3/12 개발일지

1. 엔티티 재구성

▼ 새롭게 만들어진 엔티티들

- Member.java

```
package web.web1.Member.domain;

import jakarta.persistence.*;
import lombok.*;

import java.util.ArrayList;
import java.util.List;

import web.web1.Map.domain.MemberHistory;

@Entity
@NoArgsConstructor
@Getter
@Setter
public class Member {
    @Id
    private String email; // 유저 구글 이메일을 기본키로 사용

    private String oauth2Id;
    private String name; //유저 이름
    private String password; //유저 비밀번호
    private String role; //유저 권한 (일반 유저, 관리자)
    private String provider; //공급자 (google, facebook
    private String providerId; //공급 아이디

    @OneToMany(mappedBy = "member", cascade = CascadeTy
    private List<MemberHistory> userHistories = new Arr

    @OneToOne(mappedBy = "member", cascade = CascadeTyp
    private Token token;
```

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

- Token.java

```
package web.web1.Member.domain;

import java.time.LocalDateTime;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

@Entity
@NoArgsConstructor
@Getter
@Setter
public class Token {

    @Id
    private String email; // 유저의 이메일을 기본키로 사용

    private String tokenValue;
    private LocalDateTime createdTime;
    private LocalDateTime expiredTime;
    private String tokenType;
```

```
    @OneToOne
    @JoinColumn(name = "email", referencedColumnName =
    @MapsId // Member 엔티티의 기본 키를 Token의 기본 키로 사
    private Member member;

    // 모든 필드를 포함하는 생성자
    public Token(String email, String tokenValue, Local
        this.email = email; // Member 객체의 email을 직접
        this.tokenValue = tokenValue;
        this.createdTime = createdTime;
        this.expiredTime = expiredTime;
        this.tokenType = tokenType;
        this.member = member;
    }

}
```

- MemberHistory.java

```
package web.web1.Map.domain;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

import java.time.LocalDateTime;
import java.io.Serializable;

import web.web1.Member.domain.Member;
import web.web1.Config.MemberHistoryId;

@Entity
@NoArgsConstructor
@Getter
@Setter
public class MemberHistory {
```

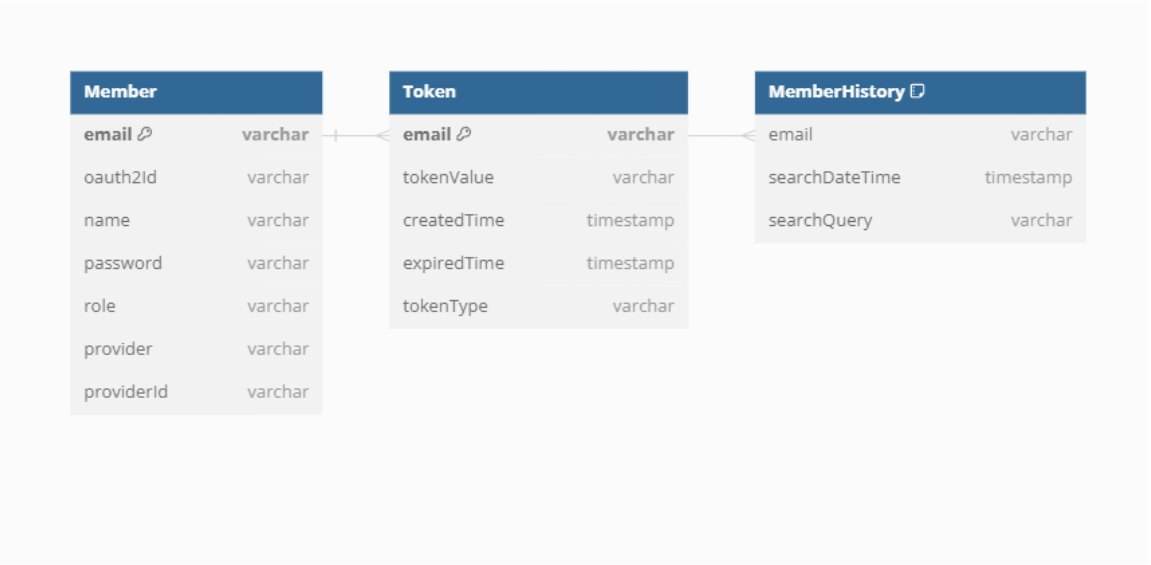
```
    @EmbeddedId
    private MemberHistoryId id;

    private String searchQuery;

    @ManyToOne(fetch = FetchType.LAZY)
    @MapsId("email") // 복합 키의 일부를 외래 키로 매핑
    @JoinColumn(name = "email", referencedColumnName =
    private Member member;


    public MemberHistory(String searchQuery, Member mem
        this.id = new MemberHistoryId(member.getEmail()
        this.searchQuery = searchQuery;
        this.member = member;
    }

    // getters and setters
}
```

- MemberHistoryId.java

```
package web.web1.Config;
import java.io.Serializable;
import java.time.LocalDateTime;
import jakarta.persistence.Embeddable;

// MemberHistory 엔티티의 복합키를 정의한 클래스
@Embeddable
public class MemberHistoryId implements Serializable {
    private String email;
    private LocalDateTime searchDateTime;

    public MemberHistoryId() {
    }

    public MemberHistoryId(String email, LocalDateTime
        this.email = email;
```

```
        this.searchDateTime = searchDateTime;
    }

    // getters and setters
}
```

▼ 보기 좋게 클래스 다이어그램으로 만들면 다음과같다.



