1.jpg"
17  },
18  {
19    "fieldId": 3,
20    "fieldName": "Al Bayt Stadium",
21    "fieldCity": "Al Khor",
22    "matchId": 1,
23    "fieldCapacity": 60000,
24    "fieldURL": "https://www.stadiumguide.com/wp-content/uploads/albayt_top.jpg"
25  }
26 ]
```

Agile Activates



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

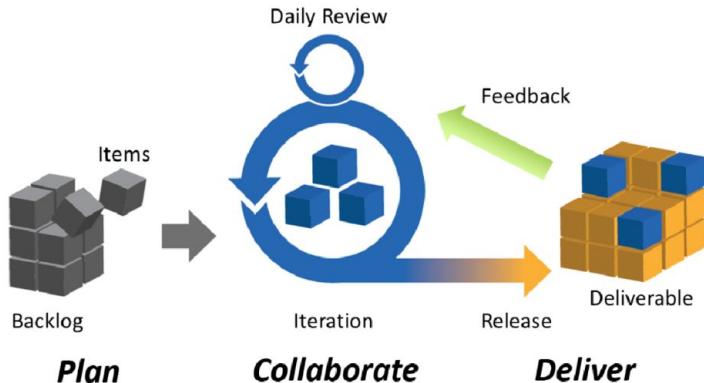
We actively learnt and applied our knowledge in whatever way we can in our project.

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

Trello Workspaces Recent Starred Templates Create Search

Board SportsEventSystems Workspace visible Share Power-Ups Aut

Product Backlog
The Product Backlog helps you estimate, refine, and prioritize everything you might sometime in the future want to complete.

Understanding the needs and requirements of end-users using Design Thinking

Website Responsiveness

Learning Agile Methodology, Agile principles and the benefits of Agile vs other SDLCs

Front-End specialised in Angular Framework to create single-page applications using TypeScript

Back-End specialised in ASP.NET Core, Entity-Framework Frameworks to create back-end services using RESTful APIs

Requirements, functionalities and needs of users specifications acquired

Sprint Backlog
The sprint backlog is a list of tasks identified by the Scrum team to be completed during the Scrum sprint. During the sprint planning meeting, the team selects some number of product backlog items, usually in the form of user stories, and identifies the tasks necessary to complete each user story.

+ Add a card

In Progress
In agile development, work in progress (WIP) limits set the maximum amount of work that can exist in each status of a workflow. Limiting the amount of work in progress makes it easier to identify inefficiency in a team's workflow.

Delete and edit of booking

Modify UML Diagrams

Add more attributes to classes (BE)

+ Add a card

Sprint Complete
Sprint Completion is the other Sprint metric often considered in conjunction with Sprint Target Completion. It is important as it shows the overall ability of the Scrum Team to hit their Sprint goals having taken into account work added after the Sprint has started.

1 Apr 2021

Back-End
Create Models, APIServices, Repositories
Started: 13 Apr

44/44

Diagram
UML

Diagram
Sequence Diagram

+ Add a card

Learning & Experiences

- ▶ Learning to build a small application in a business context from the scratch was fulfilling and help us further understand 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 emphasized 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 got 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

