Project 4: Implementation & Testing

> **팀원: 김경민 김련성 심기호 이동균 이유상 허건**
> **최종작성일자: 2023년 12월 1일**

1. Implementation

1-1. Login & Sign Up

- 인증과 인가는 JWT 토큰을 이용해서 구현한다.
- 로그인 과정
  1. 프론트에서 로그인 시 POST 요청으로 JSON body로 account와 password를 보낸다.
  2. 백엔드는 이를 받아서 password를 bcrpyt로 암호화를 하게 된다.
  3. 이후 이 암호화된 비밀번호와 account를 DB에 저장하고 payload에 PK인 user 테이블의 id와 account를 넣어JWT token을 만들어 프론트로 보내준다.
     - 이 경우 HTTP header의 authorization 항목에 “accesstoken”으로 보내준다.
     - 그러면 프론트는 WS에서 들고 있다가 로그인이 필요한 페이지들은 모두 토큰을 bearer token에 넣어서 백엔드로 보내준다.
  4. 백엔드는 bearer token에서 JWT token을 꺼내 위조가 되지 않았는지 확인한다.
     - 만약 안전하다면 payload에서 id와 account를 꺼내 유저에 맞는 정보를 다시 프론트로 보내준다.

1-2. Generating Quiz

Front-end

1. **사용자 인터페이스 및 입력 요소:**

- **퀴즈 내용 입력:** 사용자는 textarea를 사용해 퀴즈의 내용을 입력한다. 입력 길이 제한은 500~2000자이다.
- **퀴즈 옵션 설정:** 난이도, 퀴즈 유형(객관식/주관식), 퀴즈 개수 등을 설정할 수 있는 입력 필드 및 선택 상자가 제공된다.
- **난이도 선택:** 난이도는 슬라이더를 통해 'Easy', 'Medium', 'Hard' 중 선택한다.

2. **데이터 검증 및 처리:**

- **입력 길이 검증:** 퀴즈 내용 길이가 500자 미만 또는 2000자 초과인 경우, 사용자에게 경고 메시지가 표시된다.
- **퀴즈 생성 요청:** 입력 데이터가 유효한 경우, '퀴즈 생성' 버튼을 통해 GPT API 요청이 시작된다.

3. **GPT API 요청:**

- **주관식/객관식 프롬프트 생성:** 퀴즈 유형에 따라 다른 프롬프트가 생성된다.
  - 주관식: `Please create ${quizCount} subjective questions based on the following format: Q#(# is number)."" \\\\nA#(# is number)."" \\\\n Difficulty: ${difficultyLevels[difficulty-1]}\\\\n${quizContent}`
  - 객관식: `Please create ${quizCount} problems based on the following. Each question must have four options and the correct answer is one of them. The problem starts with Q.(ex. Q#(# is number). Problem a, b, c, d(where a, b, c, d are options) The answer to the question is indicated by A.(ex. A#(# is number). a) Difficulty: ${difficultyLevels[difficulty -1]}\\n The content of the question is as follows:\\n “${quizContent}”\\n Note: Make sure the problem is in multiple-choice form`
- **API 호출:** 구성된 프롬프트를 사용하여 GPT-3.5 API에 요청을 보낸다.

4. **응답 처리 및 데이터 전송:**

- **응답 파싱:** API 응답으로부터 퀴즈 문제와 답변을 추출한다.

- **서버 요청:** 추출된 데이터를 서버로 전송한다. 전송 데이터 포맷은 JSON {InputText, Difficulty, Type, DataTitle, [{QuizText, QuizAnswer}]}이다.

5. **네비게이션 및 사용자 경험:**

- **로딩 화면:** API 요청 중 로딩 상태를 표시한다.
- **퀴즈 세트 페이지로 이동:** 퀴즈 생성이 완료되면, 생성된 퀴즈 세트를 포함하여 `QuizSet` 컴포넌트로 이동한다.

6. **추가 기능 및 에러 핸들링:**

- **로그아웃 및 페이지 이동:** 사용자는 로그아웃 버튼을 통해 로그아웃할 수 있으며, 다른 페이지로의 이동도 가능하다.
- **API 요청 실패 처리:** GPT API 호출 또는 서버 요청 중 실패 시 사용자에게 적절한 경고 메시지가 표시된다.

Back-end

- 프론트엔드에서 POST 요청으로 JSON { InputText, Difficulty, Type, DataTitle, [{QuizText, QuizAnswer}]} 을 받는다.
- 해당 정보대로 Data 정보들과 해당하는 Quiz정보들을 DB에 저장한다.
- 먼저 Data를 저장하고 퀴즈에 Data를 FK로 해서 저장하기 때문에 총 DB 연결이 두번이 있다. 그래서 쿼리 2개 앞 뒤로 Transaction을 걸어준다.

