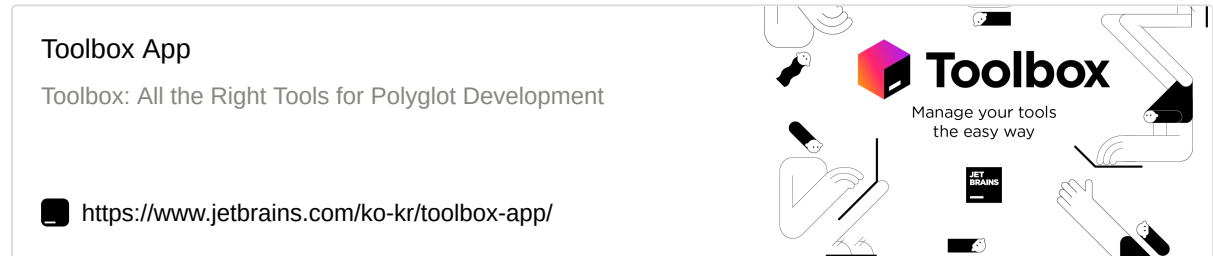


1.1. IntelliJ IDEA 설치하기

1. JetBrains Toolbox App 설치

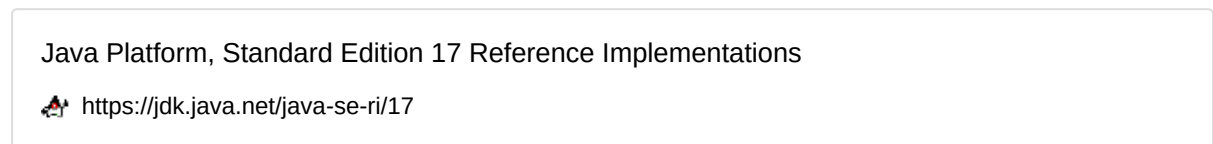


2. IntelliJ IDEA Community Edition 설치

3. OpenJDK 17 설치

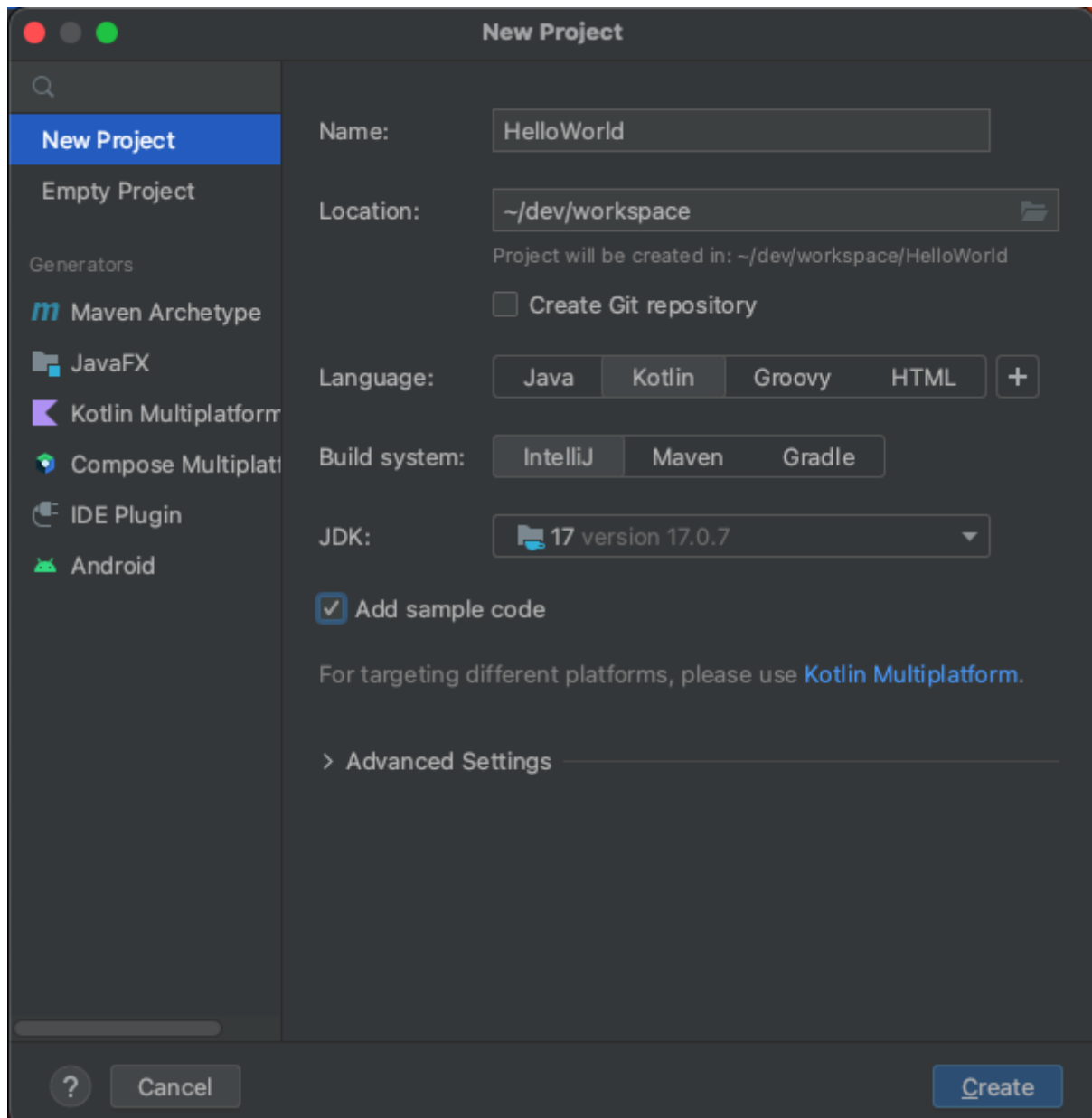
- 터미널 열어서 `brew install openjdk@17` 실행
- 완료 후 나오는 명령어 실행

3.1 Windows OpenJDK 17 설치



- Windows 10 x64 Java Development Kit 를 다운받아 원하는 위치에 압축을 풀기만 하면 끝.

