

kotlin/KTOR/JPA/QueryDsl based



Revision History

Version	Date	Description	Author
1.0.0	2021.06.19	backend 기술 설명서	강명구

목 차

- Kotlin
- Coroutine
- KTOR
- JPA
- Relation Mapping
- QueryDsl
- QueryDsl4RepositorySupport
- JWT(Json Web Token)
- Bory JWT Token
- KTOR Routing with Auth
- KTOR Routing without Auth
- JPQ/QueryDsl with DB



Kotlin vs Java

1. 변수 상수 및 Nullable 변수 구분

```
var strVar = "" // 변수  
val strVal = "" // 상수
```

```
var strNullable: String? = null (Nullable)
```

```
var strNonNull: String = "" (Non Nullable)
```

```
strNullable?.split("/") (Nullable check)
```

1.kotlin

HYPERLINK
<https://dev-images.tistory.com/36>



2. 객체 초기화

```
val testIntent = Intent(this, SecondActivity::class.java).apply { // 객체 초기화 시  
    초기 작업 수행
```

```
    putExtra("ext1", 1)  
    putExtra("ext2", 2)  
    putExtra("ext3", "3")  
    putExtra("ext4", "4")  
    putExtra("ext5", false)  
}
```

3. Data class (속성을 정의하기 위한 프로퍼티 전용 클래스)

```
data class KotlinData(var s: String?,  
                     var i: Int,
```

Kotlin 의 장점

지금까지 Kotlin과 Java를 비교해봤습니다. Java를 주로 사용하시던 분들이라면 오히려 불편하다고 생각하실 수도 있지만 Kotlin을 써보고 익숙해지면 Java와는 비교할 수 없을 정도로 편하게 사용할 수 있을 것입니다. 이 외에도 함수, For loop, 가변인자 등 자바에서도 사용되지만 사용방법이 다른것들 또는 Extension Function, let/apply/run/with/also block과 같은 Kotlin에서만 사용할 수 있는 것들이 많이 있어서 이런것들에 익숙해지신다면 훨씬 좋은 개발 능력을 기를 수 있을것이라고 생각합니다.

2. Coroutine

[HYPERLINK
https://developer.android.com/kotlin/coroutines/coroutines-adv?hl=ko](https://developer.android.com/kotlin/coroutines/coroutines-adv?hl=ko)

[HYPERLINK
http://www.gisdeveloper.co.kr/?p=10279](http://www.gisdeveloper.co.kr/?p=10279)

Coroutine is async event routine and is like thread but is not a thread: a async job lighter than java thread.

