# 생활 환경에 따른 노인 기대 수명 및 질병 정보 제공 서비스

프로젝트 수행 결과서

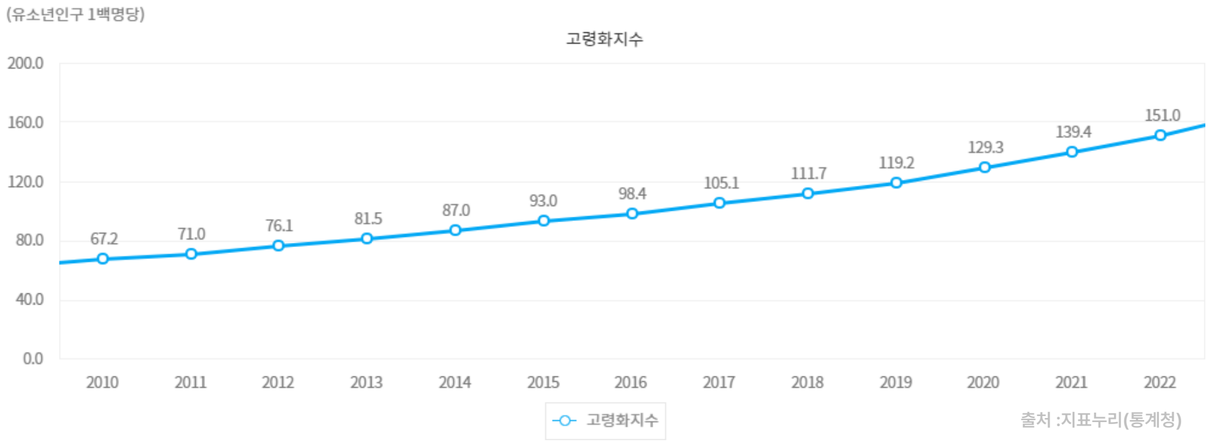
유정민

https://github.com/p1p2p1/Project_DLT

# 목차

# 기획 의도

프로젝트 기획서



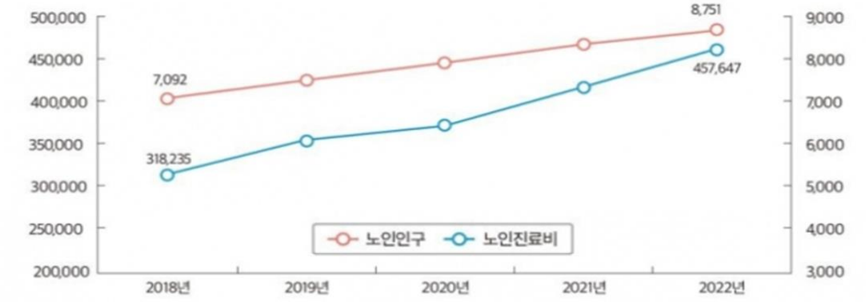

사회적 이슈로 계속해서 나타나는 **고령화 문제**

계속되는 고령화 문제로 인한 노년부양비가 계속해서 증가

실제로 노년 부양비 중 진료비는 상당한 비중을 차지

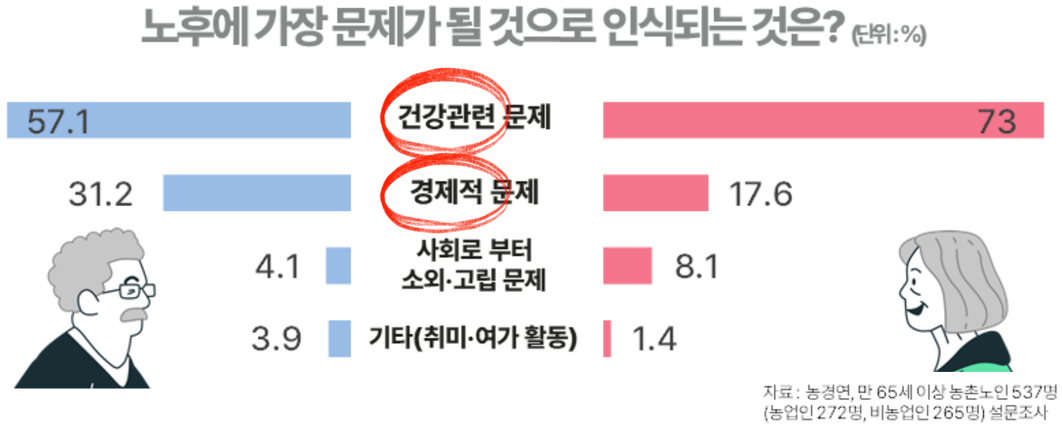
고령자 간병·치매보험 가입률 17.9%…"초고령화 시대 간병 위험 준비해야"

'고령화'에 불어나는 노인 진료비…작년 한 해 45조8000억원



# 기획 의도

프로젝트 기획서




기타

## 서울 VS 충북, 기대수명 2년 차… '건강 불평등' 때문?

전종보 헬스조선 기자

입력 2021/12/21 09:38

설문에서도 노후 걱정 중 건강, 경제적 문제를 가장 높게 인식

하지만 해당 정보의 접근성이 떨어진다.

이번 프로젝트를 통해,
지역별로 노인층의 **질병 정보**와 **정책 정보**를 제공하고,
건강과 직결되는 **기대 수명**에 대한 정보를 제공

# 개발 목표

프로젝트 기획서

## 개인별 기대 여명 예측

**연령대별**(성별, 기대 여명, 흡연 여부, 음주 여부, 고혈압 여부, 당뇨 여부) **데이터로 기대 여명을 예측할 수 있는 다중 선형 회귀 모델 훈련**

**사용자의 설문 조사를 진행**

**회귀 분석을 통해 만든 회귀식으로**

**실제 사용자의 기대 여명 예측**

## 지역별 기대 수명 예측

**지역별로** 4대 만성 질환(고지혈증, 치매, 당뇨, 고혈압), 스트레스 인지율과 음주율, 당뇨 비율, 사망률 **데이터를 분석**

**지역별 기대 수명과 연관성이 높은 특성 파악**

**비교하고 싶은 2개 지역과 미래 연도 항목을 선택**

**지역별 기대 수명을 그래프로 시각화**

# 기술 스택

사용 기술 목록

## 사용 언어


HTML5, CSS3, JavaScript

java
[11.0.9]


JSP
[2.3]


R Studio
[4.3.2]

python
[3.10.12]

## 서버 환경


Oracle DB
[11.2.0.2.0]

Apache tomcat
[9.0.84]


AWS EC2

Docker
[25.0.4]

## 개발 도구

Visual Studio Code
[1.85.2]

Google Colab

Eclipse
[2019.09]

git
[2.43.0]

Spring Framework
[4.3.9]

Spring Boot
[3.1.10]

# 프로젝트 사용 기술 경험

사용 기술 목록

## JAVA

- 다양한 미니 프로젝트 경험
- 주소록 관리 프로그램
  - 연락처 CRUD
  - 자바 컬렉션 프레임워크 사용 경험 (ArrayList, Set, Map, Stack)
- 다양한 자료구조 구현 경험
  - Single Linked List 구현 (노드 추가, 삭제, 역순으로 뒤집기 구현)
  - Queue 구현 (enqueue, dequeue 구현)
  - Stack 구현 (push, pop 구현)
  - Tree/Binary Tree 구현
- 다양한 알고리즘 구현 경험
  - Binary Search 구현
  - Linear Search 구현
  - Greedy Search 구현

## JSP, Servlet

- 개인별, 지역별 기대 수명 예측 서비스
- JSTL, ES6을 사용하여 데이터 처리 및 출력
- 세션을 사용한 로그인 세션 기능 구현
- 회원 관리 기능, 커뮤니티 게시판 기능 구현
- request, response를 사용하여 클라이언트의 요청 처리

## HTML5, CSS3

- 폼태그를 사용하여 서버와의 원활한 데이터 교환
- 그리드 시스템을 사용하여 반응형 구현
  - BootStrap을 사용하여 깔끔한 디자인 구성

## R

- 공공데이터포털을 활용한 데이터 분석
  - ggplot2를 사용한 시각화 가능
- 데이터 전처리
  - 연평균 데이터를 0과 1사이 값으로 스케일링
- 개인별 기대 수명 모델, 지역별 기대 수명 모델 훈련
  - 다중 선형 회귀 분석을 이용한 모델 훈련
- 단계적 선택법을 사용하여 특성 설정
  - stepWise() 함수를 사용하여 상관관계가 낮은 특성 제거

## JavaScript

- 클라이언트에서 서버에 데이터를 비동기적으로 요청 (AJAX, Fetch API 사용)
- Chart.js를 이용한 그래프 출력 (Chart.js 사용)
- 정규표현식을 이용하여 유효성 검사 진행

## Python

- 데이터 샘플 분석
  - matplotlib 패키지를 통해 데이터 시각화
- 데이터 전처리
  - sklearn 패키지의 StandardScaler 메서드를 사용하여 표준 점수로 전처리
- 다중 선형 회귀 모델 훈련
  - 과적합(Overfitting), 과소적합(underfitting) 개념 이해 및 모델 개선
- 특성 공학(Feature Engineering)을 통한 모델 과소적합 개선
  - sklearn 패키지의 PolynomialFeatures 메서드 사용
- 규제를 통해 모델의 과적합 방지
  - 릿지 회귀, 라쏘 회귀를 사용한 모델 과적합 방지

# 프로젝트 사용 기술 경험

사용 기술 목록

## Oracle DB

- 무결성 제약조건과 관계
  - 기본키, 외래키 설정
- DB 백업
- ERD 작성
- 인덱스 개념 이해
  - B-tree 개념 이해
- 부속질의 개념 이해
  - 중첩 쿼리
  - 스칼라 부속쿼리
- 데이터베이스 정규화 개념 학습

## Spring Boot

- Gradle을 사용한 프로젝트 관리
  - Spring 프레임워크에서 Spring Boot로 전환함으로써 프로젝트 관리 간편화
- Spring Security 경험
  - Authorization과 Authentication 차이 이해

