|  |  |
| --- | --- |
|  | UST  **Sports Event Management System** |
|  |
|  |

**Table of Contents**

Contents

[1.0 Important Instructions 2](#_Toc104980813)

[2.0 Introduction 2](#_Toc104980814)

[2.1 Purpose of this document 3](#_Toc104980815)

[2.2 Project Overview 3](#_Toc104980816)

[2.3 Scope 3](#_Toc104980817)

[2.4 Hardware and Software Requirement 4](#_Toc104980818)

[2.5 System Architecture Diagram 4](#_Toc104980819)

[2.6 Functional Requirements and High Level Use case Diagram 5](#_Toc104980820)

[3.0 System Requirements 6](#_Toc104980821)

[3.1.1 Sports Microservice 6](#_Toc104980822)

[3.1.2 Functional Requirements – Player Microservice 7](#_Toc104980823)

[3.1.3 Functional Requirements – Participation Microservice 8](#_Toc104980824)

[3.1.4 Functional Requirements – Authorization Microservice 9](#_Toc104980825)

[3.1.5 Functional Requirements – Sports event management portal 9](#_Toc104980826)

[3.1.6 Swagger 9](#_Toc104980827)

[4.0 Cloud Deployment requirements 10](#_Toc104980828)

[5.0 Design Considerations 10](#_Toc104980829)

[6.0 Reference learning 10](#_Toc104980830)

[7.0 Change Log 11](#_Toc104980831)

# Important Instructions

1. Associate must adhere to the Design Considerations specific to each Technolgy Track.
2. Associate must not submit project with compile-time or build-time errors.
3. Being a Full-Stack Developer Project, you must focus on ALL layers of the application development.
4. Unit Testing is Mandatory, and we expect a code coverage of 100%. Use Unit testing and Mocking Frameworks wherever applicable.
5. All the Microservices, Client Application, DB Scripts, have to be packaged together in a single ZIP file. Associate must submit the solution file in ZIP format only.
6. If backend has to be set up manually, appropriate DB scripts have to be provided along with the solution ZIP file.
7. A READ ME has to be provided with steps to execute the submitted solution, the Launch URLs of the Microservices in cloud must be specified.

(Importantly, the READ ME should contain the steps to execute DB scripts, the LAUNCH URL of the application)

1. Follow coding best practices while implementing the solution. Use appropriate design patterns wherever applicable.
2. You are supposed to use an In-memory database or code level data as specified, for the Microservices that should be deployed in cloud. No Physical database is suggested for Microservice.

# Introduction

The project is a Sports Event Management Portal that manages the creation and management of a sports event. It makes it easier to maintain the different aspects of a sports event.

## Purpose of this document

The purpose of the software requirement document is to systematically capture requirements for the project and the system “Sports Management System” that has to be developed. Both functional and non-functional requirements are captured in this document. It also serves as the input for the project scoping.

The scope of this document is limited to addressing the requirements from a user, quality, and non-functional perspective.

High Level Design considerations are also specified wherever applicable, however the detailed design considerations have to be strictly adhered to during implementation.

## Project Overview

The project is a Sports Event Management Portal that manages the creation and management of a sports event. It makes it easier to maintain the different aspects of a sports event. The portal provides various functionalities like creating events for various sports, request for participation, etc.

## Scope

Below are the modules that needs to be developed part of the Project:

|  |  |  |
| --- | --- | --- |
| **Req. No.** | **Req. Name** | **Req. Description** |
| REQ\_01 | Sports Event Service | This Module is a Middleware Microservice that performs following operations:   * View all the Sports * View sports by name * Create and cancel the events (based on sports name) * View all Events and search event by name |
| REQ\_02 | Player Service | This Module is a Middleware Microservice that performs the following operations:   * Add the player (based on sports name) * Remove the players * View all the players |
| REQ\_03 | Participation Service | This Module is a Middleware Microservice that performs the following operations by using the sports event service and player service.   * Request for the participation on behalf of players for the particular event. * View the participation request * Update the participation status as per the event requirement * List the participation request based on approved/ declined /pending status separately. |
| REQ\_04 | Authorization service | This microservice is used with anonymous access to Generate JWT |
| REQ\_05 | Sports Event Management portal | A Web Portal that allows a member to Login and allows to do following operations:   * Login * Add / remove the player * Add / remove the Event * View all the Players * View all the Events * Request for the participation * View all the participation requests then Approve or decline the request * View the Approved / Declined /pending status |

