포트폴리오 #1

사용자 매칭 (matching) 시스템

개요

- 매칭 시스템은 사용자 간의 매칭을 자동화하고 관리하는 시스템입니다.
- 이 시스템은 실시간 웹소켓을 통해 사용자와 상호작용하며, Redis를 사용하여 빠르고 효율적인 데이터 처리를 수행합니다.
- 주요 기능으로는 사용자 인증, 매칭, 재시도 메커니즘 등이 포함됩니다.

주요 기능

- 사용자 인증
  - 사용자는 Restful API 통해 로그인하며, 로그인 시 입력한 이름을 기준으로 신규 사용자 여부를 확인합니다.
  - 등록되지 않은 사용자는 자동으로 등록되며, 1에서 1000 사이의 무작위 점수가 부여됩니다.
  - 인증된 사용자에게는 세션을 유지할 수 있는 JWT 토큰이 발급되며, 이 토큰을 통해 웹소켓 연결 시 사용자가 인증된 상태임을 확인합니다. 이를 통해 인증된 사용자만 매칭 시스템을 이용할 수 있습니다.
- 매칭
  - 매칭 시스템은 실시간 웹소켓을 통해 사용자와 상호작용하며 매칭 요청을 처리합니다.
  - 매칭 요청 시, 매칭 대기열에 있는 다른 사용자를 탐색해 매칭을 시도합니다.
  - 대기열에 적합한 사용자가 없을 경우 일정 시간 동안 대기 후 다시 탐색하며, 대기 중인 사용자가 여러 명인 경우 무작위로 상대방을 선정합니다.
  - 매칭에 성공한 경우, 양측 사용자에게 매칭 결과를 즉시 반환합니다.
- 재시도 메커니즘
- 사용자가 매칭에 실패하면 매칭 서비스는 일정 시간 후 자동으로 재시도를 진행합니다.
- 최대 3회까지 재시도하며, 모든 시도가 실패하면 봇 매칭으로 전환됩니다.
- 봇 매칭은 데이터베이스에 저장된 사용자의 점수를 기준으로 적합한 봇을 검색해 매칭을 시도하며, 초기 점수 범위는 ±50으로 설정됩니다. 이후 범위를 늘려가며 최대 3회까지 시도해 매칭이 성사될 가능성을 높입니다.

- 매칭이 완료되면 결과를 사용자에게 실시간으로 전달하며, 매칭 실패 시에도 즉각적인 피드백을 제공합니다.

기능 상세

- **로그인**
  - 사용자는 로그인 시 사용자 이름을 입력해 접속합니다.
  - 신규 사용자는 자동 등록되며, 1에서 1000 사이의 무작위 점수가 부여됩니다.
  - 로그인 완료 시 서버는 접속 토큰과 사용자 정보를 반환합니다.
- **매칭 요청 및 검증**
  - 사용자는 웹소켓을 통해 매칭을 요청합니다.
  - **토큰 검증**: 사용자의 토큰을 검증한 뒤 매칭 대기 상태로 전환합니다.
  - **대기열 등록**: 검증이 완료된 사용자는 매칭 대기열에 등록됩니다.
- **매칭 프로세스**
  - **실제 사용자와의 매칭**:
    - 대기열에 있는 다른 사용자를 탐색하여 매칭을 시도합니다.
    - 대기 중인 사용자가 여러 명인 경우, 무작위로 상대방을 선택해 매칭합니다.
    - 매칭 성공 시, 양측 사용자에게 매칭 결과를 반환합니다.
    - 매칭 실패 시 3초간 대기 후 다시 탐색을 시작합니다.
    - **최대 3회 재시도**: 최대 3회까지 매칭이 실패할 경우 봇 매칭으로 전환됩니다.
  - **봇 매칭 전환**:
    - **점수 기반 봇 검색**: 봇 매칭으로 전환되면 사용자의 점수를 기준으로 봇을 검색합니다.
    - 검색 범위는 사용자 점수 ±50으로 시작하며, 매칭 실패 시 ±50씩 확장해 최대 3회까지 추가 검색합니다.
    - 봇 매칭이 성사되면 봇 정보를 포함한 결과를 사용자에게 반환합니다.
- **최종 매칭 결과 반환**
  - 사용자 간 매칭이 성사되면 양측에 상대 정보를 포함한 성공 결과를 반환합니다.
  - 봇 매칭이 성사되면 봇 정보를 포함한 성공 결과를 반환합니다.
  - 매칭이 최종적으로 실패하면 "매칭 실패" 상태를 사용자에게 반환합니다.

