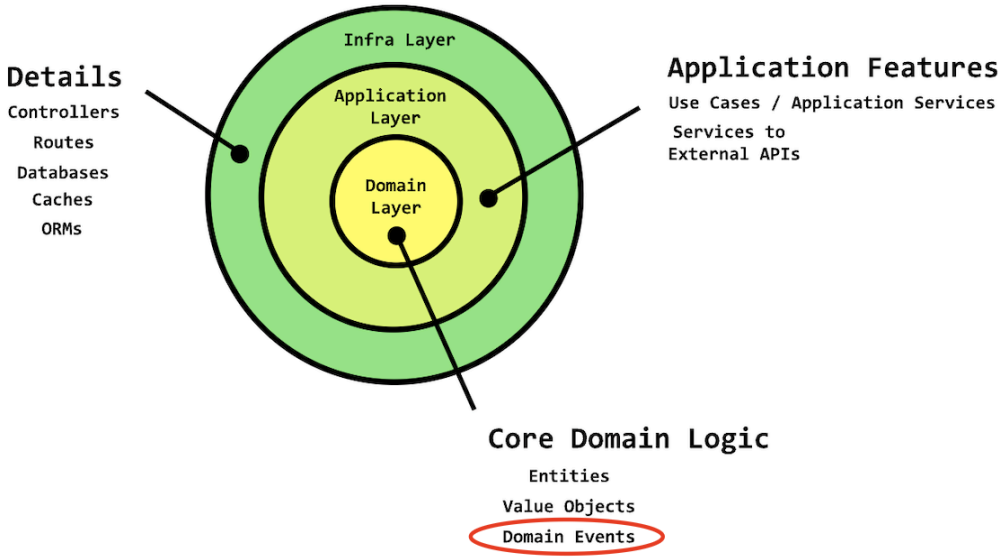
Test Task Review - Andrii Fedak

Недоліки/покращення

1. Зі структури файлів архітектура клієнтської частини не є очевидною і доволі складно буде орієнтуватись по коду, тому що він здебільшого розділений інфраструктурно (components, hooks), немає чітко виділених логічних модулів (use cases), які описують і реалізують бізнес-логіку. Хороша стаття по темі клієнтської архітектури, яка описує проблеми, що присутні в дизайн-документі, є тут. Також тут можна детальніше почитати про проблему і варіант її вирішення і ще наведемо орієнтовне бачення покращеної структури нижче:

```
src/
├── pages/
│   ├── Login.tsx
│   ├── MyGames.tsx
│   └── Game.tsx
├── features/
│   ├── auth/
│   │   └── login/
│   │       ├── ui/
│   │       │   └── LoginForm.tsx
│   │       └── model /
│   │           └── useLogin.ts
│   └── game /
│       ├── create-game/
│       │   ├── ui/
│       │   │   └── CreateGame.tsx
│       │   └── model/
│       │       └── useCreateGame.tsx
│       ├── get-games-list/
│       │   ├── ui/
│       │   │   └── ListOfGames.tsx
│       │   └── model/
│       │       └── useGames.tsx
│       ├── make-choice/
│       │   ├── ui/
│       │   │   └── GameControls.tsx
│       │   └── model/
│       │       └── useMakeChoise.tsx
│       └── ...
├── shared/
│   └── components/
│       ├── Button.tsx
│       ├── Input.tsx
│       ├── LoadingSpinner.tsx
│       └── ErrorMessage.tsx
├── App.tsx
└── index.tsx
```

2. Є цікавий момент з порушенням Dependency Rule між Application і Domain логікою тут, насправді легко виправляється перенесенням Domain Event'у на рівень домену.



3. В domain/exceptions протекло трохи інфраструктурної логіки. Це може бути проблемою, звісно не у всіх застосунках і інколи цим можна знехтувати, але загальне правильно звучить так, що domain логіка має бути незалежною від інфраструктури. Тобто в ідеальному світі це працює так:

```
src/
└── game/
    ├── domain/
    │   └── exceptions/
    │       └── FinishedGameException.ts (a pure domain exception, not extending HttpException)
    └── infrastructure/
        └── http/
            └── exception-filters/
                └── HttpExceptionFilter.ts (maps domain exceptions to HttpExceptions)
```

```
// src/domain/exceptions/FinishedGameException.ts
export class FinishedGameException extends Error {
  constructor(public readonly gameId: string) {
    super(`Game (${gameId}) is finished!`);
  }
}
```

```
// src/infrastructure/http/exception-filters/DomainExceptionFilter.ts

import { ArgumentsHost, Catch, ExceptionFilter, HttpStatus } from '@nestjs/common';
import { FinishedGameException } from '../../../domain/exceptions/FinishedGameException';
@Catch(Error)
export class DomainExceptionFilter implements ExceptionFilter {
  catch(exception: Error, host: ArgumentsHost) {
    ....
    if (exception instanceof FinishedGameException) {
      status = HttpStatus.BAD_REQUEST;
      message = exception.message;
    }
    ....
  }
}
```

4. Насправді створення окремих subdomains Auth & User виглядає ускладненням в цій ситуації, якщо Coupling між сабдоменами високий, це проявляється в цьому випадку, а наскрізні проблеми мінімальні, то має сенс тримати їх разом. Не критично, але у таких випадках об’єднання може покращити розуміння коду і просто зменшити витрати на його підтримку.

Переваги

1. Реалізовано всю функціональну частину застосунку.
2. Використано патерн Repository для роботи із базою даних.
3. В більшості дотримано SOLID принципів, CQSR на стороні API. Дуже вдало і вміло реалізовані принципи DDD, код на бекенді легко читати і буде легко масштабувати.
4. Хороший неймінг змінних в цілому по проєкту

Загальне враження

Тестове завдання по функціональних вимогах було виконано повністю. Якість коду на достатньо високому рівні. Все ж у поточній реалізації потрібно попрацювати частково над архітектурою клієнтської частини, тут не все реалізовано на перспективу і для для функціонального масштабування клієнту потрібно провести рефактор окремих його частин. З бекенд частиною все значно краще, суттєвих зауважень немає.

Додаткові посилання

1. Feature Sliced Methodology
2. Clean Architecture by Uncle Bob