

1.1. var / val

1. var

```
fun main() {  
    var a = 1  
    a = 2  
    println(a)  
}
```

- 읽기, 쓰기 가능

```
fun main() {  
    var a = 1  
    // a = "a" // Type mismatch  
    println(a::class)  
}
```

- 타입 추론으로 입력된 값에 의해 자동으로 타입을 잡아줌
- 다른 타입의 데이터를 넣으려고 하면 Type mismatch

2. val

```
fun main() {  
    val a = 1  
    // a = 2 // 변경 불가  
    println(a)  
}
```

- 읽기만 가능
- 코딩을 할때 기본적으로 val 로 하고 변경이 필요할 경우만 var 로 변경 권장

```
fun main() {  
    val a: Int  
    a = 2 // 지연 할당  
    println(a)  
}
```

- val 에 즉시 할당이 안되어도 최초 1번은 들어감

3. 타입 추론

```
fun main() {  
    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)  
}
```

- kotlin은 타입추론으로 변수에 들어오는 값을 보고 타입을 알아서 지정해줌
- 코딩을 할때 타입추론이 되더라도 직접 명시하는 것을 권장

1.2. ? / ?. / ?: / !!

1. ?

```
fun main() {  
    val str: String? = null  
    //    val str2: String = null // null 허용하지 않음  
}
```

- type에 ? 를 붙이면 null 허용

2. ?.

```
fun main() {  
    val str: String? = "ABC"  
    println(str?.length)  
  
    val str2: String? = null  
    println(str2?.length)  
}
```

- null safe call operator
- ?. 앞의 값이 null 이 아니면 뒤에 명령어 실행, null 이면 null 반환

3. ?:

```
fun main() {  
    val str: String? = "ABC"  
    str ?: println("str is null")  
  
    val str2: String? = null  
    str2 ?: println("str2 is null")  
}
```

- Elvis operator
- ?: 앞에 값이 null 이라면 뒤에 명령어 실행

4. !!

```
fun main() {  
    val str: String? = "ABC"  
    println(str!!.length)  
}
```

- null 허용하는 변수를 null이 아님을 보장
- !!로 해줬는데 실행시 null 인경우 NullPointerException 발생

1.3. if / when

1. if 문

형태

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

대소 비교

```
fun main() {  
    val a: Int = 100  
    val b: Int = 200  
  
    if (a >= b) {  
        println("a = $a")  
    } else {  
        println("b = $b")  
    }  
}
```

null 체크

```
fun main() {  
  
    val a: Int? = null  
  
    if (a == null) {  
        println("null check true")  
    } else {  
        println("a = $a")  
    }  
}
```

in 체크

```
fun main() {

    val a: Int = 100

    if (a 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() {
    val a: Int = 100

    when (a) {
        100 -> println("1. a = $a")
        200 -> println("2. a = $a")
        300 -> println("3. a = $a")
        else -> println("4. Not")
    }
}
```

범위 비교

```
fun main() {
    val a: Int = 100

    when (a) {
        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. class / data class / enum class

1. class

형태

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

생성자로 인스턴스 생성

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

2. data class

```
data class MemberDto(  
    val name: String,  
    val birthDate: LocalDate,  
    val gender: Gender,  
    val email: String,  
)
```

- DTO 만들때 사용
- getter, setter(var 인 경우), equals, hashCode, toString, copy, componentN 자동 생성

3. enum class


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

- 상수들을 모아놓은 집합

2.1. 프로젝트 생성

1. Spring Initializr

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

The screenshot shows the Spring Initializr web interface. The 'Project' section has 'Gradle - Kotlin' selected. The 'Language' section has 'Kotlin' selected. The 'Spring Boot' section has '3.1.0' selected. The 'Project Metadata' section has 'Group' as 'com.example', 'Artifact' as 'auth', 'Name' as 'auth', 'Description' as 'JWT clone coding lecture', and 'Package name' as 'com.example.auth'. The 'Packaging' section has 'Jar' selected. The 'Java' version is set to '17'. The 'Dependencies' section has 'Spring Web', 'Spring Data JPA', 'MariaDB Driver', and 'Spring Boot DevTools' selected. The 'Generate' button is highlighted.

- Project : Gradle - Kotlin
- Language : Kotlin
- Spring Boot : 3.1.0
- packaging : Jar
- Java : 17
- Dependencies
 - Spring Web
 - Spring Data JPA
 - MariaDB Driver (H2, PostgreSQL 등 다른 DB 사용해도 무관)

- Spring Boot DevTools

2. build.gradle.kts 셋팅하기

1. plugins


```
plugins {  
    val kotlinVersion = "1.8.21"  
    id("org.springframework.boot") version "3.1.0"  
    id("io.spring.dependency-management") version "1.1.0"  
    kotlin("jvm") version kotlinVersion  
    kotlin("plugin.spring") version kotlinVersion  
    kotlin("plugin.jpa") version kotlinVersion  
}
```

- 미리 구성해둔 task들의 그룹
- 빌드과정에서 필요한 정보들을 포함하고 있으며 필요에 따라 커스터 마이징 가능
- `val kotlinVersion = "1.8.21"` 를 사용해서 버전을 통일

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")  
}
```