기술 스택

- **백엔드: NestJS**

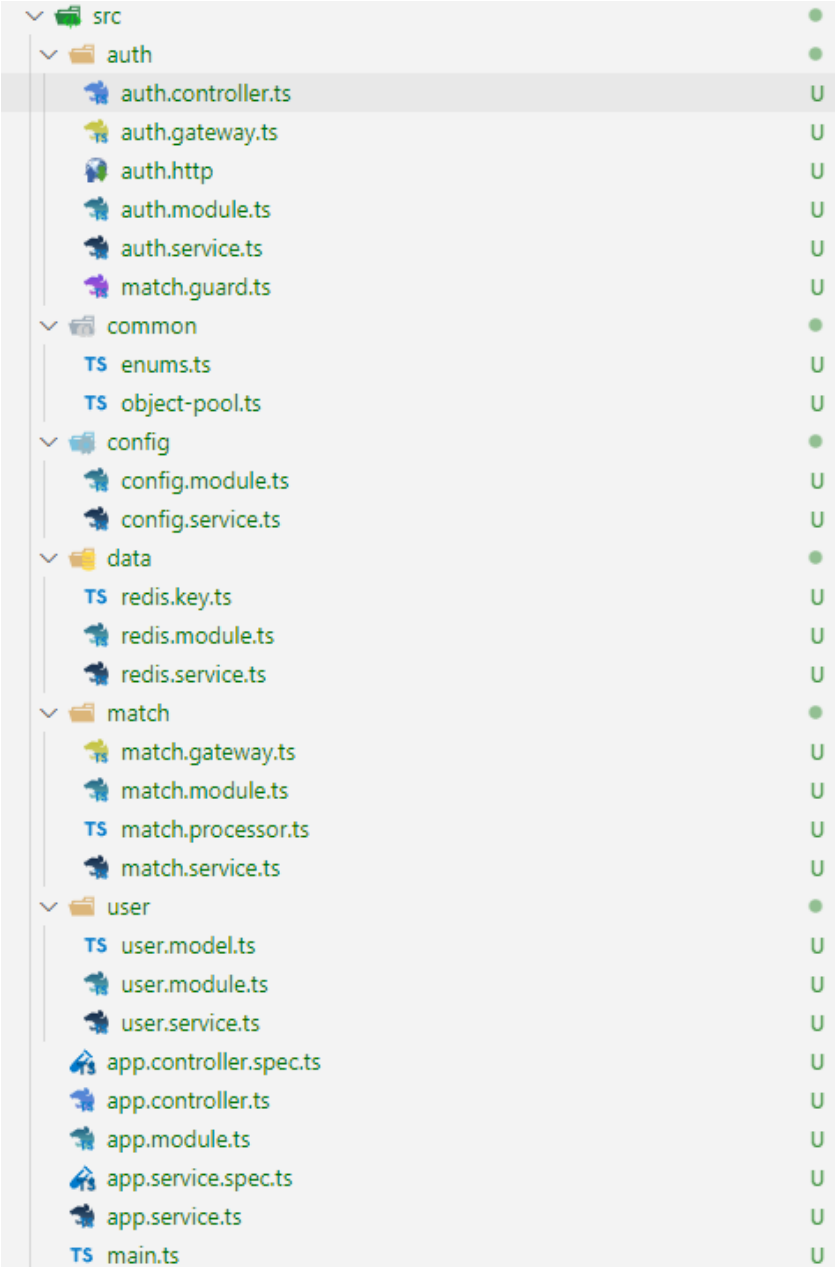
    - 애플리케이션의 핵심 로직을 처리하고 REST API, 웹소켓 게이트웨이, 서비스, 컨트롤러 등을 구현하여 높은 확장성과 유지보수성을 제공합니다.
- **데이터베이스: Redis**
    - 사용자 세션 관리, 매칭 대기 큐, 실시간 데이터 저장 및 조회에 사용되며, 빠른 데이터 접근과 저장이 필요한 실시간 매칭 시스템에 적합합니다.
- **실시간 통신: Socket.IO**
    - 사용자와 서버 간의 실시간 통신을 처리하여 매칭 상태 업데이트와 알림을 즉시 전송합니다.
- **테스트: Jest**
    - 애플리케이션의 단위 테스트 및 통합 테스트를 작성하고 실행하여 코드 안정성을 유지하고, 주요 로직이 올바르게 동작하는지 검증합니다.
- **코드 스타일: ESLint, Prettier**
    - 린팅과 코드 포맷팅을 통해 일관된 코드 스타일을 유지하고, 코드 품질을 개선하여 가독성을 높입니다.
- **외부 패키지**
    - **사용자 정보 관리 풀:** `generic-pool`
        - 사용자 객체의 효율적인 관리를 위한 풀링 메커니즘을 제공하여, 자원을 효과적으로 사용하고 성능을 최적화합니다.
    - **매칭 대기 큐:** `bull`
        - 매칭 요청을 큐에 추가하여 순차적으로 처리하고, 매칭 로직을 효율적으로 관리하여 사용자 간의 매칭을 원활하게 진행합니다.
    - **Redis 클라이언트:** `ioredis`
        - Redis 데이터베이스와의 상호작용을 관리하며, 다양한 Redis 기능을 지원해 안정적인 데이터 관리를 돕습니다.
    - **환경 설정 데이터 양식:** `yaml`
        - 애플리케이션의 설정 데이터를 YAML 형식으로 관리하여 가독성과 설정 변경의 용이성을 제공합니다.