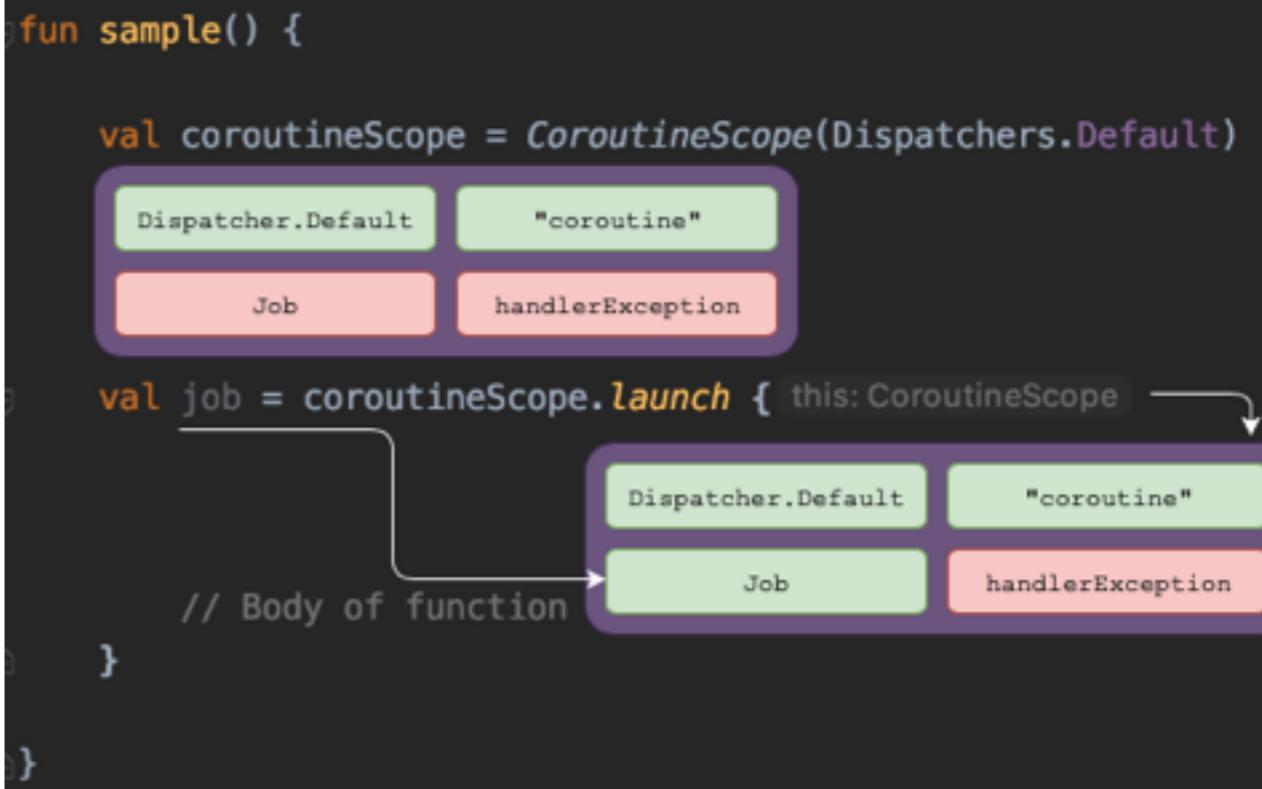
코루틴은 스레드와 기능적으로 같지만, 스레드에 비교하면 좀 더 가볍고 유연하며 한단계 더 진화된 병렬 프로그래밍을 위한 기술입니다. 하나의 스레드 내에서 여러개의 코루틴이 실행되는 개념인데, 아래의 코드는 동일한 기능을 스레드와 코루틴으로 각각 구현한 코드의 예시입니다.

```
Thread(Runnable {
    for(i in 1..10) {
        Thread.sleep(1000L)
        print("I'm working in Thread.")
    }
}).start()

GlobalScope.launch() {
    repeat(10) {
        delay(1000L)
        print("I'm working in Coroutine.")
    }
}
```



**Kotlin
Coroutines**



3. KTOR

HYPERLINK

"<https://ktor.io/>"<https://ktor.io/>

HYPERLINK

"<https://www.jetbrains.com/>"<https://www.jetbrains.com/>

Generate a Ktor project
If you're starting a new project with Ktor, you can get started quickly using **HYPERLINK** "<https://start.ktor.io/>"start.ktor.io to generate and download your project template.

Plugin for IntelliJ IDEA
If you're using IntelliJ IDEA, you can use the **HYPERLINK** "<https://plugins.jetbrains.com/plugin/16008-ktor>"[ktor](https://plugins.jetbrains.com/plugin/16008-ktor) plugin not only to generate new projects but also to get significantly more functionality.

Plugin for IntelliJ IDEA

If you're using IntelliJ IDEA, you can use the **HYPERLINK** "<https://plugins.jetbrains.com/plugin/16008-ktor>"[ktor](https://plugins.jetbrains.com/plugin/16008-ktor) plugin not only to generate new projects but also to get significantly more functionality.

Artifacts

Whether you're using **HYPERLINK** "<https://gradle.org/>"[Gradle](https://gradle.org/) or **HYPERLINK** "<https://maven.apache.org/>"[Maven](https://maven.apache.org/), you can add Ktor to an existing project by simply referencing the **HYPERLINK** "<https://package-search.jetbrains.com/search?query=ktor>"[dependencies](https://package-search.jetbrains.com/search?query=ktor).



