3/7 개발일지

1. Flyaway 잠정 보류

마이그레이션툴로 이만한 좋은 툴이 없지만, 마이그레이션 오류가 자꾸 떴다.

▼ 오류 내용

```
PS C:\projects\web1> .\gradlew flywayMigrate --stacktrace
> Task :flywayMigrate FAILED

FAILURE: Build failed with an exception.

* What went wrong:
Execution failed for task ':flywayMigrate'.
> Error occurred while executing flywayMigrate
  No database found to handle jdbc:mariadb://192.168.146.1
```

이런 오류를 고치는 방법은 다음과 같은 방법이 있다.

오류 메시지와 스택 트레이스를 살펴보면, Flyway가 `jdbc:mariadb://192.168.146.128/web1` URL을 처리하는 데 실패했다는 것을 알 수 있습니다. 이 오류는 Flyway가 제공된 JDBC URL을 인식하지 못하고, 그 결과로 데이터베이스 연결을 생성할 수 없음을 나타냅니다. 이 문제를 해결하기 위한 몇 가지 방법을 제시합니다:

JDBC URL 형식 확인

Flyway의 `flyway.url` 설정에 사용된 JDBC URL이 올바른 형식인지 다시 확인해야 합니다. 일반적으로 MariaDB JDBC URL 형식은 `jdbc:mariadb://[호스트]:[포트]/[데이터베이스명]` 입니다. 포트 번호가 누락된 것 같습니다. MariaDB의 기본 포트 번호는 `3306` 이므로, 올바른 JDBC URL은 다음과 같아야 합니다:

```
groovyCopy code
flyway {
    url = 'jdbc:mariadb://192.168.146.128:3306/web1'
    user = 'root'
    password = '5162'
    locations = ['filesystem:src/main/resources/db/migra
```

```
tion']
}
```

MariaDB JDBC 드라이버 확인

Flyway가 MariaDB 데이터베이스를 올바르게 인식하려면, MariaDB JDBC 드라이버가 프로젝트 의존성에 포함되어 있어야 합니다. `build.gradle` 파일에서 `org.mariadb.jdbc:mariadb-java-client` 의존성이 포함되어 있는지 확인하세요. 의존성 선언은 제공된 `build.gradle` 파일에 이미 포함되어 있는 것으로 보입니다. 그러나 Flyway가 실행 시 해당 드라이버를 올바르게 찾을 수 있도록, 프로젝트 설정이 올바르게 적용되었는지 확인해보세요.

Flyway 버전 확인

사용 중인 Flyway 버전이 최신 버전인지 확인하세요. 때때로 새로운 버전에서는 버그 수정이나 기능 개선이 이루어지므로, 최신 버전 사용이 문제 해결에 도움이 될 수 있습니다. `build.gradle` 파일에서 사용 중인 Flyway 버전을 확인하고 필요에 따라 업데이트할 수 있습니다.

네트워크 연결 확인

`192.168.146.128` 호스트와 `3306` 포트로의 네트워크 연결이 가능한지 확인하세요. 방화벽 설정, 네트워크 보안 그룹, 또는 가상 머신의 네트워크 설정이 연결을 차단하고 있지 않은지 검토해야 합니다.

위 단계들을 통해 문제를 해결할 수 있기를 바랍니다. 만약 문제가 지속된다면, 추가적인 진단을 위해 Flyway 로그 출력을 살펴보거나, MariaDB 서버의 로그 파일에서 관련 오류 메시지를 찾아볼 수 있습니다.

위 수행사항을 적절히 수행해도 도저히 Flyaway의 마이그레이션이 진행이 되지 않아 Spring ORM을 사용하여 마이그레이션 없이 진행하려 한다.

- 차후에 프로덕션 빌드에선 Flyaway의 마이그레이션을 지켜야 하는걸 인지한다.

2. Spring ORM을 이용한 마이그레이션 없이 스키마를 추가, 변경, 삭제

▼ 다음과 같은 과정을 따른다. (일단 3단계까지)

- 차후에 4~5 단계에 해당하는 CRUD 작업이 필요한 경우엔, Repository를 이용하여 데이터에 대해 CRUD 작업을 실시한다.

