2024-04-01 인증 서비스 시작하기

현행 문서: 인증 서비스 시작하기

1 개요

(주의) 2023년 9월 6일 기준으로 작성되었으며,
인증 프로세스가 모두 지원되지 않고 있다.
인증 이후 세션이 발생한 이후 검증하는 프로세스만 지원되고 있다.

본 문서는 am-backend의 support 컴포넌트 중 authentication의 사용 방법을 업무팀 개발자 입장에서 적은 것이다.

2 용어집

접근 제어(access control)
누군가가 무언가를 사용하는 것을 허가하거나 거부하는 기능. 식별 및, 인증, 인가를 포함한다.

식별(identification)
접근 제어 과정의 첫 단계로 행위자가 누구인지 알아내는 과정이다.

인증(authentication)
접근 제어 과정의 두 번째 단계로 식별된 행위자가 시스템에 등록된 사용자인지 확인하는 과정이다.

인가(authorization)
접근 제어의 마지막 단계로 인증된 사용자의 행동을 허가하거나 거부하는 과정이다.

세션(session)
시스템이 사용자에 대해 인증한 결과를 일정한 기간 가리키는 것이다. 본 문서에서는 서비스에서 세션의 영속성을 유지한다.

토큰(token)
세션의 식별자.

글로벌 사용자(global user)
도메인 내의 사용자를 나타낸다.

콘텐트 제공자(content provider)
도메인 서비스를 제공하는 서비스를 말한다.

신원 제공자(identify provider)
행위자를 식별하는 주체이다. 신원 제공자에 따라서 여러 유형의 자격 정보를 활용할 수 있다.

자격 정보(credential)
신원 제공자가 행위자를 식별하기 위한 단서이다. 로그인 ID와 암호 쌍을 예로 들 수 있다.

AWS STS(AWS Security Token Service)
사용자에 대한 권한이 제한된 임시 자격 증명을 요청할 수 있는 웹 서비스이다.

3 빠르게 시작하기

3.1 도메인 서버 작성

settings.gradle에 새로운 서브 프로젝트를 추가한다.

```
// ...
include 'example:authentication:infrastructure:domain:authentication-server'
// ...
```

새 서브 프로젝트를 위한 build.gradle을 작성한다.

```
plugins {
    id 'com.koreanair.java-spring-boot-convention'
}

dependencies {
    implementation project(':support:domain:authentication')
    implementation project(':support:infrastructure:domain:authentication-autoconfigure')

    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.flywaydb:flyway-core'
    implementation 'org.postgresql:postgresql'
    implementation 'software.amazon.awssdk:secretsmanager'
    implementation 'software.amazon.awssdk:sts'
    implementation 'software.amazon.jdbc:aws-advanced-jdbc-wrapper:2.2.2'
}
```

서버의 메인 모듈을 작성한다.

```
@SpringBootApplication
public class ExampleDomainAuthenticationServer {

    public static void main(final String[] args) {
        SpringApplication.run(ExampleDomainAuthenticationServer.class, args);
    }
}
```

`java.time.Clock` 를 빈으로 등록한다.

```
@Configuration
class ClockConfig {

    @Bean
    Clock clock() {
        return Clock.systemUTC();
    }
}
```

인증 도메인의 저장소 인터페이스를 구현한다.

- InHouseIdentityRepository
- ContentProviderRepository
- GlobalUserRepository
- SessionRepository

LoginRepositoryConfig를 작성한다.

```
@Configuration
class LoginRepositoryConfig {

    @Bean
    InHouseIdentityRepository iHouseIdentityRepository(
            final InHouseIdentityCrudRepository inHouseIdentityCrudRepository) {
        return new InHouseIdentitySpringDataRepository(inHouseIdentityCrudRepository);
    }
}
```

ContentProviderRepositoryConfig를 작성한다.

```
@Configuration
class ContentProviderRepositoryConfig {

    @Bean
    ContentProviderRepository contentProviderRepository(
            final ContentProviderCrudRepository contentProviderCrudRepository) {
        return new ContentProviderSpringDataRepository(contentProviderCrudRepository);
    }
}
```

GlobalUserRepositoryConfig를 작성한다.

```
@Configuration
class GlobalUserRepositoryConfig {

    @Bean
    GlobalUserRepository globalUserRepository(
            final GlobalUserCrudRepository globalUserCrudRepository) {
        return new GlobalUserSpringDataRepository(globalUserCrudRepository);
    }
}
```