4. HelloWorld 출력 해보기




- Main.kt 파일에서 실행(Ctrl + Shift + R)

```
fun main(args: Array<String>) {  
    println("Hello World!")  
}
```

5. Coding conventions 설정

Coding conventions | Kotlin

 <https://kotlinlang.org/docs/coding-conventions.html#configure-style-in-ide>



스타일 가이드 적용

1. Settings/Preferences > Editor > Code Style > Kotlin
2. Set From... 클릭
3. Kotlin style guide 선택

스타일 가이드 따르는지 확인

1. Settings/Preferences > Editor > Inspections > General
2. Incorrect formatting 켜기

IntelliJ IDEA 단축키

- Ctrl + Shift + . : 글자 크기 키우기
- Ctrl + Shift + , : 글자 크기 줄이기
- Ctrl + Shift + R : 실행
- Ctrl + Shift + D : 디버깅

1.2. 변수 알아보기

형태

```
var 변수명: 타입
```

1. var / val

- var 은 읽기, 쓰기 가능
- val 은 읽기만 가능

```
fun main(args: Array<String>) {  
  
    var i: Int = 10  
    val j: Int = 10  
  
    i = 20  
    //    j = 20    // val은 변경 불가능  
  
    println(i)  
    println(j)  
}
```

2. Int / Int?

- kotlin의 Int는 null을 허용하지 않음
- null을 허용 하려면 타입에 ? 붙여서 선언

```
fun main(args: Array<String>) {  
  
    var i: Int = 10  
    var j: Int? = 10  
  
    //    i = null    // Int는 null을 허용하지 않음  
    j = null  
  
    println(i)  
    println(j)  
}
```

3. String / String?

- kotlin의 String는 null을 허용하지 않음
- null을 허용 하려면 타입에 ? 붙여서 선언

```
fun main(args: Array<String>) {  
  
    var i: String = "ABC"  
    var j: String? = "ABC"  
  
    //    i = null    // String는 null을 허용하지 않음  
    j = null  
  
    println(i)  
    println(j)  
}
```

4. 타입 추론

- kotlin은 타입추론으로 변수에 들어오는 값을 보고 타입을 알아서 지정해줌

```
fun main(args: Array<String>) {  
  
    val s = "ABC"  
    val i = 1  
    val l = 1L  
    val d = 1.0  
    val f = 1.0f  
  
    println("s = " + s::class)  
    println("i = " + i::class)  
    println("l = " + l::class)  
    println("d = " + d::class)  
    println("f = " + f::class)  
}
```

1.3. if 와 when 알아보기

1. if 문

형태

```
if ( 조건식 ) {  
    // 조건식이 true인 경우 실행  
} else {  
    // 조건식이 false인 경우 실행  
}
```

대소 비교

```
fun main(args: Array<String>) {  
  
    val priceA: Int = 100  
    val priceB: Int = 200  
  
    if (priceA >= priceB) {  
        println("priceA = $priceA")  
    } else {  
        println("priceB = $priceB")  
    }  
}
```

null 체크

```
fun main(args: Array<String>) {  
  
    val price: Int? = null  
  
    if (price == null) {  
        println("null check true")  
    } else {  
        println("price = $price")  
    }  
}
```

in 체크

```
fun main(args: Array<String>) {  
  
    val price: Int = 100  
  
    if (price in arrayOf(100, 200, 300)) {  
        println("contain")  
    } else {  
        println("not contained")  
    }  
}
```

2. when 문

형태

```
when ( 변수 ) {  
    조건1 -> 조건1 만족시 실행 후 when 밖으로 이동  
    조건2 -> 조건2 만족시 실행 후 when 밖으로 이동  
    조건3 -> 조건3 만족시 실행 후 when 밖으로 이동  
    // ...  
    else -> 아무것도 만족하지 않을때 실행  
}
```

값 비교

```
fun main(args: Array<String>) {  
  
    val price: Int = 100  
  
    when (price) {  
        100 -> println("1. price = $price")  
        200 -> println("2. price = $price")  
        300 -> println("3. price = $price")  
        else -> println("4. Not")  
    }  
}
```

범위 비교

```
fun main(args: Array<String>) {  
  
    val price: Int = 100  
  
    when (price) {  
        in 100..199 -> println("1. 100 ~ 199")  
        in 200..299 -> println("2. 200 ~ 299")  
        in 300..399 -> println("3. 300 ~ 399")  
        else -> println("4. Not")  
    }  
}
```


1.4. function 알아보기

형태

```
fun 함수명(매개변수: 타입): 반환타입 {  
    // 본문  
    return 반환값  
}
```

예시

```
fun main(args: Array<String>) {  
  
    val price1: Int = 100  
    val price2: Int = 200  
  
    val price3 = sumPrice(price1, price2)  
    println("price3 = $price3")  
}  
  
fun sumPrice(price1: Int, price2: Int): Int = price1 + price2
```

1.5. class 와 Interface 알아보기

1. class

형태

```
class 클래스명 {  
    // 프로퍼티와 메소드  
}
```

생성자

```
fun main(args: Array<String>) {  
  
    val item = Item("BOOK", 10_000)  
    println("Item name is ${item.name}, price is ${item.price}")  
}  
  
class Item(  
    val name: String,  
    val price: Int  
)
```

enum class

```
enum class Color {  
    RED,  
    GREEN,  
    BLUE  
}
```

2. Interface

형태

```
interface 인터페이스명 {  
    fun 함수명()  
}
```

예시

```
fun main(args: Array<String>) {  
  
    val item = Item("BOOK", 10_000)  
    println("Item name is ${item.name}, price is ${item.price}")  
  
    item.buy()  
    item.sell()  
}  
  
class Item(  
    val name: String,  
    val price: Int  
) : ItemTrade {  
    override fun buy() {  
        println("[buy] $name")  
    }  
  
    override fun sell() {  
        println("[sell] $name")  
    }  
}  
  
interface ItemTrade {  
    fun buy()  
    fun sell()  
}
```