1단계: 의존성 추가

`build.gradle` 파일에 Spring Data JPA와 MariaDB JDBC 드라이버 의존성을 추가합니다. 이를 통해 필요한 라이브러리를 프로젝트에 포함시킵니다.

```
groovyCopy code
dependencies {
    implementation 'org.springframework.boot:spring-boot
-starter-data-jpa'
    runtimeOnly 'org.mariadb.jdbc:mariadb-java-client'
}
```

2단계: application.properties 설정

Spring Boot `application.properties` 파일에 MariaDB 연결 설정을 추가합니다. 이 설정을 통해 애플리케이션은 MariaDB 데이터베이스에 연결할 수 있습니다. 파일 경로는 `src/main/resources/application.properties`입니다.

```
sqlCopy code
spring.datasource.url=jdbc:mariadb://localhost:3306/your
DatabaseName
spring.datasource.username=yourUsername
spring.datasource.password=yourPassword
spring.datasource.driver-class-name=org.mariadb.jdbc.Dri
ver

# Hibernate DDL Auto (none, update, create, create-drop)
spring.jpa.hibernate.ddl-auto=update

# SQL 문을 콘솔에서 보여주는 옵션
spring.jpa.show-sql=true
```

3단계: 엔티티 클래스 생성

엔티티 클래스를 이미 생성하셨으므로, 이 단계는 이미 완료하신 것으로 보입니다. `@Entity` 어노테이션을 사용하여 클래스를 엔티티로 정의하고, `@Id` 및 `@GeneratedValue` 어노테이션을 사용하여 기본 키를 설정합니다.

4단계: 리포지토리 인터페이스 생성

Spring Data JPA를 사용하면, 데이터 접근을 위한 리포지토리 인터페이스를 쉽게 정의할 수 있습니다. 예를 들어, `Address` 엔티티에 대한 리포지토리를 생성하려면 다음과 같이 인터페이스를 정의합니다.

```
javaCopy code
package web.web1.repository;

import web.web1.model.Address;
import org.springframework.data.jpa.repository.JpaReposi
tory;

public interface AddressRepository extends JpaRepository
<Address, Long> {
    // 필요한 쿼리 메소드를 여기에 정의할 수 있습니다.
}
```

5단계: 애플리케이션 실행

위의 설정을 완료한 후, 애플리케이션을 실행하면, Spring Boot는 자동으로 `application.properties` 파일의 설정을 읽고, 지정한 데이터베이스에 테이블을 생성 또는 업데이트합니다. `spring.jpa.hibernate.ddl-auto=update` 설정은 기존 스키마를 업데이트하며, 데이터베이스에 새로운 엔티티가 있을 경우 테이블을 생성합니다.

이러한 단계를 통해, 마리아DB 데이터베이스에 자동으로 엔티티의 구조에 맞는 테이블을 마이그레이션할 수 있습니다. 데이터베이스 작업을 위한 CRUD 작업은 `AddressRepository`를 통해 쉽게 처리할 수 있습니다.

▼ 성공한 모습이다.

web1 데이터베이스 안의 User 엔티티 모델 생성.

1단계: Spring ORM을 이용한 스키마 작성 후 빌드

```java
package web.web1.model;

import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import java.util.List;

@Entity
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long userId;
    private String username;
    private String password;

    // @OneToMany(mappedBy = "user")
    // private List<SearchHistory> searchHistories;

```

2단계: 클라이언트측(localhost)에서 192.168.146.129:3306으로 접속(MariaDB서버)

```
C:\Users\sdkim>mysql -h 192.168.146.129 -u root -p
Enter password: ****
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 2
Server version: 5.5.68-MariaDB MariaDB Server

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| web1               |
+--------------------+
4 rows in set (0.002 sec)

MariaDB [(none)]> use web1
Database changed
MariaDB [web1]> show tables;
```