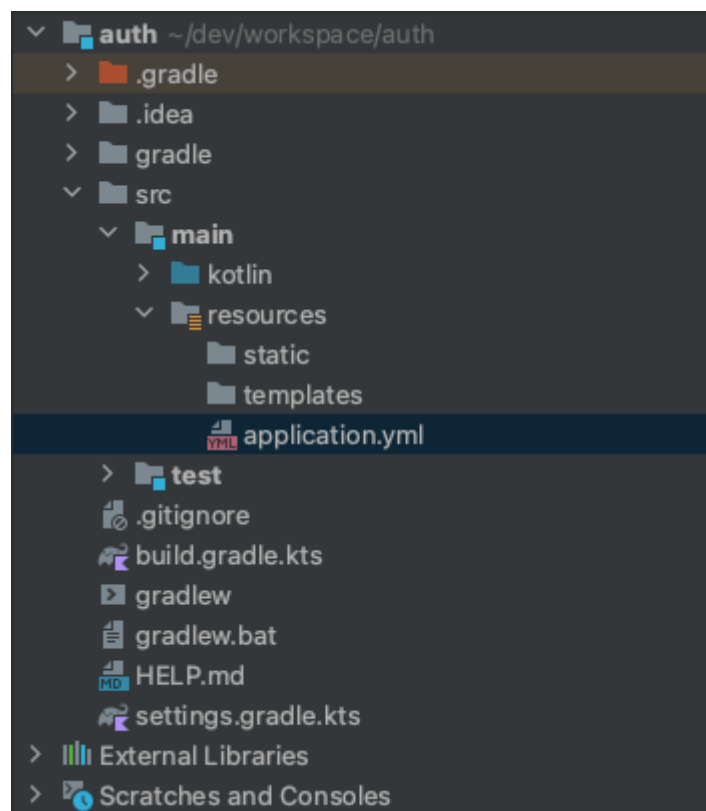
noArg

- 매개변수가 없는 생성자를 자동으로 추가

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

3. application.yml 셋팅하기

1. resources 아래에 yml 파일 생성



2. application.yml

```
server:  
  port: 8080  
  servlet:  
    context-path: /  
    encoding:  
      charset: UTF-8  
      enabled: true  
      force: true  
spring:  
  datasource:
```

```

driver-class-name: org.mariadb.jdbc.Driver
url: jdbc:mariadb://localhost:3306/study
username: root
password: 1234
jpa:
  open-in-view: true
  hibernate:
    ddl-auto: create
  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:
        SQL: debug
        type.descriptor.sql: trace

```

server : 서버 관련 설정

- `server.port` : 서버 포트 (default. 8080)
- `server.servlet.encoding.charset` : HTTP 요청과 응답의 문자 집합 (default. UTF-8)
- `server.servlet.encoding.enabled` : HTTP 인코딩 지원 여부 (default. true)
- `server.servlet.encoding.force` : HTTP 요청과 응답에서 문자 집합에 인코딩을 강제할 지 여부

spring.datasource : Database 접속 정보

spring.jpa : jpa 설정 정보

- `spring.jpa.open-in-view` : true 일 경우 영속성 컨텍스트가 트랜잭션 범위를 넘어선 레이어까지 유지
- `spring.jpa.hibernate.ddl-auto`
 - create : 기존테이블 삭제 후 다시 생성 (DROP + CREATE)
 - create-drop : create와 같으나 종료시점에 테이블 DROP
 - update : 변경분만 반영(운영DB에서는 사용하면 안됨)
 - validate : 엔티티와 테이블이 정상 매핑되었는지만 확인(운영DB)
 - none: 사용하지 않음(사실상 없는 값이지만 관례상 none이라고 한다.)

- `spring.jpa.properties.hibernate.show_sql` : System.out 에 sql 로그 출력
- `spring.jpa.properties.hibernate.format_sql` : sql을 보기좋게 줄맞춤
- `spring.jpa.properties.hibernate.highlight_sql` : sql 색상 표시 추가

logging : 로그에 관한 정보

- `logging.pattern.console` : 기본 로그의 형태 지정
- `logging.level.org.hibernate.SQL` : logger 에 sql 로그 출력(권장)
- `logging.level.org.hibernate.type.descriptor.sql` : trace로 하면 sql에 바인딩 되는 값을 확인

2.2. 회원가입 기능 만들기

0. 요구사항

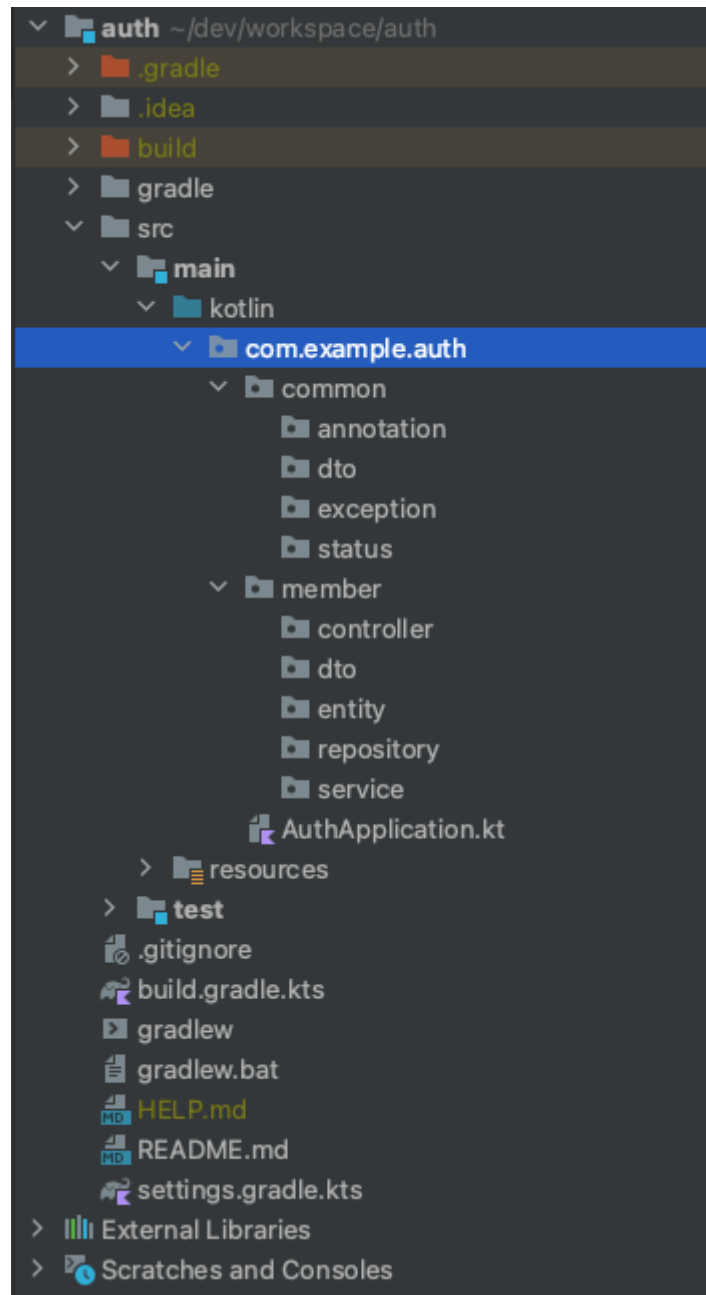
EndPoint

POST /api/member/signup

회원가입시 입력받을 정보

Name	Description	Required
loginId	로그인 아이디	O
password	비밀번호, 영문/숫자/특수문자를 포함한 8~20자리	O
name	이름	O
birthDate	생년월일, YYYY-MM-DD 형식	O
gender	성별, MAN(남) 이나 WOMAN(여) 중 하나	O
email	이메일, 이메일 형식	O