2.1. MariaDB 설치하기

1. MariaDB 설치

1. Homebrew 설치

Homebrew

The Missing Package Manager for macOS (or Linux).

 https://brew.sh/index_ko




1. 터미널 열어서 스크립트 실행
 - `/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"`
2. 설치가 끝나면 ==> **Next steps:** 다음에 나오는 명령어들 순서대로 실행
3. `brew --version` 으로 설치가 잘되어 버전 정보 나오는지 확인

2. Homebrew MariaDB 설치

- mac은 mariaDB 설치 파일이 없어서 Homebrew를 사용해서 MariaDB를 설치 해야함

Homebrew로 macOS에 MariaDB Server 설치


Homebrew 패키지 매니저를 이용해서 MariaDB Server를 macOS (이전 Mac OS X) 설치할 수 있습니다. MariaDB Server는 미리 컴파일된 Homebrew "bottle" 패키지로 이용 가능하며, 소스 빌드가 필요 없어 시간을 절약 해줄 수 있습니다. Homebrew 설치 후에는 ...

 <https://mariadb.com/kb/ko/installing-mariadb-on-macos-using-homebrew/>

Windows MariaDB 설치

Download MariaDB Server

REST API Release Schedule Reporting Bugs ... Continue reading


 <https://mariadb.org/download/?t=mariadb&p=mariadb&r=11.1.0>



- 다운받아서 실행하면 끝

2. DBeaver 설치

Download

 <https://dbeaver.io/download/>

2.2. SQL 알아보기

1. create table

- DB에 테이블을 만들 때 사용

```
CREATE TABLE WORDCOUNT (  
  WORD VARCHAR(255) NOT NULL,  
  CNT INT(11) DEFAULT NULL,  
  PRIMARY KEY (`WORD`)  
)
```

2. select

- 테이블에서 데이터를 읽을 때 사용

```
SELECT *  
FROM WORDCOUNT
```

3. insert

- 테이블에 데이터를 넣을 때 사용

```
INSERT INTO WORDCOUNT  
VALUE('코틀린', 1)
```

4. update

- 테이블에 이미 들어가있는 데이터를 변경할 때 사용

```
UPDATE WORDCOUNT  
SET CNT = CNT + 1  
WHERE WORD = '코틀린'
```

5. delete

- 테이블에 있는 데이터를 삭제할 때 사용

```
DELETE FROM WORDCOUNT  
WHERE WORD = '코틀린'
```

3.1. Spring Initializr 써보기

- 기본적인 Spring Project를 만들어서 다운로드

<https://start.spring.io/>

The screenshot shows the Spring Initializr web interface with the following configuration:

- Project:** ☒ Gradle - Kotlin, ☐ Maven
- Language:** ☐ Java, ☒ Kotlin, ☐ Groovy
- Spring Boot:** ☐ 3.1.0 (SNAPSHOT), ☐ 3.1.0 (RC2), ☐ 3.1.0 (M2), ☐ 3.0.7 (SNAPSHOT), ☒ 3.0.6, ☐ 2.7.12 (SNAPSHOT), ☐ 2.7.11
- Project Metadata:**
 - Group: com.example
 - Artifact: kakaoapi
 - Name: kakaoapi
 - Description: Kotlin project for Spring Boot
 - Package name: com.example.kakaoapi
 - Packaging: ☒ Jar, ☐ War
 - Java: ☐ 20, ☒ 17, ☐ 11, ☐ 8
- Dependencies:**
 - Spring Web** WEB: Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
 - Spring Data JPA** SQL: Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.
 - MariaDB Driver** SQL: MariaDB JDBC and R2DBC driver.
 - Spring Boot DevTools** DEVELOPER TOOLS: Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

3.2. build.gradle.kts 셋팅하기


1. plugins

- 미리 구성해둔 task들의 그룹
- 빌드과정에서 필요한 정보들을 포함하고 있으며 필요에 따라 커스터 마이징 가능

2. dependencies

- 의존성 관리
- Maven Repository 에서 필요한 것 검색 후 추가

Maven Repository: Search/Browse/Explore

 <https://mvnrepository.com/>

3. JPA 사용하기 위해 allOpen, noArg 추가

allOpen

- plugin.spring에서 Open 해주는 것 외에 추가로 Open 해줄 것 명시

```
allOpen {  
    annotation("jakarta.persistence.Entity")  
    annotation("jakarta.persistence.MappedSuperclass")  
    annotation("jakarta.persistence.Embeddable")  
}
```

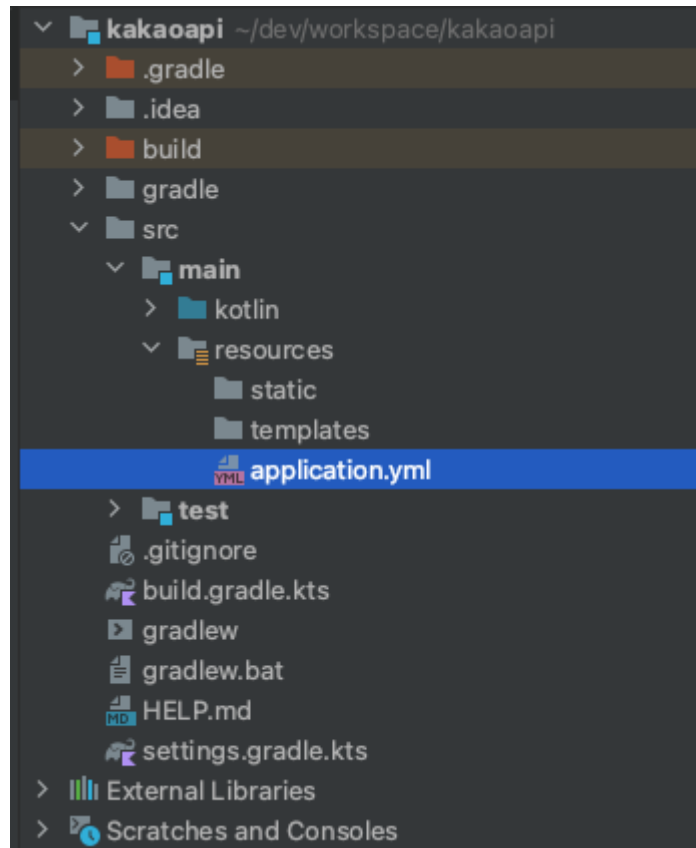
noArg

- 매개변수가 없는 생성자를 자동으로 추가

```
noArg {  
    annotation("jakarta.persistence.Entity")  
}
```