1-3. QuizSpace

Front-end

1. **퀴즈 데이터 로드:**
   - 페이지 접속 시 서버에 GET 요청을 보내 전체 퀴즈 데이터를 불러온다.
   - 불러온 퀴즈 데이터는 상태 변수에 저장되어 UI에 표시된다.
2. **퀴즈 항목 클릭 동작:**
   - 퀴즈 항목 클릭 시, 해당 퀴즈의 `dataId`를 URL 매개변수로 사용하여 "Destination" 페이지로 이동한다.
   - "Destination" 페이지에서는 `dataId`를 사용하여 해당 퀴즈 세트의 상세 정보를 서버로부터 불러온다.
3. **카테고리 관련 작업:**
   - 카테고리 추가, 수정, 삭제, 퀴즈의 카테고리 변경 등은 모두 서버에 요청을 보내 처리된다.
   - 각 작업 후에는 전역 `authInfo` 상태를 업데이트하여 UI에 반영된다.
4. **카테고리 탭 및 필터링:**
   - 카테고리 별로 탭을 생성하고, 활성 탭에 따라 퀴즈를 필터링하여 표시한다.
   - `activeTab` 상태 변수를 사용하여 현재 활성화된 카테고리를 관리한다.
5. **동적 UI 요소 및 사용자 인터랙션:**
   - 카테고리 추가, 수정 등의 작업은 모달 또는 입력 필드를 통해 수행된다.
   - 각 작업에 대한 버튼 클릭 시, 적절한 핸들러 함수가 호출된다.

Back-end

1. **카테고리 생성**
   - POST 요청으로 JSON : {Department} 을 받아서 DB에서 새롭게 Department를 가진 레코드를 생성해준다.
2. **카테고리 이름 변경**
   - PATCH 요청으로 url 경로 (/category/:categoryID) 로 categoryID를 path Variable로 받고 JSON으로 {Department}을 받아서 해당 categoryID에 해당하는 레코드의 Department를 JSON에 있는 Department로 변경한다.
3. **데이터의 카테고리 변경**
   - PATCH 요청으로 JSON {DataID, nextCID} 를 받아서 DataID로 해당 Data의 categoryID를 nextCID로 수정한다.

4. **카테고리 삭제**
    - DELETE 요청으로 url 경로 (/category/categoryID)를 받아서 categoryID에 해당하는 카테고리를 삭제한다. 카테고리를 지울 때 해당하는 데이터와 퀴즈도 다 삭제해야 하므로 cascade 옵션을 준다. 바로 삭제하기 보다는 soft-remove 기능을 써서 deleted_at에 Date를 추가하는 형식으로 지운다.

1-4. Trouble Shooting

Front-end

1. 서버와 프론트 모두 port 3000을 사용하여 conflict가 일어난다.

    → 프론트에서의 proxy 설정을 통해 해결한다.

2. Chat-GPT API를 사용하기 위한 prompt 포맷을 맞추는 과정이 다소 원활하지 못했다.

    → format 예제를 두고 영어로 prompt 작성 후 gpt 모델을 3에서 3-5로 바꿨다.

    → 이 과정에서 GPT 모델(3, 3-5)마다 API 호출 형식이 달라 다소 어려움을 겪었다.

3. 문제 생성 버튼을 누른 후 이동하는 화면을 Quiz Space에서 문제 상세 페이지로 재활용하고자 한다.

    → 두 페이지의 랜더링 방식이 달라 실패했다.

4. 서버와의 데이터 필드 대소문자 구분에서 문제가 발생했다.

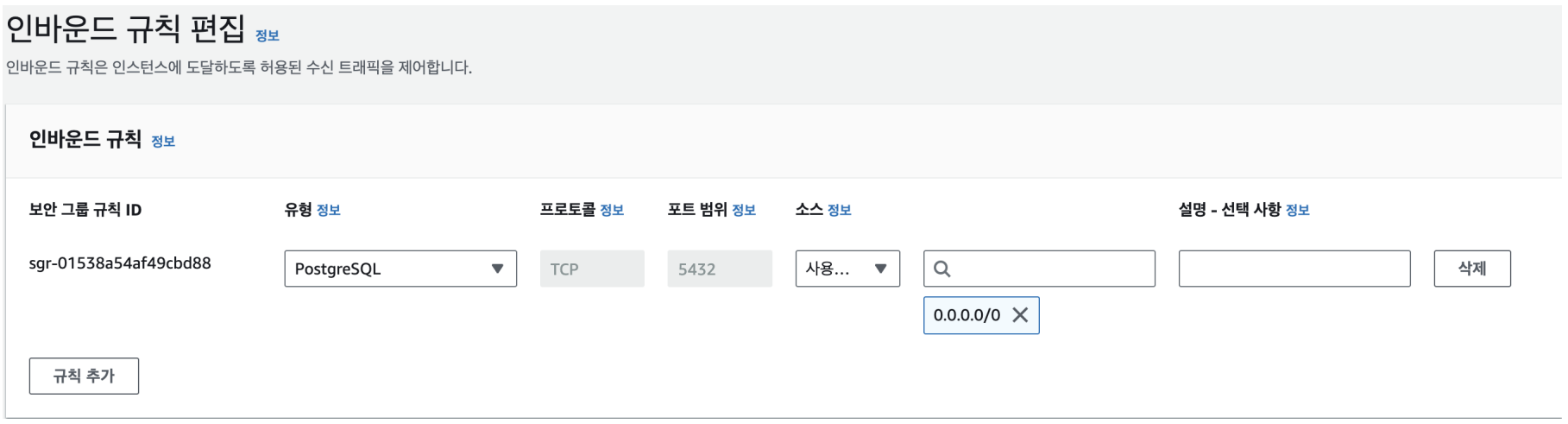
    → 정확한 필드 이름 공유를 통해 해결한다.