## Spring

- STS3를 사용하여 스프링 프레임워크 사용
  - Servlet model2 -> Spring Web MVC 변경
- Spring IoC
- Spring DI
  - XML 파일과 Annotation 기법의 차이를 이해하고 빈을 관리
- Spring AOP
  - AOP 사용 경험 및 이해
- Spring JDBC
  - Repository 클래스를 사용하여 Oracle 데이터베이스 조회 및 조작 경험
- Spring Web MVC
  - 프로젝트 아키텍처인 MVC 패턴에 대한 이해 및 실제 프로젝트 적용
- MyBatis
  - ORM 프레임워크 사용으로 데이터베이스 상호작용 개선
  - 프로젝트에서 Query문 분리로 가독성 향상
- Spring Web MVC
  - 프로젝트 아키텍처인 MVC 패턴에 대한 이해 및 실제 프로젝트 적용
- maven을 사용한 라이브러리 종속성 관리

## Docker

- MicroService 아키텍처 구성
  - tomcat9과 Oracle DB 컨테이너 구성
- Docker 네트워크 설정
  - 컨테이너끼리 같은 네트워크로 설정하여 컨테이너 간 통신을 원활하게 설정

## Git

- 프로젝트 버전 관리
  - Git의 전체적인 구조 이해 및 사용
- Git의 다양한 커맨드 이해 및 사용
  - 스테이징, 커밋 개념 이해 및 브랜치 관리

## Linux

- VM(Virtual Machine)을 통한 리눅스 환경 서버 배포 실습
  - Oracle Virtual Box 사용
  - Ubuntu 사용
- 리눅스 디렉토리 개념 이해
  - i-node
- 리눅스 기본 명령어 실습
  - vim과 같은 편집기 사용 경험

## AWS EC2

- EC2 인스턴스를 생성하여 클라우드 서버 생성
- 통신 포트를 통한 서버 트래픽 제어
  - 인바운드, 아웃바운드 규칙 설정
  - 8181 : 서버 포트
  - 23 : Telnet
- SSH를 사용하여 서버 쉘 접속
  - 쉘을 사용한 Docker Container 관리
- FTP를 통해 EC2 서버에 파일 전송
  - FileZilla 사용

# 개발 스케쥴표

기획, 1차(UI, R, Python)

| 분류 | 담당자 | 작업 | 시작일 | 종료일 | 작업기간 |
|---|---|---|---|---|---|
| 기획 | 공통 | 아이디어 구상 | 2023-11-15 | 2023-11-21 | 6일 |
| | | 기획안 작성 | 2023-11-21 | 2023-11-22 | 2일 |
| | 박초윤 | 화면설계서 작성 | 2023-11-22 | 2023-11-30 | 8일 |
| | 유정민 | 요구사항 명세서/분석서 작성 | 2023-12-18 | 2023-12-21 | 4일 |
| 1차 UI | 유정민 | 기대 수명 페이지 구현 | 2023-12-07 | 2023-12-09 | 3일 |
| | | 커뮤니티 목록 페이지 구현 | 2023-12-11 | 2023-12-18 | 7일 |
| | 박초윤 | 정책 메인 페이지 구현 | 2023-12-07 | 2023-12-09 | 3일 |
| | | 커뮤니티 작성 페이지 구현 | 2023-12-09 | 2023-12-14 | 5일 |
| | | 커뮤니티 게시글 확인 페이지 구현 | 2023-12-15 | 2023-12-19 | 4일 |
| | 이연지 | 정책 지역별 상세 페이지 구현(서울, 전남) | 2023-11-29 | 2023-12-06 | 7일 |
| | | 정책 예산 비교 페이지 구현 | 2023-12-07 | 2023-12-11 | 4일 |
| | 박의범 | 지역별 통계 페이지 구현 | 2023-11-30 | 2023-12-06 | 6일 |
| | | 메인 페이지 구현 | 2023-12-06 | 2023-12-15 | 9일 |
| | | UI 페이지 최종 점검 | 2023-12-16 | 2023-12-18 | 2일 |
| 1차 빅데이터 분석 | 유정민 | [통계청]지역별 질병, 기대 수명, 건강 상태, 환경 데이터 수집 | 2023-11-22 | 2023-11-29 | 7일 |
| | | [기대 수명 예측]수집 데이터 전처리 | 2023-11-30 | 2023-12-06 | 6일 |
| | | [기대 수명 예측]지역 건강, 환경적 요소 분석 | 2023-12-09 | 2023-12-12 | 3일 |
| | | [기대 수명 예측]지역별 기대 수명 예측 | 2023-12-13 | 2023-12-20 | 7일 |
| | | [기대 수명 예측]빅데이터 분석서 작성 및 검토 | 2023-12-20 | 2023-12-21 | 2일 |

# 개발 스케쥴표

2차(Servlet model2), 3차(Spring Web MVC)

| | | | | | |
|---|---|---|---|---|---|
| 2차 서비스 구현 (Servlet model2) | 유정민 | 기대 수명 예측 서비스 구현 | 2024-01-15 | 2024-01-17 | 3일 |
| | | 로그인 세션, 회원가입 서비스 구현 | 2024-01-17 | 2024-01-22 | 4일 |
| | | 커뮤니티 댓글 작성 서비스 구현 | 2024-01-22 | 2024-01-24 | 3일 |
| | | 커뮤니티 글 작성 서비스 구현 | 2024-01-24 | 2024-01-27 | 4일 |
| | | 서비스 점검 및 리팩토링 | 2024-01-29 | 2024-01-31 | 3일 |
| | 박의범 | 지역별 질병률 데이터 제공 서비스 구현 | 2024-01-15 | 2024-01-20 | 6일 |
| | | 커뮤니티 글 목록 조회 서비스 구현 | 2024-01-22 | 2024-01-27 | 6일 |
| | | 개인정보 수정 서비스 구현 | 2024-01-29 | 2024-01-30 | 2일 |
| 3차 서비스 전환 (Spring) | 유정민 | Mapping 설계 및 사용 라이브러리 의존성 정리 | 2024-02-26 | 2024-02-26 | 1일 |
| | | 프로젝트 아키텍처 개선 | 2024-02-27 | 2024-02-27 | 1일 |
| | | 공통 페이지(navigation, footer) 전환 | 2024-02-27 | 2024-02-27 | 1일 |
| | | 커뮤니티 서비스 전환 | 2024-02-28 | 2024-02-29 | 2일 |
| | | 메인 페이지 전환 | 2024-03-04 | 2024-03-04 | 1일 |
| | | 질병 서비스 전환 | 2024-03-04 | 2024-03-05 | 2일 |
| | | 기대 수명 서비스 전환 및 개선 | 2024-03-05 | 2024-03-12 | 5일 |
| | | AWS EC2 인스턴스 생성 및 서버 구성 | 2024-03-12 | 2024-03-15 | 4일 |
| | | 클라우드 서버 배포 | 2024-03-15 | 2024-03-18 | 3일 |

# 요구사항 정의서

목차, 공통 기능, 메인 기능

## 목차



## 0. 공통 기능

▶ 메뉴의 클릭을 통해 다른 기능으로 이동할 수 있어야 한다.

▷건강·정책(질병 정보, 정책 정보 확인), 삶의 질 분석(기대 수명 예측, 당신의 만족도는?), 커뮤니티, 로그인 순으로 메뉴가 구성되어 있어야 한다.

▷질병 정보는 서울과 전남의 4대 질병 데이터를 그래프로 확인할 수 있어야 한다.

▷기대수명 예측은 개인별 기대 수명 정보를 예측해주고 지역별로 예측한 기대수명 데이터를 비교하여 그래프로 확인할 수 있어야 한다.

▷커뮤니티는 사용자들끼리의 정보와 지식을 공유할 수 있어야 한다.

▷메뉴는 웹 페이지 상단에 고정될 수 있어야 한다.

▶ 로고 이미지 클릭 시, 메인화면으로 이동할 수 있어야 한다.

## 1. 메인 기능

▶움직이는 화면(캐러셀)이 있어야 하고 각 화면을 이동할 수 있는 버튼이 있어야 한다.

▷캐러셀 화면은 총 3개의 화면으로 구성될 수 있어야 한다.

▷각 캐러셀 화면에는 주요 기능인 지역별 통계, 기대수명 예측, 정책을 설명할 수 있어야 한다.

▷각 캐러셀 화면에는 주요 기능인 지역별 통계, 기대수명 예측, 정책 기능으로 이동할 수 있는 버튼이 있어야 한다.

▶ 커뮤니티로 이동할 수 있는 기능이 있어야 한다.

▷커뮤니티에 대한 간단한 설명과 기능 소개가 있어야 한다.

▷커뮤니티의 사용자가 쓴 글을 5개만 구성 되어야 한다.

▶ 각 서비스에 맞는 이미지와 서비스로 이동할 수 있는 기능이 있어야 한다.

# 요구사항 정의서

서비스 요구사항

## 2. 질병 정보 제공 기능

▶ 맵 기능을 통해 지역을 선택할 수 있어야 한다.

▷지역을 선택할 경우 작은 웹 페이지가 만들어지고 데이터를 보여줘야 한다.

### 2.1. 지역별 질병 정보 제공 기능

▶만들어진 웹 페이지 해당 지역의 그래프가 해당 조건에 맞추어 데이터를 제공해야 한다.

▶연도를 설정하면 해당 연도에 맞게 데이터 그래프가 정보를 제공해야 한다.

▷시작 연도를 설정할 수 있어야 한다.

▷마지막 연도를 설정할 수 있어야 한다.

▶해당 지역의 시/군/구를 설정할 수 있어야 한다.

▶질병을 선택할 수 있는 버튼이 있어야 한다.

▷고지혈증 버튼을 이용하여 선택할 수 있도록 하고 데이터를 그래프에서 제공해야 한다.

▷치매 버튼을 이용하여 선택할 수 있도록 하고 데이터를 그래프에서 제공해야 한다.

▷당뇨 버튼을 이용하여 선택할 수 있도록 하고 데이터를 그래프에서 제공해야 한다.

▷고혈압 버튼을 이용하여 선택할 수 있도록 하고 데이터를 그래프에서 제공해야 한다.

▶차트 단위를 확인할 수 있는 버튼이 있어야 하고 단위에 대한 설명을 제공해야 한다.

▶각 질병들에 대한 설명이 있어야 한다.

▶왼쪽 아래 화살표 버튼을 통해 화면의 상단으로 이동할 수 있어야 한다.

## 3. 기대 수명 예측 기능

▶사용자의 정보를 받아서 개인별 기대 수명 데이터를 예측하여 제공할 수 있도록 해야 한다.