3.3. application.yml 셋팅하기

1. resources 아래에 yml 파일 생성



2. application.yml

- server : 서버 관련 설정
- spring.datasource : Database 접속 정보
- spring.jpa : jpa 설정 정보
- logging : 로그에 관한 정보

```
server:
  port: 8080
  servlet:
    context-path: /
    encoding:
      charset: UTF-8
      force: true
spring:
  datasource:
    driver-class-name: org.mariadb.jdbc.Driver
    url: jdbc:mariadb://localhost:3306/study
    username: study
    password: study123
  jpa:
    open-in-view: true
    hibernate:
      ddl-auto: create
      use-new-id-generator-mappings: false
    properties:
      hibernate:
        show_sql: false
        format_sql: true
        highlight_sql: true
  logging:
    pattern:
      console: "[%d{HH:mm:ss.SSS}][%-5level][%logger.%method:line%line] - %msg%n"
    level:
      org:
        hibernate:
          type.descriptor.sql: trace
          SQL: debug
```

4.1. kakao developers 써보기

1. kakao developers 접속

Kakao Developers

카카오 API를 활용하여 다양한 어플리케이션을 개발해보세요. 카카오 로그인, 메시지 보내기, 친구 API, 인공지능 API 등을 제공합니다.

 <https://developers.kakao.com/>

kakao developers

2. 사용할 API 문서 확인

- 로그인 > 문서 > 좌측 검색의 Daum 검색 > Rest API > 블로그 검색하기

블로그 검색하기

기본 정보

```
GET /v2/search/blog HTTP/1.1
Host: dapi.kakao.com
Authorization: KakaoAK ${REST_API_KEY}
```

다음 블로그 서비스에서 질의어로 게시물을 검색합니다. 원하는 검색어와 함께 결과 형식 파라미터를 선택적으로 추가할 수 있습니다. 응답 바디는 `meta`, `documents` 로 구성된 `JSON` 객체입니다.

Request

Parameter

Name	Type	Description	Required
query	String	검색을 원하는 질의어 특정 블로그 글만 검색하고 싶은 경우, 블로그 url과 검색어를 공백(' ') 구분자로 넣을 수 있음	O
sort	String	결과 문서 정렬 방식, accuracy(정확도순) 또는 recency(최신순), 기본 값 accuracy	X
page	Integer	결과 페이지 번호, 1~50 사이의 값, 기본 값 1	X
size	Integer	한 페이지에 보여질 문서 수, 1~50 사이의 값, 기본 값 10	X

3. REST API KEY 받기

1. 상단에 “내 애플리케이션” 클릭
2. “애플리케이션 추가하기” 클릭

전체 애플리케이션 (1)

애플리케이션 이름

+

애플리케이션 추가하기

3. 정보 입력 후 저장

애플리케이션 추가하기

앱 아이콘

이미지
업로드

파일 선택

JPG, GIF, PNG
권장 사이즈 128px, 최대 250KB

앱 이름

study

사업자명

study

✓

[서비스 이용이 제한되는 카테고리](#), [금지된 내용](#), [금지된 행동](#) 관련 운영정책을 위반하지 않는 앱
입니다.