KTOR은 kotlin 기반의 경량의 고속 서버입니다. Java/SpringBoot에 비해서 소스와 라이브러리를 포함한 사이즈가 훨씬 작으며 속도가 빠른 event bus 방식 서버입니다.

HYPERLINK "<http://ktor.io>"ktor.io 및 **HYPERLINK** "<http://jetbrains.com>"jetbrains.com 등의 사이트 등이 모두 KTOR 기반으로 만들어졌으며, intelliJ-IDEA를 만든 jetbrains사에서 Java를 대체하기 위해서 만들어진 kotlin 언어 기반으로 구현되어 있습니다. KTOR은 서버개발에 대한 이해가 약한 초급자들도 간단하게 웹(또는 웹소켓) 서버를 만들 수 있습니다. 다만 ORM(JPA/QueryDSL) 및 Routing에 대한 이해가 있어야 Enterprize에서 사용하는 비즈니스 서버를 만들 수 있습니다. 그러나 이러한 기술의 적용이 Spring에 비해 단순하고 명료해서 배우기 용이합니다.

HYPERLINK

"<https://blog.sapzil.org/2017/11/02/kotlin-jpa-pitfalls/>" "<https://blog.sapzil.org/2017/11/02/kotlin-jpa-pitfalls/>"

HYPERLINK

"<https://effectivesquid.tistory.com/entry/Kotlin-JPA-%EC%82%A%EC%9A%A9%EC%8B%9C-Entity-%EC%A0%95%EC%9D%98>" "<https://effectivesquid.tistory.com/entry/Kotlin-JPA-%EC%82%A%EC%9A%A9%EC%8B%9C-Entity-%EC%A0%95%EC%9D%98>"

HYPERLINK

"<https://velog.io/@javajav28/%EC%97%94%ED%8B%90%ED%8B%90Entity>" "<https://velog.io/@javajav28/%EC%97%94%ED%8B%90%ED%8B%90Entity>"

HYPERLINK

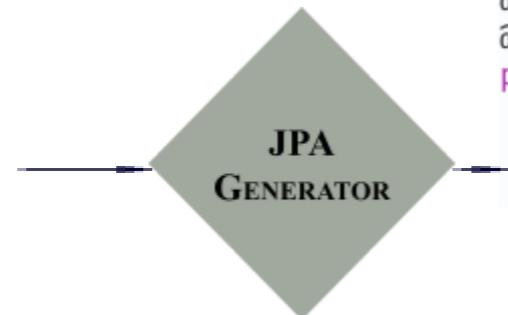
"<https://velog.io/@javajav28/%EC%97%94%ED%8B%90%ED%8B%90Entity>" "



Kotlin JPA에서 가장 중요한 부분은 entity 생성입니다: DB 테이블 생성 후
→ JPA Generator plugin을 이용해서 {Entity}.java를 자동 생성한 후 → JavaToKotlin 자동변환을 이용해서 {Entity}.kt 파일을 만듭니다.

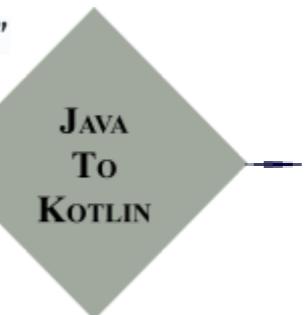
컬럼명	타입	Null
uuid	char(36)	Not null
email_hash	char(64)	Not null
mobile_has_h	char(64)	Not null
Password	varchar(255)	Not Null

DB Table(b1_member)



Java Entity(member.java)

```
@Entity
@Table(Env.tablePrefix + "member")
public class Member {
    @Id
    String uuid;
    String emailHash;
    String mobileHash;
    String password;
}
```



Kotlin Entity(member.kt)

```
@Entity
@Table(Env.tablePrefix + "member")
class Member(
    @Id
    String uuid,
    String emailHas,
    String mobileHash,
    String password,
)
```