1. 패키지 생성



- com.example.auth.common : 공통적으로 사용할 수 있는 기능 분류
 - annotation : 사용자 생성 어노테이션
 - dto : 어플리케이션 전반에 공통적으로 사용할 수 있는 DTO
 - exception : exception 처리
 - status : 어플리케이션에서 사용할 status
- com.example.auth.member : 회원정보 관련 기능 분류
 - controller : Request 를 받을 End Point

- dto : 회원정보 관련 DTO
- entity : 회원정보 관련 Entity
- repository : 회원정보 관련 Repository
- service : 비즈니스 로직

2. Gender enum class 생성

```
package com.example.auth.common.status

// Gender 클래스로 남, 여 상태 구분
enum class Gender(val desc: String) {
    MAN("남"),
    WOMAN("여")
}
```

3. DTO 생성

```
package com.example.auth.member.dto

import com.example.auth.common.status.Gender
import java.time.LocalDate

// 회원가입시 입력받을 정보
data class MemberDtoRequest(
    val id: Long?,
    val loginId: String,
    val password: String,
    val name: String,
    val birthDate: LocalDate,
    val gender: Gender,
    val email: String,
)
```

4. entity 생성

```
package com.example.auth.member.entity

import com.example.auth.common.status.Gender
import jakarta.persistence.*
import java.time.LocalDate

@Entity
```

```

@Table(
    uniqueConstraints = [
        UniqueConstraint(name = "uk_member_login_id", columnNames = ["loginId"])
    ]
)
class Member(
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    var id: Long? = null,

    @Column(nullable = false, length = 30, updatable = false)
    val loginId: String,

    @Column(nullable = false, length = 100)
    val password: String,

    @Column(nullable = false, length = 10)
    val name: String,

    @Column(nullable = false)
    @Temporal(TemporalType.DATE)
    val birthDate: LocalDate,

    @Column(nullable = false, length = 5)
    @Enumerated(EnumType.STRING)
    val gender: Gender,

    @Column(nullable = false, length = 30)
    val email: String,
)

```

5. repository 생성

```

package com.example.auth.member.repository

import com.example.auth.member.entity.Member
import org.springframework.data.jpa.repository.JpaRepository

interface MemberRepository : JpaRepository<Member, Long> {
    // ID 중복 검사를 위해 필요
    fun findByLoginId(loginId: String): Member?
}

```

6. service 생성

```

package com.example.auth.member.service

import com.example.auth.member.dto.MemberDtoRequest
import com.example.auth.member.entity.Member

```

```

import com.example.auth.member.repository.MemberRepository
import jakarta.transaction.Transactional
import org.springframework.stereotype.Service

@Transactional
@Service
class MemberService(
    private val memberRepository: MemberRepository
) {
    /**
     * 회원가입
     */
    fun signUp(memberDtoRequest: MemberDtoRequest): String {
        // ID 중복 검사
        var member: Member? = memberRepository
            .findByLoginId(memberDtoRequest.loginId)
        if (member != null) {
            return "이미 등록된 ID 입니다."
        }

        member = Member(
            null,
            memberDtoRequest.loginId,
            memberDtoRequest.password,
            memberDtoRequest.name,
            memberDtoRequest.birthDate,
            memberDtoRequest.gender,
            memberDtoRequest.email
        )

        memberRepository.save(member)

        return "회원가입이 완료 되었습니다."
    }
}

```

7. controller 생성

```

package com.example.auth.member.controller

import com.example.auth.member.dto.MemberDtoRequest
import com.example.auth.member.service.MemberService
import org.springframework.web.bind.annotation.PostMapping
import org.springframework.web.bind.annotation.RequestBody
import org.springframework.web.bind.annotation.RequestMapping
import org.springframework.web.bind.annotation.RestController

@RequestMapping("/api/member")
@RestController
class MemberController(
    private val memberService: MemberService
) {

```

```
/**
 * 회원가입
 */
@PostMapping("/signup")
fun signUp(@RequestBody memberDtoRequest: MemberDtoRequest): String {
    return memberService.signUp(memberDtoRequest)
}
}
```

2.3. validation 추가하기

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

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

- validation을 사용하기 위해 Maven Repository 에서 spring validation 검색 후 추가

2. Enum 체크할 Validator 생성

```
package com.example.auth.common.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<*>>
)

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
        }
        return enumValues.any { it.name == value.toString() }
```

```
}
}
```

- @Target : 어노테이션이 지정되어 사용할 종류를 정의
- @Retention : 어노테이션을 컴파일된 클래스 파일에 저장할 것인지 런타임에 반영할 것인지 정의
 - SOURCE: binary 파일로 명시되지 않음
 - BINARY: binary 파일에는 명시되지만 reflection 에는 명시되지 않음
 - RUNTIME: binary 파일과 reflection 둘다 명시
- @MustBeDocumented : API의 일부분으로 문서화하기 위해 사용.

3. DTO에 Validation 추가

```
package com.example.auth.member.dto

import com.example.auth.common.annotation.ValidEnum
import com.example.auth.common.status.Gender
import com.fasterxml.jackson.annotation.JsonProperty
import jakarta.validation.constraints.Email
import jakarta.validation.constraints.NotBlank
import jakarta.validation.constraints.Pattern
import java.time.LocalDate
import java.time.format.DateTimeFormatter

data class MemberDtoRequest(
    val id: Long?,

    @field:NotBlank
    @JsonProperty("loginId")
    private val _loginId: String?,

    @field:NotBlank
    @field:Pattern(
        regexp="^(?=.*[a-zA-Z])(?=.*[0-9])(?=.*[!@#\\$%^&*])[a-zA-Z0-9!@#\\$%^&*]{8,20}\\$",
        message = "영문, 숫자, 특수문자를 포함한 8~20자리로 입력해주세요"
    )
    @JsonProperty("password")
    private val _password: String?,

    @field:NotBlank
    @JsonProperty("name")
    private val _name: String?,

    @field:NotBlank
    @field:Pattern(
        regexp = "^[12]\\d{3}-(0[1-9]|1[0-2])-(0[1-9]|[12]\\d|3[01])$",
        message = "날짜형식(YYYY-MM-DD)을 확인해주세요"
    )
)
```

```

    )
    @JsonProperty("birthDate")
    private val _birthDate: String?,

    @field:NotBlank
    @field:ValidEnum(enumClass = Gender::class,
        message = "MAN 이나 WOMAN 중 하나를 선택해주세요")
    @JsonProperty("gender")
    private val _gender: String?,

    @field:NotBlank
    @field:Email
    @JsonProperty("email")
    private val _email: String?,
) {
    val loginId: String
        get() = _loginId!!

    val password: String
        get() = _password!!

    val name: String
        get() = _name!!

    val birthDate: LocalDate
        get() = _birthDate!!.toLocalDate()

    val gender: Gender
        get() = Gender.valueOf(_gender!!)

    val email: String
        get() = _email!!

    private fun String.toLocalDate(): LocalDate =
        LocalDate.parse(this, DateTimeFormatter.ofPattern("yyyy-MM-dd"))
}