취소

저장

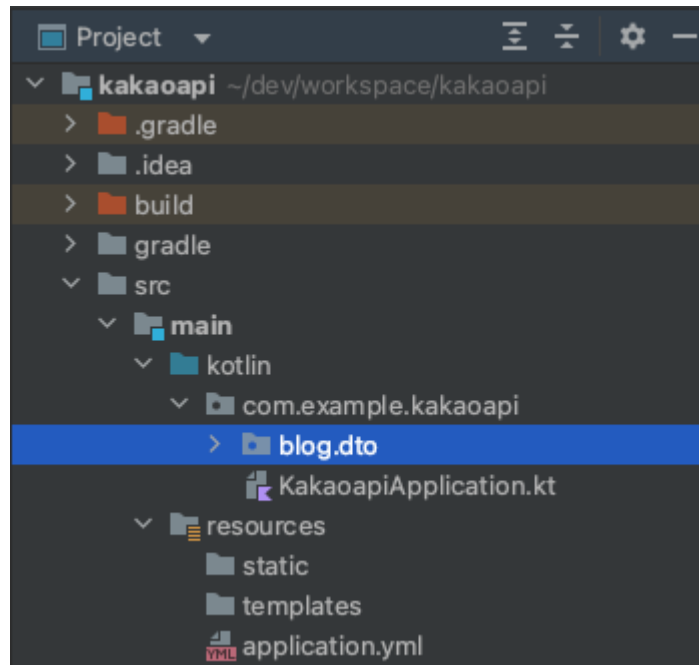
4. REST API 키 값 확인

앱 키

네이티브 앱 키	8a254443ed50591c30bd30
REST API 키	34ecd1ab62304546495dd

4.2. DTO 만들기

1. 패키지 만들기



2. data class 만들기

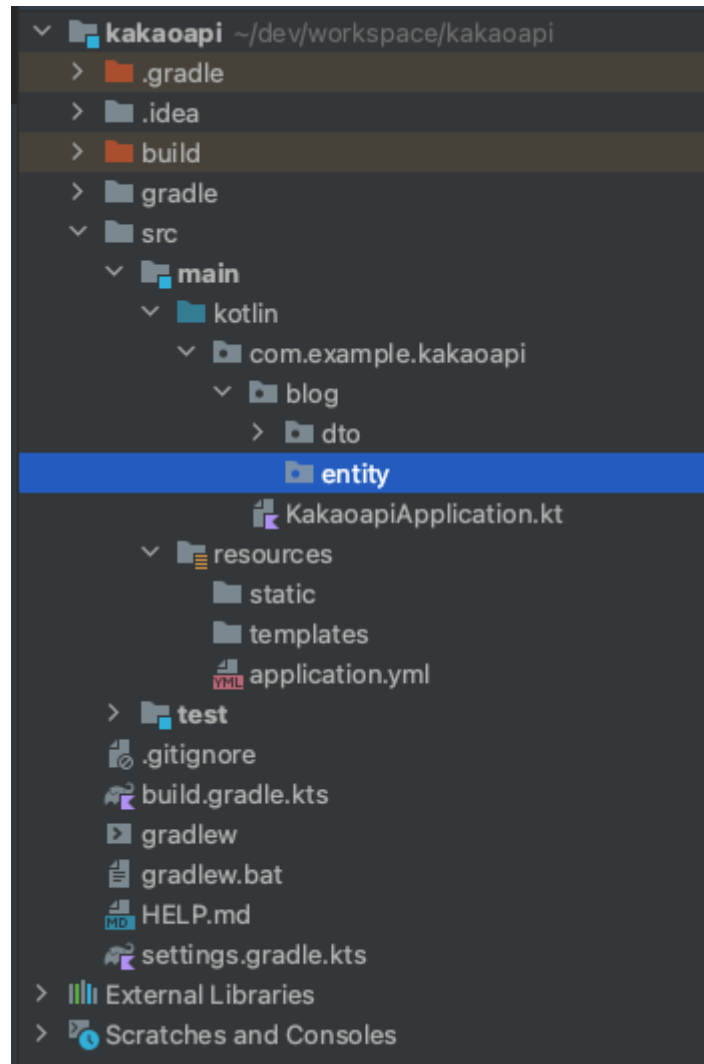
- 사용자로부터 받은 Request를 담아서 controller, service 에서 데이터를 주고받을 때 사용

```
package com.example.kakaoapi.blog.dto

data class BlogDto(
    val query: String,
    val sort: String,
    val page: Int,
    val size: Int
)
```

4.3. entity 만들기

1. 패키지 만들기



2. Entity 만들기

- 데이터베이스 테이블에 매핑되어 테이블에 CRUD할 때 사용

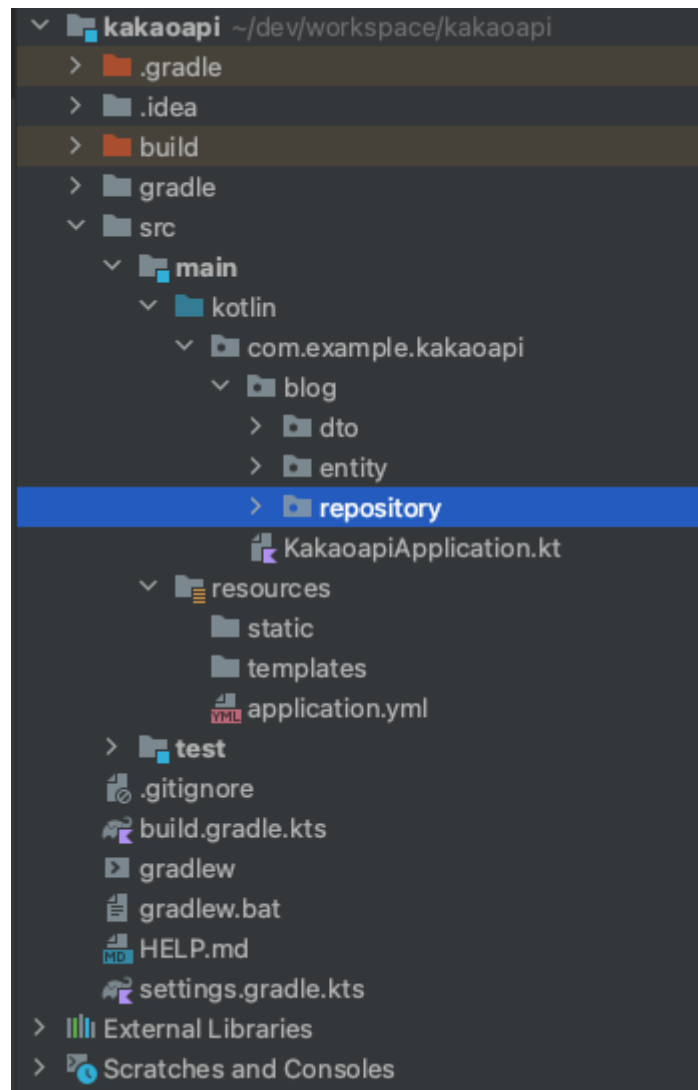
```
package com.example.kakaoapi.blog.entity

import jakarta.persistence.Entity
import jakarta.persistence.Id

@Entity
class Wordcount(
    @Id val word: String,
    var cnt: Int = 0
)
```