▷개인별 기대 수명을 예측하고 해당 기대 수명 데이터에 대한 회귀식과 출처에 대한 정보를 제공해야 한다.

▶두 지역의 미래의 예측한 기대수명 데이터를 동시에 제공하여 비교할 수 있도록 그래프 제공해야 한다.

▶지역을 설정할 수 있어야 한다.

▷첫 번째 지역을 설정할 수 있어야 한다.

▷두 번째 지역을 설정할 수 있어야 한다.

▶연도를 설정하면 해당 연도에 맞게 데이터 그래프가 정보를 제공한다.

▷기준 연도를 설정할 수 있어야 한다.

▶과거 기대수명 데이터로 그래프를 제공해야 한다.

▶지역을 설정할 수 있어야 한다.

▶차트에 대한 부가 설명이 있어야 한다.

▷각 데이터에 대한 출처와 예측에 사용된 데이터에 대한 설명이 있어야 한다.

## 4. 게시판 기능

▶게시판에 등록된 게시글 목록을 불러올 수 있어야 한다.

▶게시판에 글을 등록하려면 로그인을 할 수 있어야 한다

▷아이디, 비밀번호에 대한 유효성 검사를 진행할 수 있어야 한다.

▶로그인을 하기 위한 회원 가입 기능이 있어야 한다.

▷아이디, 비밀번호, 이메일, 휴대폰 번호에 대한 유효성 검사를 진행할 수 있어야 한다.

▶게시판에 글을 수정할 수 있어야 한다.

▷아이디, 비밀번호, 이메일, 휴대폰 번호에 대한 유효성 검사를 진행할 수 있어야 한다.

▶게시판에 글을 삭제할 수 있어야 한다.

▶게시판에 글에 댓글을 작성할 수 있어야 한다.

▶게시판에 글에 답글을 작성할 수 있어야 한다.

# 화면 설계서

## 기대 수명 페이지

| 프로젝트 명 | 생활 환경에 따른 노인 건강 분석 및 정책 제공 서비스 | 페이지 | 14-1 |
|---|---|---|---|
| 설명 | 기대수명 예측 페이지 (상단) | | |


| 기대수명 예측 페이지 Description | |
|---|---|
| 1 | 현재 본인의 나이를 직접 입력 혹은 ▲▼ 아이콘을 클릭하여 최소 20세에서 최대 84세까지 입력 가능하도록 설정 |
| 2 | 체크박스를 통해 본인에게 맞는 선택지를 선택할 수 있음 |
| 3 | 작성 완료 버튼 클릭 시 기대 수명, 기대 여명 확인 가능 |
| 4 | 값 도출 식 보기 버튼을 사용하여 기대 여명 예측 계산식에 대한 정보 확인 가능 |
| 5 | |

[기대 수명 제공 페이지(개인)]

| 프로젝트 명 | 생활 환경에 따른 노인 건강 분석 및 정책 제공 서비스 | 페이지 | 14-2 |
|---|---|---|---|
| 설명 | 기대수명 예측 페이지 (상단, 결과 화면) | | |


| 기대수명 예측 페이지 Description | |
|---|---|
| 1 | 작성 완료 버튼을 누르고 남은 기대 수명 수치와 기대 여명 수치 값을 표시 |
| 2 | 값 도출 식에 대한 설명으로 이동할 수 있는 버튼 표시 |
| 3 | |
| 4 | |
| 5 | |

[기대 수명 제공 페이지(개인, 결과)]

# 화면 설계서

## 기대 수명 페이지

| 프로젝트 명 | 생활 환경에 따른 노인 건강 분석 및 정책 제공 서비스 | 페이지 | 14-3 |
|---|---|---|---|
| 설명 | 기대수명 예측 페이지 (하단) | | |


| 기대수명 예측 페이지 Description | |
|---|---|
| 1 | 기대수명 예측 정보를 차트로 나타냄 |
| 2 | 셀렉트 박스를 이용해 확인할 지역과 비교할 지역 선택 가능 |
| 3 | 셀렉트 박스를 이용해 2024년 부터 2026년까지 선택 가능 |
| 4 | |
| 5 | |

[미래 기대 수명 제공 페이지(지역)]

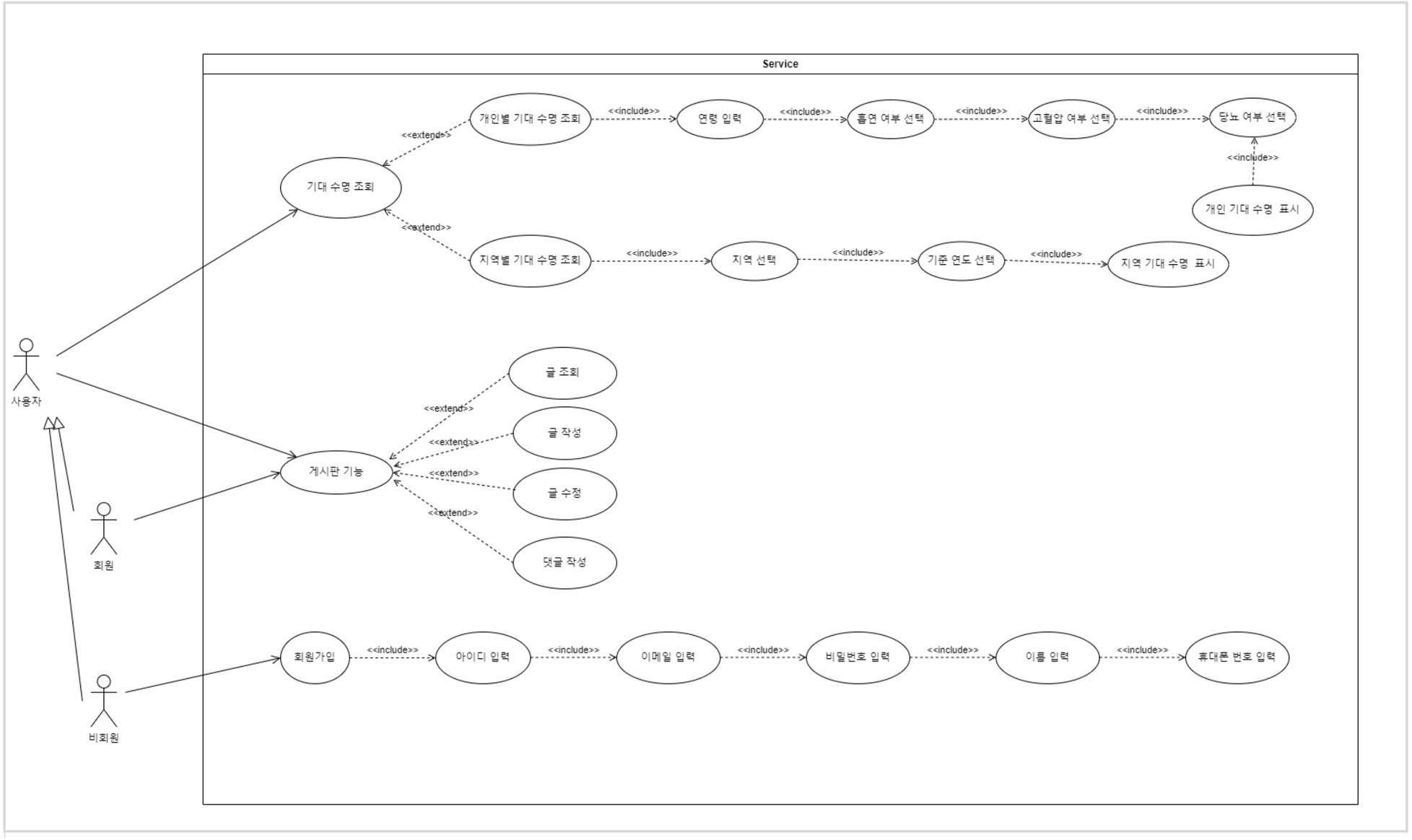
| 프로젝트 명 | 생활 환경에 따른 노인 건강 분석 및 정책 제공 서비스 | 페이지 | 14-4 |
|---|---|---|---|
| 설명 | 기대수명 예측 페이지 (중단) | | |


| 기대수명 예측 페이지 Description | |
|---|---|
| 1 | 과거 수명 정보를 차트로 나타냄 |
| 2 | 셀렉트 박스를 이용해 지역 설정 가능 |
| 3 | |
| 4 | |
| 5 | |

[과거 실제 수명 제공 페이지(지역)]