```

4. controller에 @Valid 추가

```

/**
 * 회원가입
 */
@PostMapping("/signup")
fun signUp(@RequestBody @Valid memberDtoRequest: MemberDtoRequest): String {
    return memberService.signUp(memberDtoRequest)
}
}

```

- @Valid 가 있어야 DTO에 데이터를 담은 후 유효성 검사

2.4. BaseResponse 만들기

1. ResultCode enum class 생성

```
package com.example.auth.common.status

enum class Gender(val desc: String) {
    MAN("남"),
    WOMAN("여")
}

// 추가
enum class ResultCode(val msg: String) {
    SUCCESS("정상 처리 되었습니다."),
    ERROR("에러가 발생했습니다.")
}
```

2. BaseResponse 생성

```
package com.example.auth.common.dto

import com.example.auth.common.status.ResultCode

data class BaseResponse<T>(
    val resultCode: String = ResultCode.SUCCESS.name,
    val data: T? = null,
    val message: String = ResultCode.SUCCESS.msg,
)
```

- resultCode : 결과 코드
- data : 조회시 데이터를 담아서 반환해줄 data
- message : 처리 메시지

2.5. ExceptionHandler 만들기

1. InvalidInputException 생성

```
package com.example.auth.common.exception

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

- @Valid 외에 필드값이 문제가 있어서 exception 을 발생시킬때 사용

2. CustomExceptionHandler 생성

```
package com.example.auth.common.exception

import com.example.auth.common.dto.BaseResponse
import com.example.auth.common.status.ResultCode
import org.springframework.http.HttpStatus
import org.springframework.http.ResponseEntity
import org.springframework.validation.FieldError
import org.springframework.web.bind.MethodArgumentNotValidException
import org.springframework.web.bind.annotation.ExceptionHandler
import org.springframework.web.bind.annotation.RestControllerAdvice

@RestControllerAdvice
class CustomExceptionHandler {

    @ExceptionHandler(MethodArgumentNotValidException::class)
    protected fun handleValidationExceptions(ex: MethodArgumentNotValidException):
        ResponseEntity<BaseResponse<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 ?: "Not Exception Message"
        }
        return ResponseEntity(BaseResponse(
            ResultCode.ERROR.name,
            errors,
            ResultCode.ERROR.msg
        ), HttpStatus.BAD_REQUEST)
    }

    @ExceptionHandler(InvalidInputException::class)
    protected fun invalidInputException(ex: InvalidInputException):
```

```

        ResponseEntity<BaseResponse<Map<String, String>>> {
            val errors = mapOf(ex.fieldName to (ex.message ?: "Not Exception Message"))
            return ResponseEntity(BaseResponse(
                ResultCode.ERROR.name,
                errors,
                ResultCode.ERROR.msg
            ), HttpStatus.BAD_REQUEST)
        }

        @ExceptionHandler(Exception::class)
        protected fun defaultException(ex: Exception):
            ResponseEntity<BaseResponse<Map<String, String>>> {
                val errors = mapOf("미처리 예러" to (ex.message ?: "Not Exception Message"))
                return ResponseEntity(BaseResponse(
                    ResultCode.ERROR.name,
                    errors,
                    ResultCode.ERROR.msg
                ), HttpStatus.BAD_REQUEST)
            }
        }
    }
}

```

- MethodArgumentNotValidException : @Valid에서 걸린 경우 발생
- InvalidInputException : 사용자 생성 exception
- Exception : 그외의 모든 exception

3. 기존 코드 수정

MemberDtoRequest에 entity 생성하는 함수 추가

```

fun toEntity(): Member =
    Member(id, loginId, password, name, birthDate, gender, email)

```

MemberService signUp 함수 수정

```

/**
 * 회원가입
 */
fun signUp(memberDtoRequest: MemberDtoRequest): String {
    var member = memberRepository.findByLoginId(memberDtoRequest.loginId)
    if (member != null) {
        throw InvalidInputException("loginId", "이미 등록된 ID 입니다.")
    }

    member = memberDtoRequest.toEntity()
    memberRepository.save(member)
}

```

```
        return "회원가입이 완료되었습니다."
    }
```

MemberController signUp 함수 수정

```
/**
 * 회원가입
 */
@PostMapping("/signup")
fun signUp(@RequestBody @Valid memberDtoRequest: MemberDtoRequest):
    BaseResponse<Unit> {
    val resultMsg: String = memberService.signUp(memberDtoRequest)
    return BaseResponse(message = resultMsg)
}
```

3.1. 권한 관리 방법 알아보기

인증과 인가

- 인증 (Authentication) : 해당 사용자가 본인이 맞는지 확인하는 절차
- 인가 (Authorization) : 인증된 사용자가 요청한 자원에 접근 가능한지를 결정하는 절차

Spring Security

- 어플리케이션의 보안을 담당하는 스프링 하위 프레임워크
- 인증과 인가(권한)에 대한 부분을 Filter 흐름에 따라 처리

JWT

JWT.IO

JSON Web Tokens are an open, industry standard RFC 7519 method for representing claims securely between two parties.

 <https://jwt.io/>



JSON Web Tokens are an open, industry standard RFC 7519 method for representing claims securely between two parties. JWT.IO allows you to decode, verify and generate JWT.

- Json Web Token의 약자로 클라이언트와 서버 통신시 인증, 인가를 위해 사용하는 토큰
- Json Web Token은 두 당사자 간의 클레임을 안전하게 표현하기 위한 개방형 업계 표준 RFC 7519 방법
 - header : 토큰의 유형과 서명 알고리즘 명시
 - payload : claim이라고 불리는 사용자 인증, 인가 정보
 - signature : 헤더와 페이로드가 비밀키로 서명

생성

1. TokenInfo

- 로그인시 토큰 정보를 담아 클라이언트에게 전달하는 용도

2. JwtTokenProvider

- Token 생성, Token 정보 추출, Token 검증

3. JwtAuthenticationFilter

- GenericFilterBean 상속
- Filter로 Token 정보를 검사하고 SecurityContextHolder에 authentication을 기록

4. SecurityConfig

- 인증 및 인가 관리 config

5. CustomUserDetailsService

- UserDetailsService 구현
- loadUserByUsername override

3.2. JwtToken 만들기

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

