# 3. Design




| Student No. | 22311977 |
|---|---|
| Name | 송서현 |
| E-mail | angels0315@daum.net |

# [ Revision history ]

| Revision date | Version # | Description | Author |
|---|---|---|---|
| 05/06/2025 | 1.0 | First Draft | 송서현 |
| 14/06/2025 | 1.1 | 공개 코스 기능 추가 | |
| | | | |
| | | | |
| | | | |
| | | | |

= Contents =

1. Introduction .....4

2. Class diagram .....5

3. Sequence diagram .....11

4. State machine diagram .....17

5. Implementation requirements .....18

6. Glossary .....19

7. References .....19

# 1. Introduction

## 1) Summary

에전에 갔던 곳을 다시 가고 싶어도 기억이 잘 안나는 경우가 많다. Tripiary는 이러한 문제를 방지하여준다. 친구나 연인, 가족 등과 함께 여행을 다녀온 후, 자신만의 여행 코스를 지도에 남기고 공유할 수 있는 웹 애플리케이션이다. 사용자는 지도를 통해 자신만의 여행 코스를 생성하고, 해당 위치에 마커를 추가하거나 메모를 남기며 여행 일지를 작성할 수 있다.

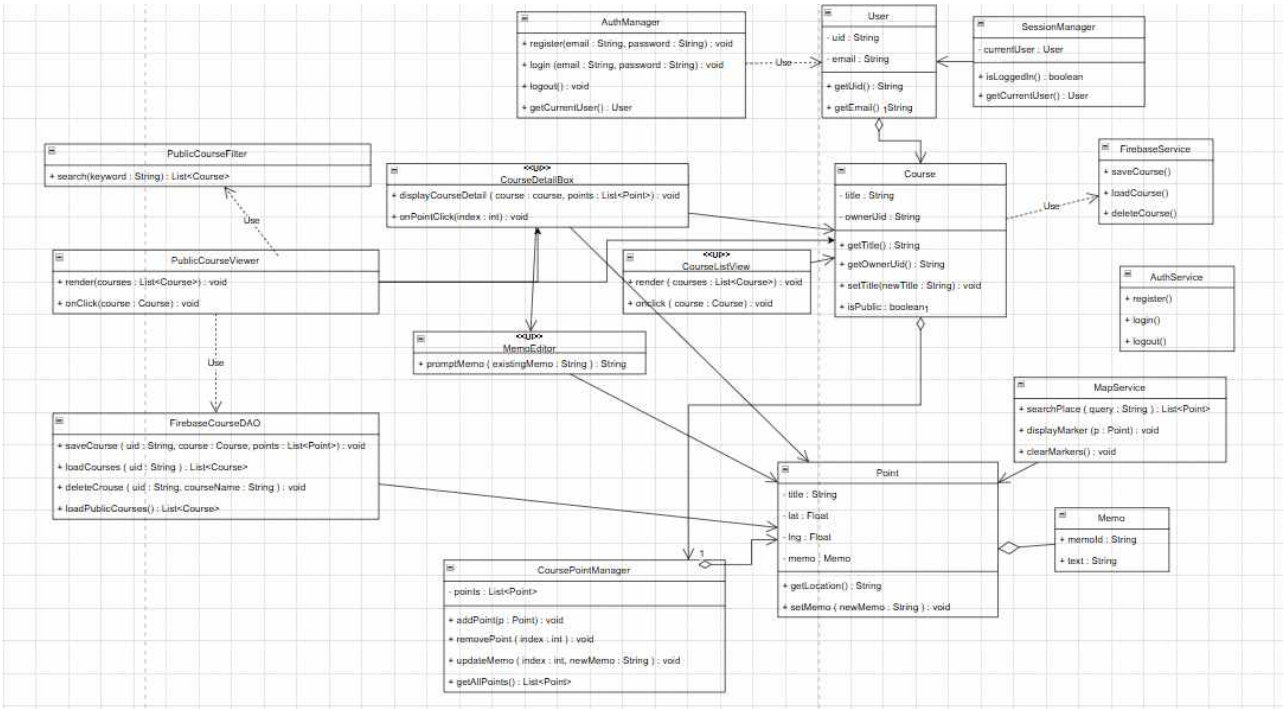
본 Design 문서는 Tripiary 시스템의 각 기능을 실제로 '어떻게 구현할 것인가'에 대한 내용을 다루며, 구현 단계에서 필요한 구조적 설계와 흐름을 구체화하였다.