4.4. repository 만들기

1. 패키지 만들기



2. Repository 만들기

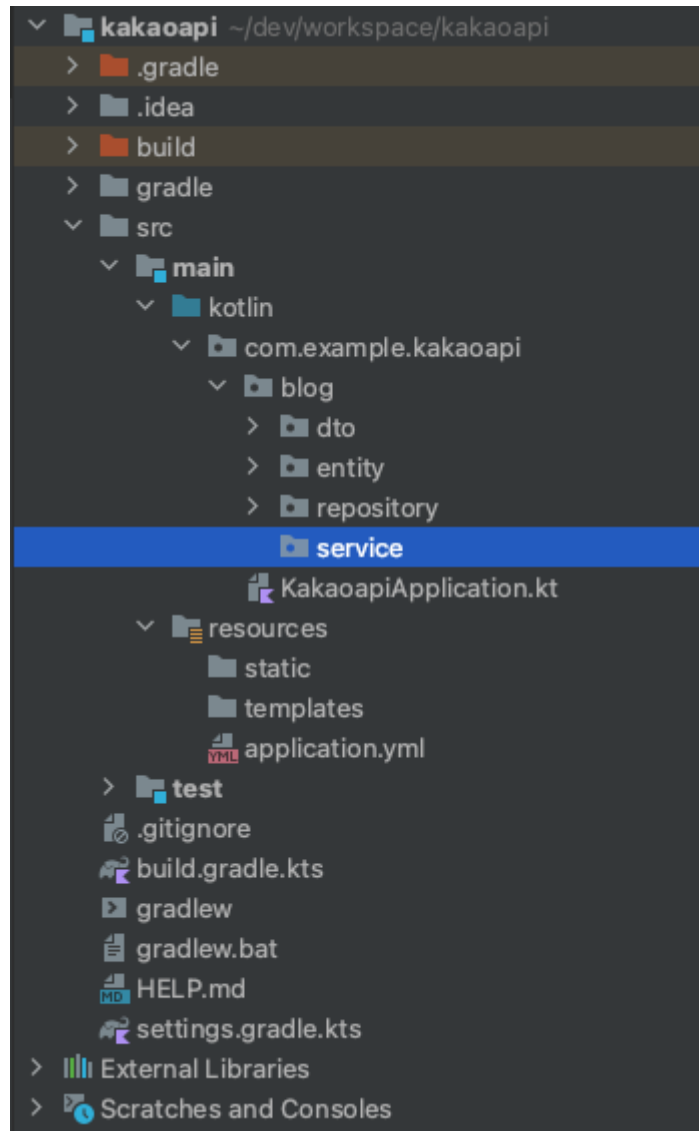
- entity를 가지고 CRUD

```
package com.example.kakaoapi.blog.repository
import com.example.kakaoapi.blog.entity.Wordcount
import org.springframework.data.repository.CrudRepository

interface WordRepository : CrudRepository<Wordcount, String>
```

4.5. service 만들기

1. 패키지 만들기



2. Service 만들기

- 일반적인 비즈니스 로직이 들어가는 부분

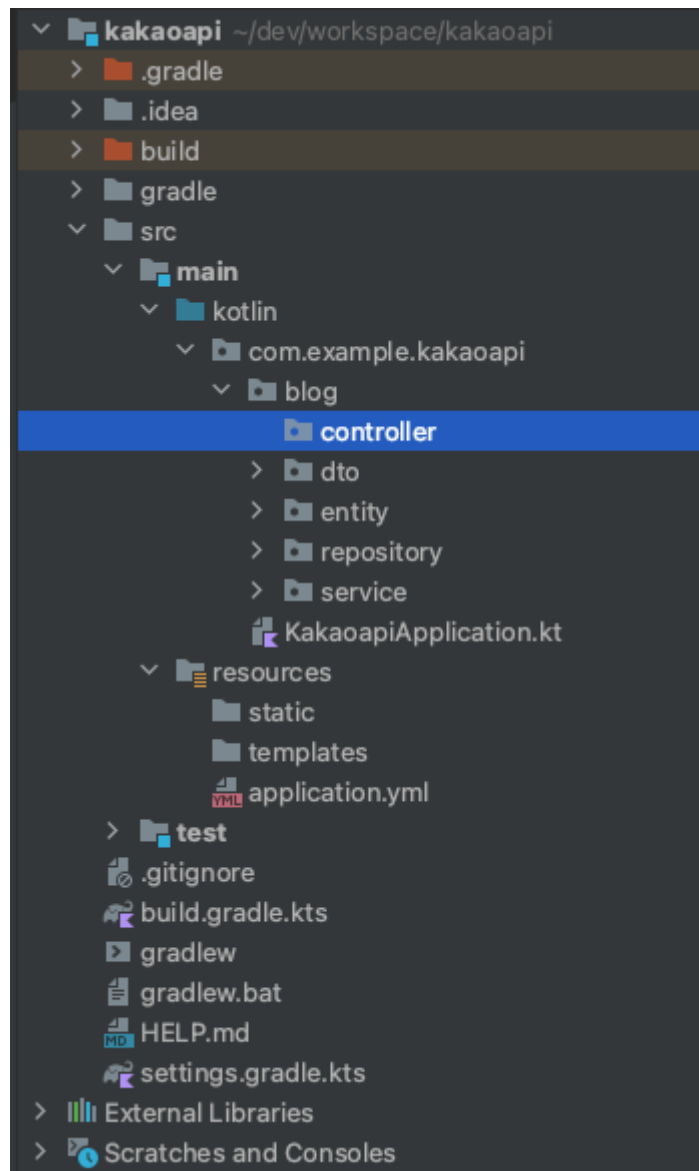
```
package com.example.kakaoapi.blog.service

import com.example.kakaoapi.blog.dto BlogDto
import org.springframework.stereotype.Service

@Service
class BlogService {
    fun searchKakao(blogDto: BlogDto): String? {
        return "SearchKakao"
    }
}
```

4.6. controller 만들기

1. 패키지 만들기



2. Controller 만들기

- 요청을 받아서 필요한 Service에 전달

```
package com.example.kakaoapi.blog.controller

import com.example.kakaoapi.blog.dto.BlogDto
import com.example.kakaoapi.blog.service.BlogService
import org.springframework.web.bind.annotation.GetMapping
import org.springframework.web.bind.annotation.RequestBody
import org.springframework.web.bind.annotation.RequestMapping
import org.springframework.web.bind.annotation.RestController

@RequestMapping("/api/blog")
@RestController
class BlogController(
    val blogService: BlogService
) {
    @GetMapping("")
    fun search(@RequestBody blogDto: BlogDto): String? {
        val result: String? = blogService.searchKakao(blogDto)
        return result
    }
}
```

4.7. WebClient로 kakao API 호출하기

1. build.gradle.kts > dependencies 에 의존성 추가

- WebClient 를 사용하기 위해 Maven Repository 에서 spring webflux 검색 후 추가

```
implementation("org.springframework:spring-webflux")
```