5. 카테고리 수정/추가/삭제 시 즉시 화면에 반영되지 않고 다른 페이지를 다녀와야 반영이 가능하다.

    → 이 부분은 해결하지 못했다.

Back-end

1. AWS RDS 서비스를 이용한 팀원의 local environment와 DB 공유 과정에서의 DB 연결 문제

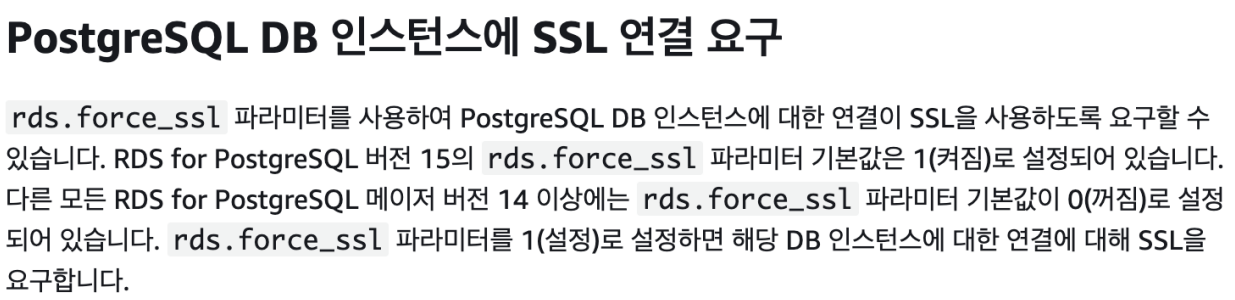


RDS의 인바운드 규칙을 모든 TCP 주소를 가능하게 설정함에도 불구하고 아래와 같은 에러가 발생했다.

```
[오후 7:26:04] Starting compilation in watch mode...

[오후 7:26:06] Found 0 errors. Watching for file changes.

[Nest] 6410  - 2023. 11. 15. 오후 7:26:07     LOG [NestFactory] Starting Nest application...
[Nest] 6410  - 2023. 11. 15. 오후 7:26:07     LOG [InstanceLoader] TypeOrmModule dependencies initialized +12ms
[Nest] 6410  - 2023. 11. 15. 오후 7:26:07     LOG [InstanceLoader] ConfigHostModule dependencies initialized +0ms
[Nest] 6410  - 2023. 11. 15. 오후 7:26:07     LOG [InstanceLoader] AppModule dependencies initialized +0ms
[Nest] 6410  - 2023. 11. 15. 오후 7:26:07     LOG [InstanceLoader] AuthModule dependencies initialized +0ms
[Nest] 6410  - 2023. 11. 15. 오후 7:26:07     LOG [InstanceLoader] DataModule dependencies initialized +0ms
[Nest] 6410  - 2023. 11. 15. 오후 7:26:07     LOG [InstanceLoader] CategoryModule dependencies initialized +0ms
[Nest] 6410  - 2023. 11. 15. 오후 7:26:07     LOG [InstanceLoader] QuizModule dependencies initialized +0ms
[Nest] 6410  - 2023. 11. 15. 오후 7:26:07     LOG [InstanceLoader] ConfigModule dependencies initialized +0ms
[Nest] 6410  - 2023. 11. 15. 오후 7:26:07     LOG [InstanceLoader] ConfigModule dependencies initialized +1ms
[Nest] 6410  - 2023. 11. 15. 오후 7:26:07   ERROR [TypeOrmModule] Unable to connect to the database. Retrying (1)...
error: no pg_hba.conf entry for host "58.239.2.136", user "dbsghgudvos123", database "quizgenerator", no encryption
    at Parser.parseErrorMessage (/Users/giho/projects/QuizGenerator/server/node_modules/pg-protocol/src/parser.ts:369:69)
    at Parser.handlePacket (/Users/giho/projects/QuizGenerator/server/node_modules/pg-protocol/src/parser.ts:188:21)
    at Parser.parse (/Users/giho/projects/QuizGenerator/server/node_modules/pg-protocol/src/parser.ts:103:30)
    at Socket.<anonymous> (/Users/giho/projects/QuizGenerator/server/node_modules/pg-protocol/src/index.ts:7:48)
    at Socket.emit (node:events:514:28)
    at addChunk (node:internal/streams/readable:324:12)
    at readableAddChunk (node:internal/streams/readable:297:9)
    at Socket.Readable.push (node:internal/streams/readable:234:10)
    at TCP.onStreamRead (node:internal/stream_base_commons:190:23)
```



PostgreSQL DB 인스턴스에 SSL 연결 요구

`rds.force_ssl` 파라미터를 사용하여 PostgreSQL DB 인스턴스에 대한 연결이 SSL을 사용하도록 요구할 수 있습니다. RDS for PostgreSQL 버전 15의 `rds.force_ssl` 파라미터 기본값은 1(켜짐)로 설정되어 있습니다. 다른 모든 RDS for PostgreSQL 메이저 버전 14 이상에는 `rds.force_ssl` 파라미터 기본값이 0(꺼짐)로 설정되어 있습니다. `rds.force_ssl` 파라미터를 1(설정)로 설정하면 해당 DB 인스턴스에 대한 연결에 대해 SSL을 요구합니다.