HYPERLINK

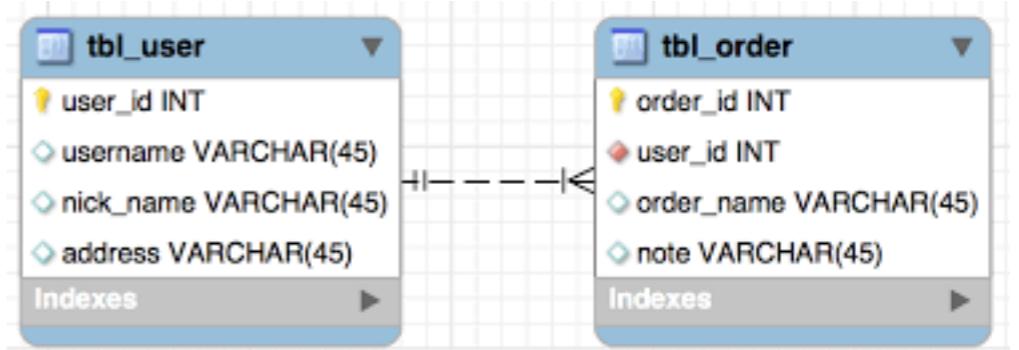
"<https://velog.io/@swchoi0329/Spring-Boot%EC%97%90%EC%84%9C-JPA-%EC%82%AC%EC%9A%A9%E%95%98%EA%B8%B0>" "<https://velog.io/@swchoi0329/Spring-Boot%EC%97%90%EC%84%9C-JPA-%EC%82%AC%EC%9A%A9%ED%95%98%EA%B8%B0>"

JPA/QueryDsl에 대한 개념은
SpringBoot/JPA/QueryDsl 관련 문서를 참고해도
좋습니다. 개념은 동일합니다.

5. Relation Mapping

```
@Entity 부모  
@ApiModel("사용자")  
@Table(name = Env.tablePrefix + "member")  
@JsonInclude(JsonInclude.Include.NON_NULL)  
class Member  
    @Column(name = "mb_email_hash", nullable = false)  
    var mbEmailHash: String = "",  
    ...  
    @OneToOne(fetch = FetchType.LAZY, mappedBy = "member"  
    , cascade = [CascadeType.ALL])  
    var detail: MemberDetail? = null  
): BaseEntity()
```

```
@Entity  
@ApiModel("사용자 상세정보")  
@Table(name = Env.tablePrefix + "member_detail")  
@JsonInclude(JsonInclude.Include.NON_NULL)  
class MemberDetail (  
    @JsonIgnore  
    @OneToOne(fetch = FetchType.LAZY)  
    @JoinColumn(name = "mb_id")  
    var member: Member? = null,  
    @Id  
    @Column(name = "mb_id", nullable = true)  
    var mbId: String? = null,
```



Relation 은 부모 entity(table)과 자식(관련) entity(table)의 의존성(foreign key)을 JPA ORM에서 @annotation으로 표현하는 방법입니다.

@OneToOne 과 @OneToMany 만 존재 합니다. @ManyToOne의 의존관계를 구성하고 계신다면, DB 스키마를 잘못 설계하셨거나 JPA ORM Entity의 의존관계를 잘못 이해하고 계신 겁니다.

부모 entity는 mappedBy로 자식 entity와의 연결을 설정하고 자식은 부모와 @OneToOne(1:1) 또는 @ManyToOne(1:N)로 의존관계를 설정합니다.

@OneToOne 매핑관계이며 자식 entity가 부모 entity의 상세 정보를 별도 관리(성능 및 보안 이유)하는 경우에는 부모의 ID 값을 Foreign key로 해서 ID를 가지고 별도의 고유 ID를 생성하지 않습니다.

6.QueryDsl

QueryDsl4RepositorySupport 클래스:

QueryDsl 주요 기능은 이 Custom Base 클래스가 모두 제공합니다.