2. BlogService에 WebClient로 카카오 API 호출

```
package com.example.kakaoapi.blog.service

import com.example.kakaoapi.blog.dto.BlogDto
import org.springframework.http.HttpHeaders
import org.springframework.http.MediaType
import org.springframework.stereotype.Service
import org.springframework.web.reactive.function.client.WebClient
import org.springframework.web.reactive.function.client.bodyToMono

@Service
class BlogService {
    fun searchKakao(blogDto: BlogDto): String? {
        val webClient: WebClient = WebClient
            .builder()
            .baseUrl("https://dapi.kakao.com")
            .defaultHeader(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_JSON_VALUE)
            .build()

        val response = webClient
            .get()
            .uri { it.path("/v2/search/blog")
                .queryParams("query", blogDto.query)
                .queryParams("sort", blogDto.sort)
                .queryParams("page", blogDto.page)
                .queryParams("size", blogDto.size)
                .build() }
            .header("Authorization", "KakaoAK ac5d9a371616d5c1a4c00e6e81230a6e")
            .retrieve()
            .bodyToMono<String>()

        val result = response.block()
        return result
    }
}
```

3. REST_API_KEY application.yml에 관리

1. application.yml 에 key 추가

```
REST_API_KEY: "ac5d9a371616d5c1a4c00e6e81230a6e"
```

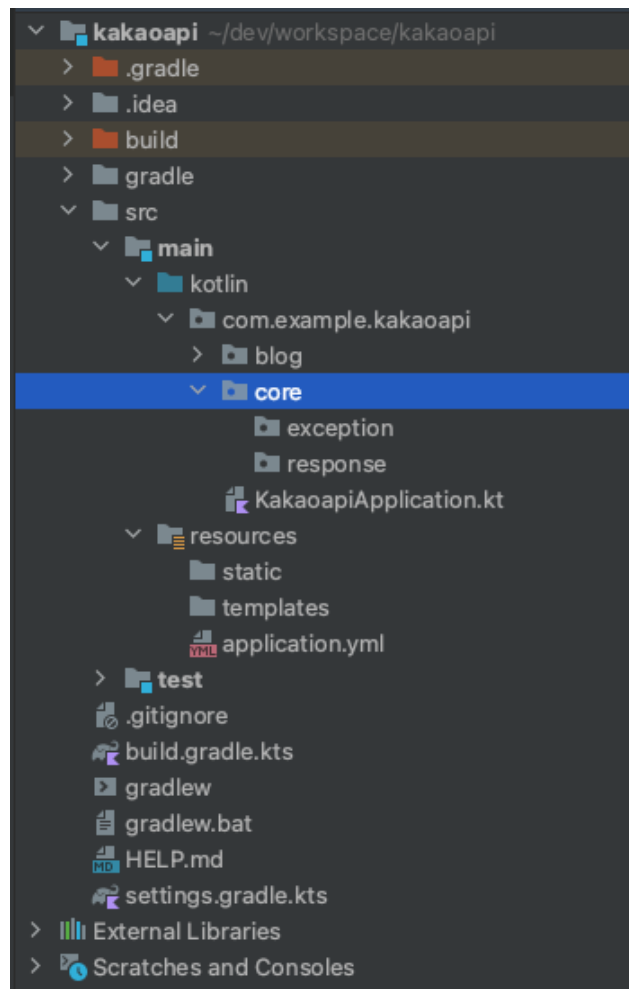
2. BlogService에 @Value 로 key 불러와서 사용

```
@Value("\${REST_API_KEY}")  
lateinit var restApiKey: String
```

```
val response = webClient  
    .get()  
    .uri { it.path("/v2/search/blog")  
        .queryParams("query", blogDto.query)  
        .queryParams("sort", blogDto.sort)  
        .queryParams("page", blogDto.page)  
        .queryParams("size", blogDto.size)  
        .build() }  
    .header("Authorization", "KakaoAK $restApiKey")  
    .retrieve()  
    .bodyToMono<String>()
```


4.8. ExceptionHandler 만들기

1. 패키지 만들기



2. ErrorResponse 만들기

```
package com.example.kakaoapi.core.response

const val INVALID_ARGUMENT: String = "Invalid Argument"

data class ErrorResponse(
    val message: String,
    val errorType: String = INVALID_ARGUMENT
)
```

3. InvalidInputException 만들기

```
package com.example.kakaoapi.core.exception

class InvalidInputException(
    message: String = "Invalid Input"
) : RuntimeException(message)
```

4. CustomExceptionHandler 만들기

```
package com.example.kakaoapi.core.exception

import com.example.kakaoapi.core.response.ErrorResponse
import org.springframework.http.HttpStatus
import org.springframework.http.ResponseEntity
import org.springframework.web.bind.annotation.ExceptionHandler
import org.springframework.web.bind.annotation.RestControllerAdvice

@RestControllerAdvice
class CustomExceptionHandler {

    @ExceptionHandler(InvalidInputException::class)
    protected fun invalidInputException(ex: InvalidInputException): ResponseEntity<ErrorResponse> {
        val errors = ErrorResponse(ex.message ?: "Not Exception Message")
        return ResponseEntity(errors, HttpStatus.BAD_REQUEST)
    }
}
```

5. BlogService 에서 값 체크

1. enum class 추가

```
private enum class ExceptionMsg(val msg: String) {
    EMPTY_QUERY("query parameter required"),
    NOT_IN_SORT("sort parameter one of accuracy and recency"),
    LESS_THAN_MIN("page is less than min"),
    MORE_THAN_MAX("page is more than max")
}
```

2. searchKakao 함수에 값 체크 추가

```
val msgList = mutableListOf<ExceptionMsg>()

if (blogDto.query.trim().isEmpty()) {
    msgList.add(ExceptionMsg.EMPTY_QUERY)
}

if (blogDto.sort.trim() !in arrayOf("accuracy", "recency")) {
    msgList.add(ExceptionMsg.NOT_IN_SORT)
}

when {
    blogDto.page < 1 -> msgList.add(ExceptionMsg.LESS_THAN_MIN)
    blogDto.page > 50 -> msgList.add(ExceptionMsg.MORE_THAN_MAX)
}

if (msgList.isNotEmpty()) {
    val message = msgList.joinToString { it.msg }
    throw InvalidInputException(message)
}
```

