MatchMaking (Test Task)

Overview

For a certain game, a solution is needed to group players into matches.
Requirements:

1. Users must be able to make a request to search for a new match.Request rate: max 1 request per 100 milliseconds.
2. Users must be able to retrieve information about the match formed for their last successful search request.

Technical Description

Two .NET 9 solutions must be implemented:

1. MatchMaking.Service
2. MatchMaking.Worker

MatchMaking.Service

Overview

This service provides an HTTP API for everything related to MatchMaking.

The service should have two HTTP endpoints:

1. Match Search Request
2. Retrieve Match Information

Match Search Request

Request:

Parameters:

1. userId: string

Response:

HTTP Status Codes:

1. 204
2. Optional: 400

Retrieve Match Information

Request:

Parameters:

1. userId: string

Response:

Body:

```
{
  "matchId": "45ae548e-d72f-438d-bf1a-f1692a699a81",
  "userIds": ["example_user_id_1", "example_user_id_2", "example_user_id_3"]
}
```

HTTP Status Codes:

1. 200
2. 404
3. Optional: 400

MatchMaking.Worker

Overview

The worker communicates with **MatchMaking.Service** via Kafka.
Workflow:

1. The worker consumes the **matchmaking.request** topic (messages containing userId).

2. When the number of players is sufficient to start a match, a message is sent back to Kafka on the **matchmaking.complete** topic with userIds and a matchId (a generated GUID string).
3. The **matchmaking.complete** topic is consumed by MatchMaking.Service.

Configuration:

1. The number of users required per match is stored in the MatchMaking.Worker configuration (e.g., appsettings.json). Default 3.

General Solution

Overview

The solution must be submitted as a Pull Request on GitHub, in a new repository, from the **feature/initial** branch to the **main** branch. The repository should include a **README.md** file with instructions on how to run the solution.

Infrastructure Setup

The entire infrastructure must be launched using **docker compose** and include the following components:

1. MatchMaking.Service (1 instance)
2. MatchMaking.Worker (2 instances)
3. Kafka
4. Redis (used as a database for the worker and service, if needed)

For all technical and business-related questions (including partial implementation if it's not possible to complete the task within the given timeframe), reach out to the specialist who assigned this task.

Code style requirements (optional)

1. Use records for POCO
2. csproj:
    a. <WarningsAsErrors>Nullable</WarningsAsErrors>
    b. <ImplicitUsings>enable</ImplicitUsings>
    c. <Nullable>enable</Nullable>
3. Logs for failed operations
4. Some logs for successful operations (e.g. match making completed)