RDS 공식문서에서 RDS의 엔진버전이 15이상부터는 ssl의 Default Value가 1이어서 SSL이 강제되었다.

→ 해당 서비스 내 db 접속 네트워크 관리 파일(pg_hba.conf)에서 외부 접속이 가능하도록 `rds.force_ssl` parameter가 0으로 설정된 RDS for PostgreSQL 버전 14를 사용하여 이 문제를 해결한다.

2. 백엔드에서 JWT을 httponly 옵션을 넣어서 쿠키로 보낼 때 프론트에서의 확인 불가 문제

```
@Post('signin')
  async signIn(@Body() signInDto: SignInDto, @Res({ passthrough: true }) res: Response) {
    const loginRes: LoginResultType = await this.authService.signIn(signInDto.account, signInDto.password);
    res.cookie('AccessToken', loginRes.accessToken, loginRes.cookieOption);
    return await this.authService.returnSignIn(loginRes.user);
  }
```

```
async signIn(account, pass): Promise<LoginResultType> {
    const user = await this.userService.findbyAccount(account);

    //비크립트 비교
    const isPasswordMatching = await bcrypt.compare(pass, user.password);
    if (!isPasswordMatching) {
      throw new UnauthorizedException();
    }
    user.password = undefined; //유저 비번 가리기
    const payload = { sub: user.id, account: user.account };
    const accessToken = await this.jwtService.signAsync(payload);
    const result: LoginResultType = {
      user: user,
      accessToken: accessToken,
      cookieOption: this.getCookieOption(),
    };
    return result;
  }
```

백엔드에서 httponly로 쿠키를 보내면 클라이언트에서 직접 확인이 불가능하다.

→ cookie에 넣어서 보내지 않고 Authorization 헤더로 보내어 해결한다.

3. env 파일 관련 `MissingDriverError: Wrong driver: "undefined" given` 에러

```
[Nest] 42064  - 2023. 12. 01. 오전 11:34:42   ERROR [TypeOrmModule] Unable to connect to the database. Retrying (1)...
MissingDriverError: Wrong driver: "undefined" given. Supported drivers are: "aurora-mysql", "aurora-postgres", "better-sqlite3", "capacitor", "cockroachdb", "cordova", "expo", "mariadb", "mongodb", "mssql", "mysql", "nativescript", "oracle", "postgres", "react-native", "sap", "sqlite", "sqljs", "spanner".
    at DriverFactory.create (/Users/giho/projects/QuizGenerator/server/src/driver/DriverFactory.ts:72:23)
    at new DataSource (/Users/giho/projects/QuizGenerator/server/src/data-source/DataSource.ts:145:43)
    at createTypeormDataSource (/Users/giho/projects/QuizGenerator/server/node_modules/@nestjs/typeorm/dist/typeorm-core.module.js:164:23)
    at /Users/giho/projects/QuizGenerator/server/node_modules/@nestjs/typeorm/dist/typeorm-core.module.js:168:42
    at Observable._subscribe (/Users/giho/projects/QuizGenerator/server/node_modules/rxjs/src/internal/observable/defer.ts:55:15)
    at Observable._trySubscribe (/Users/giho/projects/QuizGenerator/server/node_modules/rxjs/src/internal/Observable.ts:244:19)
    at /Users/giho/projects/QuizGenerator/server/node_modules/rxjs/src/internal/Observable.ts:234:18
    at Object.errorContext (/Users/giho/projects/QuizGenerator/server/node_modules/rxjs/src/internal/util/errorContext.ts:29:5)
    at Observable.subscribe (/Users/giho/projects/QuizGenerator/server/node_modules/rxjs/src/internal/Observable.ts:220:5)
    at subscribeForRetryWhen (/Users/giho/projects/QuizGenerator/server/node_modules/rxjs/src/internal/operators/retryWhen.ts:74:25)
```

→ app.module에서 configmodule의 env file path의 정확한 명시가 이루어지지 않아 생긴 문제; 정확한 명시를 통해 문제 해결한다.