SessionRepositoryConfig를 작성한다.

```
@Configuration
class SessionRepositoryConfig {

    @Bean
    SessionRepository reactiveSessionRepository(
            final SessionCrudRepository sessionCrudRepository) {
        return new SessionSpringDataRepository(sessionCrudRepository);
    }
}
```

이중 추가로 SessionRepository 구현에 대한 예시는 아래와 같다.

```
class SessionSpringDataRepository implements SessionRepository {
    private final SessionCrudRepository sessionCrudRepository;

    SessionSpringDataRepository(final SessionCrudRepository sessionCrudRepository) {
        this.sessionCrudRepository = Objects.requireNonNull(sessionCrudRepository);
    }

    @Override
    public Token save(
            final Session session) {
        final var entityToStore = toEntity(session);

        final var storedEntity = sessionCrudRepository.save(entityToStore);

        return fromEntity(storedEntity).token();
    }

    @Override
    public Optional<Session> find(
            final Token token) {
        final var entityId = token.value();

        return sessionCrudRepository.findById(entityId)
                .map(SessionSpringDataRepository::fromEntity);
    }

    @Override
    public void delete(final Token token) {
        final var entityId = token.value();

        sessionCrudRepository.deleteById(entityId);
    }

    private static SessionEntity toEntity(final Session it) {
        return new SessionEntity(
                it.token().value(),
                it.issuedAt(),
                it.expiredAt(),
                it.subject().value(),
                it.issuer().value(),
                it.audience(),
                it.userAgent().osFamily(),
                it.userAgent().osVersion(),
                it.userAgent().agentFamily(),
                it.userAgent().agentVersion(),
                it.userAgent().deviceFamily());
    }

    private static Session fromEntity(final SessionEntity it) {
        return new Session(
                new Token(it.getToken()),
                it.getIssuedAt(),
                it.getExpiredAt(),
                new GlobalUserId(it.getSubject()),
                new GlobalUserId(it.getIssuer()),
                it.getAudience(),
                new UserAgent(
                        it.getUserAgentOsFamily(),
                        it.getUserAgentOsVersion(),
                        it.getUserAgentAgentFamily(),
                        it.getUserAgentAgentVersion(),
                        it.getUserAgentDeviceFamily()
                )
        );
    }
}
```

```
interface SessionCrudRepository extends CrudRepository<SessionEntity, UUID> {
}
```

```
@Entity
@Table(name = "session")
class SessionEntity {
    @Id
    private UUID token;
    @Column
    private Instant issuedAt;
    @Column
    private Instant expiredAt;
    @Column
    private UUID subject;
    @Column
    private UUID issuer;
    @Column
    private String audience;
    @Column
    private String userAgentOsFamily;
    @Column
    private String userAgentOsVersion;
    @Column
    private String userAgentAgentFamily;
    @Column
    private String userAgentAgentVersion;
    @Column
    private String userAgentDeviceFamily;

    SessionEntity(
            final UUID token,
            final Instant issuedAt,
            final Instant expiredAt,
            final UUID subject,
            final UUID issuer,
            final String audience,
            final String userAgentOsFamily,
            final String userAgentOsVersion,
            final String userAgentAgentFamily,
            final String userAgentAgentVersion,
            final String userAgentDeviceFamily
    ) {
        this.token = Objects.requireNonNull(token);
        this.issuedAt = Objects.requireNonNull(issuedAt);
        this.expiredAt = Objects.requireNonNull(expiredAt);
        this.subject = Objects.requireNonNull(subject);
        this.issuer = Objects.requireNonNull(issuer);
        this.audience = Objects.requireNonNull(audience);
        this.userAgentOsFamily = Objects.requireNonNull(userAgentOsFamily);
        this.userAgentOsVersion = Objects.requireNonNull(userAgentOsVersion);
        this.userAgentAgentFamily = Objects.requireNonNull(userAgentAgentFamily);
        this.userAgentAgentVersion = Objects.requireNonNull(userAgentAgentVersion);
        this.userAgentDeviceFamily = Objects.requireNonNull(userAgentDeviceFamily);
    }

    protected SessionEntity() {
    }

    UUID getToken() {
        return token;
    }

    Instant getIssuedAt() {
        return issuedAt;
    }
```

```
    Instant getExpiredAt() {
        return expiredAt;
    }

    UUID getSubject() {
        return subject;
    }

    UUID getIssuer() {
        return issuer;
    }

    String getAudience() {
        return audience;
    }

    String getUserAgentOsFamily() {
        return userAgentOsFamily;
    }

    String getUserAgentOsVersion() {
        return userAgentOsVersion;
    }

    String getUserAgentAgentFamily() {
        return userAgentAgentFamily;
    }

    String getUserAgentAgentVersion() {
        return userAgentAgentVersion;
    }

    String getUserAgentDeviceFamily() {
        return userAgentDeviceFamily;
    }
}
```