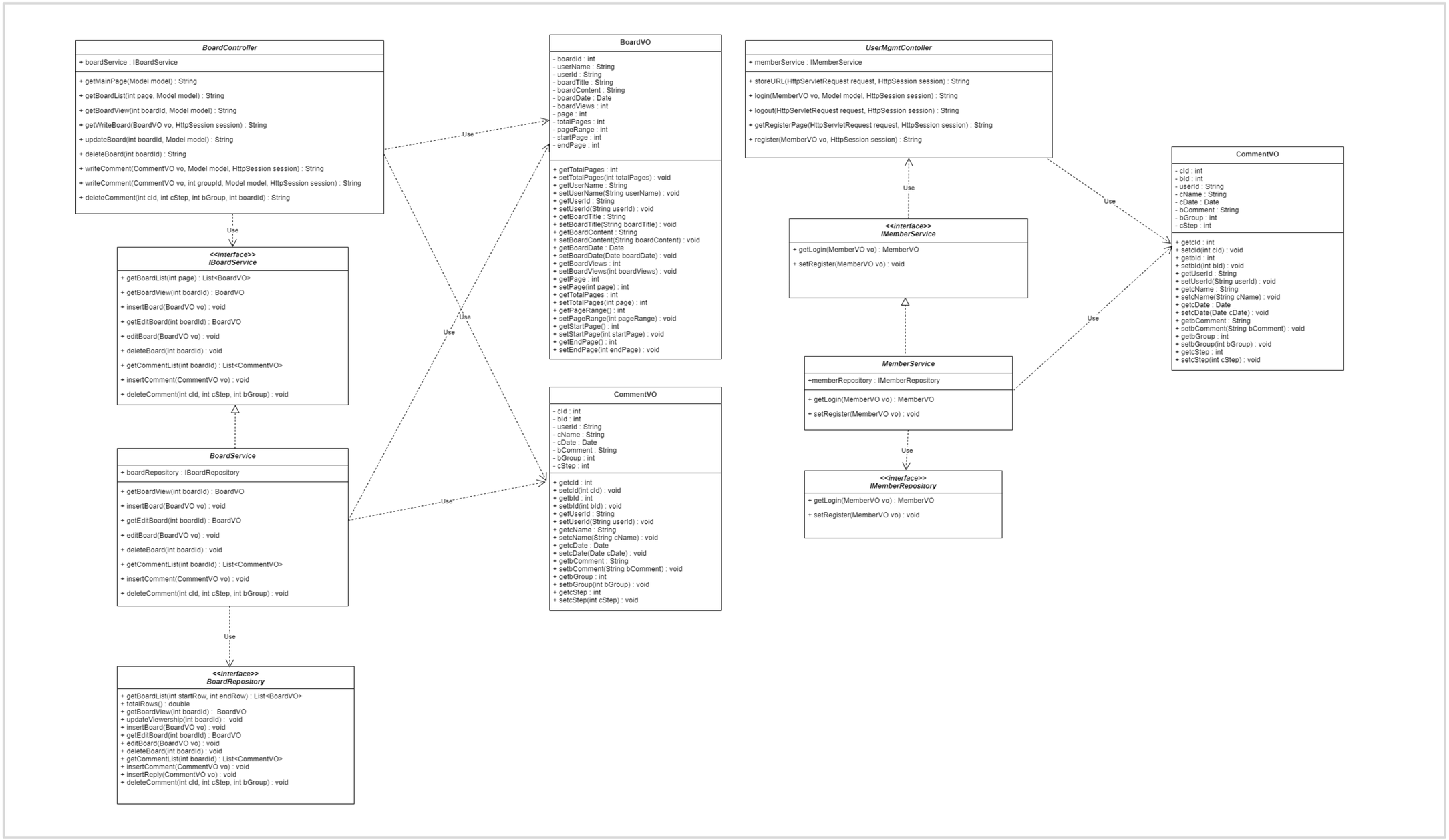
# UML

## Usecase Diagram



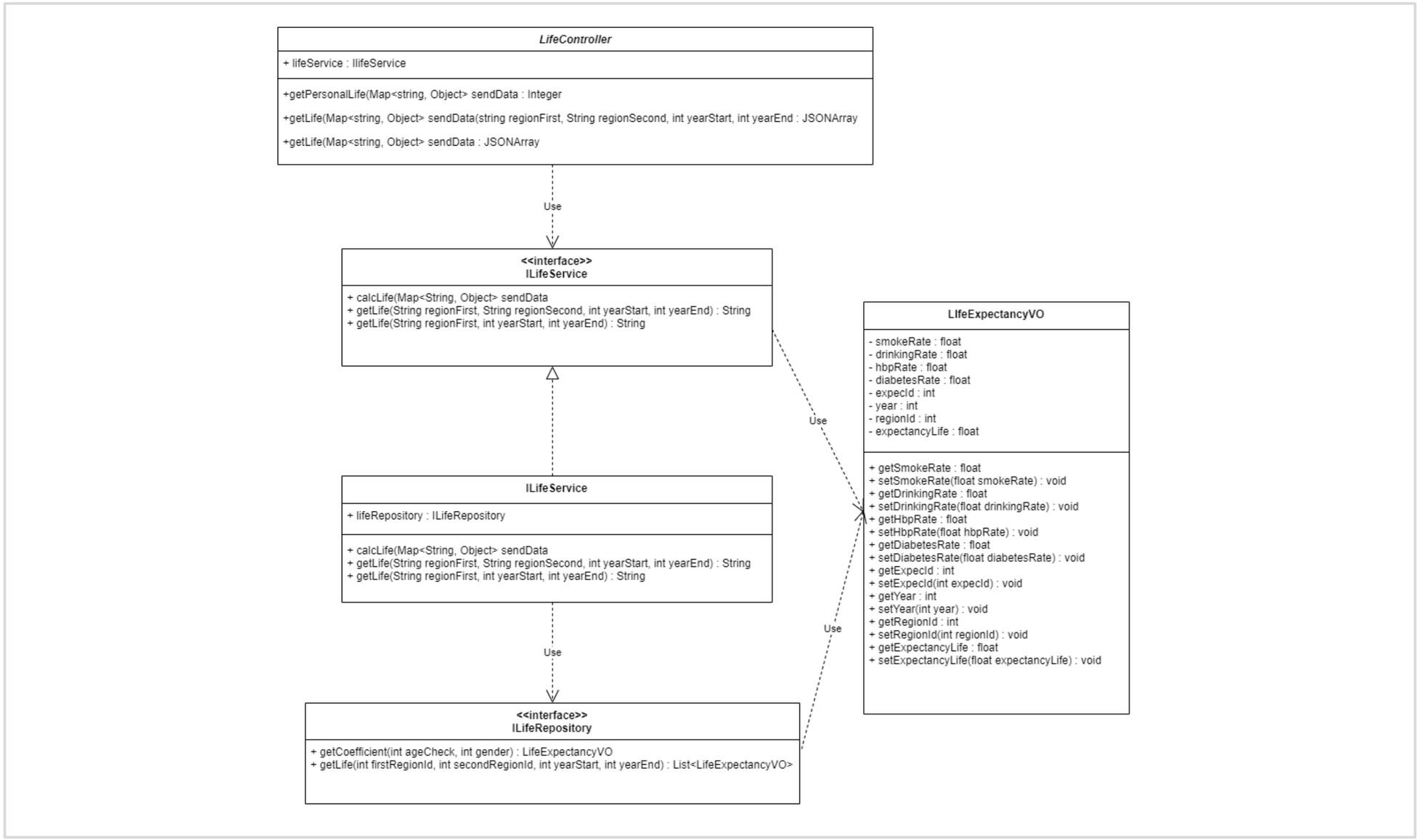
# UML

## Class Diagram(게시판, 회원관리)



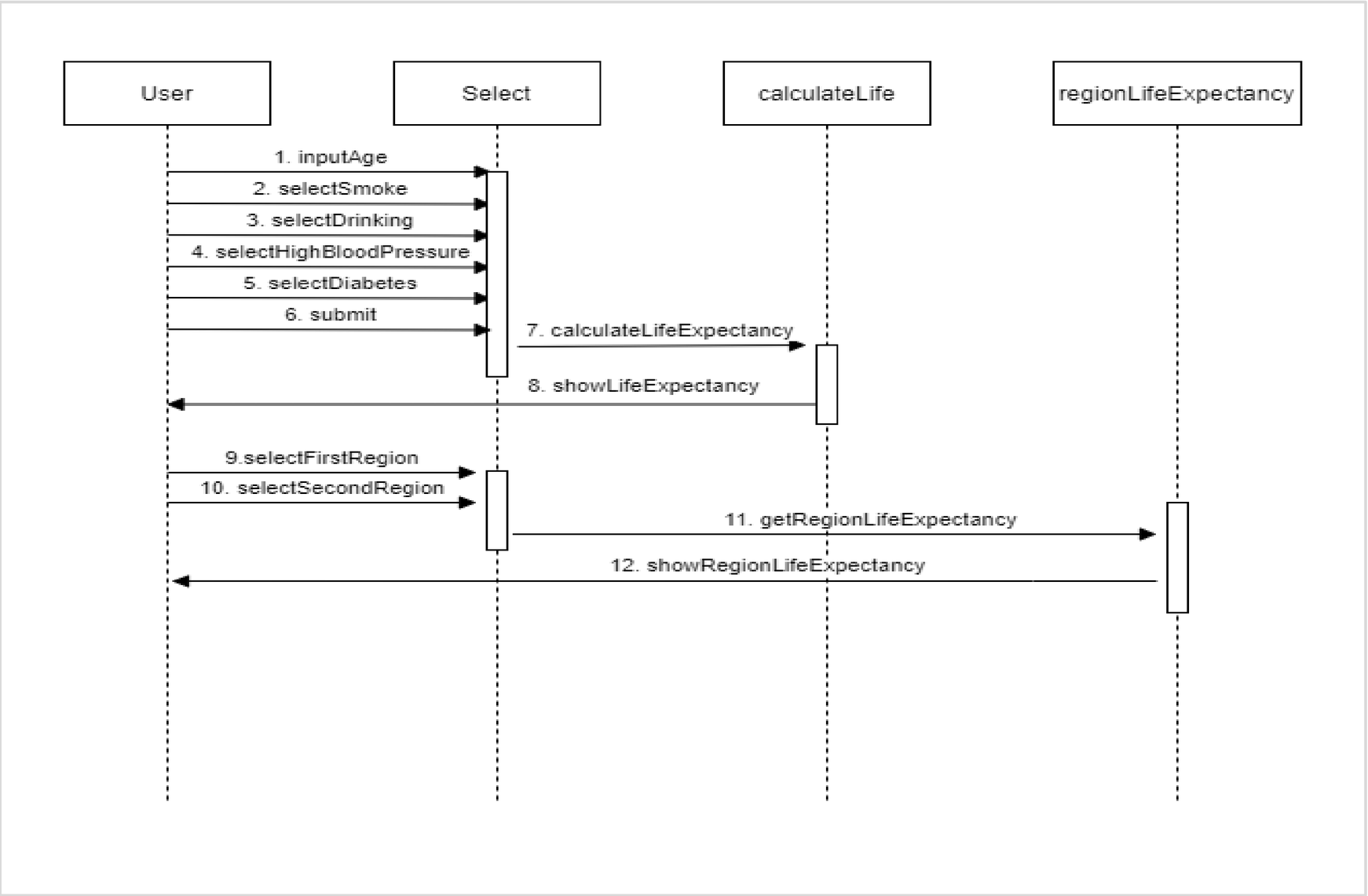
# UML

## Class Diagram(기대 수명 서비스)



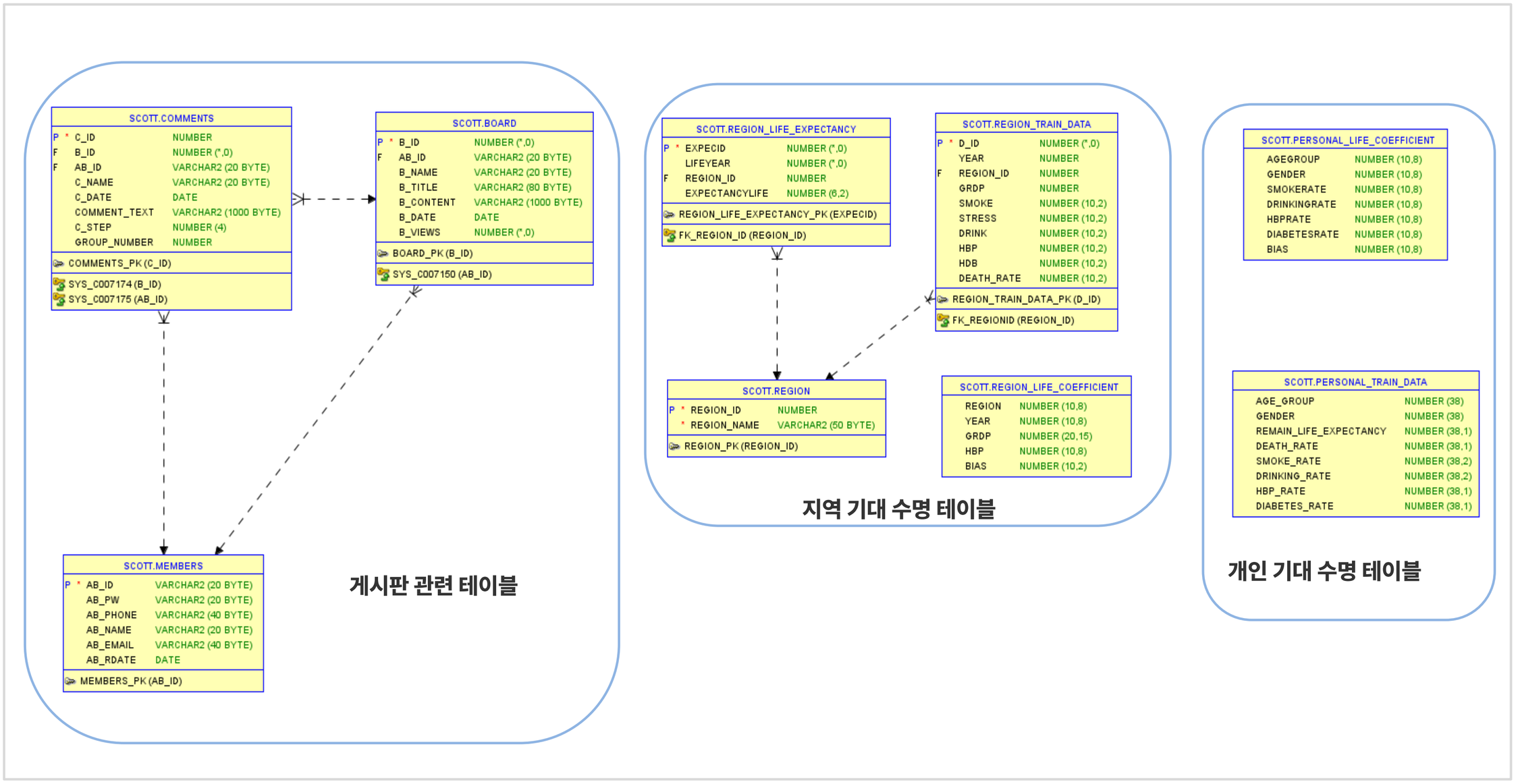
# UML

## Sequence Diagram



# ERD(Entity Relationship Diagram)

Oracle DB 테이블 구성



# 주요 기능 설명

주요 서비스 기능(개인별 기대 여명 정보 제공)


**사용자가 입력한 값**(나이, 성별, 흡연 여부, 음주 여부, 고혈압 여부)**을**
**릿지 회귀 모델을 사용하여 사용자의 기대 여명을 예측**

**사용자의 입력 값이 0과 1의 값으로**
**전처리되어 클라이언트에서**
**서버의 서비스 모델로 전송**

**기대 여명 계산 방법에 대한 정보를 버튼을 통해서**
**누구든지 접근 가능**

# 주요 기능 설명

주요 서비스 기능(개인별 기대 여명 정보 제공)


통계청의 연평균 데이터를 사용하여 훈련 시킨 회귀 모델을 사용

(지역, 연도별 GRDP, 고혈압 비율, 스트레스 인지율, 음주율, 당뇨 비율 데이터 사용)

지역별 기대 수명 정보를 시각화하여 정보 제공

사용자의 기대 수명과 지역별 기대 수명과 비교 가능

사용자가 비교하고 싶은 지역을 선택하여
지역별로 기대 수명을 비교 가능

해당 서비스의 정보는 비동기 방식을 사용하여 제공

# 주요 기능 설명

RequestMapping(기대 수명 예측 서비스)

| 기능 | Repository | URL | 요청 파라미터 | 요청방식 | View | 비고 |
|---|---|---|---|---|---|---|
| 기대 수명 페이지 조회 | 없음 | life | 없음 | GET | life/lifeService | |
| 개인 기대 수명 계산 | int calcLife | /life/lifeCalc | Map<String, Object> | POST | 없음 | ajax 요청 |
| 미래 기대 수명 계산 | String getLife | /life/getLife/{regionFirst}/{regionSecond}/{yearStart}/{yearEnd} | regionFirst, regionSecond, yearStart, yearEnd | GET | 없음 | ajax 요청 |
| 과거 기대 수명 계산 | String getLife | /life/getLife/{regionFirst}/{yearStart}/{yearend} | regionFirst, yearStart yearEnd | GET | 없음 | ajax 요청 |

# 주요 기능 설명

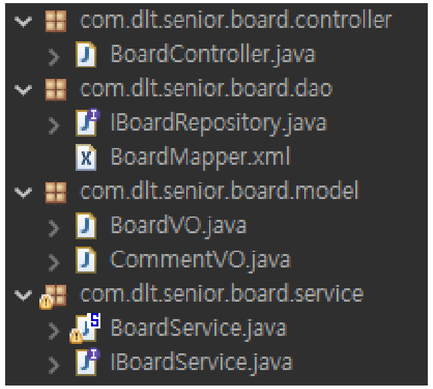
## RequestMapping(게시판 서비스)

| 기능 | Repository | URL | 요청 파라미터 | 요청방식 | View | 비고 |
|---|---|---|---|---|---|---|
| 메인 페이지 조회 | List<BoardVO> getBoardList()<br>double totalRows() | / | page | GET | index | |
| 글 리스트 조회 | BoardVO getBoardList() | /boardList/{page} | boardId | GET | board/boardList | |