```
// Spring Security 사용시 필요
implementation("org.springframework.boot:spring-boot-starter-security")

// JWT 사용시 필요
implementation("io.jsonwebtoken:jjwt-api:0.11.5")
runtimeOnly("io.jsonwebtoken:jjwt-impl:0.11.5")
runtimeOnly("io.jsonwebtoken:jjwt-jackson:0.11.5")
```

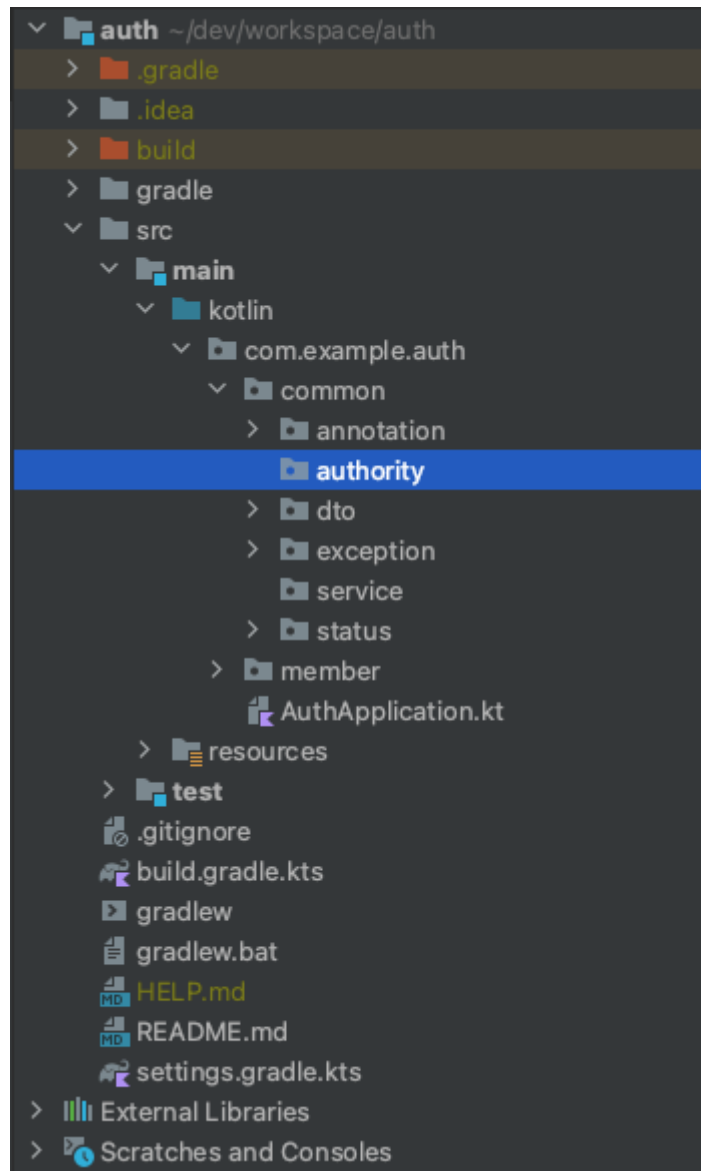
- 권한 관리를 위해 Spring Security + JWT 사용

2. application.yml 에 secretKey 추가

```
jwt:
  secret: DadFufN40ui8Bfv3ScFj6R9fyJ9hD45E6AGFsXgFsRhT4YSdSb
```

- 향후 암호화 할때 사용, HS256 알고리즘을 사용하기 위해, 256비트 보다 커야함(32글자 이상)

3. 패키지 생성



- com.example.auth.common : 공통적으로 사용할 수 있는 기능 분류
 - authority : 권한 관련 기능 분류
 - service : CustomUserDetailsService 생성

4. Token 정보 담을 data class 생성

```
package com.example.auth.common.authority

data class TokenInfo(
    val grantType: String,
    val accessToken: String,
)
```

- grantType : JWT 권한 인증 타입(ex. Bearer)
- accessToken : 실제 검증할 때 확인할 토큰
- 해당 실습에서 refreshToken은 사용하지 않음

5. JwtTokenProvider 생성

```
package com.example.auth.common.authority

import io.jsonwebtoken.*
import io.jsonwebtoken.io.Decoders
import io.jsonwebtoken.security.Keys
import io.jsonwebtoken.security.SecurityException
import org.springframework.beans.factory.annotation.Value
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken
import org.springframework.security.core.Authentication
import org.springframework.security.core.GrantedAuthority
import org.springframework.security.core.authority.SimpleGrantedAuthority
import org.springframework.security.core.userdetails.User
import org.springframework.security.core.userdetails.UserDetails
import org.springframework.stereotype.Component
import java.util.*

const val EXPIRATION_MILLISECONDS: Long = 1000 * 60 * 60 * 12

@Component
class JwtTokenProvider {
    @Value("\${jwt.secret}")
    lateinit var secretKey: String

    private val key by lazy {
        Keys.hmacShaKeyFor(Decoders.BASE64.decode(secretKey))
    }

    /**
     * token 생성
     */
    fun createToken(authentication: Authentication): TokenInfo {
        val authorities: String = authentication
            .authorities
            .joinToString(", ", transform = GrantedAuthority::getAuthority)

        val now = Date()
        val accessExpiration = Date(now.time + EXPIRATION_MILLISECONDS)

        // Access Token
        val accessToken = Jwts.builder()
            .setSubject(authentication.name)
            .claim("auth", authorities)
            .setIssuedAt(now)
            .setExpiration(accessExpiration)
            .signWith(key, SignatureAlgorithm.HS256)
```



```

        .compact()

        return TokenInfo("Bearer", accessToken)
    }

    /**
     * token 정보 추출
     */
    fun getAuthentication(token: String): Authentication {
        val claims: Claims = getClaims(token)

        val auth = claims["auth"] ?: throw RuntimeException("잘못된 토큰 입니다.")