로컬 환경에서는 추가로 PostgreSQL을 구동한다.
만약 기존에 활용하던 컨테이너가 있다면 데이터베이스만 추가하여 활용해도 좋다.

```
nerdctl run \
-p 5432:5432 \
-e "POSTGRES_USER=user" \
-e "POSTGRES_PASSWORD=pass" \
-e "POSTGRES_DB=authentication" \
-d \
postgres:15.2
```

실행에 필요한 환경 변수는 아래와 같다.

```
SPRING_DATASOURCE_URL: PostgreSQL
SPRING_DATASOURCE_USERNAME: PostgreSQL
SPRING_DATASOURCE_PASSWORD: PostgreSQL
SPRING_JPA_HIBERNATE_DDLAUTO: JPA   DDL
```

개발자 로컬 환경에서의 서버 실행 명령어는 아래와 같다.

```
SPRING_DATASOURCE_URL=jdbc:postgresql://localhost:5432/authentication \
SPRING_DATASOURCE_USERNAME=user \
SPRING_DATASOURCE_PASSWORD=pass \
SPRING_JPA_HIBERNATE_DDLAUTO=validate \
./gradlew :example:authentication:infrastructure:domain:authentication-server:bootRun
```

3.2 게이트웨이 서버 작성

settings.gradle에 새로운 서브 프로젝트를 추가한다.

```
// ...
include 'example:authentication:infrastructure:gateway:authentication-server'
// ...
```

새 서브 프로젝트를 위한 build.gradle을 작성한다.

```
plugins {
    id 'com.koreanair.java-spring-boot-convention'
}

dependencies {
    implementation project(':support:domain:authentication')
    implementation project(':support:infrastructure:domain:authentication-client')
    implementation project(':support:infrastructure:gateway:authentication-autoconfigure')

    implementation 'org.springframework.cloud:spring-cloud-starter-gateway'
}
```

서버 실행을 위한 메인 모듈을 작성한다.

```
@SpringBootApplication
public class ExampleAuthenticationGatewayServer {

    public static void main(final String[] args) {
        SpringApplication.run(ExampleAuthenticationGatewayServer.class, args);
    }
}
```

실행에 필요한 환경 변수는 아래와 같다.

```
SUPPORT_GATEWAY_AUTHENTICATION_BYPASSENABLED:     (: false)
SUPPORT_GATEWAY_AUTHENTICATION_ORIGIN:    Origin( +  + )
SUPPORT_GATEWAY_AUTHENTICATION_APIENTRYPOINT:     API
SUPPORT_GATEWAY_AUTHENTICATION_UPSTREAMAPIENTRYPOINT:   API
SUPPORT_DOMAIN_AUTHENTICATION_CLIENT_BASEURL:
```

일반적인 개발자 로컬 환경에서의 서버 실행 명령어는 아래와 같다.

```
SPRING_DATASOURCE_URL=jdbc:postgresql://localhost:5432/authentication \
SPRING_DATASOURCE_USERNAME=user \
SPRING_DATASOURCE_PASSWORD=pass \
SPRING_JPA_HIBERNATE_DDLAUTO=validate \
./gradlew :example:authentication:infrastructure:domain:authentication-server:bootRun
```

3.2.1 (필요시) 스텁 세션 구성

운영 환경이 아닌 곳에서 고정적인 세션을 유지하여 개발 과정을 보다 더 원활하게 해야 할 필요가 있을 수 있다.
이런 경우에만 추가로 스텁 세션을 구성한다.