문서 전반에는 Use Case Diagram, Sequence Diagram, State Diagram 등을 포함하여 Tripiary의 주요 기능등을 시각화하였다. Tripiray는 웹 기반 어플리케이션으로 HTML/CSS/JavaScript를 기반으로 하여 구현된다.

## 2) Important Points of Design

- 코스 생성 기능 : 사용자가 지도를 클릭하여 코스를 생성하고, 코스 이름을 설정할 수 있다.
- 장소 검색 및 마커 추가 : kakao map api를 사용하여 장소를 검색하고, 검색 결과를 코스에 마커로 추가할 수 있다.
- 마커 메모 작성 기능 : 마커마다 텍스트 메모를 남길 수 있어 여행 당시의 추억이나 정보를 기록할 수 있다.
- 코스 편집 기능 : 기존에 만든 코스를 다시 수정하거나 장소를 삭제/추가할 수 있다.
- 공개 코스 기능 : 사용자가 자신의 코스를 공개코스로 설정할 수 있고 공개코스 페이지에서 다른 사용자의 코스들을 확인할 수 있다.

2. Class diagram




1) User

| Attributes |
|---|
| uid : String : uid 정보<br>email : String : email 정보 |
| Methods |
| getUid() : String : 현재 사용자 객체에 저장된 고유 식별자(uid)를 문자열 형태로 반환한다. 이 UID는 사용자의 인증과 데이터 구분에 사용된다.<br>getEmail() : String : 현재 사용자 객체에 저장된 이메일 주소를 반환한다. 이메일은 로그인 시 인증 정보로 사용되며 사용자 식별의 주요 수단이다. |
| Description |
| User 클래스는 Firebase 인증을 통해 생성된 사용자 계정을 표현하는 데이터 클래스이다. 각 사용자는 고유한 UID와 이메일 주소를 가지며, 이 정보는 코스 작성 시 소유자 식별 및 인증 처리에 활용된다. 사용자 관련 데이터를 저장하고, 외부에서 사용자 정보를 쉽게 접근할 수 있도록 getter 메서드를 제공한다. |

2) SessionManager

| Attributes |
| --- |
| currentUser : User : currentUser 정보 |
| Methods |
| isLoggedIn() : boolean : 현재 로그인된 사용자가 존재하는지를 boolean 값으로 반환한다. 세션 유지 여부를 판별할 때 사용되며, 로그인 상태 확인 기능을 제공한다.<br>getCurrentUser() : User ; 현재 로그인한 사용자(User 객체)를 반환한다. 해당 객체를 통해 사용자 정보를 가져오거나 데이터에 접근할 수 있다. |
| Description |
| SessionManager 클래스는 클라이언트 측에서 로그인 세션을 관리하는 역할을 한다. 현재 로그인된 사용자의 정보를 보관하고 있으며, 로그인 상태 여부를 확인하거나 현재 사용자 객체를 반환하는 기능을 제공한다. 이 클래스는 인증 후 사용자 식별과 접근 권한 설정에 활용된다. |

3) AuthManager

| Attributes |
| --- |
| firebaseAuth : Object : Firebase 인증 모듈 인스턴스 |
| Methods |
| register (email : String, password : String) : void : Firebase를 통해 새 사용자를 등록한다.<br>login (email : String, password : String) : void : 사용자 인증 후 세션을 생성한다.<br>logout() : void : 현재 로그인된 사용자를 로그아웃 시킨다.<br>getCurrentUser() : User : 현재 로그인된 사용자 정보를 가져온다. |
| Description |
| AuthManager 클래스는 Firebase 인증 시스템과 연결되어 사용자 로그인, 로그아웃, 회원가입 등의 기능을 제공하는 서비스성 클래스이다. firebaseAuth 속성을 통해 외부 인증 모듈 인스턴스를 관리하며, 모든 기능은 이 객체를 통해 실행된다. |