위 클래스를 상속받은 {Entity}Repository 클래스에서는 SQL이나 JPQL 대신 QueryDsl 문법대로 Select 쿼리를 작성하기만 하면 됩니다.

https://gitlab.bory.io:90/b2021_everydata/everytalk-backend-server/-/blob/master/src/main/kotlin/io/bory/server/jpa/common/repository/Querydsl4RepositorySupport.kt

```
/**  
 * Querydsl 4.x 버전에 맞춘 Querydsl 지원 라이브러리  
 *  
 */  
@Suppress("UNCHECKED_CAST")  
open class Querydsl4RepositorySupport<T>(val entityManager: EntityManager, val domainClass: KClass<Any>)  
{  
    lateinit var queryFactory: JPAQueryFactory  
    lateinit var builder: PathBuilder<*>  
    // lateinit var querydsl: Querydsl  
    val batchSize = 300  
  
    init {  
        val path: EntityPath<T> = EntityPathBase<T>(domainClass as Class<T>, "entity")  
        builder = PathBuilder(path.type, path.metadata)  
        // querydsl = Querydsl(  
        //     entityManager,  
        //     builder  
        // )  
        queryFactory = JPAQueryFactory(entityManager)  
    }  
    ::::::::::::
```



7. Querydsl4RepositorySupport

```
class MemberRepository : Querydsl4RepositorySupport<Member>(Env.em, Member::class as KClass<Any>) {  
  
    val member: QMember = QMember("member")  
  
    fun findByUuid(uuid: String): Member? {  
        return selectFrom(member).where(member.uuid.eq(uuid)) as Member  
    }  
  
    fun findByEmail(email: String): Member? {  
        return selectFrom(member).where(member.mbEmailHash.eq(email.sha256O)) as Member  
    }  
  
    fun findByMobile(mobile: String): Member? {  
        return selectFrom(member).where(member.mbMobileHash.eq(mobile.sha256O)) as Member  
    }  
}
```

QueryDsl 작성에 있어서 가장 중요한 개념은 `Q{Entity}` 입니다. QueryDsl Entity는 소스 빌드 시점에 java 파일 형태로 자동으로 생성되어 kotlin 코드와 함께 컴파일 됩니다. 자동으로 생성되는 `Q{Entity}` 는 QueryDsl 쿼리에서 Sql 문의 Table 닉네임과 같은 역할을 합니다.

8. JWT(Json Web Token)

base64(\${header}).base64(\${payload}).\${signature}

[헤더.내용.서명] 형식의 토큰으로 규격화 되어 있으며 내용(Payload)은 커스텀이 가능하다. JWT Signature 안에 expire time 값이 같이 암호화 되어 있어서 JWT 토큰의 유효기간을 디코딩 과정에서 동시에 체크할 수 있다. JWT 토큰을 Authorization 헤더에 넣어서 api 인증 토큰(accessToken)으로 사용하면 토큰의 유효성 및 유효기간을 동시에 체크할 수 있어서 편리하다.



[HYPERLINK](https://jwt.io/) "<https://jwt.io/>" <https://jwt.io/> (Json Web Token Online Parser)

[HYPERLINK](#)

<https://github.com/ktorio/ktor/blob/main/ktor-features/ktor-auth-jwt/jvm/src/io/ktor/auth/jwt/JWTAuth.kt> <https://github.com/ktorio/ktor/blob/main/ktor-features/ktor-auth-jwt/jvm/src/io/ktor/auth/jwt/JWTAuth.kt>

[HYPERLINK](https://velopert.com/2389) "<https://velopert.com/2389>" <https://velopert.com/2389>