스텁 세션 구성 작성

```
@Configuration
class StubTokenVerificationReactiveServiceConfiguration {

    @Bean
    @Primary
    TokenVerificationReactiveService compositeTokenVerificationReactiveService(
            @SupportTokenVerificationReactiveService final TokenVerificationReactiveService supportOne) {
        final String adminGlobalUserIdUuidString = "E4AFBE21-0FED-4A70-98A8-E3A11E96CB22";
        final var sessionsBuilder =
                new SessionsBuilder(configurer ->
                        configurer
                                .issuedAt(Instant.MIN)
                                .expiredAt(Instant.MAX)
                                .audience("<http://localhost:8081>")
                                .issuer(adminGlobalUserIdUuidString)
                                .userAgent(new UserAgent("Mac OS X", "10,15,7", "Edge", "116,0,1938", "Mac")));

        // session for admin
        final String adminTokenUuidString = "798FDCF9-ED69-4B7C-8043-E90232DCBD50";
        sessionsBuilder.add(builder -> builder.token(adminTokenUuidString).subject
(adminGlobalUserIdUuidString));

        // session for user
        final String userTokenUuidString = "7F98CB7D-8623-4E54-8971-3EE40CC5A02B";
        final String userGlobalUserIdUuidString = "1DE51F73-3C41-47FF-A483-E753C28CD5D8";
        sessionsBuilder.add(builder -> builder.token(userTokenUuidString).subject(userGlobalUserIdUuidString));

        final var stubOne = new StubTokenVerificationReactiveService(sessionsBuilder.build());

        return new CompositeTokenVerificationService(
                stubOne,
                supportOne);
    }
}
```

3.2.2 (필요시) 추가 세션 구성

기본적으로 제공되는 세션 외에 공통적으로 필요한 정보는 AdditionalSessionConfigurer를 확장하여 구현한다.
이후 AdditionalSessionConfigurer를 구현한 클래스는 빈으로 등록하면,
자동 구성을 통해 집계 서버로 추가 세션 정보가 전달된다.
아래는 관련된 자동 구성의 일부이다.

```
@AutoConfiguration
@Import(SupportGatewayAuthenticationPropertiesConfig.class)
class SupportGatewayAuthenticationAutoConfiguration {

    // ...

    @Bean
    SessionEncoder sessionEncoder(
            final ObjectMapper objectMapper,
            final Optional<AdditionalSessionConfigurer> additionalSessionConfigurer) {
        return additionalSessionConfigurer
                .map(it -> new DefaultSessionEncoder(objectMapper, it))
                .orElseGet(() -> new DefaultSessionEncoder(objectMapper));
    }

    // ...
}
```

3.3 집계 서버의 핸들러 메소드에서 세션 사용

집계 서버의 build.gradle에 아래와 같이 의존성을 추가한다.

```
// ...

dependencies {
    // ...

    implementation project(':support:infrastructure:aggregation:session-resolver')

    // ...
}
```

세션 사용이 필요한 컨트롤러의 핸들러 메소드에 아래와 같이 매개 변수를 선언한다.

```
@RestController
@RequestMapping("/some")
class SomeController {

    @GetMapping
    List<Some> retrieveAll(
            @RequestSession final Session session) {
        // ..
    }
}
```

3.3.1 (필요시) 집계 서버의 핸들러 메소드에서 추가 세션 사용

추가 세션 사용이 필요한 컨트롤러의 핸들러 메소드에 아래와 같이 매개 변수를 선언한다.

```
@RestController
@RequestMapping("/some")
class SomeController {

    @GetMapping
    List<Some> retrieveAll(
            // ...
            @RequestAditionalSession final InfoDefinedByYou additionalSession) {
        // ..
    }
}
```

4 기능

4.1 도메인 서버

4.1.1 세션 생성

Ref. com.koreanair.support.authentication.SessionService#issue
Ref. com.koreanair.support.authentication.SessionReactiveService#issue

```
curl localhost:8082/sessions \
        -X POST \
        -H 'Content-Type: application/json' \
        -H 'Accept: application/json' \
        -d @- <<EOF
{
  "sessionLifeTime": "PT1H",
  "audience": "<http://localhost:8081>",
  "userAgent": {
    "osFamily": "osFamily",
    "osVersion": "osVersion",
    "agentFamily": "agentFamily",
    "agentVersion": "agentVersion",
    "deviceFamily": "deviceFamily"
  },
  "subject": {
    "value": "A7FA49AE-7570-430D-ABB3-F03F7E11C309"
  },
  "issuer": {
    "value": "D23F3D6F-8E60-4811-BD86-7B62BB987376"
  }
}
EOF
```

4.1.2 세션 조회

Ref. com.koreanair.support.authentication.SessionService#retrieve
Ref. com.koreanair.support.authentication.SessionReactiveService#retrieve

```
curl localhost:8082/sessions/e87ac830-3fdb-4e80-a6d9-8b8b50c48a36 \
        -X GET \\
        -H 'Accept: application/json'
```

(주의) 이 기능은 단순 조회 기능으로 세션이 현재 요청과 관련된 것인지 확인하지 않는다.