3단계: 데이터베이스에 추가된 User모델 확인

```
MariaDB [(none)]> use web1
Database changed
MariaDB [web1]> show tables;
+----------------+
| Tables_in_web1 |
+----------------+
| address        |
| user           |
+----------------+
2 rows in set (0.002 sec)

MariaDB [web1]> select * from user
    -> ;
Empty set (0.002 sec)

MariaDB [web1]> select * from address;
Empty set (0.002 sec)

MariaDB [web1]> describe user;
+----------+--------------+------+-----+---------+----------------+
| Field    | Type         | Null | Key | Default | Extra          |
+----------+--------------+------+-----+---------+----------------+
| user_id  | bigint(20)   | NO   | PRI | NULL    | auto_increment |
| password | varchar(255) | YES  |     | NULL    |                |
| username | varchar(255) | YES  |     | NULL    |                |
+----------+--------------+------+-----+---------+----------------+
3 rows in set (0.005 sec)
```

3. 기본 인터페이스 작업 시작

▼ 네비게이션 바

- 코드로만 존재.
- 차후에 로그인 서비스 구현되면 실제 구현할 예정

```
// import { useState, useEffect } from "react";
// import '../css/navbar.css'
// import { NavLink } from 'react-router-dom';

// const NavBar = ({}) => { // 차후에 isloggedin등등을 파리

//     const [fade, setFade] = useState(false);
//     const [word, setWord] = useState('Project by sdk
//     const [isKorean, setIsKorean] = useState(false);
```

```
//     const words = (latin, korean, mouseover) => {
//         if (mouseover) {
//             setFade(true);
//             setTimeout(() => {
//                 setWord(korean);
//                 setFade(false);
//                 setIsKorean(true);  // 한국어 문장 출력
//             }, 500);
//         } else {
//             setFade(true);
//             setTimeout(() => {
//                 setWord(latin);
//                 setFade(false);
//                 setIsKorean(false);  // 라틴어 문장 출력
//             }, 500);
//         }
//     }

//     return(
//         <nav>
//             {/* <p
//                 className={`words ${fade ? 'fade' :
//                 onMouseOver={() => words('Project by
//                 onMouseOut={() => words('Project by
//             >
//                 {word}
//             </p> */}

//         </nav>
//     )
// }

// export default NavBar;
```

▼ 메인페이지

```
const MainPage = () => {
    return(
        <h1>hi</h1>
    );
}

export default MainPage;
```

hi

This is footer

▼ 푸터

```
const Footer = () => {
    return(
        <h1>This is footer</h1>
    );
}

export default Footer;
```

4. 소셜 로그인 작업 시작

▼ 소셜 로그인 작업

- 기존에 Django를 쓰던 나는 Spring Boot가 익숙하지 않아 여러 예제를 참고해서 만들었다.
- 아직은 익숙하지 않아 예제를 복붙하는 수준에 지나지 않는다.
- SecurityConfig (Spring Security 라이브러리, 전체적인 로그인의 흐름을 관장함)