구조

- 디렉토리 구조
    - **auth**
        - 사용자 접속 및 인증 관리를 위한 로직을 포함하여, 로그인 및 토큰 검증을 처리합니다.
    - **user**
        - 사용자 정보 및 활동과 관련된 로직을 관리하여, 사용자 세부 정보와 프로필 업데이트 등과 같은 기능을 포함합니다.

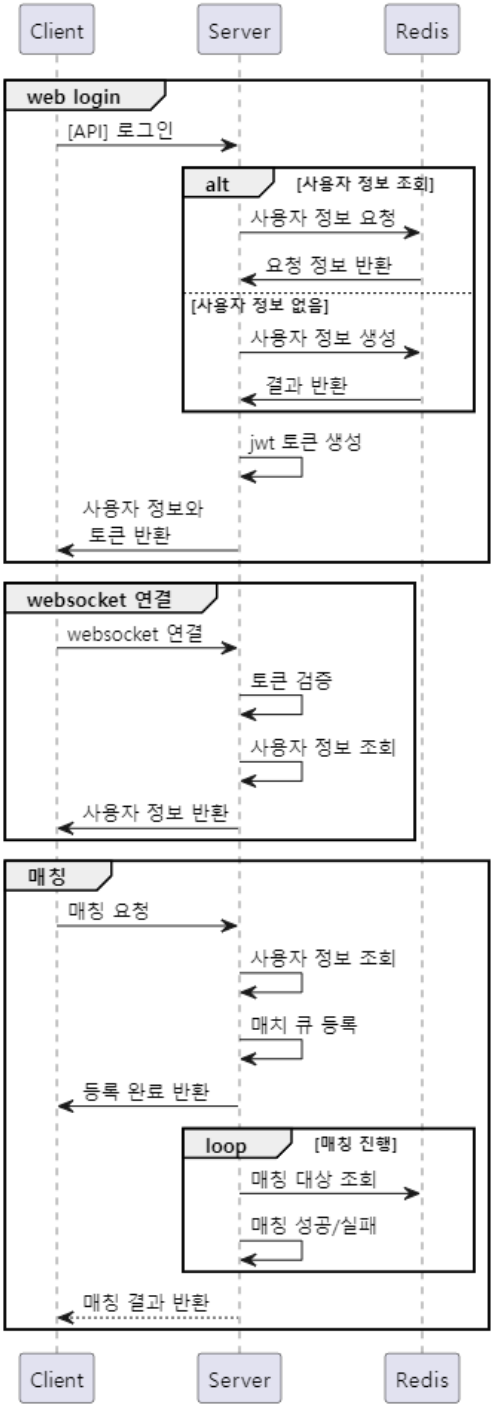
    - **match**
        - 유저 간 또는 봇과의 매칭 로직을 처리하여 매칭의 전체 흐름을 관리합니다.
    - **data**
        - 데이터베이스와 관련된 코드를 모아 관리하며, 데이터베이스 모델과 연결 설정이 포함됩니다.
    - **common**
        - 공통 유틸리티와 서비스를 제공하는 디렉토리로, 여러 모듈에서 재사용 가능한 코드와 기능을 포함합니다.
    - **config**
        - 애플리케이션의 환경 설정 및 구성 관련 코드로, 설정 파일 및 초기화 로직을 관리합니다.