4.1.3 세션 파기

Ref. com.koreanair.support.authentication.SessionService#revoke
Ref. com.koreanair.support.authentication.SessionReactiveService#revoke

```
curl localhost:8082/sessions/e87ac830-3fdb-4e80-a6d9-8b8b50c48a36 \
        -X DELETE
```

4.1.4 세션 검증

Ref. com.koreanair.support.authentication.TokenVerificationService#invoke
Ref. com.koreanair.support.authentication.TokenVerificationReactiveService#invoke

```
curl localhost:8082/validate \
        -X POST \
        -H 'Content-Type: application/json' \
        -H 'Accept: application/json' \
        -H 'Origin: <http://localhost:8081>' \
        -d @- <<EOF
{
  "value": "d35e0f0b-f58e-4945-ac21-1fc48b88897c"
}
EOF
```

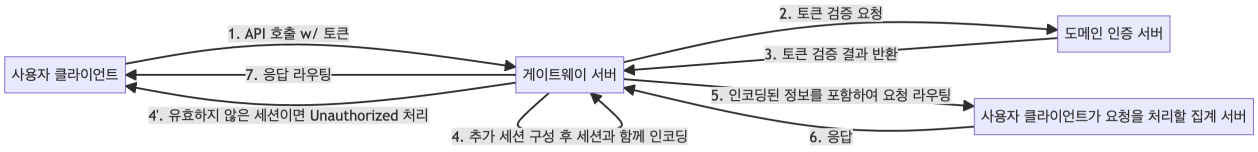
4.1.5 자체 ID/PW 로그인

Ref. com.koreanair.support.authentication.InHouseLoginService#login

```
curl localhost:8082/authenticate \
        -X POST \
        -H 'Content-Type: application/json' \
        -H 'Accept: application/json' \
        -H 'Origin: <http://localhost:8081>' \
        -H 'User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/116.0.0.0 Safari/537.36 Edg/116.0.1938.69' \
        --url-query 'type=in_house' \
        -d @- <<EOF
{
  "id": "login",
  "password": "pass"
}
EOF
```

4.2 게이트웨이 서버

4.2.1 토큰 검증 후 API 라우팅



(link)

```
curl localhost:8081/api/hello \
        -H 'Authorization: Bearer 7F98CB7D-8623-4E54-8971-3EE40CC5A02B'
```

5 설명

5.1 도메인 서버

- 주 관심사는 인증을 지원하기 위한 서비스 구현이다. (support:domain:authentication)
- 범업무적으로 작성된 자동 구성을 사용한다. (support:infrastructure:**domain**:authentication-autoconfigure)
    - 서비스 구성, API 노출이 기본적으로 구성된다.
    - 하지만, 도메인 저장소 구현은 사용 측에서 진행해야 한다.
- 도메인 서버는 Spring WebMVC 기반으로 동작한다. (org.springframework.boot:spring-boot-starter-web)
- 도메인 저장소 구현을 위해서 기본적으로 Spring Data JPA를 사용한다. (org.springframework.boot:spring-boot-starter-data-jpa)
    - PostgreSQL을 위한 JDBC 드라이버를 추가한다. (org.postgresql:postgresql)
    - AWS JDBC 드라이버 래퍼를 추가한다. (aws-advanced-jdbc-wrapper)
        - AWS의 Secrets Manger SDK를 추가한다. (software.amazon.awssdk:secretsmanager)
        - AWS의 STS SDK를 추가한다. (software.amazon.awssdk:sts)
    - RDBMS의 스키마 관리를 위하여 Flyway를 사용한다. (org.flywaydb:flyway-core)