4) AuthService

| Attributes |
| --- |
| authManager : AuthManager : 인증 기능을 위임하는 객체 |
| Methods |
| register(email: String, password: String): void : 사용자의 이메일과 비밀번호를 받아 Firebase에 회원가입 요청을 보낸다. 성공 시 해당 사용자는 인증 데이터베이스에 등록된다.<br>login(email: String, password: String): void : 사용자 입력 정보를 바탕으로 로그인 요청을 수행하고, 성공 시 세션이 생성된다.<br>logout(): void : 현재 로그인된 사용자의 세션을 해제하고, 클라이언트에서 사용자 상태를 초기화한다.<br>getCurrentUser(): User : Firebase에서 현재 인증된 사용자 정보를 받아 User 객체로 반환한다. 이 객체는 이후 데이터 접근과 사용자 구분에 사용된다. |
| Description |
| AuthManager 클래스는 Firebase 인증 시스템과 직접 통신하며 사용자 인증 기능을 수행하는 클래스이다. 회원가입, 로그인, 로그아웃 등의 기능을 제공하며, 사용자 인증과 관련된 핵심 로직을 담당한다. 내부적으로 Firebase 인증 API를 래핑하여 UI 또는 서비스 계층에서 쉽게 호출할 수 있도록 돕는다. |

5) Course

| Attributes |
| --- |
| title : String : title 정보<br>ownerUid : String : ownerUid 정보 |
| Methods |
| getTitle(): String : 코스의 제목을 반환한다. 이 값은 사용자 인터페이스에서 코스를 식별하는 데 사용된다.<br>getOwnerUid(): String : 코스를 생성한 사용자의 UID를 반환한다. 이는 사용자별 코스를 구분하거나 권한을 검증할 때 사용된다.<br>setTitle(newTitle: String): void : 코스의 제목을 새로운 문자열로 설정한다. 코스 편집 시 사용자가 제목을 수정할 수 있도록 한다. |
| Description |
| Course 클래스는 사용자가 만든 여행 코스 데이터를 표현하는 도메인 객체이다. 각 코스는 제목과 소유자 UID 정보를 가지고 있으며, 장소 리스트 및 메모 정보와 연결되어 저장된다. 이 클래스는 사용자 맞춤형 코스를 생성하고 저장할 때 기본 단위를 제공한다. |

6) FirebaseService

| Attributes |
| --- |
| firestore : Object : Firestore DB 인스턴스 |
| Methods |
| saveCourse() : void : Firestore에 코스 데이터를 저장한다.<br>loadCourse() : void : Firestore에서 코스 데이터를 불러온다.<br>deleteCourse() : void : 선택한 코스 데이터를 Firestore에서 삭제한다. |
| Description |
| FirebaseService 클래스는 Firebase Firestore와 직접 통신하여 데이터를 저장하고 불러오는 역할을 한다. firestore 속성은 데이터베이스 연결을 유지하고 각 메서드에서 사용한다. |

7) FirebaseCourseDAO

| Attributes |
| --- |
| firebaseService : FirevaseService : Firebase 연동 객체 |
| Methods |
| saveCourse( uid : String, course : Course, points : List<Point> ) : void : 특정 사용자의 코스를 Firestore에 저장한다.<br>loadCourses ( uid : String ) : List<Course> : 사용자의 모든 코스 목록을 불러온다.<br>deleteCourse(uid: String, courseName : String ) : void : 선택한 코스를 Firestore에서 삭제한다. |
| Description |
| FirebaseCourseDAO 클래스는 데이터 저장 및 조회 기능을 실제로 구현하는 DAO 계층이다. FirebaseService를 통해 Firestore에 접근하며, 코스 저장, 불러오기, 삭제 등의 구체적인 작업을 수행한다. |

8) CoursePointManager

| Attributes |
| --- |
| points : List<Point> : points 정보 |
| Methods |
| addPoint(p:Point) : void : 새로운 장소를 현재 코스에 추가한다.<br>removePoint ( index : int) : void : 지정한 인덱스에 위치한 장소를 코스에서 제거한다.<br>updateMemo ( index : int, newMemo : String ) : void : 특정 장소에 연결된 메모를 수정한다.<br>getAllPoints() : List<Point> : 현재 코스에 등록된 모든 장소 리스트를 반환한다. |
| Description |
| CoursePointManager 클래스는 사용자가 코스를 편집할 때, 해당 코스에 추가된 장소들을 관리하는 역할을 한다. 장소를 추가하거나 삭제하고, 특정 장소에 메모를 부여하거나 수정하는 기능을 포함하며, 코스 내 장소 정보를 일관되게 유지하는 데 핵심적인 역할을 한다. 이 클래스는 UI와 데이터 저장 계층 사이의 중간 역할을 하며, 장소 리스트의 상태 관리를 책임진다. |