2. Controller 구성

MemberController

`MemberController`는 회원 관련 기능을 제공합니다. 주요 기능으로는 일반 로그인, 회원가입, 로그아웃이 있습니다.

일반 로그인

- 경로: `/member/login`
- 메서드: `POST`
- 기능: 사용자의 이메일과 비밀번호를 확인하여 로그인을 처리합니다. 로그인 성공 시 JWT를 생성하고, 이를 `Authorization` 헤더에 담아 응답합니다.

회원가입

- 경로: `/member/join`
- 메서드: `POST`
- 기능: 사용자 정보를 받아 신규 회원가입을 처리합니다. 비밀번호는 BCrypt 암호화를 통해 안전하게 저장됩니다.

로그아웃

- 경로: `/member/logouts`
- 메서드: `POST`
- 기능: 사용자의 JWT를 검증하여 유효하지 않거나 만료된 경우 삭제합니다. 로그아웃 요청 시에는 토큰의 유효성과 관계없이 삭제합니다.

EgovSampleController

`EgovSampleController`는 외부 API 호출을 통해 주소 정보 및 지오코딩 서비스를 제공합니다.

주소 정보 가져오기

- 경로: `/map/getaddress`
- 메서드: `POST`
- 기능: 사용자가 입력한 키워드를 기반으로 주소 정보를 조회합니다. 조회된 결과는 JSON 형태로 응답됩니다.

지오코딩 정보 가져오기

- 경로: `/map/getgeocoding`
- 메서드: `POST`
- 기능: 주소를 입력받아 해당 주소의 위도와 경도 정보를 조회합니다. 조회된 결과는 JSON 형태로 응답됩니다.

공통 요소

- `BCryptPasswordEncoder`: 비밀번호 암호화에 사용됩니다.
- `TokenFactory` 인터페이스와 구현체 (`GoogleTokenFactory`, `NormalTokenFactory`): JWT 생성 및 관리에 사용됩니다.
- `ApplicationContext`: 빈 관리 및 접근에 사용됩니다.
- `MemberRepository`, `TokenRepository`: 데이터베이스 접근에 사용됩니다.

주요 처리 로직

- **로그인 시**, 입력받은 비밀번호와 데이터베이스에 저장된 비밀번호의 해시 값을 비교하여 로그인을 처리합니다. 로그인 성공 시 JWT를 생성하여 클라이언트에 반환합니다.
- **회원가입 시**, 입력받은 비밀번호를 BCrypt로 암호화하여 저장합니다. 이메일 중복 검사를 통해 이미 가입된 계정인지 확인합니다.
- **로그아웃 시**, 클라이언트로부터 받은 JWT를 검증하여 유효하지 않거나 만료된 경우 토큰을 삭제합니다. 로그아웃 과정은 토큰의 유효성과 무관하게 수행됩니다.
- **주소 정보 및 지오코딩 정보 조회 시**, 외부 API를 호출하여 필요한 데이터를 조회하고, 이를 클라이언트에 JSON 형태로 반환합니다.

이 구성은 웹 어플리케이션의 사용자 인증 및 주소 관련 기능을 효과적으로 제공합니다. 또한, 외부 API를 통한 데이터 조회 기능은 사용자에게 추가적인 가치를 제공합니다.

3. JWT 구성

▼ JWT 클래스 및 메소드 구성

JWT 구성 상세

JWT (JSON Web Tokens) 인증 시스템은 `TokenFactory` 인터페이스를 통해 구현되며, 이는 여러 구현체를 통해 다양한 토큰 생성 및 관리 기능을 제공합니다. `AbstractTokenFactory`는 `TokenFactory`의 기본 기능을 구현하며, 구체적인 토큰 생성 로직은 `GoogleTokenFactory`와 `NormalTokenFactory`에 의해 제공됩니다.

TokenFactory 인터페이스

`TokenFactory`는 JWT 생성 및 관리를 위한 기본 메소드를 정의합니다. 주요 메소드는 다음과 같습니다:

- `createToken(Member member, String code)`: 주어진 사용자 정보와 코드를 기반으로 JWT를 생성합니다.
- `decodeToken(String token)`: 토큰을 디코드하여 클레임을 반환합니다.
- `checkToken(String tokenValue, List<String> allowedRoles)`: 토큰의 유효성 및 역할을 검사합니다.
- `isExpired(String exp)`: 토큰의 만료 여부를 검사합니다.
- `deleteToken(String oauth2Id)`: 주어진 OAuth2 ID를 기반으로 토큰을 삭제합니다.