9. JWT Token

```
package io.bory.server.auth

import com.auth0.jwt.JWT
import com.auth0.jwt.JWTVerifier
import com.auth0.jwt.algorithms.Algorithm
import io.bory.server.jpa.everytalk.dto.UserDto
import java.util.*

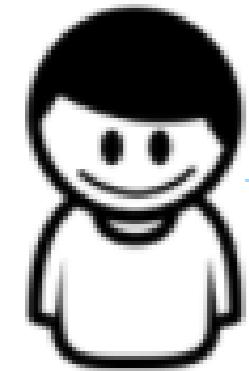
object JwtConfig {

    const val authHeader = "Authorization"
    private const val secret = "xElx1olx8qflc1iYtcRd"
    private const val subject = "sso:::authentication"
    private const val issuer = "everytalk.io"
    private const val audience = "everytalk:::web:::mobile:::client"
    const val realm = issuer
    private const val validityInMs = 36_000_000 // 10 hours
    private val algorithm = Algorithm.HMAC256(secret)

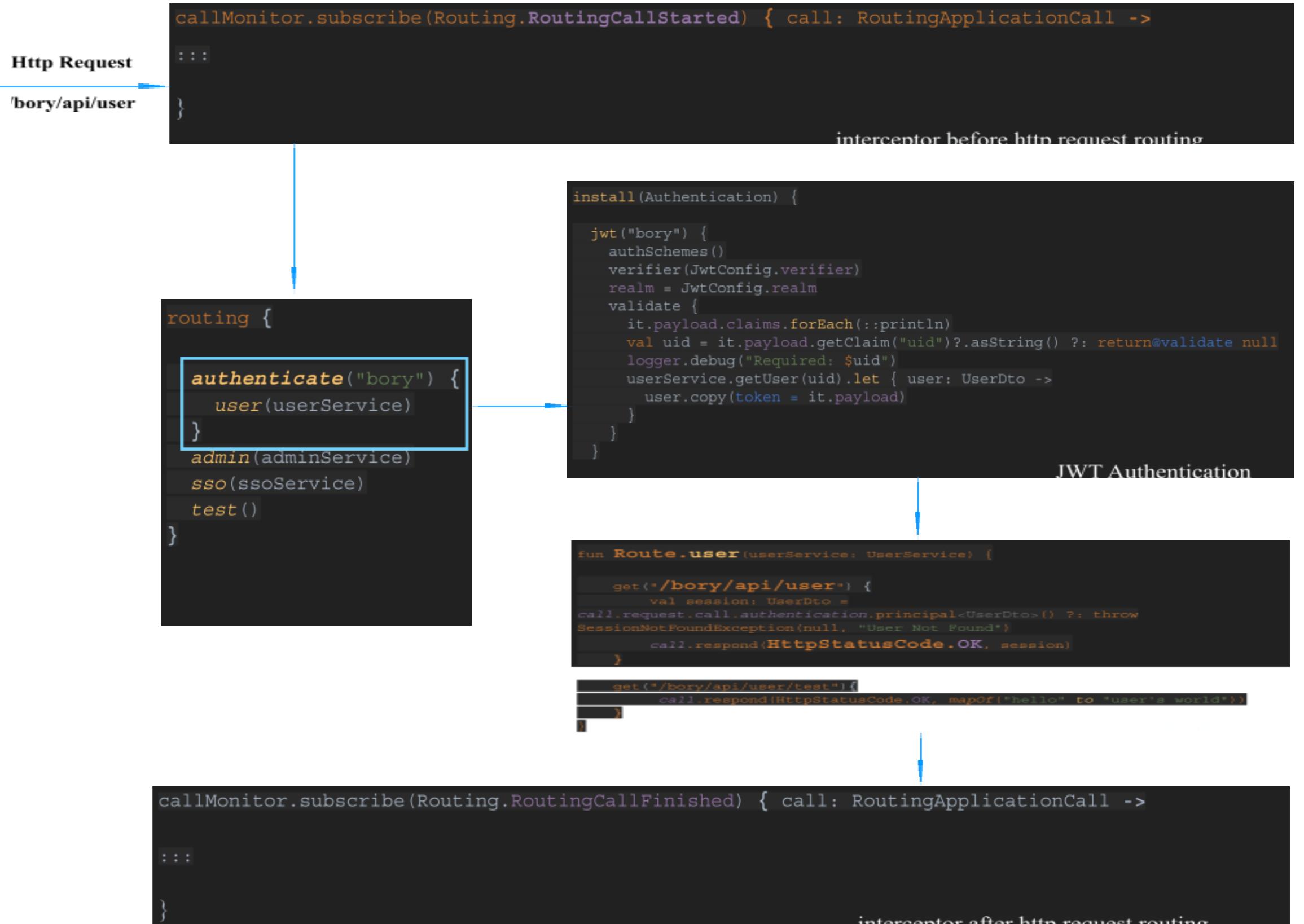
    val verifier: JWTVerifier = JWT
        .require(algorithm)
        .withAudience(audience)
        .withIssuer(issuer)
        .build()

    /**
     * Produce a token for this combination of User and Account
     */
    fun makeToken(user: UserDto, jti: String? = null): String =
        JWT.create()
            .withSubject(subject)
            .withAudience(audience)
            .withIssuer(issuer)
            .withJWTId(if (jti == null) UUID.randomUUID().toString() else
                jti)
            .withClaim("uid", user.id)
            .withClaim("name", user.name)
            .withClaim("level", user.level)
            .withClaim("status", user.status)
            .withExpiresAt(getExpiration())
            .sign(algorithm)
```