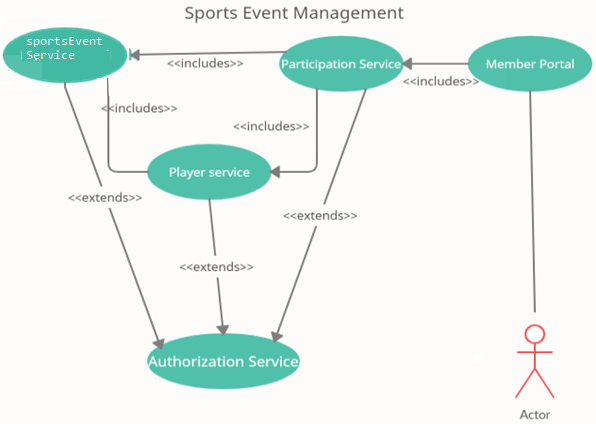
## Hardware and Software Requirement

1. Hardware Requirement:
2. Developer Desktop PC with 8GB RAM
3. Software Requirement (Java)
4. Spring Tool Suite (STS) Or any Latest Eclipse
5. Have PMD Plugin, EclEmma Code Coverage Plugin and AWS Code Commit Enabled
6. Configure Maven in Eclipse
7. Maven
8. Docker (Optional)
9. Postman Client in Chrome
10. Software Requirement (Dotnet)
11. Visual studio 2017 enterprise edition
12. SQL Server 2014
13. Postman Client in Chrome /Swagger client
14. Azure/AWS cloud access

## System Architecture Diagram



## Functional Requirements and High Level Use case Diagram



# System Requirements

### Sports Microservice

|  |  |
| --- | --- |
| **Sports Event Management system** | **Sports Event Microservice** |
| **Functional Requirements**  The intent of this Microservice is to provide the   * View all the Sports * View sports by name * Create, update and cancel the events (based on sports name) * View all Events and search event by id/name | |
| **Entities**    **Sports**     1. **sportsId** 2. **noOfPlayers** 3. **sportsName** 4. **sportsType**   **Event**     1. eventId 2. eventDate 3. eventName 4. noofSlots 5. sportsName       **REST End Points**  **Sports Event Microservice**   * GET: / sports(Input: None| Output: ListAllSportsDetails) * GET:/getSportsByName(Input:sports name|Output: Sports) * POST: /addEvent (Input: Event object | Output: Sports Event +” added successfully”) * DELETE: /deleteEvent (Input: Event id which is already exists | Output: Event Object +” deleted successfully”) * GET: / events(Input: None| Output: ListAllEvents) * GET: / eventsByName(Input: eventName| Output: Event) | |
|  | |
| **Steps and Actions**   1. Lists the sports services for it to be shown on the User interface  |  |  |  |  | | --- | --- | --- | --- | | **Sports Id** | **No.of. Players** | **Sports Name** | **Sports Type** | | 1001 | 2 | Chess | Indoor | | 1002 | 2 | Carrom | Indoor | | 1003 | 22 | Cricket | Outdoor | | 1004 | 22 | Hockey | Outdoor |      1. List the sports based on sports name. 2. Create the event based on sports name   The web application provides input to  for a new in event service request, get the   * Event name, event date, no.of.slots and sports name.  1. Remove the event based on eventId 2. List the all events 3. Search the event by its name | |
| **Non-Functional Requirement:**   * Only Authorized requests can access these REST End Points | |

### Functional Requirements – Player Microservice

|  |  |
| --- | --- |
| **Sports Event Management System** | **Player Microservice** |
| **Functional Requirements**  This microservice should be invoked from the Web application. It allows the following operations:   The web application provides input to   * For a new in-player service request, get the   + Player detail – Name, Age ,gender, contact number ,email and sports name * Remove the players based on player id. * View all the players | |
| **Entity**    **Player**     1. Playerid 2. **playerName** 3. **age** 4. **contact number** 5. **email** 6. **gender** 7. **sportsname**     **REST End Points**  **Player Microservice**  **Sports Player Microservice**   * POST: /addPlayer (Input: Player object | Output: Player +” added successfully”) * DELETE: /deletePlayer (Input: player id which is already exists | Output: Player Object +” removed successfully”) * GET: / players(Input: None| Output: ListAllPlayers) | |
| **Trigger** – Can be invoked from Sports event management web portal | |
| **Steps and Actions**   * This microservice will be invoked from the web portal with the JWT obtained from the Authentication service. * Player detail along with sports name should be saved in database. * Player detail can be removed from the database. * View all the players | |
| **Non-Functional Requirement:**  Only Authorized requests can access these REST End Points | |