9) Point

| Attributes |
| --- |
| title : String : title 정보<br>lad : Float : 위도 정보<br>lng : Float : 경도 정보<br>memo : Memo : memo |
| Methods |
| getLocation() : String : 장소의 위도/경도를 문자열로 반환한다.<br>setMemo (newMemo, String) ; void : 이 장소에 연결된 메모 내용을 설정한다. |
| Description |
| Point 클래스는 코스에 포함되는 단일 장소를 표현하는 데이터 객체이다. 각 장소는 제목, 위도, 경도 정보를 포함하고, 선택적으로 메모를 가질 수 있다. 이 클래스는 지도 상의 특정 위치를 나타내며, 사용자 경험을 개인화하는 메모 기능과 연동된다. |

10)Memo

| Attributes |
| --- |
| memold : String : memold 정보<br>text : String : text 정보 |
| Methods |
| |
| Description |
| Memo 클래스는 특정 장소(point)에 연결되어 작성되는 텍스트 메모 데이터를 표현한다. 이 메모는 사용자가 장소에 대해 기록하고 싶은 설명, 일정 등을 자유롭게 입력하는 용도로 사용되며, 코스 편집 및 조회 시 함께 출력된다. Memo 객체는 pointId에 의해 각 장소와 연결된다. |

11) MapService

| Attributes |
| --- |
| mapAPI : Object : 카카오 지도 API 인스턴스 |

| Methods |
| --- |
| searchPlace(query : String) : List<Point> : 사용자가 입력한 키워드(query)를 기반으로 카카오 지도 API를 통해 장소를 검색한다. 검색 결과는 Point 객체들의 리스트로 반환되며, 지도 상에 표시될 수 있다.<br>displayMarker ( p : Point ) : void : 주어진 장소(Point)의 위치에 마커를 지도 위에 표시한다. 마커에는 장소명과 클릭 이벤트가 포함될 수 있다.<br>clearMarker() : void : 현재 지도에 표시된 모든 마커를 제거하여 화면을 초기화한다. 코스 변경이나 검색 초기화 시 사용된다. |
| Description |
| MapService 클래스는 외부 지도 API(카카오 지도)를 활용하여 장소 검색, 마커 표시, 마커 초기화 등 지리적 기능을 제공한다. 사용자가 장소를 검색하거나 코스를 편집할 때 필요한 지도 상의 시각적 피드백을 담당하며, 지도와 사용자의 상호작용의 중심적인 역할을 한다. |

12) CourseListView

| Attributes |
| --- |
| |
| Methods |
| render ( course : List<Course> ): void : 사용자가 보유한 코스 목록을 리스트의 형태로 UI에 출력한다. 각 코스는 제목과 식별자로 표시되며 클릭 가능하다.<br>onClick(course:Course): void : 리스트에서 코스를 클릭하면 해당 코스의 상세 정보 화면으로 이동하거나 표시되도록 처리한다. |
| Description |
| CourseListView 클래스는 사용자 인터페이스에서 전체 코스 목록을 출력하고, 선택된 코스를 처리하는 역할을 한다. 각 코스를 클릭하면 해당 코스의 상세 내용을 볼 수 있도록 연결되며, 사용자와의 상호작용이 주로 발생하는 프론트엔드 컴포넌트이다. |

13) CourseDetailBox

| Attributes |
| --- |
| |
| Methods |
| displayCourseDetail ( course : Course, point: List<Point>) : void : 선택된 코스의 제목과 포함된 장소(Point) 목록을 상세 화면에 시각적으로 표시한다. 이 정보는 사용자가 해당 코스를 수정하거나 확인할 수 있게 해준다.<br>onPointClick ( index : int ) : void : 상세 정보 화면 내 장소 리스트에서 특정 장소를 클릭하면 관련 이벤트를 발생시킨다. |
| Description |
| CourseDetailBox 클래스는 사용자가 선택한 코스의 상세정보를 시각적으로 보여주는 UI 컴포넌트이다. 포함된 장소들의 리스트를 출력하고, 각 장소에 대한 클릭 이벤트도 제공하여 직관적인 사용자 경험을 가능하게 한다. |