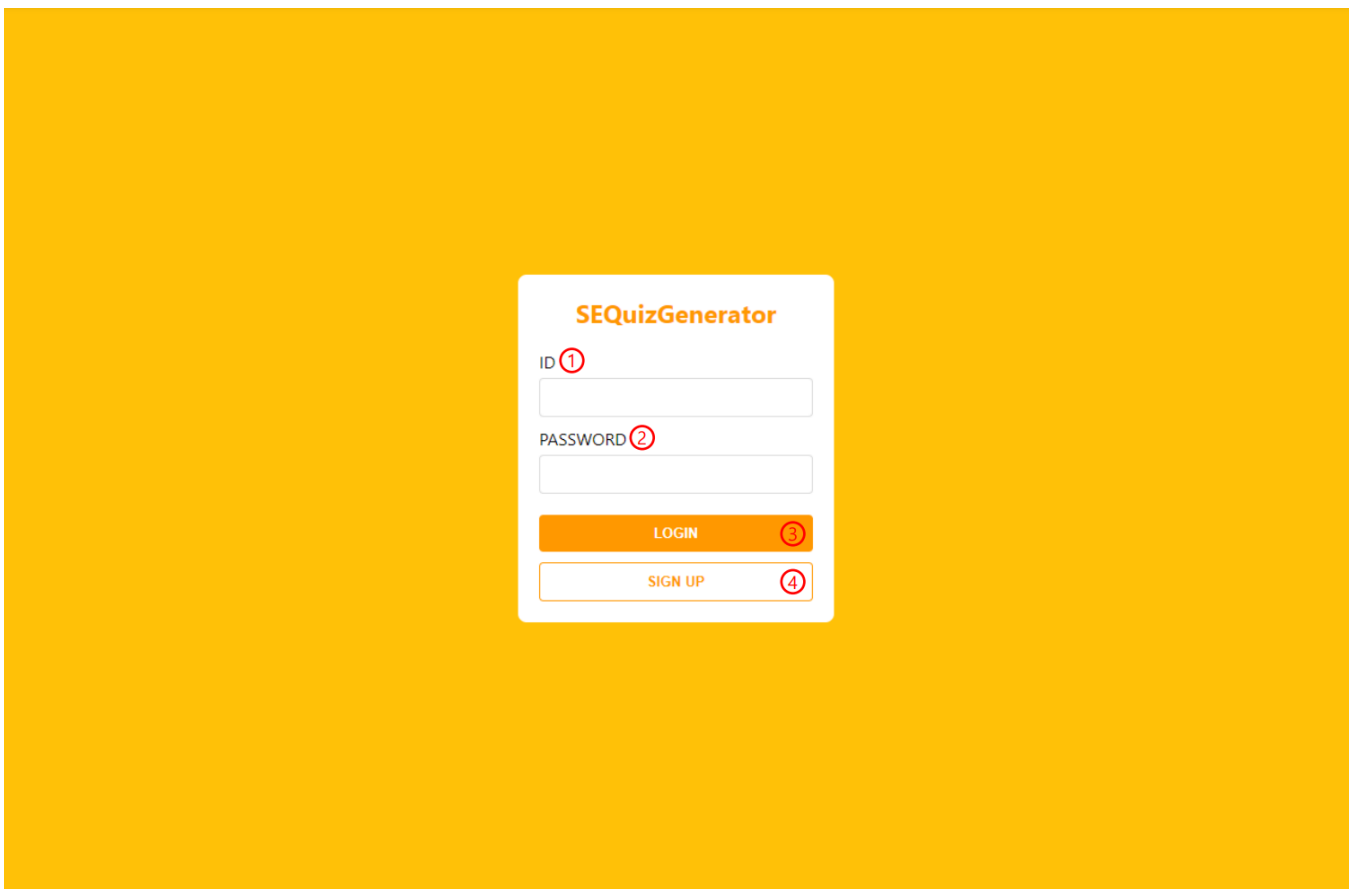
4. getData API 관련 500 error

→ 기존 data를 프론트에게 넘길 때 categoryId를 같이 줘야 하는 상황에서 data가 category가 없이 만들어진 경우 category가 null로 나타나 생겼던 오류

→ 이후 category의 null checking 이후 안전하게 null이나 categoryId를 반환하여 해결한다.\

2. User Manual

2-1. Login



Login 화면. 로그인하여 QuizGenerator화면으로 이동하거나, SIGN UP화면으로 이동한다.