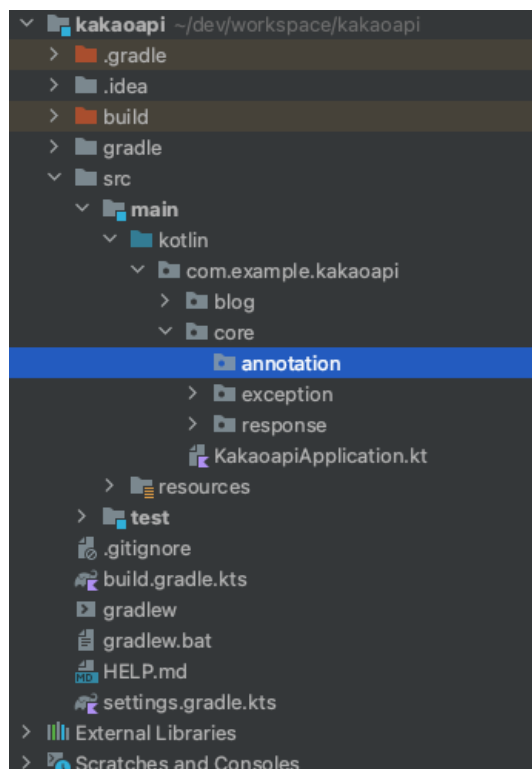
4.9. validation 추가하기

1. build.gradle.kts > dependencies 에 의존성 추가

- WebClient 를 사용하기 위해 Maven Repository 에서 spring validation 검색 후 추가

```
implementation("org.springframework.boot:spring-boot-starter-validation")
```

2. annotation 패키지 만들기



3. Enum 체크할 Validator 생성

```
package com.example.kakaoapi.core.annotation

import jakarta.validation.Constraint
import jakarta.validation.ConstraintValidator
import jakarta.validation.ConstraintValidatorContext
import jakarta.validation.Payload
import kotlin.reflect.KClass

@Target(AnnotationTarget.FIELD)
@Retention(AnnotationRetention.RUNTIME)
@MustBeDocumented
@Constraint(validatedBy = [ValidEnumValidator::class])
annotation class ValidEnum(
    val message: String = "Invalid enum value",
    val groups: Array<KClass<*>> = [],

```

```

        val payload: Array<KClass<out Payload>> = [],
        val enumClass: KClass<out Enum<*>> // specify the enum class to be validated
    )

    class ValidEnumValidator : ConstraintValidator<ValidEnum, Any> {
        private lateinit var enumValues: Array<out Enum<*>>

        override fun initialize(annotation: ValidEnum) {
            enumValues = annotation.enumClass.java.enumConstants
        }

        override fun isValid(value: Any?, context: ConstraintValidatorContext): Boolean {
            if (value == null) {
                return true // null values are validated with the @NotNull annotation
            }
            return enumValues.any { it.name == value.toString() }
        }
    }
}

```

4. BlogDto에 validation 추가

```

package com.example.kakaoapi.blog.dto

import com.example.kakaoapi.core.annotation.ValidEnum
import com.fasterxml.jackson.annotation.JsonProperty
import jakarta.validation.constraints.Max
import jakarta.validation.constraints.Min
import jakarta.validation.constraints.NotBlank
import jakarta.validation.constraints.NotNull

data class BlogDto(
    @field:NotBlank(message = "query parameter required")
    @JsonProperty("query")
    private val _query: String?,

    @field:NotBlank(message = "sort parameter required")
    @field:ValidEnum(enumClass = EnumSort::class, message = "sort parameter one of ACCURACY and RECENCY")
    @JsonProperty("sort")
    private val _sort: String?,

    @field:NotNull(message = "page parameter required")
    @field:Max(50, message = "page is more than max")
    @field:Min(1, message = "page is less than min")
    @JsonProperty("page")
    private val _page: Int?,

    @field:NotNull(message = "size parameter required")
    @JsonProperty("size")
    private val _size: Int?
) {
    val query: String
        get() = _query!!
    val sort: String
        get() = _sort!!
    val page: Int
        get() = _page!!
    val size: Int
        get() = _size!!

    private enum class EnumSort {
        ACCURACY,
        RECENCY
    }
}

```

5. validation 할 DTO 앞에 @Valid 추가

```
class BlogController(  
    val blogService: BlogService  
) {  
    @GetMapping("")  
    fun search(@RequestBody @Valid blogDto: BlogDto): String? {  
        val result: String? = blogService.searchKakao(blogDto)  
        return result  
    }  
}
```

6. validation 에서 걸린 exception 처리

```
@ExceptionHandler(MethodArgumentNotValidException::class)  
protected fun handleValidationExceptions(ex: MethodArgumentNotValidException): ResponseEntity<Map<String, String>> {  
    val errors = mutableMapOf<String, String>()  
    ex.bindingResult.allErrors.forEach { error ->  
        val fieldName = (error as FieldError).field  
        val errorMessage = error.getDefaultMessage()  
        errors[fieldName] = errorMessage ?: "NULL"  
    }  
    return ResponseEntity(errors, HttpStatus.BAD_REQUEST)  
}
```

4.10 검색어 순위 추가하기

1. BlogService 생성자에 Repository 추가

```
val wordRepository: WordRepository
```

2. 카카오 API 호출 후 검색어 카운트 추가

```
val lowQuery: String = blogDto.query.lowercase()
val word: Wordcount = wordRepository.findById(lowQuery).orElse(Wordcount(lowQuery))
word.cnt++

wordRepository.save(word)
```

3. Repository 에 function 추가

```
fun findTop10ByOrderByCntDesc(): List<Wordcount>
```

4. Service 에 function 추가

```
fun searchWordRank(): List<Wordcount> = wordRepository.findTop10ByOrderByCntDesc()
```

5. Controller 에 end point 추가

```
@GetMapping("/rank")
fun searchWordRank(): List<Wordcount> = blogService.searchWordRank()
```