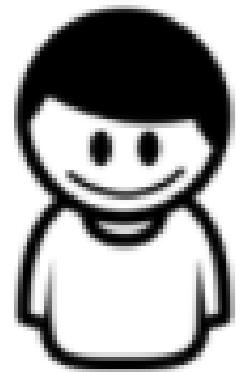
10. KTOR Routing with Auth



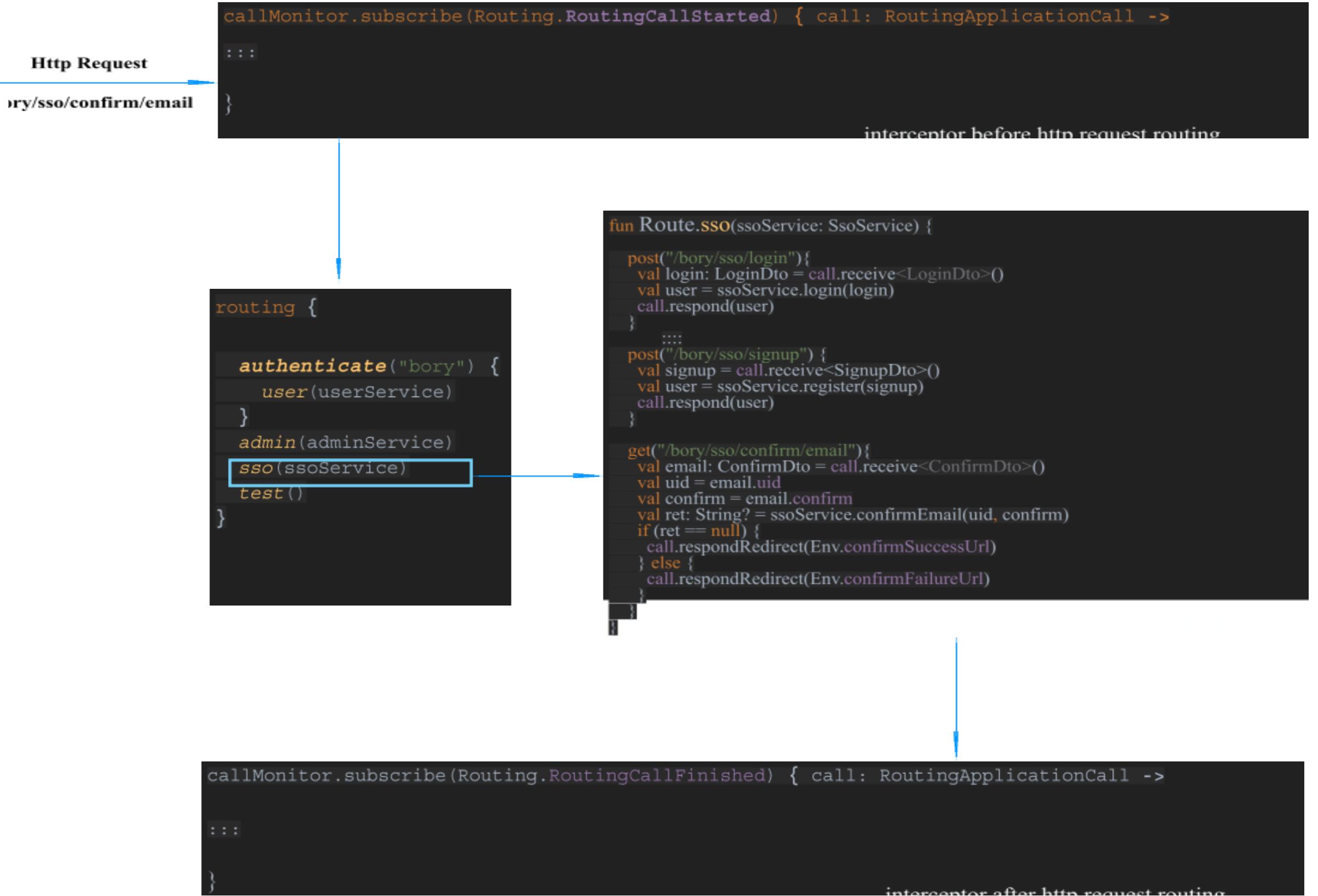
Client



11. KTOR Routing without Auth



Client



12. JPA/QueryDsl with DB

JPA Entity 부터 DB Table 까지 관련 api 연동 순서 :
Entity - Repository - QueryDsl - JPQL - SQL - DB Table

```
package  
io.bory.server.jpa.everytalk.entity  
  
:  
  
@Entity  
@ApiModel("사용자")  
@Table(name = Env.tablePrefix + "member")  
@JsonInclude(JsonInclude.Include.NON_NULL)  
class Member(  
  
    @Column(name = "mb_email_hash",  
    columnDefinition = "char(64)", nullable =  
    false)  
    var mbEmailHash: String = "",  
  
    @Column(name = "mb_mobile_hash",  
    columnDefinition = "char(64)", nullable =  
    false)  
    var mbMobileHash: String = "",  
  
    @Column(name = "mb_password", nullable =  
    false)  
    var mbPassword: String = "",  
  
    @Column(name = "mb_name", nullable =  
    false)  
    var mbName: String = "",  
  
    @ApiModelProperty("member level flat  
(0:guest, 1:user, 10:company, 11:company  
user, 30:agent, 31:agent user,  
1000:admin)")  
    @Column(name = "mb_level", nullable =  
    false)  
    var mbLevel: Int = 1,  
  
    @Column(name = "mb_point", nullable =  
    false)  
    var mbPoint: Long = 0L,
```

```