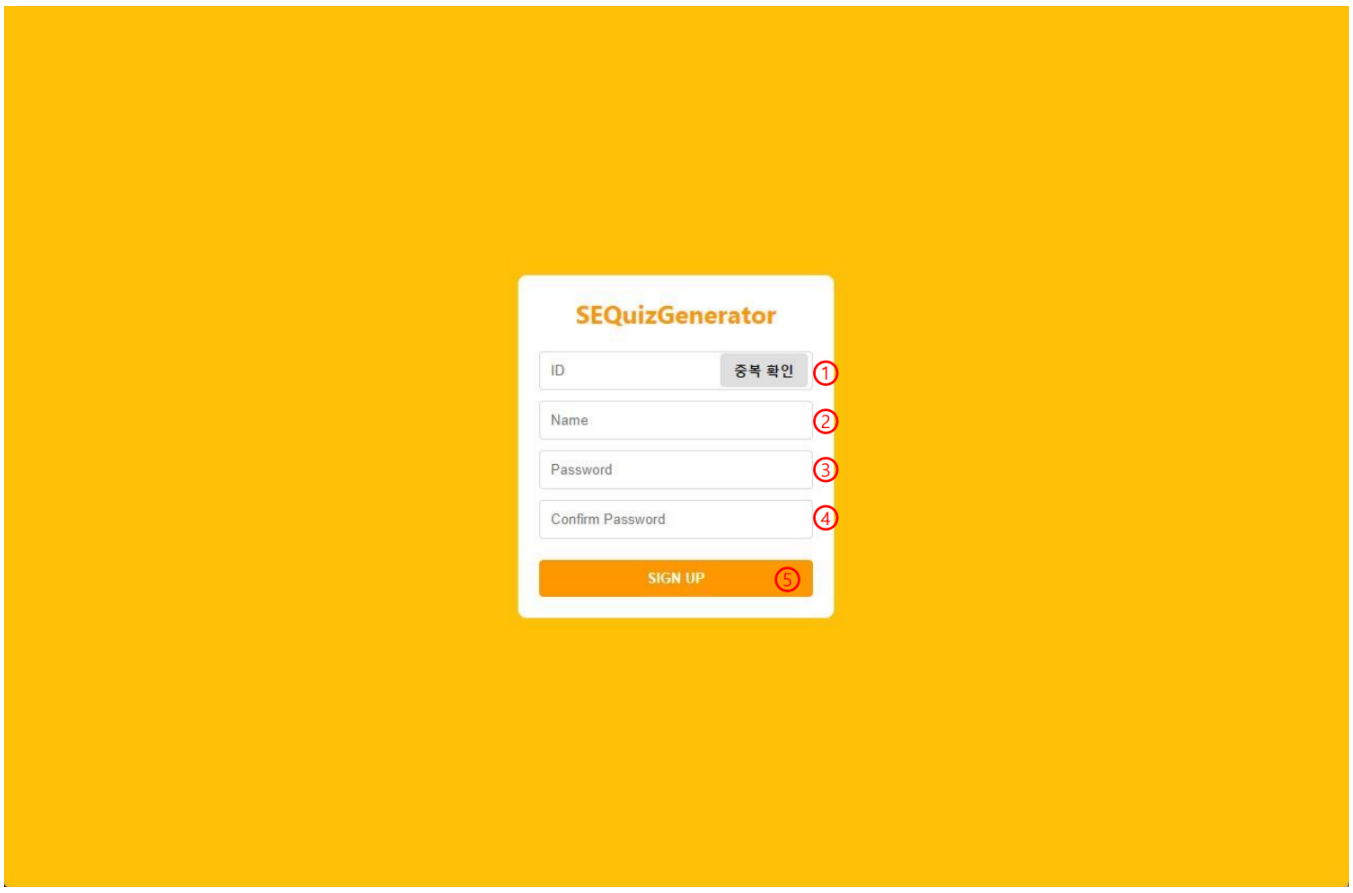
1, 2, 3번

- 회원가입시 입력하였던 ID/Password를 입력후 로그인을 할 수 있다.

4번

- 회원가입이 필요하다면 “SIGN UP”버튼을 눌러 회원가입을 진행할 수 있다.

2-2. Sign up



Sign up화면. 사용자의 ID, Name, Password 입력을 통해 회원가입을 진행한다.

1번

- ID를 입력 후 중복확인을 체크한다.

2번

- 사용자의 이름을 입력한다.

3, 4번

- Password를 입력하고, Password 재입력을 통해 입력한 Password를 확인한다.

2-3. QuizGenerator

QuizGenerator화면. 로그인 직후 보이는 화면이다.

1번

- 로그인 화면으로 이동한다.

2번

- Quiz Space화면으로 이동한다.

3~8번

3. QuizSet의 바탕, 사용자가 학습하고자 하는 텍스트를 입력한다.(입력제한: 500~2000자)
4. 생성하는 QuizSet의 제목을 입력한다.
5. QuizSet의 문제 유형을 선택한다.(객관식 또는 주관식)
6. QuizSet의 난이도를 선택한다.(Easy/Medium/Hard)
7. QuizSet의 퀴즈 개수를 입력한다. (입력 text에 따른 퀴즈 최대 개수: 500~1000자: 3문제/1000~1500자: 4문제/1500~2000자: 5문제)
8. Quiz 생성 버튼을 눌러 3~7번의 입력을 바탕으로 퀴즈를 생성한다.

2-4 QuizSet

퀴즈가 생성된 후 퀴즈와 퀴즈의 정답을 확인할 수 있다.