| 상세 글 조회 | BoardVO getBoardView()<br>List<commentVO> getCommentList() | /boardView/{boardId} | boardId | GET | board/boardView | |
| 글 작성 페이지 조회 | 없음 | /getWriteBoard | 없음 | GET | board/boardWrite | |
| 글 작성 | void insertBoard() | /writeBoard | BoardVO | POST | redirect:/boardList/1 | |
| 글 수정 페이지 조회 | BoardVO getEditBoard() | /getEditBoard/{boardId} | boardId | GET | board/boardEdit | |
| 글 수정 | void editBoard() | /editBoard | BoardVO | POST | redirect:/boardView/ + boardId | |
| 글 삭제 | void deleteBoard() | /deleteBoard/{boardId} | Void | GET | redirect:/boardList/1 | |
| 댓글 작성 | void insertComment() | /boardView/writeComment | CommentVO | POST | redirect:/boardView/ + boardId | |
| 답글 작성 | void insertComment() | /boardView/writeComment/{groupId} | CommentVO | POST | redirect:/boardView/ + boardId | |
| 댓글 삭제 | void deleteComment() | /boardView/deleteComment/{cId}/{cStep}/{bGroup}/{boardId} | cId, cStep, cGroup | GET | redirect:/boardView/ + boardId | |

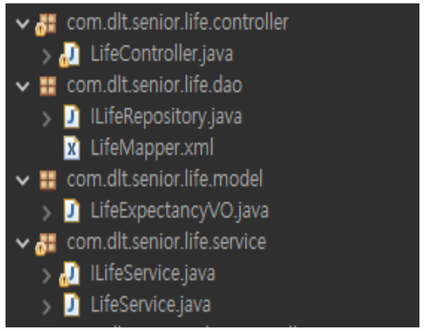
# 주요 기능 설명

## 프로젝트 패키지 구조

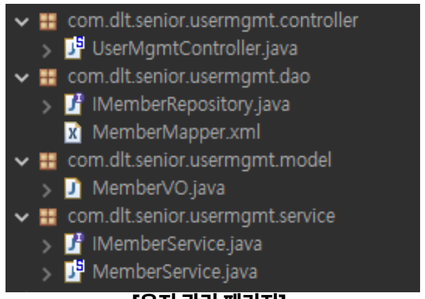
### Controller, Model




[FrontController, 게시판 패키지]

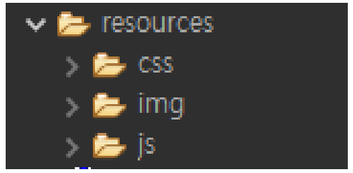



[기대 수명 관련 패키지]

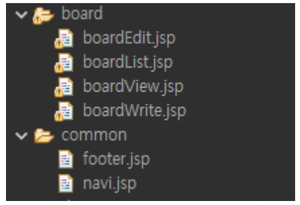


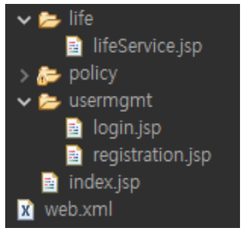

[유저 관리 패키지]

### View




[static 파일 ]




[전체적인 View]

[프로젝트 패키지]

## Controller

**com.dlt.senior.board.controller**

- BoardController.java : 게시판 관련 요청 처리

**com.dlt.senior.life.controller**

- LifeController.java : 기대 수명 관련 요청 처리

**com.dlt.senior.board.controller**

-UserMgmtController.java : 유저 관리 관련 요청 처리

## View

- index.jsp : 메인 페이지

**common 디렉토리** (jsp:include로 모든 페이지에 삽입)

- footer.jsp : footer 공통 페이지
- navi.jsp : navigation 공통 페이지

**board 디렉토리**

- boardEdit.jsp : 게시글 수정 페이지
- boardList.jsp : 게시글 목록 페이지
- boardView.jsp : 게시글 상세 페이지
- boardWrite.jsp : 게시글 작성 페이지