14) MemoEditor

| Attributes |
| --- |
| |
| Methods |
| promptMemo(existingMemo : String) : String : 기존에 작성된 메모가 잇을 경우 팝업 창에 해당 내용을 미리 띄운다. 사용자는 이를 수정하거나 새로 작성하여 저장할 수 있으며, 결과 문자열이 반환된다. |
| Description |
| MemoEditor 클래스는 사용자가 특정 장소에 대한 메모를 입력하거나 수정할 수 있도록 돕는 UI 요소이다. 팝업 창 형식으로 작동하며, 사용자는 친화적인 방식으로 메모 내용을 편집하고 적용할 수 있도록 지원한다. |

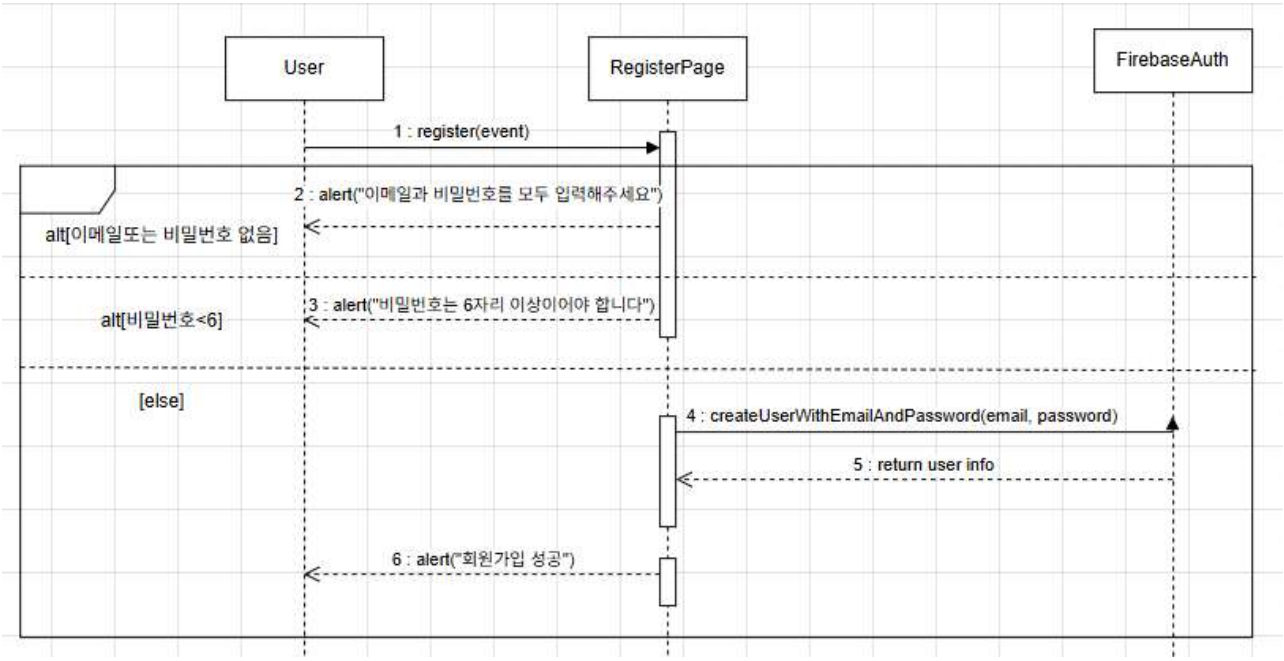
15) PublicCourseViewer

| Attributes |
| --- |
| course : List<Course> : 현재 로드된 공개 코스 목록을 저장하는 리스트이다. |
| Methods |
| render(Course: List<Course>) : void : 전달받은 공개 코스 리스트를 화면에 렌더링한다.<br>onClick(course : Course) : void : 공개 코스 카드 중 하나를 클릭했을 때, 해당 코스의 상세 보기 화면으로 이동하거나 로직을 수행한다. |
| Description |
| PublicCourseViewer 클래스는 전체 Tripiary 사용자들이 공유한 공개 코스들을 화면에 렌더링하고 사용자와의 인터랙션을 관리하는 UI 컴포넌트이다. 이 클래스는 FirebaseCourseDAO로부터 공개 코스를 불러와 내부 courses 리스트에 저장하고, 각각의 코스를 카드 형태로 사용자에게 표시한다. 사용자가 코스를 클릭하면 상세 페이지로 연결하는 역할을 하며, 사용자 관점에서 “공개 코스 탐색” 기능의 핵심 뷰어 역할을 수행한다. |