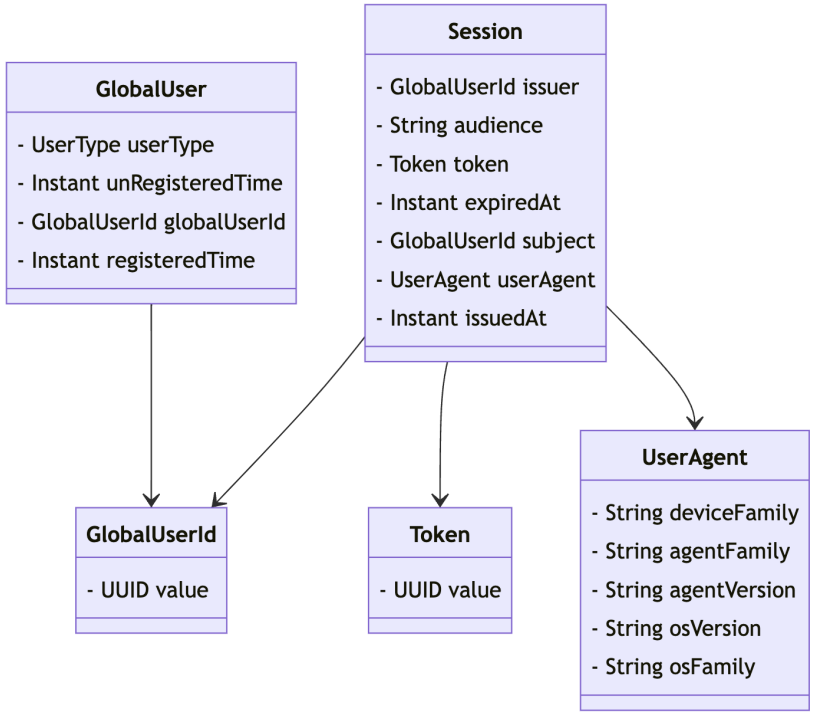
5.2 게이트웨이 서버

- 주 관심사는 인증을 지원하기 위한 서비스를 사용하는 것이다. (support:domain:authentication)
- 범업무적으로 작성된 자동 구성을 사용한다. (support:infrastructure:**gateway**:authentication-autoconfigure)
- 도메인 서비스는 클라이언트 자동 구성 사용을 권장한다. (support:infrastructure:domain:authentication-client)
- 게이트웨이 서버는 Spring Cloud Gateway 기반으로 동작한다. (org.springframework.cloud:spring-cloud-starter-gateway)
- 고정적인 세션을 제공해야 하는 경우 TokenVerificationReactiveService를 통해 구현할 수 있다.
    - 세션 생성을 돕기 위한 유틸 클래스가 있다. (com.koreanair.support.authentication.SessionsBuilder)
    - 고정적인 세션만으로 동작하는 토큰 인증 서비스가 있다. (com.koreanair.support.authentication.
      StubTokenVerificationReactiveService)
    - 다수의 토큰 인증 서비스를 결합하는 구현이 있다. (com.koreanair.support.authentication.CompositeTokenVerificationService)

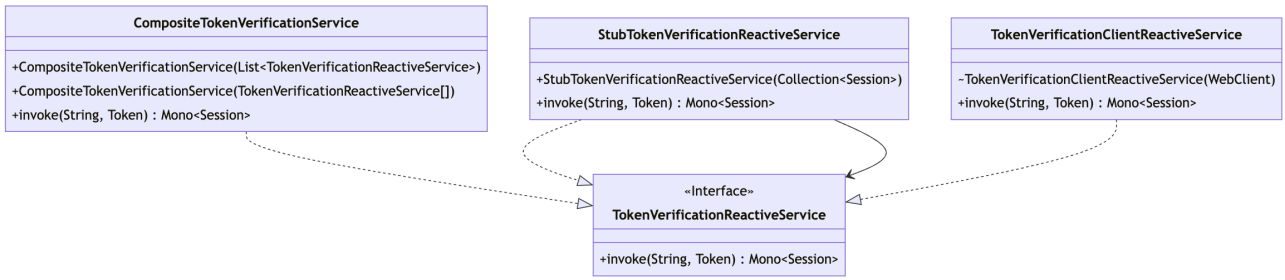
5.3 글로벌 사용자와 콘텐트 제공자

- 글로벌 사용자는 애플리케이션 내에서 유일한 사용자를 가리킨다.
- 콘텐트 제공자는 애플리케이션 내에서 유일한 서비스를 가리킨다.
    - 게이트웨이마다 콘텐트 제공자가 하나씩 설정된다.
- 인증은 글로벌 사용자가 요청하고 콘텐트 제공자 별로 진행된다.

5.4 글로벌 사용자, 세션, 토큰



(link)



(link)

.

- 세션은 인증을 요청한 글로벌 사용자와 인증된 글로벌 사용자의 정보를 가지고 있다.
- 세션은 생성 시점과 만료 시점을 가진다.
- 세션은 인증을 요청한 클라이언트의 정보를 포함하고 있다.
- 세션은 만료 전에 폐기될 수 있다.

6 참조 문헌

- https://en.wikipedia.org/wiki/Access_control
- https://en.wikipedia.org/wiki/Session_(computer_science)