        // 권한 정보 추출
        val authorities: Collection<GrantedAuthority> = (auth as String)
            .split(",")
            .map { SimpleGrantedAuthority(it) }

        val principal: UserDetails = User(claims.subject, "", authorities)

        return UsernamePasswordAuthenticationToken(principal, "", authorities)
    }

    /**
     * Token 검증
     */
    fun validateToken(token: String): Boolean {
        try {
            getClaims(token)
            return true
        } catch (e: Exception) {
            when (e) {
                is SecurityException -> {} // Invalid JWT Token
                is MalformedJwtException -> {} // Invalid JWT Token
                is ExpiredJwtException -> {} // Expired JWT Token
                is UnsupportedJwtException -> {} // Unsupported JWT Token
                is IllegalArgumentException -> {} // JWT claims string is empty
                else -> {} // else
            }
            println(e.message)
        }
        return false
    }

    private fun getClaims(token: String): Claims =
        Jwts.parserBuilder()
            .setSigningKey(key)
            .build()
            .parseClaimsJws(token)
            .body
    }

```

6. JwtAuthenticationFilter 생성

```

package com.example.auth.common.authority

import jakarta.servlet.FilterChain
import jakarta.servlet.HttpServletRequest
import jakarta.servlet.HttpServletResponse
import jakarta.servlet.http.HttpServletRequest
import org.springframework.security.core.context.SecurityContextHolder
import org.springframework.util.StringUtils
import org.springframework.web.filter.GenericFilterBean

class JwtAuthenticationFilter(
    private val jwtTokenProvider: JwtTokenProvider
) : GenericFilterBean() {
    override fun doFilter(
        request: ServletRequest?,
        response: ServletResponse?,
        chain: FilterChain?
    ) {
        val token = resolveToken(request as HttpServletRequest)

        if (token != null && jwtTokenProvider.validateToken(token)) {
            val authentication = jwtTokenProvider.getAuthentication(token)
            SecurityContextHolder.getContext().authentication = authentication
        }

        chain?.doFilter(request, response)
    }

    private fun resolveToken(request: HttpServletRequest): String? {
        val bearerToken = request.getHeader("Authorization")

        return if (StringUtils.hasText(bearerToken) &&
            bearerToken.startsWith("Bearer")) {
            bearerToken.substring(7)
        } else {
            null
        }
    }
}

```

3.3. Spring Security로 권한 관리 만들기

1. 권한 관리 클래스 생성

```
package com.example.auth.common.authority

import org.springframework.context.annotation.Bean
import org.springframework.context.annotation.Configuration
import org.springframework.security.config.annotation.web.builders.HttpSecurity
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity
import org.springframework.security.config.http.SessionCreationPolicy
import org.springframework.security.crypto.factory.PasswordEncoderFactories
import org.springframework.security.crypto.password.PasswordEncoder
import org.springframework.security.web.SecurityFilterChain
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter

@Configuration
@EnableWebSecurity
class SecurityConfig(
    private val jwtTokenProvider: JwtTokenProvider
) {
    @Bean
    fun filterChain(http: HttpSecurity): SecurityFilterChain {
        http
            .httpBasic { it.disable() }
            .csrf { it.disable() }
            .sessionManagement {
                it.sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            }
            .authorizeHttpRequests {
                it.requestMatchers("/api/member/signup").anonymous()
                .anyRequest().permitAll()
            }
            .addFilterBefore(
                JwtAuthenticationFilter(jwtTokenProvider),
                UsernamePasswordAuthenticationFilter::class.java
            )

        return http.build()
    }

    @Bean
    fun passwordEncoder(): PasswordEncoder =
        PasswordEncoderFactories.createDelegatingPasswordEncoder()
}
```

- `.httpBasic { it.disable() }` : basic auth 끄기
- `.csrf { it.disable() }` : csrf 끄기

- .sessionManagement {
 it.sessionCreationPolicy(SessionCreationPolicy.STATELESS) } : JWT를 사용하기 때문에 세션은 사용하지 않기
- .addFilterBefore(A, B) : B 필터를 실행하기 전에 A 필터 실행하기(A가 통과하면 B는 실행 안함)

hasRole('role1')	권한(role1)을 가지고 있는 경우
hasAnyRole('role1', 'role2')	권한들(role1, role2) 하나라도 가지고 있을 경우 (갯수는 제한없다)
permitAll()	권한 있든 말든 모두 접근 가능하다.
denyAll()	권한 있든 말든 모두 접근 불가능하다.
anonymous()	Anonymous 사용자일 경우 (인증을 하지 않은 사용자)
rememberMe()	Remember-me 기능으로 로그인한 사용자일 경우
authenticated()	Anonymous 사용자가 아닐 경우 (인증을 한 사용자)
fullyAuthenticated()	Anonymous 사용자가 아니고 Remember-me 기능으로 로그인 하지 않은 사용자 일 경우

4.1. 회원가입시 권한 부여

1. ROLE enum class 추가

```
package com.example.auth.common.status

enum class Gender(val desc: String) {
    MAN("남"),
    WOMAN("여")
}

enum class ResultCode(val msg: String) {
    SUCCESS("정상 처리 되었습니다."),
    ERROR("에러가 발생했습니다.")
}

// 추가
enum class ROLE {
    MEMBER
}
```

2. MemberRole Entity 추가

```
package com.example.auth.member.entity

import com.example.auth.common.status.Gender
import com.example.auth.common.status.ROLE
import jakarta.persistence.*
import java.time.LocalDate

@Entity
@Table(
    uniqueConstraints = [
        UniqueConstraint(name = "uk_member_login_id", columnNames = ["loginId"])
    ]
)
class Member(
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    var id: Long? = null,

    @Column(nullable = false, length = 30, updatable = false)
    val loginId: String,

    @Column(nullable = false, length = 100)
    val password: String,

    @Column(nullable = false, length = 10)
```

```

        val name: String,

        @Column(nullable = false)
        @Temporal(TemporalType.DATE)
        val birthDate: LocalDate,

        @Column(nullable = false, length = 5)
        @Enumerated(EnumType.STRING)
        val gender: Gender,

        @Column(nullable = false, length = 30)
        val email: String,
    ) {
        // 추가
        @OneToMany(fetch = FetchType.LAZY, mappedBy = "member")
        val memberRole: List<MemberRole>? = null
    }