**life 디렉토리**

- lifeService.jsp : 기대 수명 서비스 페이지
- login.jsp : 로그인 페이지
- registration.jsp : 회원가입 페이지

## Model

[게시판]

**com.dlt.senior.board.dao**
- IBoardRepository.java
- BoardMapper.xml : myBatis Mapper

**com.dlt.senior.board.model**
- BoardVO.java
- CommentVO.java

**com.dlt.senior.board.service**
- IBoardService.java
- BoardService.java : 게시판 비즈니스 로직 처리

[기대 수명]

**com.dlt.senior.life.dao**
- ILifeRepostiory.java
- BoardMapper.xml : myBatis Mapper

**com.dlt.senior.life.model**
- LifeExpectancyVO

**com.dlt.senior.life.service**
- ILifeService.java
- LifeService.java : 기대 수명 비즈니스 로직 처리

[유저 관리]

**com.dlt.senior.userMgmt.dao**
- IMemberRepository.java
- MemberMapper.xml : myBatis Mapper

**com.dlt.senior.userMgmt.model**
- MemberVO.java

**com.dlt.senior.userMgmt.service**
- IMemberService.java
- MemberService.java

# 코드 설명

## [Controller] Java/ LifeController.java

```java
@Controller
public class LifeController {

    @Autowired
    ILifeService lifeService;

    //기대 수명 페이지 불러오기
    @GetMapping("life")
    public String getLifePage() {
        return "life/lifeService";
    }

    //개인 기대 수명 계산
    @PostMapping("lifeCalc")
    public ResponseEntity<Integer> getPersonalLife(@RequestBody Map<String,Object> sendData) {
        return ResponseEntity.ok()
        .contentType(new MediaType(MediaType.APPLICATION_JSON, StandardCharsets.UTF_8))
        .body(lifeService.calcLife(sendData));
    }

    //미래 기대 수명
    @GetMapping("getLife/{regionFirst}/{regionSecond}/{yearStart}/{yearEnd}")
    @ResponseBody
    public ResponseEntity<String> getLife(@PathVariable String regionFirst,@PathVariable String regionSecond,
        String json = lifeService.getLife(regionFirst, regionSecond, yearStart, yearEnd);
        return ResponseEntity.ok()
        .contentType(new MediaType(MediaType.APPLICATION_JSON, StandardCharsets.UTF_8))
        .body(json);
    }

    //과거 기대 수명
    @GetMapping("getLife/{regionFirst}/{yearStart}/{yearEnd}")
    @ResponseBody
    public ResponseEntity<String> getLife(@PathVariable String regionFirst, @PathVariable int yearStart, @Pat
        String json = lifeService.getLife(regionFirst, yearStart, yearEnd);
        return ResponseEntity.ok()
        .contentType(new MediaType(MediaType.APPLICATION_JSON, StandardCharsets.UTF_8))
        .body(json);
    }

}
```

Annotation을 사용하여 스프링 프레임워크에 Controller라고 알림

Spring DI를 사용하여 프레임워크가 인스턴스 생성을 관리하도록 설정

@GetMapping을 사용하여 Get 요청을 명시하고, String 타입을 return 함으로써 jsp 페이지를 클라이언트에 response

@PathVariable을 사용하여 클라이언트로부터 파라미터 값을 받아와서 처리

@ResponseBody를 사용하여 JSON 데이터를 처리

## [View] JavaScript/ lifeService.js

```javascript
fetch('lifeCalc', {
    headers : {
        "Content-Type" : "application/json",
    },
    method: 'POST',
    body: JSON.stringify(sendData)
})
    .then(response => response.json())
    .then(data => {
        const lifeContainer = document.getElementById('inputLife');
        if(data < 0){
            data = 1;
        }
        lifeContainer.innerHTML = '<div class = "col mb-1" style="font-weight : bold;">
    })
    .catch(error => {
        console.log(error);
        console.log("fetch가 제대로 작동하지 않습니다.");
    });
```

Fetch API를 사용하여 클라이언트에서 서버측으로 비동기적으로 요청을 보냄

## [View] navi.jsp

```jsp
<div>
    <c:forEach var="dto" items="${boardList}" varStatus="status">
        <div class="num">${dto.getBoardId() }</div>
        <div class="title">
            <a href="./boardView.do?boardId=${dto.getBoardId()}" style="text-decoration:none">${dto.getBoardTitle()}</a>
        </div>
        <div class="writer">${dto.getUserName() }</div>
        <div class="date">${dto.getBoardDate()}</div>
        <div class="count">${dto.getBoardViews()}</div>
    </c:forEach>
</div>
```

JSTL와 EL을 사용하여 데이터를 출력하거나 로그인 세션 관리

## [Model] ILifeRespository.java

```java
public interface ILifeRepository {
    public LifeExpectancyVO getCoefficient(@Param("ageCheck") int ageCheck, @Param("gender") int gender);
    public List<LifeExpectancyVO> getLife(@Param("firstRegionId") int firstRegionId, @Param("secondRegion]
}
```

@Param을 사용하여 메소드의 파라미터를 명시적으로 지정

## [Model] LifeMapper.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org/DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace = "com.dlt.senior.life.dao.ILifeRepository">
    <resultMap id = "lifeExpectancyMapping" type = "com.dlt.senior.life.model.LifeExpectancyVO">
        <result property = "smokeRate" column = "smoke_rate"/>
        <result property = "drinkingRate" column = "drinking_rate"/>
        <result property = "hbpRate" column = "hbp_rate"/>
        <result property = "diabetesRate" column = "diabetes_rate"/>
        <result property = "expecId" column = "expec_id"/>
        <result property = "year" column = "life_year"/>
        <result property = "regionId" column = "region_id"/>
        <result property = "expectancyLife" column = "expectancy_life"/>
    </resultMap>

    <select id = "getCoefficient" parameterType = "int" resultType = "com.dlt.senior.life.model.LifeExpectancyVO">
        SELECT smoke_rate AS smokeRate, drinking_rate AS drinkingRate, hbp_rate AS hbpRate, diabetes_rate AS diabetesRate FROM personal_train_data WHERE ag
    </select>

    <select id = "getLife" parameterType = "int" resultType = "com.dlt.senior.life.model.LifeExpectancyVO">
        <![CDATA[
            SELECT life_year AS year, region_id AS regionId, expectancy_life AS expectancyLife FROM region_life_expectancy WHERE life_year >= #{yearStart}
        ]]>
    </select>
</mapper>
```

## [Model] SLifeService.java

```java
@Service
public class LifeService implements ILifeService {
    @Autowired
    ILifeRepository lifeRepository;

    //개인별 기대 수명 계산
    @Override
    public int calcLife(Map<String, Object> sendData) {
        //R에서 계산한 계수(DB 커넥션 변경 예정)
        float ageGroup = (float) -0.8800585;
        float gender = (float) 1.8732418;
        float smokeRate = (float) -0.1005591;
        float drinkingRate = (float) -0.0948583;
        float hbpRate = (float) -0.0779488;
        float diabetesRate = (float) -0.0672881;
        float bias = (float) 82.9454053;

        int ageCheck = Integer.parseInt((String) sendData.get("personal_age"))/5 * 5;
        int genderCheck = Integer.parseInt((String) sendData.get("genderCheck"));
        int smokingCheck = Integer.parseInt((String) sendData.get("smokeCheck"));
        int drinkingCheck = Integer.parseInt((String) sendData.get("drinkingCheck"));
        int hbpCheck = Integer.parseInt((String) sendData.get("hbpCheck"));
        int diabetesCheck = Integer.parseInt((String) sendData.get("diabetesCheck"));

        LifeExpectancyVO vo = lifeRepository.getCoefficient(ageCheck, genderCheck);

        float ageX = ageGroup * ageCheck;
        float genderX = gender * genderCheck;
        float smokeX = vo.getSmokeRate() * smokingCheck * smokeRate;
        float drinkX = vo.getDrinkingRate() * drinkingCheck * drinkingRate;
        float hbpX = vo.getHbpRate() * hbpCheck * hbpRate;
        float diaX = vo.getDiabetesRate() * diabetesCheck * diabetesRate;



        //흡연율, 음주율, 고혈압 비율, 당뇨병 비율 -> 여부로 데이터 변환
        float lifeExpectancy = Math.round(bias + (ageX + genderX + smokeX + drinkX + hbpX + diaX));
        lifeExpectancy = lifeExpectancy - 10;
```

비즈니스 로직을 처리하는 Service 계층으로써,

Repository에서 데이터를 가져와서 계산

## [Model] LifeExpectancyVO.java

```java
public class LifeExpectancyVO {
    //계수(기대 수명 계산기)
    private float smokeRate;
    private float drinkingRate;
    private float hbpRate;
    private float diabetesRate;

    //기대 수명
    private int expecId;
    private int year;
    private int regionId;
    private float expectancyLife;

    public float getSmokeRate() {
        return smokeRate;
    }
    public void setSmokeRate(float smokeRate) {
        this.smokeRate = smokeRate;
    }
    public float getDrinkingRate() {
        return drinkingRate;
    }
```

VO를 사용하여

Controller와 View, Model 간의 데이터 교환

# 주요 기능 설명

프로젝트 코드 설명(Python, 개인별 기대 수명 회귀 모델)

## 활용 데이터

- 연령대, 성별 기대 여명 데이터(통계청)
- 연령대, 성별 연평균 흡연율, 음주율, 고혈압  유병률, 당뇨 비율(통계청)

(개인정보 데이터는 구할 수 없기에 샘플의 수가 부족하여 임의로 샘플의 수를 늘려 진행)