16) PublicCourseFilter

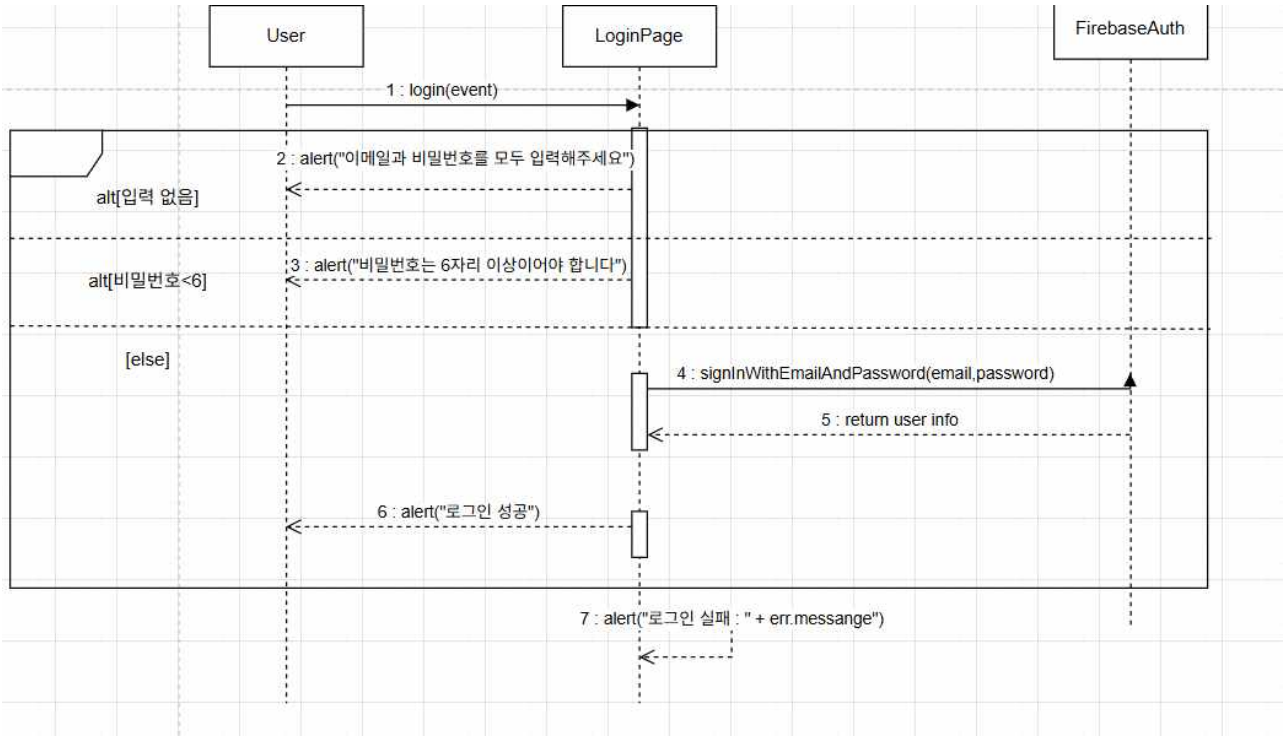
| Attributes |
| --- |
| keyword : String : 사용자로부터 입력받은 검색 키워드 ( 코스 제목 또는 장소명 검색에 사용 ) |
| Methods |
| search(keyword :String, courses : List<Course>) : List<Course> : 주어진 코스 리스트 중 키워드가 제목 또는 장소명에 포함된 공개 코스를 필터링하여 반환한다. |
| Description |
| PublicCourseFilter 클래스는 사용자 입력 기반의 검색 기능을 수행하며, 공개 코스 중 조건에 부합하는 코스만 선별하는 역할을 담당한다. 이 클래스는 PublicCourseViewer와 연동되어 동작하며, 사용자가 입력한 키워드를 기준으로 Course.title 또는 Point.title 안에 해당 문자열이 포함된 코스만 필터링한다. UI와 직접 연결되기보다는 로직 중심의 유틸리티 클래스이며, render() 전에 필터링된 결과를 전달하는 중간 처리 계층으로 활용된다. |