    // 추가
    @Entity
    class MemberRole(
        @Id
        @GeneratedValue(strategy = GenerationType.AUTO)
        var id: Long? = null,

        @Column(nullable = false, length = 30)
        @Enumerated(EnumType.STRING)
        val role: ROLE,

        @ManyToOne(fetch = FetchType.LAZY)
        @JoinColumn(foreignKey = ForeignKey(name = "fk_user_role_member_id"))
        val member: Member,
    )

```

3. MemberRole Repository 추가

```

package com.example.auth.member.repository

import com.example.auth.member.entity.Member
import com.example.auth.member.entity.MemberRole
import org.springframework.data.jpa.repository.JpaRepository

interface MemberRepository : JpaRepository<Member, Long> {
    fun findByLoginId(loginId: String): Member?
}

// 추가
interface MemberRoleRepository : JpaRepository<MemberRole, Long>

```

4. 회원가입시 권한 저장 추가

```

package com.example.auth.member.service

import com.example.auth.common.exception.InvalidInputException
import com.example.auth.common.status.ROLE
import com.example.auth.member.dto.MemberDtoRequest
import com.example.auth.member.entity.Member
import com.example.auth.member.entity.MemberRole
import com.example.auth.member.repository.MemberRepository
import com.example.auth.member.repository.MemberRoleRepository
import jakarta.transaction.Transactional
import org.springframework.stereotype.Service

@Transactional
@Service
class MemberService(
    private val memberRepository: MemberRepository,
    private val memberRoleRepository: MemberRoleRepository
) {
    /**
     * 회원가입
     */
    fun signUp(memberDtoRequest: MemberDtoRequest): String {
        var member: Member? = memberRepository
            .findByLoginId(memberDtoRequest.loginId)
        if (member != null) {
            throw InvalidInputException("loginId", "이미 등록된 ID 입니다.")
        }

        // 사용자 정보 저장
        member = memberDtoRequest.toEntity()
        memberRepository.save(member)

        // 권한 저장
        val memberRole = MemberRole(null, ROLE.MEMBER, member)
        memberRoleRepository.save(memberRole)

        return "회원가입이 완료되었습니다."
    }
}

```

4.2. 로그인 후 Token 발행

0. 요구사항

EndPoint

```
POST /api/member/login
```

로그인시 입력받을 정보

Name	Description	Required
loginId	로그인 아이디	O
password	비밀번호	O

1. Login DTO 생성

```
data class LoginDto(  
    @field:NotBlank  
    @JsonProperty("loginId")  
    private val _loginId: String?,  
  
    @field:NotBlank  
    @JsonProperty("password")  
    private val _password: String?,  
) {  
    val loginId: String  
        get() = _loginId!!  
    val password: String  
        get() = _password!!  
}
```

2. service에 로그인 기능 추가

```
@Transactional  
@Service  
class MemberService(  
    private val memberRepository: MemberRepository,  
    private val memberRoleRepository: MemberRoleRepository,
```



```

        // 추가
        private val authenticationManagerBuilder: AuthenticationManagerBuilder,
        private val jwtTokenProvider: JwtTokenProvider,
    ) {

        ...

        /**
         * 로그인
         */
        fun login(loginDto: LoginDto): TokenInfo {
            val authenticationToken =
                UsernamePasswordAuthenticationToken(loginDto.loginId, loginDto.password)
            val authentication =
                authenticationManagerBuilder.`object`.authenticate(authenticationToken)

            return jwtTokenProvider.createToken(authentication)
        }
    }

```

3. controller에 로그인 EndPoint 추가

```

    /**
     * 로그인
     */
    @PostMapping("/login")
    fun login(@RequestBody @Valid loginDto: LoginDto): BaseResponse<TokenInfo> {
        val tokenInfo = memberService.login(loginDto)
        return BaseResponse(data = tokenInfo)
    }

```

4. CustomUserDetailsService 생성

```

package com.example.auth.common.service

import com.example.auth.member.entity.Member
import com.example.auth.member.repository.MemberRepository
import org.springframework.security.core.authority.SimpleGrantedAuthority
import org.springframework.security.core.userdetails.User
import org.springframework.security.core.userdetails.UserDetails
import org.springframework.security.core.userdetails.UserDetailsService
import org.springframework.security.core.userdetails.UsernameNotFoundException
import org.springframework.security.crypto.password.PasswordEncoder
import org.springframework.stereotype.Service

@Service
class CustomUserDetailsService(
    private val memberRepository: MemberRepository,
    private val passwordEncoder: PasswordEncoder,

```

```

) : UserDetailsService {
    override fun loadUserByUsername(username: String): UserDetails =
        memberRepository.findByLoginId(username)
            ?.let { createUserDetails(it) }
            ?: throw UsernameNotFoundException("해당하는 유저를 찾을 수 없습니다.")

    private fun createUserDetails(member: Member): UserDetails =
        User(
            member.loginId,
            passwordEncoder.encode(member.password),
            member.memberRole!!.map { SimpleGrantedAuthority("ROLE_${it.role}") }
        )
}

```

5. 로그인 실패시 Exception 처리 추가

```

@ExceptionHandler(BadCredentialsException::class)
protected fun badCredentialsException(ex: BadCredentialsException):
    ResponseEntity<BaseResponse<Map<String, String>>> {
    val errors = mapOf("로그인 실패" to "아이디 혹은 비밀번호를 다시 확인하세요.")
    return ResponseEntity(BaseResponse(
        ResultCode.ERROR.name,
        errors,
        ResultCode.ERROR.msg
    ), HttpStatus.BAD_REQUEST)
}

```

5.1. 내 정보 조회 기능 만들기

0. 요구사항

EndPoint

```
GET /api/member/info
Authorization: Bearer ${accessToken}
```

입력받을 정보

Name	Description	Required
id	회원번호	O

조회된 정보

Name	Description
id	회원번호
loginId	로그인 아이디
name	이름
birthDate	생년월일, YYYYMMDD 형식
gender	성별, 남 / 여 중 하나
email	이메일, 이메일 형식

1. 정보 답을 DTO 생성

```
data class MemberDtoResponse(  
    val id: Long,  
    val loginId: String,  
    val name: String,  
    val birthDate: String,  
    val gender: String,  
    val email: String,  
)
```

2. Entity에 DTO 변경 기능 추가