```
from google.colab import files
from google.colab import drive
import pandas as pd
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
#데이터 로딩
new_data = pd.read_excel('/content/drive/MyDrive/refined_samples.xlsx')
new_data.head()
```

R을 사용하여 전처리한 샘플을 사용하여 훈련

| | age_group | gender | smoking | drinking | obesity | diabetes | hypertension | remain_life_expectancy |
|---|---|---|---|---|---|---|---|---|
| 0 | 50.606061 | 1 | 0 | 1 | 1 | 1 | 0 | 36.279175 |
| 1 | 60.202020 | 2 | 0 | 1 | 1 | 0 | 1 | 32.069439 |
| 2 | 71.313131 | 2 | 0 | 0 | 1 | 1 | 1 | 14.090213 |
| 3 | 79.393939 | 2 | 0 | 1 | 1 | 0 | 0 | 11.577030 |
| 4 | 61.212121 | 2 | 1 | 1 | 0 | 0 | 0 | 35.430143 |

특성 데이터

타겟 데이터

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

total_data = new_data.drop('remain_life_expectancy', axis=1)
life_data = new_data['remain_life_expectancy']


train_input, test_input, train_target, test_target = train_test_split(total_data, life_data, random_state=40)

ss = StandardScaler()
ss.fit(total_data)
train_input = ss.transform(train_input)
test_input = ss.transform(test_input)

lr = LinearRegression()
lr.fit(train_input, train_target)
```

train_test_split() 메서드를 사용하여

훈련 샘플과 테스트 샘플을 분리

간단한 다중 선형 회귀 모델 훈련

```
train_score :  0.933205725406334
test_score :  0.8851358200128143
```

훈련 세트의 점수가 테스트 점수에 비해 높게 나와서 과적합으로 보일 수 있지만, 두 점수가 전부 낮기 때문에 과소적합 의심

특성 공학(Feature Engineering)을 통해 과소적합 문제 해결

# 주요 기능 설명

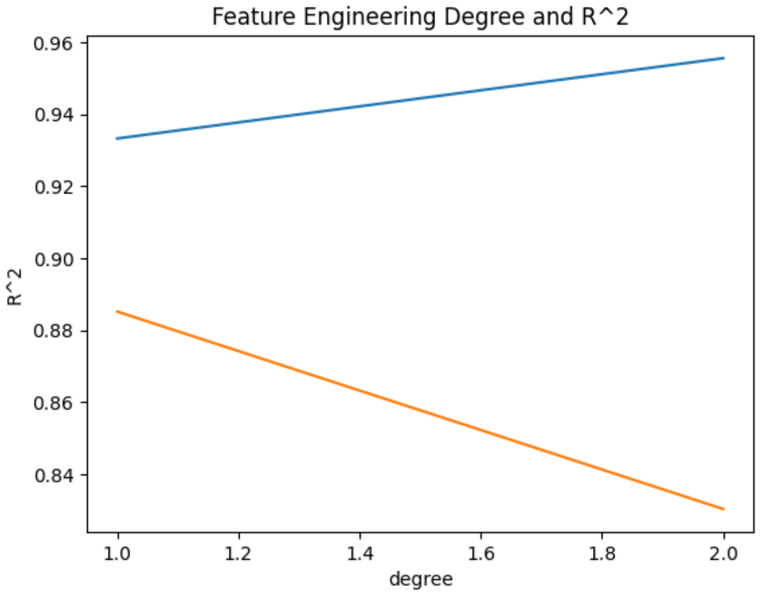
프로젝트 코드 설명(Python, 개인별 기대 수명 회귀 모델)

## 특성 공학(Feature Engineering) 적용

```
import matplotlib.pyplot as plt
import numpy as np
print(train_poly_test)
print(test_poly_test)

plt.plot(range(1,3), train_poly_test[:2])
plt.plot(range(1,3), test_poly_test[:2])
plt.xlabel("degree")
plt.ylabel("R^2")
plt.show()
```

```
[0.933205725406334, 0.9555119386671708, 0.9930676533106392, 1.0]
[0.8851358200128148, 0.8303001788227004, -1.7103183281782873e+21, -48.3642682656666]
```

[Degree 수치에 의한 훈련 세트와 테스트 세트 점수 수치 비교]




[Degree 수치에 의한 훈련 세트와 테스트 세트 점수 그래프 비교]

계수 값이 제곱(degree = 2)인 경우가 가장 모델이 좋다고 판단

```
poly = PolynomialFeatures(degree = 2,include_bias = False)
poly.fit(train_input)
train_poly = poly.transform(train_input)
test_poly = poly.transform(test_input)
lr.fit(train_poly, train_target)
print(lr.score(train_poly, train_target))
print(lr.score(test_poly, test_target))
```

```
0.9555119386671708
0.8303001788227004
```

모델이 훈련 세트를 더 잘 학습하게 되었지만, 테스트 세트의 값이 떨어져서 과적합(Overfitting) 의심

과적합을 개선하기 위해 규제(Regularzation)가 적용 된 릿지(Ridge) 회귀 모델과 라쏘(Lasso) 회귀 모델 사용

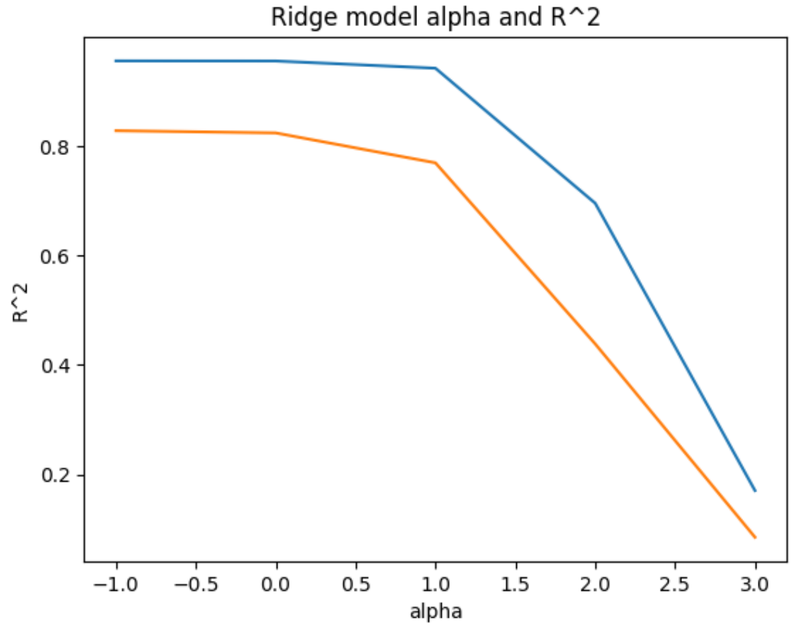
stepAIC(단계적 선택법)을 활용하여 모델 성능 향상

# 주요 기능 설명

프로젝트 코드 설명(Python, 개인별 기대 수명 회귀 모델)

## 릿지(Ridge) 회귀

```
import matplotlib.pyplot as plt
import numpy as np
plt.plot(np.log10(alpha_list), train_score)
plt.plot(np.log10(alpha_list), test_score)
plt.xlabel('alpha')
plt.ylabel('R^2')
plt.title("Ridge model alpha and R^2")
plt.show()
```




[Ridge 규제 정도에 따른 훈련 세트와 테스트 세트의 점수 비교]

```
ridge = Ridge(alpha = 10)
ridge.fit(train_poly, train_target)
print(ridge.score(train_poly, train_target))
print(ridge.score(test_poly, test_target))
```

0.9421483498023238
0.7692720770102084

릿지 회귀를 적용하고 훈련 세트의 점수는 감소했지만,

테스트 점수의 감소 폭이 너무 심하여 오히려 모델이 악화

릿지 회귀는 모델의 계수(가중치)들을 0으로 만들지 않고 전부 다 사용하기 때문에 모델이 개선되지 않고 오히려 악화

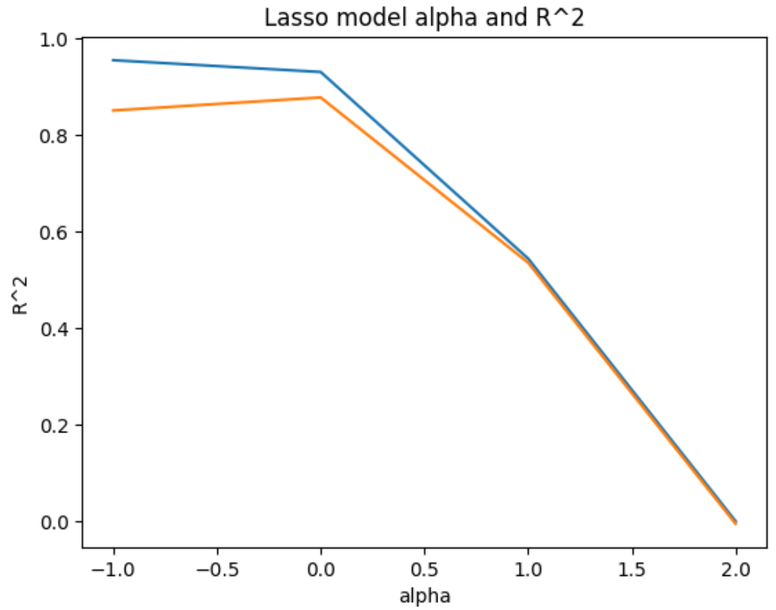
가중치를 0으로 만드는 라쏘 회귀 모델을 사용하여 점수를 비교

# 주요 기능 설명

프로젝트 코드 설명(Python, 개인별 기대 수명 회귀 모델)

## 라쏘(Lasso) 회귀

```
plt.plot(np.log10(alpha_list), train_score)
plt.plot(np.log10(alpha_list), test_score)
plt.xlabel('alpha')
plt.ylabel('R^2')
plt.title("Lasso model alpha and R^2")
plt.show()
```




[Lasso 규제 정도에 따른 훈련 세트와 테스트 세트의 점수 비교]

```
lasso = Lasso(alpha = 0.1)
lasso.fit(train_poly, train_target)
print(lasso.score(train_poly, train_target))
print(lasso.score(test_poly, test_target))
```

0.9540579519743183
0.8500151792722059

→ 라쏘 회귀 모델을 사용했더니

훈련 세트, 테스트 세트 모두 점수가 증가

훈련 세트, 테스트 세트 모두 점수가 증가함으로써 모델이 개선되었음을 알 수 있다.

이는 릿지 회귀와 다르게 라쏘 회귀에서는 모델이 필요없다고 생각한 계수들의 값을 0으로 만들어 규제하기 때문이다.