- 파일 구조

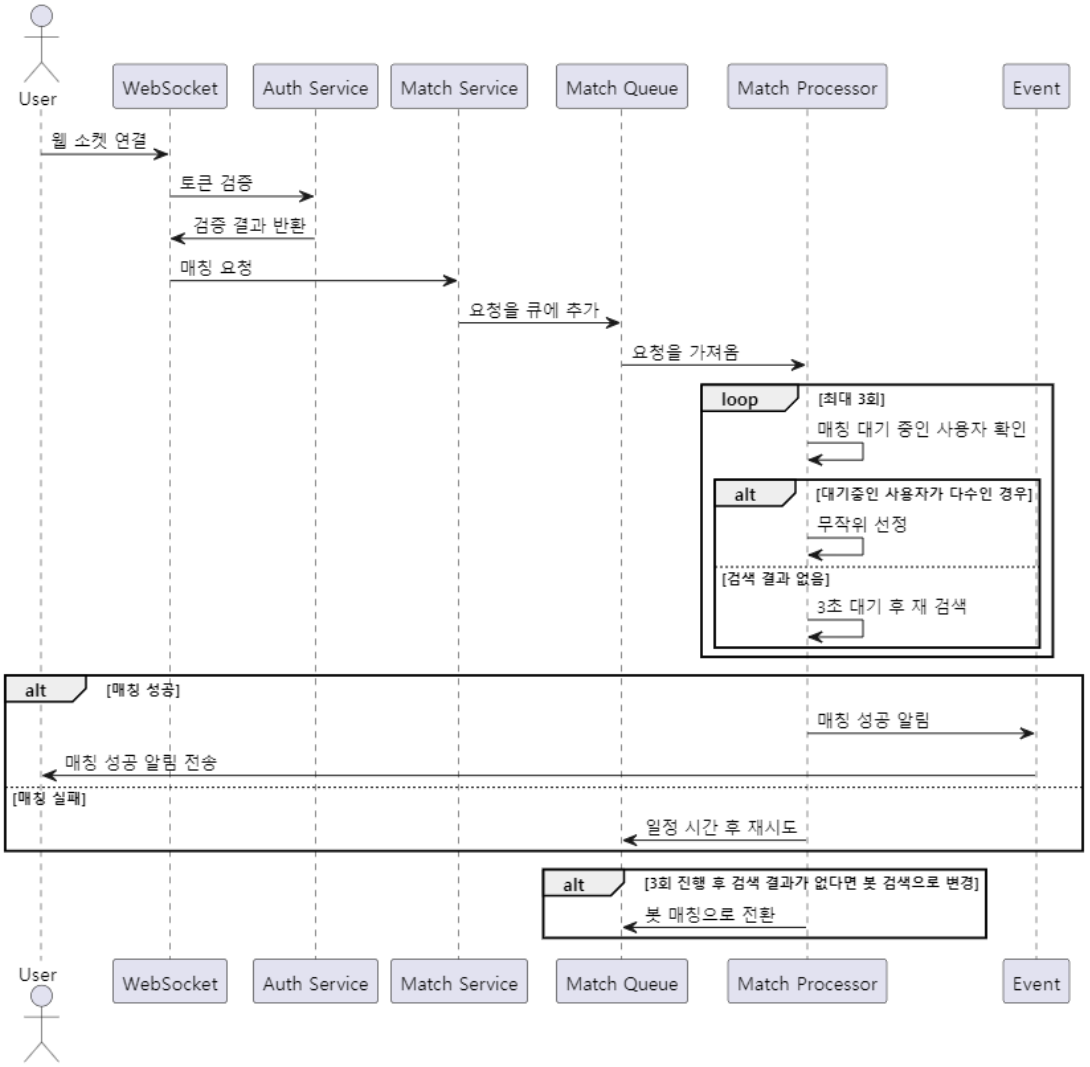

src
auth
auth.controller.ts U
auth.gateway.ts U
auth.http U
auth.module.ts U
auth.service.ts U
match.guard.ts U
common
enums.ts U
object-pool.ts U
config
config.module.ts U
config.service.ts U
data
redis.key.ts U
redis.module.ts U
redis.service.ts U
match
match.gateway.ts U
match.module.ts U
match.processor.ts U
match.service.ts U
user
user.model.ts U
user.module.ts U
user.service.ts U
app.controller.spec.ts U
app.controller.ts U
app.module.ts U
app.service.spec.ts U
app.service.ts U
main.ts U