```
@Entity
@Table(
    uniqueConstraints = [
        UniqueConstraint(name = "uk_member_login_id", columnNames = ["loginId"])
    ]
)
class Member(
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    var id: Long? = null,

    @Column(nullable = false, length = 30, updatable = false)
    val loginId: String,

    @Column(nullable = false, length = 100)
    val password: String,

    @Column(nullable = false, length = 10)
    val name: String,

    @Column(nullable = false)
    @Temporal(TemporalType.DATE)
    val birthDate: LocalDate,

    @Column(nullable = false, length = 5)
    @Enumerated(EnumType.STRING)
    val gender: Gender,

    @Column(nullable = false, length = 30)
    val email: String,
) {
    @OneToMany(fetch = FetchType.LAZY, mappedBy = "member")
    val memberRole: List<MemberRole>? = null

    // 추가
    private fun LocalDate.formatDate(): String =
        this.format(DateTimeFormatter.ofPattern("yyyyMMdd"))

    fun toDto(): MemberDtoResponse =
        MemberDtoResponse(
            id!!,
            loginId,
            name,
            birthDate.formatDate(),
            gender.desc,
            email
        )
}
```

3. service에 내 정보 보기 기능 추가

```

/**
 * 내 정보 보기
 */
fun searchMyInfo(id: Long): MemberDtoResponse {
    val member = memberRepository.findByIdOrNull(id)
        ?: throw InvalidInputException("id", "회원번호(${id})가 존재하지 않는 유저입니다.")
    return member.toDto()
}

```

4. controller에 EndPoint 추가

```

/**
 * 내 정보 보기
 */
@GetMapping("/info/{id}")
fun searchMyInfo(@PathVariable id: Long): BaseResponse<MemberDtoResponse> {
    val response = memberService.searchMyInfo(id)
    return BaseResponse(data = response)
}

```

5. Spring Security 권한 변경

```

@Bean
fun filterChain(http: HttpSecurity): SecurityFilterChain {
    http
        .httpBasic { it.disable() }
        .csrf { it.disable() }
        .sessionManagement {
            it.sessionCreationPolicy(SessionCreationPolicy.STATELESS)
        }
        .authorizeHttpRequests {
            it.requestMatchers("/api/member/signup", "/api/member/login").anonymous()
            it.requestMatchers("/api/member/info/**").hasRole("MEMBER")
            it.anyRequest().permitAll()
        }
        .addFilterBefore(
            JwtAuthenticationFilter(jwtTokenProvider),
            UsernamePasswordAuthenticationFilter::class.java
        )

    return http.build()
}

```

5.2. CustomUser로 Token에 User ID 관리하기

1. CustomUser 생성

```
package com.example.auth.common.dto

import org.springframework.security.core.GrantedAuthority
import org.springframework.security.core.userdetails.User

class CustomUser(
    val userId: Long,
    userName: String,
    password: String,
    authorities: Collection<GrantedAuthority>
) : User(userName, password, authorities)
```

2. createUserDetails 변경

```
private fun createUserDetails(member: Member): UserDetails =
    CustomUser(
        member.id!!,
        member.loginId,
        passwordEncoder.encode(member.password),
        member.memberRole!!.map { SimpleGrantedAuthority("ROLE_${it.role}") }
    )
```

3. Token 생성시 userId 정보도 기록

```
/**
 * token 생성
 */
fun createToken(authentication: Authentication): TokenInfo {
    val authorities: String = authentication
        .authorities
        .joinToString(",", transform = GrantedAuthority::getAuthority)

    val now = Date()
    val accessExpiration = Date(now.time + EXPIRATION_MILLISECONDS)

    // Access Token
```

```

        val accessToken = Jwts.builder()
            .setSubject(authentication.name)
            .claim("auth", authorities)
            .claim("userId", (authentication.principal as CustomUser).userId)
            .setIssuedAt(now)
            .setExpiration(accessExpiration)
            .signWith(key, SignatureAlgorithm.HS256)
            .compact()

        return TokenInfo("Bearer", accessToken)
    }

```

4. Token 정보 추출시 userId도 포함

```

/**
 * token 정보 추출
 */
fun getAuthentication(token: String): Authentication {
    val claims: Claims = getClaims(token)

    val auth = claims["auth"] ?: throw RuntimeException("잘못된 토큰 입니다.")
    val userId = claims["userId"] ?: throw RuntimeException("잘못된 토큰 입니다.")

    // 권한 정보 추출
    val authorities: Collection<GrantedAuthority> = (auth as String)
        .split(",")
        .map { SimpleGrantedAuthority(it) }

    val principal: UserDetails =
        CustomUser(userId.toString().toLong(), claims.subject, "", authorities)

    return UsernamePasswordAuthenticationToken(principal, "", authorities)
}

```

5. controller에 userId 가져오게 수정

```

/**
 * 내 정보 보기
 */
@GetMapping("/info")
fun searchMyInfo(): BaseResponse<MemberDtoResponse> {
    val userId = (SecurityContextHolder
        .getContext()
        .authentication
        .principal as CustomUser)
        .userId

    val response = memberService.searchMyInfo(userId)
    return BaseResponse(data = response)
}

```



5.3. 내 정보 변경 기능 만들기

0. 요구사항

EndPoint

```
PUT /api/member/info
```

회원가입시 입력받을 정보

Name	Description	Required
loginId	로그인 아이디	O
password	비밀번호, 영문/숫자/특수문자를 포함한 8~20자리	O
name	이름	O
birthDate	생년월일, YYYY-MM-DD 형식	O
gender	성별, MAN(남) 이나 WOMAN(여) 중 하나	O
email	이메일, 이메일 형식	O

1. service에 내 정보 수정 기능 추가

```
/**
 * 내 정보 수정
 */
fun saveMyInfo(memberDtoRequest: MemberDtoRequest): String {
    val member = memberDtoRequest.toEntity()
    memberRepository.save(member)
    return "수정 완료되었습니다."
}
```

2. controller에 EndPoint 추가

```
/**
 * 내 정보 저장
 */
@PutMapping("/info")
```

```
fun saveMyInfo(@RequestBody @Valid memberDtoRequest: MemberDtoRequest):  
    BaseResponse<Unit> {  
    val userId = (SecurityContextHolder  
        .getContext()  
        .authentication  
        .principal as CustomUser)  
        .userId  
    memberDtoRequest.id = userId  
    val resultMsg: String = memberService.saveMyInfo(memberDtoRequest)  
    return BaseResponse(messase = resultMsg)  
}
```