package io.bory.server.jpa.everytalk.repository  
  
:  
  
class MemberRepository :  
    Querydsl4RepositorySupport<Member>(Env.em,  
    Member::class as KClass<Any>) {  
    val path: PathBuilder<Member> =  
        PathBuilder<Member>(Member::class.java, "member")  
    val member: QMember = QMember.member  
  
    fun findByUuid(uuid: String): Member? {  
        return  
            selectFrom(member).where(member.uuid.eq(uuid)).fe  
            tchOne() as Member?  
    }  
  
    fun findByEmail(email: String): Member? {  
        return  
            selectOne().from(member).where(member.mbMobil  
eHash.eq(email.sha256())).fetchOne() as Member?  
    }  
  
    fun findByMobile(mobile: String): Member? {  
        val m = mobile.replace("-", "")  
        return  
            selectFrom(member).where(member.mbMobileHash.eq(m  
.sha256())).fetchOne() as Member?  
    }  
  
}
```

QueryDsl

Select member1 from Member member1 where
member1.mbMobileHash = ?1

JPQL

Select * from member m where
m.mb_mobile_hash = ?1

b1_member	
id	binary(16)
mb_email_hash	char(64)
mb_mobile_hash	char(64)
mb_password	varchar(255)
mb_name	varchar(64)
mb_level	int(11)
mb_point	bigint(20)
mb_status	int(11)
mb_reg_datetime	datetime
mb_mod_datetime	datetime

Repository