AbstractTokenFactory 클래스

`AbstractTokenFactory` 클래스는 `TokenFactory` 인터페이스의 추상 구현체로, 공통된 토큰 관리 로직을 제공합니다. JWT의 디코드, 만료 검사 및 삭제 등의 기능을 구현합니다.

```
public abstract class AbstractTokenFactory implements To
kenFactory {
    protected static final String secretKey = SecretConf
ig.SECRET;
    ...
}
```

GoogleTokenFactory와 NormalTokenFactory

이들 클래스는 구체적인 토큰 생성 로직을 구현합니다. 각각 Google OAuth와 일반 인증 방식에 대한 JWT를 생성하는 기능을 제공합니다.

GoogleTokenFactory

Google OAuth를 사용하는 사용자를 위한 JWT를 생성합니다. 토큰에는 사용자 역할, OAuth2 ID 등이 포함됩니다.

```
@Service("googleTokenFactory")
public class GoogleTokenFactory extends AbstractTokenFac
tory {
    ...
    @Override
    public String createToken(Member member, String cod
e) {
        ...
    }
}
```

NormalTokenFactory

일반 인증 방식(예: 사용자 이름과 비밀번호)을 사용하는 사용자를 위한 JWT를 생성합니다. 이 클래스 역시 사용자의 역할과 OAuth2 ID를 토큰에 포함합니다.

```
@Service("web1TokenFactory")
public class NormalTokenFactory extends AbstractTokenFac
tory {
    ...
    @Override
    public String createToken(Member member, String cod
e) {
        ...
    }
}
```

주요 기능

- **토큰 생성**: 사용자의 인증 정보를 기반으로 보안 JWT를 생성합니다. 이 토큰은 사용자의 세션 관리와 API 접근 제어에 사용됩니다.
- **토큰 검증 및 디코드**: 서버는 토큰을 검증하고 디코드하여 사용자의 신원 및 권한을 확인합니다.
- **토큰 만료 관리**: 만료된 토큰은 자동으로 시스템에서 제거됩니다, 이를 통해 보안성을 강화합니다.

JWT 인증 시스템은 웹 애플리케이션의 보안을 강화하며, 사용자 인증 및 세션 관리를 효율적으로 수행할 수 있게 합니다.

Member와 Token 엔티티의 관계 설명

웹 어플리케이션에서 사용자의 인증 정보와 세션 관리는 중요한 부분입니다. `Member`와 `Token` 엔티티를 통해 이를 효과적으로 관리할 수 있습니다. 이들 엔티티 간의 관계는 어플리케이션의 보안과 직접적으로 연결됩니다.

Member 엔티티

`Member` 엔티티는 사용자의 기본 정보를 저장합니다. 사용자의 이메일을 기본키로 사용하여 유니크한 식별자로 활용합니다. 또한, 사용자의 소셜 로그인 정보, 이름, 비밀번호, 역할(관리자 또는 일반 사용자), 공급자(예: Google, Facebook) 및 공급자 ID를 포함합니다. 사용자의 프로필 이미지 경로도 저장할 수 있습니다.

```
@Entity
public class Member {
    @Id
    @Column(length = 191)
    private String email;

    private String oauth2Id;
    private String name;
    private String password;
    private String role;
    private String provider;
    private String providerId;
    private String photoPath;

    // 관계 설정
    @OneToMany(mappedBy = "member", cascade = CascadeTyp
e.ALL, orphanRemoval = true)
    private List<MemberHistory> userHistories = new Arra
yList<>();

    @OneToMany(mappedBy = "member", cascade = CascadeTyp
e.ALL, orphanRemoval = true)
    private List<Recommend> userRecommed = new ArrayList
<>();

    @OneToOne(mappedBy = "member", cascade = CascadeTyp
e.ALL, orphanRemoval = true)
    private Token token;
}
```

Token 엔티티

`Token` 엔티티는 사용자의 세션 및 인증 토큰 정보를 관리합니다. 토큰의 유니크한 ID, 사용자 이메일, 토큰 값, 생성 시간, 만료 시간, 토큰 타입(예: 액세스, 리프레시)을 저장합니다. `Member` 엔티티와 `OneToOne` 관계를 가지며, 이는 사용자당 하나의 토큰 정보만을 가질 수 있음을 의미합니다.

```
javaCopy code
@Entity
public class Token {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, unique = true, length = 19
1)
    private String email;

    private String tokenValue;
    private LocalDateTime createdTime;
    private LocalDateTime expiredTime;
    private String tokenType;

    @OneToOne
    @JoinColumn(name = "email", referencedColumnName =
"email", insertable = false, updatable = false)
    private Member member;
}
```

관계 설명

`Member`와 `Token` 엔티티 사이의 `OneToOne` 관계는 사용자 정보와 토큰 정보 간의 밀접한 연결을 나타냅니다. 이 관계를 통해 사용자는 토큰을 통한 인증 과정에서 자신의 신원을 증명할 수 있으며, 시스템은 사용자의 세션을 안전하게 관리할 수 있습니다. 토큰의 만료 시간은 시스템의 보안을 강화하며, 만료된 토큰은 시스템에서 자동으로 제거되어야 합니다.

이 구조는 웹 어플리케이션의 사용자 인증 및 세션 관리에 필수적이며, 사용자의 보안과 직접적으로 관련됩니다.