시퀀스

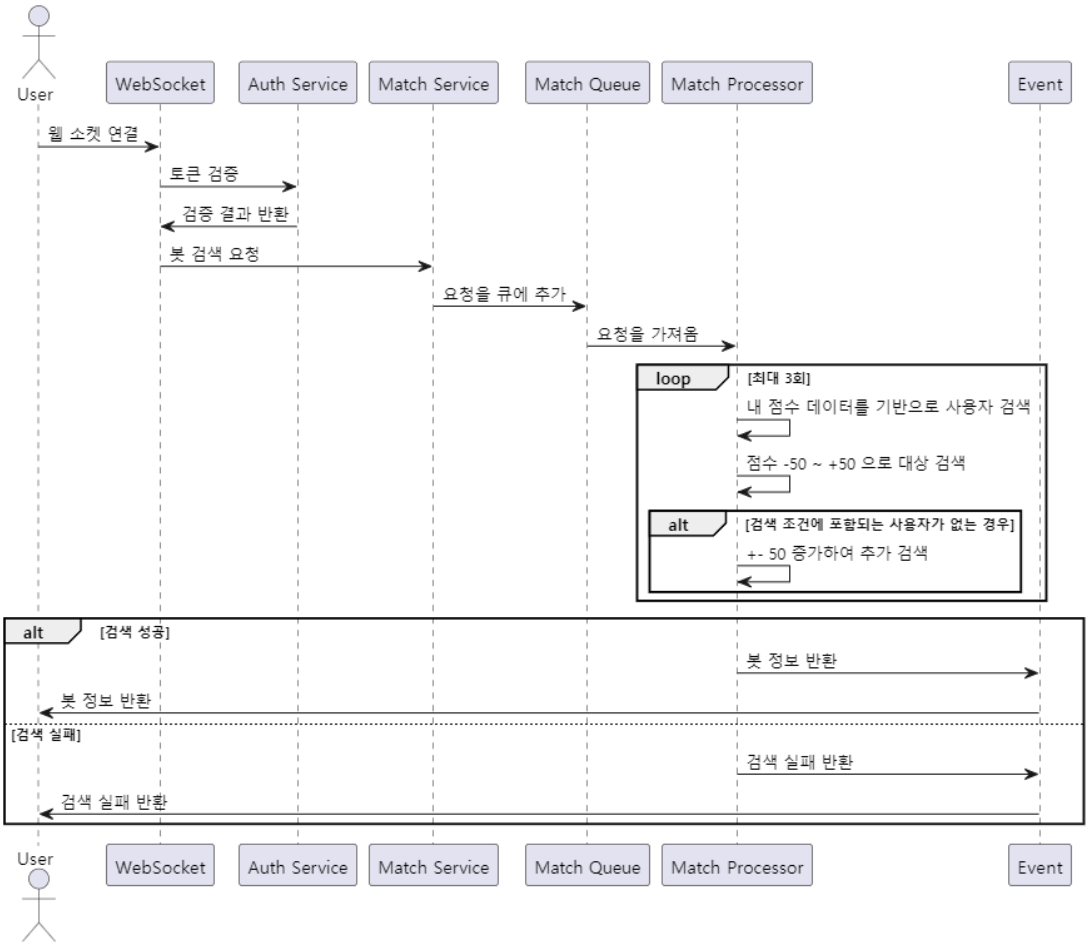
- 앱 기능


Client
Server
Redis
web login
[API] 로그인
alt
[사용자 정보 조회]
사용자 정보 요청
요청 정보 반환
[사용자 정보 없음]
사용자 정보 생성
결과 반환
jwt 토큰 생성
사용자 정보와
토큰 반환
websocket 연결
websocket 연결
토큰 검증
사용자 정보 조회
사용자 정보 반환
매칭
매칭 요청
사용자 정보 조회
매치 큐 등록
등록 완료 반환
loop
[매칭 진행]
매칭 대상 조회
매칭 성공/실패
매칭 결과 반환
Client
Server
Redis

- 유저 매칭


User
WebSocket
Auth Service
Match Service
Match Queue
Match Processor
Event
웹 소켓 연결
토큰 검증
검증 결과 반환
매칭 요청
요청을 큐에 추가
요청을 가져옴
loop
[최대 3회]
매칭 대기 중인 사용자 확인
alt
[대기중인 사용자가 다수인 경우]
무작위 선정
[검색 결과 없음]
3초 대기 후 재 검색
alt
[매칭 성공]
매칭 성공 알림
매칭 성공 알림 전송
[매칭 실패]
일정 시간 후 재시도
alt
[3회 진행 후 검색 결과가 없다면 봇 검색으로 변경]
봇 매칭으로 전환
User
WebSocket
Auth Service
Match Service
Match Queue
Match Processor
Event

- 봇 매칭



예상 문제와 해결 방안

- 예상 문제
  1. **부하 증가**

     사용자 수가 증가함에 따라, 매칭 요청, 인증 요청, 실시간 통신 등 다양한 요청이 동시에 발생하게 됩니다. 이로 인해 서버에 부하가 증가하고 성능 저하가 발생할 수 있습니다.

  2. **데이터 일관성**