### Functional Requirements – Participation Microservice

|  |  |
| --- | --- |
| **Sports Event Management system** | **Participation Microservice** |
| **Functional Requirements**  The intent of this Microservice is to provide ,   * Request for the participation * View the participation request * Update the participation status * List the participation request based on approved/ declined /pending status separately. | |
| **Entities**    **Participation**   * 1. Participation\_id   2. Player\_id   3. Player\_name   4. Event\_id   5. Event\_name   6. Sports\_id   7. Sports\_name   8. Status   **REST End Points**  **Participation Microservice**   * GET: /addparticipation (Input: participation | Output: participation object +”added successfully”) * GET: /getParticipations (Input: None| Output: List all the Participations) * PUT: /updateStatus (Input: participationId | Output: Approved/declined successfully) * GET: /getApprovedParticipations (Input: approved status| Output: List all the Approved Participations) * GET: /getDeclinedParticipations (Input: declined status| Output: List all the Declined Participations) * GET: /getPendingParticipations (Input: pending status| Output: List all the Pending Participations) | |
| **Trigger** – Can be invoked from sports event management Web portal | |
| **Steps and Actions**   * This microservice allows admin to add the participation request on behalf of the players. * This microservice should be invoked to get all the participation requests. * The microservice allows admin to approve/decline based on the event requirement. * Admin view the separate list of approved / declined /pending participation requests. | |
| **Non-Functional Requirement:**  Only Authorized requests can access these REST End Points | |

### Functional Requirements – Authorization Microservice

|  |  |
| --- | --- |
| **Sports Event Management system** | **Authorization Microservice** |
| **Security Requirements**   * Create JWT * Have the token expired after specific amount of time say 30 minutes * Has anonymous access to get the token detail | |

### Functional Requirements – Sports event management portal

|  |  |
| --- | --- |
| **Sports Event Management system** | **Sports event management Portal** |
| **Client Portal Requirements**   * Sports Event Management Portal  must allow the admin to Login. Once successfully logged in, the member do the following operations: * View the list of Sports and Sports by name. * Add / remove the players. * Add / remove the Event based on the business requirements * View all the Players * View all the Events * Request for the participation on behalf of players for the particular event. * View all the participation requests then approve or decline the request as per the business requirement. * View the Approved / Declined /pending status separately * Each of the above operations will reach out to the middleware Microservices that are hosted in cloud. | |

### Swagger

|  |  |
| --- | --- |
| **Sports Event Management** | **Swagger** |
| **Documentation Requirements (Java)**   * All the Microservices must be configured with Swagger for documentation * Register the swagger resources in the Swagger Microservice and enable them as REST end points * Configure this service along with other services in the cloud | |

# Cloud Deployment requirements

* All the Microservices must be deployed in Cloud
* All the Microservices must be independently deployable. They have to use In-memory database or data in the application wherever applicable
* The Microservices has to be dockerized and these containers must be hosted in Cloud using CI/CD pipelines
* The containers have to be orchestrated using AWS/Azure Kubernetes Services.
* These services must be consumed from an MVC app running in a local environment.

# Design Considerations

Java and Dotnet specific design considerations are attached here. These design specifications, technology features have to be strictly adhered to.



# Reference learning

Please go through all of these k-point videos for

Microservices deployment into Azure Kubernetes Service.

|  |
| --- |
| [AzureWithCICD-1](https://cognizant.kpoint.com/app/video/gcc-19532393-d4e0-4fd9-8a0c-80ecbdb349d3) |
| [AzureWithCICD-2](https://cognizant.kpoint.com/app/video/gcc-6633a958-ab72-4c69-b926-fe832e4b56a1) |
| [AzureWithCICD-3](https://cognizant.kpoint.com/app/video/gcc-553eb186-c1cf-448e-96fc-a96fe37b2e6a) |
| [AzureWithCICD-4](https://cognizant.kpoint.com/app/video/gcc-fad7d4af-d651-4501-99c6-2785190670c2) |

**Other References:**

|  |  |
| --- | --- |
| Java 8 Parallel Programming | <https://dzone.com/articles/parallel-and-asynchronous-programming-in-java-8> |
| Feign client | [https://dzone.com/articles/Microservices-communication-feign-as-rest-client](https://dzone.com/articles/microservices-communication-feign-as-rest-client) |
| Swagger | [https://dzone.com/articles/centralized-documentation-in-Microservice-spring-b](https://dzone.com/articles/centralized-documentation-in-microservice-spring-b) |
| ECL Emma Code Coverage | <https://www.eclipse.org/community/eclipse_newsletter/2015/august/article1.php> |
| Lombok Logging | <https://javabydeveloper.com/lombok-slf4j-examples/> |
| Spring Security | <https://dzone.com/articles/spring-boot-security-json-web-tokenjwt-hello-world> |
| H2 In-memory Database | <https://dzone.com/articles/spring-data-jpa-with-an-embedded-database-and-spring-boot>  <https://www.baeldung.com/spring-boot-h2-database> |
| AppInsights logging | <https://www.codeproject.com/Tips/1044948/Logging-with-ApplicationInsights> |
| Error response in WebApi | <https://stackoverflow.com/questions/10732644/best-practice-to-return-errors-in-asp-net-web-api> |
| Read content from CSV | <https://stackoverflow.com/questions/26790477/read-csv-to-list-of-objects> |
| Access app settings key from appSettings.json in .Netcore application | <https://www.c-sharpcorner.com/article/reading-values-from-appsettings-json-in-asp-net-core/>    <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/configuration/?view=aspnetcore-3.1> |