**Technical Design Document**

**Poker Game Checker Microservice**

***Table of Contents***

*1. Introduction*

*2. Architecture*

*3. Project Structure*

*4. API Documentation*

*5. Class Descriptions*

*6. Dependencies*

*7. Error Handling*

*8. Swagger Integration*

*9. Testing*

*10. Conclusion*

**1. Introduction**

The Poker Game Checker Microservice is an API that allows users to determine the winner of a Poker game. The API accepts Poker game data, including the number of players and their card hands, and calculates the winner based on Poker hand rankings.

**2. Architecture**

The microservice follows a simple three-tier architecture:

Presentation Layer: Handled by the `PokerGameController`, which serves as the API endpoint.

Business Logic Layer: Implemented by the `Poker` class, responsible for calculating the winner.

Data Access Layer: There is no direct data access, as the microservice does not store data persistently.

**3. Project Structure**

Controllers: Contains the `PokerGameController` for API endpoints.

Models: Contains the core data models: `Poker`, `PokerDeck`, and `PokerPlayer`.

Interfaces: Contains the interfaces for the data models.

Constants : Contains CardConstants, that is used as a static class for card ranking/suits and calculation of ranks for given cards

**4. API Documentation**

API documentation is provided through Swagger. The `PokerGameController` and its endpoints are documented, including input parameters and expected responses.

[XML Documentation](https://github.com/sudeepcb/CardGamesCheckerAPI/blob/master/PokerGameCheckerMicroservice%20-%20Documentation/PokerGameCheckerMicroservice.xml)

**5. Class Descriptions**

PokerGameController: API controller that exposes an endpoint to calculate the winner of a Poker game.

Poker: Represents the core class representing the Poker game, including determining the winner.

PokerDeck: Represents a deck of cards for the game, including associated players and the total number of cards.

PokerPlayer: Represents a player in the game, storing their name, cards in hand, and card rank.

CardConstants: A static class that defines constants and methods for working with poker cards, such as card ranking/suits and calculations of card rank.

**6. Dependencies**

The project uses the following dependencies:

ASP.NET Core API for building the API.

Swashbuckle.AspNetCore for Swagger documentation.

XUnit for Testing

Interfaces for defining data model contracts.

Static class for constants for working with poker cards

**7. Error Handling**

Error handling is performed for various cases, such as invalid input data and insufficient players. Detailed exception messages are returned in the API responses to aid in debugging.

Ex:

**if (\_decksInHand.Player.Count != \_players)**

**{**

**throw new ArgumentNullException("Total Players does not match Decks of Players");**

**}**

**8. Swagger Integration**

Swagger is integrated for API documentation. It provides a user-friendly interface to interact with the API, and understand the input and output data models.

**9. Testing**

For now testing has been done for main functionally but more tests related to during the process of creating models, as well as testing different test cases should be performed in the future.

**Tests:**

PokerGameChecker\_CalculateWinner\_ReturnsIActionResult()

CardConstants\_CheckIfCardRankingIsCorrectForGame\_ReturnsRankWithGivenCards()

**10. To Do**

Add Authentication Layer, for Access Tokens enablement in API Controller using AzureAD

Integrate API Manager from Azure as a gateway to API’s in the Project, as well as enabling secure access with Access Tokens with API Permission, integration with AzureAD

With more time, make use of Interfaces, to allow for extensibility to additional addition on card games, like BlackJack, GoldFish, and etc.

Integrate Azure Web App Services / Azure Kubernetes Service for use with Azure Container Registry to deploy web services to the Web, and allow only approved requests using CORS.

Implement UI Layer to interact with API, and setting up History Databases via Microservices Layer for User Information from UI Layer, as well as allow for Visual Capture of Deck of Cards into acceptable parameters, using AI (Setup microservices for User Information/History Purposes), with Database Instances for each purpose.

**11. Technical Diagram**

[**Diagram**](https://github.com/sudeepcb/CardGamesCheckerAPI/blob/master/PokerGameCheckerMicroservice%20-%20Documentation/Technical%20Diagram%20-%20PokerGameCheckerMicroservice.pdf)

**12. Conclusion**

The Poker Game Checker Microservice is a simple yet effective API for determining the winner in a Poker game. It follows a clear architecture, provides error handling, and is well-documented through Swagger. Future improvements may include data storage, authentication, and additional features for a more comprehensive Poker game application.

This technical design document serves as an overview of the project, its structure, and key components, helping developers understand and maintain the codebase.