     여러 서버 인스턴스가 동시에 작동할 때, 데이터 일관성을 유지하는 것이 어려울 수 있습니다. 특히 Redis와 같은 인메모리 데이터베이스를 사용할 경우, 데이터 동기화 문제가 발생할 수 있으며 이는 사용자 경험에 영향을 미칩니다.

  3. **실시간 통신**

     실시간 통신을 위한 웹소켓 연결이 증가함에 따라, 연결 관리와 메시지 전달에서 지연이 발생할 수 있습니다. 이는 매칭이나 게임 진행에 필요한 실시간 상호작용에 부정적인 영향을 미칠 수 있습니다.

- 해결 방안
  1. **부하 분산**

- **로드 밸런서**: 로드 밸런서를 사용하여 트래픽을 여러 서버 인스턴스로 분산합니다. 이를 통해 각 서버의 부하를 줄이고 시스템의 전체 성능을 향상시킬 수 있습니다.
- **오토 스케일링**: 클라우드 환경에서 오토 스케일링을 설정하여 트래픽 증가에 따라 서버 인스턴스를 자동으로 추가하거나 제거합니다. 이는 급격한 트래픽 변화에 유연하게 대응할 수 있도록 합니다.

2. **데이터 일관성 유지**
   - **Redis 클러스터**: Redis 클러스터를 사용하여 데이터를 분산 저장하고 복제를 통해 데이터 일관성을 유지합니다. 클러스터링을 통해 여러 인스턴스에서 데이터에 접근할 때 동기화 문제를 최소화합니다.
   - **트랜잭션**: Redis의 트랜잭션 기능을 활용하여 데이터의 원자성을 보장하고, 일관성을 유지합니다.
3. **실시간 통신 최적화**
   - **웹소켓 클러스터링**: 웹소켓 연결을 여러 서버 인스턴스로 분산시키기 위해 웹소켓 클러스터링을 사용합니다. 예를 들어, Socket.IO의 Redis Adapter를 통해 여러 서버 간 메시지 전달을 최적화할 수 있습니다.
   - **메시지 큐**: 메시지 큐를 사용하여 실시간 메시지를 효율적으로 관리하고, 지연을 최소화하여 사용자 간의 즉각적인 상호작용을 보장합니다.