SecurityConfig.java

```
package web.web1.Oauth.config;

import web.web1.Oauth.domain.OAuth2MemberService;
import lombok.RequiredArgsConstructor;
// import org.springframework.beans.factory.annotation.Aut
// import org.springframework.context.annotation.Bean;
// import org.springframework.context.annotation.Configura
// import org.springframework.security.config.annotation.w
// import org.springframework.security.config.annotation.w
// import org.springframework.security.web.SecurityFilterC

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuratio
import org.springframework.security.config.annotation.web.
import org.springframework.security.config.annotation.web.
import org.springframework.security.core.userdetails.UserD
import org.springframework.security.web.SecurityFilterChai
import org.springframework.security.web.authentication.log

@Configuration
@RequiredArgsConstructor
@EnableWebSecurity
public class SecurityConfig {
    private final OAuth2MemberService oAuth2MemberService;

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSec
        http
            .httpBasic().disable()
            .csrf().disable()
```

```
            .cors().and()
            .authorizeRequests()
                .requestMatchers("/private/**").authentica
                .requestMatchers("/admin/**").hasRole("ADM
                .anyRequest().permitAll()
            .and()
                .formLogin()
                .loginPage("/loginForm")
                .loginProcessingUrl("/login")
                .defaultSuccessUrl("/home", true)
            .and()
                .logout()
                .logoutSuccessUrl("/loginForm?logout") //
                .invalidateHttpSession(true) // HTTP 세션 두
                .deleteCookies("JSESSIONID") // JSESSIONID
                .permitAll()
            .and()
                .oauth2Login()
                .loginPage("http://localhost:3000/login")
                .defaultSuccessUrl("http://localhost:3000/
                .userInfoEndpoint()
                .userService(oAuth2MemberService);

        return http.build();
    }

    // 로그아웃 성공 핸들러 구현이 필요한 경우 여기에 추가합니다.
    // 예: 커스텀 로그아웃 성공 핸들러
    // .logoutSuccessHandler(customLogoutSuccessHandler())

    private LogoutSuccessHandler customLogoutSuccessHandle
        return (request, response, authentication) -> {
            // 커스텀 로그아웃 성공 로직 구현
            response.sendRedirect("http://localhost:3000/l
        };
    }
}
```

```
// @Configuration
// @RequiredArgsConstructor
// @EnableWebSecurity
// public class SecurityConfig {
//     private final OAuth2MemberService oAuth2MemberServi

//     @Bean
//     public SecurityFilterChain securityFilterChain(Http
//         return httpSecurity
//                 .httpBasic().disable()
//                 .csrf().disable()
//                 .cors().and()
//                 .authorizeRequests()
//                 .requestMatchers("/private/**").authent
//                 .requestMatchers("/admin/**").access("h
//                 .anyRequest().permitAll() //나머지 uri는
//                 .and()
//                 .formLogin() // form login 관련 설정
//                 .loginPage("/loginForm")
//                 .usernameParameter("name") // Member가 u
//                 .loginProcessingUrl("/login") // 로그인 요
//                 .defaultSuccessUrl("/home") // 로그인 성공
//                 .and().oauth2Login()//oauth2 관련 설정
//                 .loginPage("/loginForm") //로그인이 필요한
//                 .defaultSuccessUrl("http://localhost:30
//                 .userInfoEndpoint()//로그인 완료 후 회원 정
//                 .userService(oAuth2MemberService).and()
//                 .logout((logout) -> logout.logoutUrl)
//                     .logoutSuccessUrl("http://localhost
//                     .invalidateHttpSession(true) // 세션
//                     .clearAuthentication(true) // 인증 정
//                     .and();

//     }
// }
```

OAuthController.java

- Django의 url, view의 역할을 관장함.

```
package web.web1.Oauth.controller;


import web.web1.Oauth.domain.Member;
import web.web1.Oauth.domain.MemberRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.security.crypto.bcrypt.BCryptPa
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping

@Controller
@RequiredArgsConstructor
public class OAuthController {
    private final BCryptPasswordEncoder encoder;
    private final MemberRepository memberRepository;
    @GetMapping("/loginForm")
    public String home() {
        return "loginForm";
    }
    @GetMapping("/joinForm")
    public String joinForm() {
        return "joinForm";
    }
    @PostMapping("/join")
    public String join(Member member) {
        String rawPwd = member.getPassword();
        System.out.println("member = " + member);
        member.setRole("ROLE_USER");
        member.setPassword(encoder.encode(rawPwd));
        memberRepository.save(member);
        return "redirect:/loginForm";
    }

    @GetMapping("/private")
```

```
    public String privatePage() {
        return "privatePage";
    }
    @GetMapping("/admin")
    public String adminPage() {
        return "adminPage";
    }
}
```

- Member(유저 엔티티)

Member.java

```
package web.web1.Oauth.domain;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import lombok.Builder;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

@Entity
@NoArgsConstructor
@Getter
@Setter
public class Member {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id; //기본키
    private String oauth2Id;
    private String name; //유저 이름
    private String password; //유저 비밀번호
    private String email; //유저 구글 이메일
    private String role; //유저 권한 (일반 유저, 관리자)
    private String provider; //공급자 (google, facebook ...)
    private String providerId; //공급 아이디
```

```
    @Builder
    public Member(String oauth2Id, String name, String pas
        this.oauth2Id=oauth2Id;
        this.name = name;
        this.password = password;
        this.email = email;
        this.role = role;
        this.provider = provider;
        this.providerId = providerId;
    }
}
```