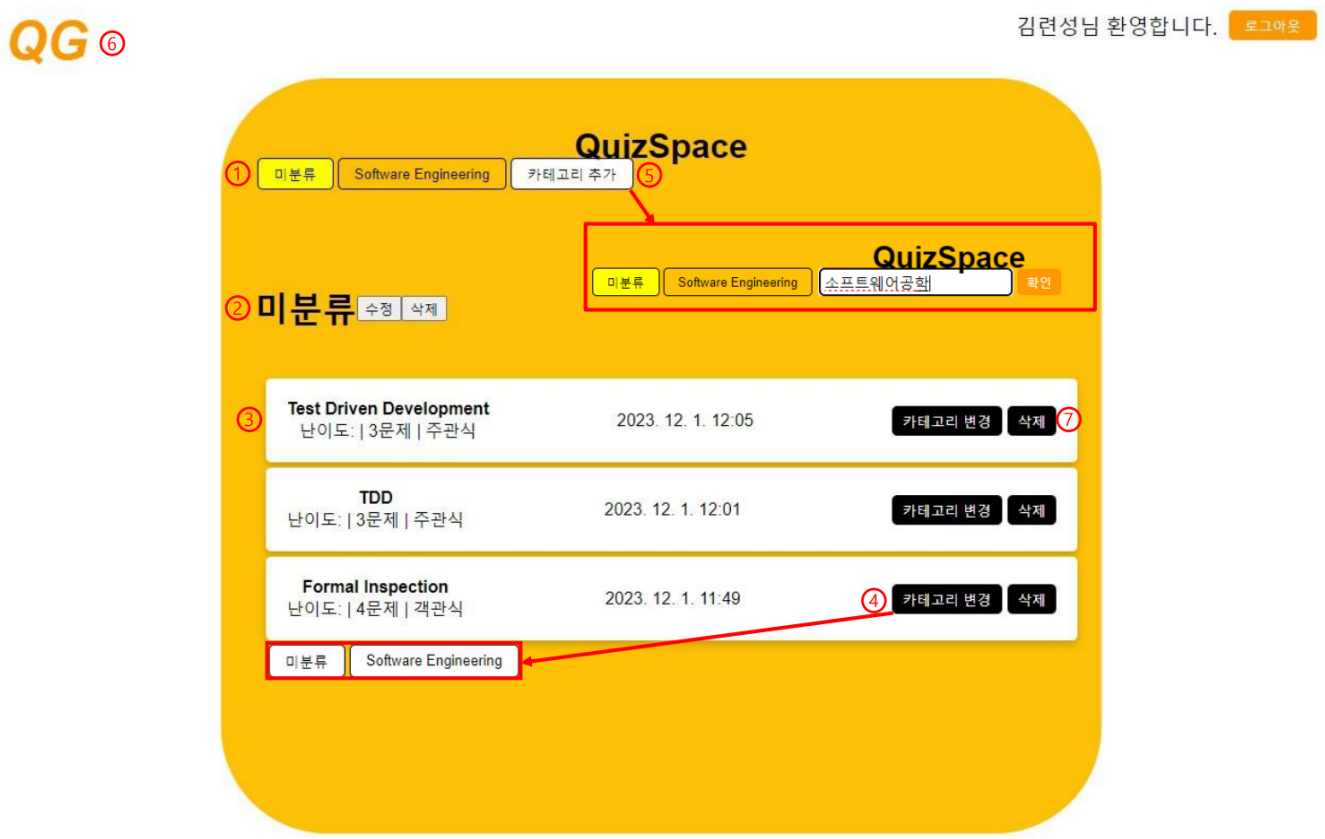
1, 2번

- 버튼을 클릭하여 퀴즈의 정답을 확인하거나 숨길 수 있다.

3번

- 클릭하여 QuizGenerator 화면으로 이동할 수 있다.

2-5. QuizSpace



QuizSpace화면. 생성된 퀴즈들을 카테고리별로 분류하거나, 삭제할 수 있다

1번

- 퀴즈를 카테고리별로 분류할 수 있다.

2번

- 카테고리의 이름을 수정하거나 카테고리를 삭제할 수 있다.

3번

- 카테고리내의 퀴즈들을 확인할 수 있다. 클릭하면 QuizSet화면으로 이동한다.

4번

- 카테고리 변경 버튼을 클릭하면 카테고리를 선택을 통해 카테고리를 변경할 수 있다.

5번

- 카테고리 목록 옆 ‘카테고리 추가’ 버튼을 클릭하면 새로운 카테고리 생성이 가능하다.

6번

- QuizGenerator 화면으로 이동할 수 있다.

7번

- 삭제 버튼을 클릭하여 퀴즈를 삭제할 수 있다.

3. How to install & use our software

- 배포를 가정한 서비스 사용 방법 :
  - 웹사이트 접속을 활용한 서비스를 제공한다.
  - 이후 회원가입을 통한 회원제 서비스를 운영한다.
- 로컬에서의 실행방법 :
  - node.js를 설치한 후 npm ci명령어를 통해서 package.json에 있는 모든 설치 module을 로컬에 설치한다. 그 후 다음의 명령어를 실행시킨다.
  - 프론트 실행: QuizGenerator/quiz 디렉토리로 이동하여 터미널에 npm start 명령어 입력
  - 백엔드 실행: QuizGenerator/server 디렉토리로 이동하여 터미널에 npm run start:dev 명령어 입력

4. Test Plan Document

4-1. Test Purpose & Range

- 테스트 목적
  - 많은 사용자로 인한 traffic이 서비스의 안정성에 커다란 영향을 끼치지 않는지 체크한다.
  - 서비스 내 모든 기능이 원하던 바에 따라 제대로 동작하는지 확인한다.
  - 모듈 간 상호작용(데이터 전송) 과정에서 잘못된 데이터 전송이나 inconsistency가 존재하지 않는지 확인한다.
- 테스트 범위
  - UI 및 로그인/회원가입 기능을 포함한 서비스의 모든 기능

4-2. Test Environment

- 대부분의 테스트는 테스트 서버를 활용한다.
- 보안 문제 관련 테스트는 데이터베이스 서버 위주로 사용한다.
- 인수 테스트는 사용자와 비슷한 상황을 상정하기 위해 실제 서버를 사용한다.