# 주요 기능 설명

프로젝트 코드 설명(R, 개인별 기대 수명 회귀 모델)

## 활용 데이터

- 지역별 0세 기준 기대 여명 데이터(통계청)
- 연도, 지역별 GRDP, 흡연율, 스트레스 인지율, 음주율, 고혈압 유병률, 당뇨율, 사망률(통계청)

```r
#전체 지역 기대 수명 예측 모델 훈련
life_expectancy_model <- lm(life.expectancy ~ ., data = dataTotal)
life_expectancy_model

summary(life_expectancy_model)
```

```r
#전체 지역 기대 수명 예측 모델 단계적 선택법 적용
life_expectancy_model2 <- stepAIC(life_expectancy_model)
life_expectancy_model2

summary(life_expectancy_model2)
```

[R Script]

```
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -9.633e+02  6.749e+02  -1.427   0.1840
year         5.260e-01  3.393e-01   1.550   0.1521
region      -4.059e+00  1.998e+00  -2.031   0.0697 .
GRDP        -8.226e-06  7.897e-06  -1.042   0.3221
smoking     -2.025e-02  1.579e-01  -0.128   0.9005
stress      -4.860e-02  7.148e-02  -0.680   0.5119
alcohol     -7.208e-03  5.463e-02  -0.132   0.8977
hbp         -9.726e-02  2.804e-01  -0.347   0.7359
diabetes    -2.257e-01  8.339e-01  -0.271   0.7921
death.rate  -6.208e-04  3.038e-03  -0.204   0.8422
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.2716 on 10 degrees of freedom
Multiple R-squared:  0.9835,    Adjusted R-squared:  0.9687
F-statistic: 66.42 on 9 and 10 DF,  p-value: 9.818e-08
```

stepwise(단계적 선택법)

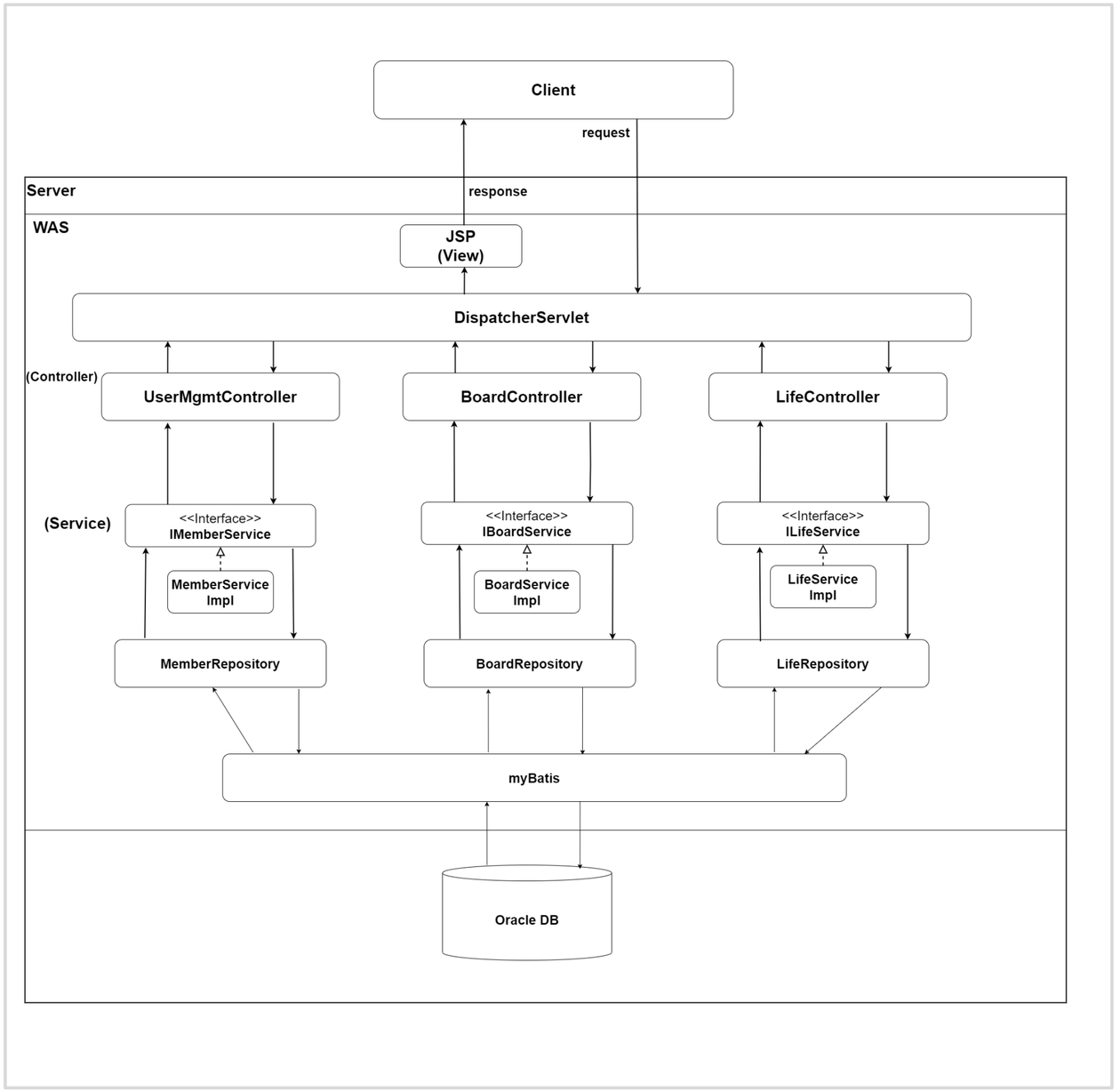
```
> summary(life_expectancy_model2)

Call:
lm(formula = life.expectancy ~ year + region + GRDP + hbp, data = dataTotal)

Residuals:
     Min       1Q   Median       3Q      Max
-0.79972 -0.03190  0.02279  0.10866  0.17409

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -8.406e+02  1.861e+02  -4.516 0.000410 ***
year         4.627e-01  9.358e-02   4.944 0.000176 ***
region      -3.601e+00  8.793e-01  -4.096 0.000955 ***
GRDP        -5.633e-06  2.973e-06  -1.895 0.077547 .
hbp         -1.269e-01  1.003e-01  -1.266 0.224996
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.2282 on 15 degrees of freedom
Multiple R-squared:  0.9826,    Adjusted R-squared:  0.9779
F-statistic: 211.4 on 4 and 15 DF,  p-value: 5.394e-13
```

stepAIC(단계적 선택법)을 활용하여 모델 성능 향상

[Model Summary]

# Software Architecture

## Spring Framework



# 향후 개발 계획

## “Spring Security 프레임워크 적용”

- Spring Security 프레임워크를 사용하여 프로젝트의 부실한 회원 관리 시스템의 보안 취약점 개선
- Session -> JWT 로그인 전환, 보안 헤더 설정을 통해 애플리케이션의 보안 강화

## “모델의 개선을 위한 다양한 기법 적용”

- 서비스에 사용된 모델을 개선하기 위해 다양한 기법을 공부하여 적용
- 결정 트리와 같은 비선형 모델이나 경사 하강법을 사용함으로써 손실 함수를 낮춰 모델 개선

## “ Spring 애플리케이션과 python 연결”

- 계수를 직접 서비스 계층에 입력하는게 아닌 python으로 구성 된 스크립트를 Flask를 사용하여 서비스 계층을 구성하고 Spring 기반 애플리케이션을 Restful API로 연결하도록 구성
- 모델을 발전시킬 때마다 서비스 계층의 코드를 수정하는 불편함 개선

## “ React 라이브러리를 사용하여 클라이언트 개선”

- 컴포넌트 기반 구성인 React를 사용하여 반복 사용된 클라이언트 측 코드를 정리하여 재사용성과 가독성 향상
- Virtual DOM을 사용하여, 브라우저의 데이터 변화 발생 시에 필요한 부분만 업데이트하여 로딩 속도 개선 및 효율 향상

# 프로젝트 수행 소감

## 선형 회귀 모델 개발 경험

선형 회귀 모델을 개발하는 도중 어려웠던 점은 데이터 탐색, 데이터 전처리 과정이었습니다. 모델을 훈련시키기 위한 맞춤형 데이터는 존재하지 않았고, 데이터를 여러 곳에서 가져오며 사용했기에 모델이 과적합되거나 과소적합이 되는 문제가 발생했습니다.

모델의 데이터 샘플이 잘못되었다고 생각하여 샘플만을 찾아다니며 모델의 정확도를 높이는데만 집중했습니다. 하지만, 어떤 모델이든 정확도가 100%가 될 수 없는 것을 깨닫고, 지금 가진 데이터 샘플을 통해 모델의 개선에 집중하여 특성 공학, 규제와 같은 다양한 기법들을 공부하며 모델에 적용하여, 점차 모델을 개선시킬 수 있었습니다.

이 경험을 통해 프로젝트 진행 중이라 할지라도 지속적인 학습이 필요하고 학습을 통해 쌓은 지식을 실제로 사용하였을 때의 즐거움을 깨달았습니다.

## 팀장이라는 역할에 대한 고민

처음 진행하는 웹 개발 프로젝트의 팀장을 맡으며 여러가지 걱정이 되었습니다.

프로젝트 진행 도중에도 제 서비스도 구현하며 다른 팀원들의 작업 일정 조율이나 팀원들의 이탈로 인한 나머지 인원들의 부담, 팀원들의 Spring 프레임워크 학습 문제와 같은 여러 가지 다양한 문제가 발생했습니다.

하지만, 팀원들에게 제가 이해한 개념들을 설명해주고 도와주며 각자 할수 있는 만큼 집중한 결과, 무사히 프로젝트를 마칠 수 있었습니다.

이 경험을 통해 예상치 못한 상황에 대한 대처 능력 전체적인 상황을 넓게 볼 수 있는 안목과 자신감을 얻을 수 있었습니다.

## 설계에 대한 중요성

프로젝트를 시작하고 기획에 대한 문서를 작성했습니다. 프로젝트 기획서부터 화면 설계서, 개발 스케줄 등 프로젝트의 전체적인 설계를 진행하고 프로젝트 코드 구현을 진행했습니다.

대학 재학 중에는 프로젝트를 완료하고 문서를 작성했다면, 프로젝트를 진행하기 위해 문서를 작성하는 과정은 조금 버거웠습니다.

하지만, 이런 문서화 과정으로 인해서 프로젝트를 실제로 구현하는 과정에서 작성한 문서들을 참고하여 진행했기에 프로젝트가 원활하게 진행되었습니다.

이 경험을 통해 프로젝트 기획과 설계에 대한 문서화 작업의 중요성을 깨달았습니다.

# 감사합니다

유정민

https://github.com/p1p2p1/Project_DLT