4-3. Test Method Specification

- **Protection test**
  - 테스트 목적

        - 유저가 다른 유저의 퀴즈 카테고리에 접근해서는 안 된다.
        - 유저의 로그인 정보는 타 유저에 대해 절대 공개되어서는 안 된다.
    - 테스트 절차
        1. 유저가 아무도 존재하지 않도록 데이터베이스를 초기화한다.
        2. A, B, C의 세 유저 계정을 생성한다.
        3. 각각의 계정으로 서로 다른 9가지(각 계정당 3가지) 텍스트로 3문제씩 생성한다.
        4. 이후 각 계정으로 로그인하여 각 계정으로 만든 3가지 이외 문제가 표시되는지 확인한다.
    - 이외에 사용자 문의사항으로 다른 유저 카테고리 표시 문제가 발생한다면 해당 문제 상황을 바탕으로 코드를 수정한다.
- **Accurency(Quiz) test**
    - 테스트 목적
        - 유저가 제공한 자료(텍스트)와 전혀 다른 퀴즈를 생성해서는 안 된다.
    - 테스트 절차
        1. 테스트용 텍스트를 준비하여 이를 바탕으로 퀴즈(3~5개)를 생성한다.
        2. 생성된 퀴즈의 키워드를 추출하여, 이 키워드들이 자료 텍스트에 모두 존재하는지 확인한다.
        3. 1-2 과정을 3~5회 반복한다.
    - 이외에 사용자 문의사항으로 부정확한 퀴즈 생성 문제가 발생한다면 해당 문제 상황을 바탕으로 코드를 수정한다.
- **DataBase test**
    - 테스트 목적
        - 서버의 데이터베이스에 문제가 생겨 유저 데이터나 퀴즈 내용이 손실되어서는 안 된다.
    - 테스트 절차
        - 임의의 상황 설정 및 테스트가 힘들어 관리 및 유지보수에 중점을 둔다.
        - 꾸준한 모니터링을 통해 데이터베이스와 서비스 간 consistency를 유지한다.
- **Acceptance test**
    - 테스트 목적
        - 사용자가 실제로 서비스를 사용하는 경우를 상정하여, 배포 이전 사용 시나리오에 따른 테스트를 해보고자 한다.
    - 테스트 절차
        1. ID 중복확인 테스트용 계정(ID : “test”)을 제외한 모든 유저가 존재하지 않도록 데이터베이스를 초기화한다.
        2. 로그인의 “SIGN UP” 버튼을 사용하여 회원가입 페이지로 이동한다.
        3. 테스트 ID/비밀번호로 회원가입을 진행한다.
            - 이 과정에서, ID 중복확인 기능 확인을 위해 ID 입력란에 “test”입력 후 중복확인 여부를 테스트한다.
        4. 생성한 ID/비밀번호로 로그인 후, 테스트용 텍스트로 Quiz를 생성한다.
        5. 생성된 Quiz가 미분류 카테고리에 존재하는지 확인한다.
        6. 테스트용 카테고리를 생성한다.
        7. 4번 과정에서 생성한 미분류 카테고리에 존재하는 Quiz 하나의 카테고리를 6에서 만든 카테고리로 이동한다.
        8. 7번 과정에서 이동한 Quiz의 카테고리가 정상적으로 변경되었는지 확인한다.

5. 기존 SRS, SDD에서 변경된 사항

공유 기능 삭제

- 기존 계획: 사용자들간 퀴즈 공유가 가능하다.
- 변경 후: 퀴즈공유 기능을 삭제했다.
- 사유: 사용자들 간의 공유를 가능하게 만들려고 했지만, token없이 페이지를 로드하지 못하는 문제가 발생하여 공유 기능을 삭제했다.

redis 기능 삭제

- 기존 계획: 백엔드에서 Redis를 사용해서 DB를 접근하지 않고 미리 캐싱해놓은 데이터를 프론트에 반환한다.
- 변경 후: Redis를 사용하지 않는다.
- 사유: 개발 기간 부족 및, Write 작업이 많을 것으로 예상되어 Cache 를 이용해서 얻는 성능이 그렇게 크지 않을 것으로 판단했다.

배포 계획 철회

- 기존 계획: 서비스 웹사이트를 배포하여 사용자들이 자유롭게 서비스 이용이 가능하도록 해준다.
- 변경 후: 해당 서비스를 배포하지 않기로 결정했다.
- 사유: Chat-GPT API 사용 비용(문제 생성 1회당 평균 100원) 문제로 인해 부득이하게 배포는 하지 못하게 되었다.