## 3. Sequence diagram



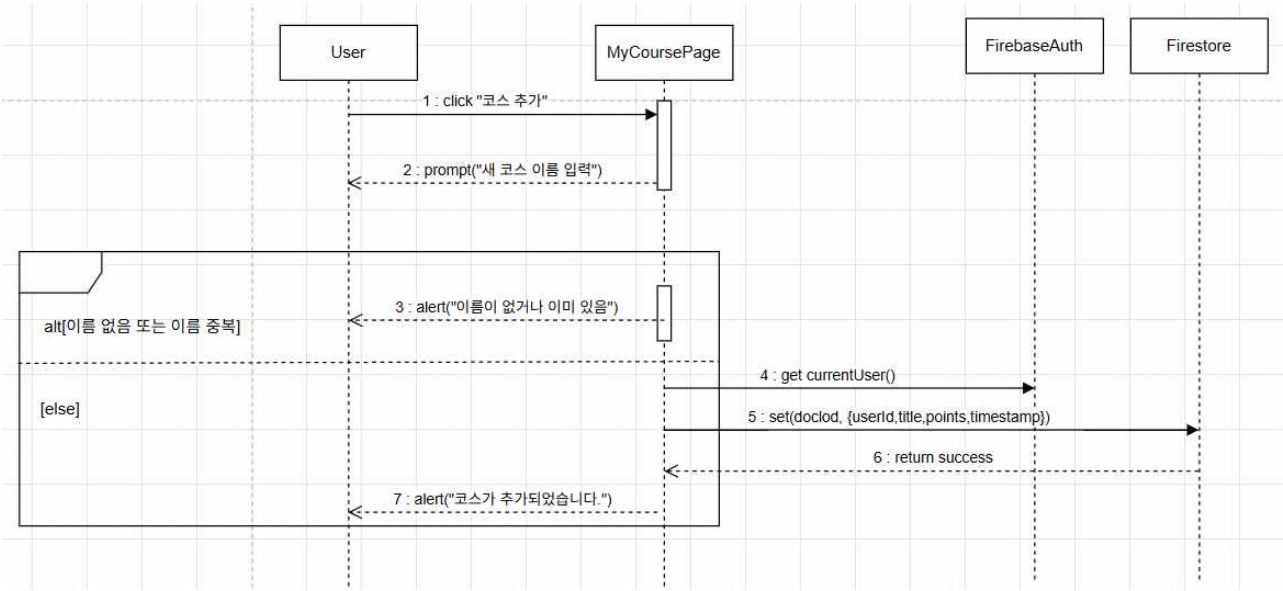

1) Register

이 시퀀스 다이어그램은 사용자가 회원가입을 시도할 때의 흐름을 나타낸다. User, Register, FirebaseAuth 세 객체가 상호작용하며, 입력값 검증과 Firebase 인증 처리를 중심으로 구성된다. 사용자가 이메일과 비밀번호를 입력하고 버튼을 누르면 register(event) 함수가 실행되며, 조선 분기를 통해 입력값이 비어있더나 비밀번호가 6자리 미만일 경우 각각 경고 메시지가 출력된다. 입력이 정상이 ㄹ 경우 Firebase의 createUserWithEmailAndPassword 함수가 호출되고, 회원가입이 성공하면 “회원가입 성공” 메시지가 사용자에게 표시된다.




2) login

다이어그램에는 User, LoginPage, FirebaseAuth 세 객체가 등장하며, 로그인 버튼 클릭부터 성공 또는 실패 여부 처리까지의 흐름이 나타나 있다. 사용자가 이메일과 비밀번호를 입력하고 로그인 버튼을 누르면 login(event) 함수가 호출되며, 입력값에 대한 유효성 검사가 먼저 수행된다. 이 과정은 alt 조건 분기 프레임을 통해 나뉘며, 이메일이나 비밀번호가 비어 있을 경우 "이메일과 비밀번호를 모두 입력해주세요"라는 알림이 표시되고, 비밀번호가 6자리 미만인 경우에는 "비밀번호는 6자리 이상이어야 합니다"라는 경고창이 출력된다. 이 두 조건을 모두 통과한 경우에만 Firebase의 signInWithEmailAndPassword() 함수가 호출되며, 로그인 요청이 전송된다. Firebase에서 사용자 정보가 성공적으로 반환되면 "로그인 성공"이라는 메시지가 출력되고, 메인 페이지로 이동한다. 반면 로그인 실패 시에는 "로그인 실패: " + err.message 형식의 경고창이 사용자에게 표시된다.

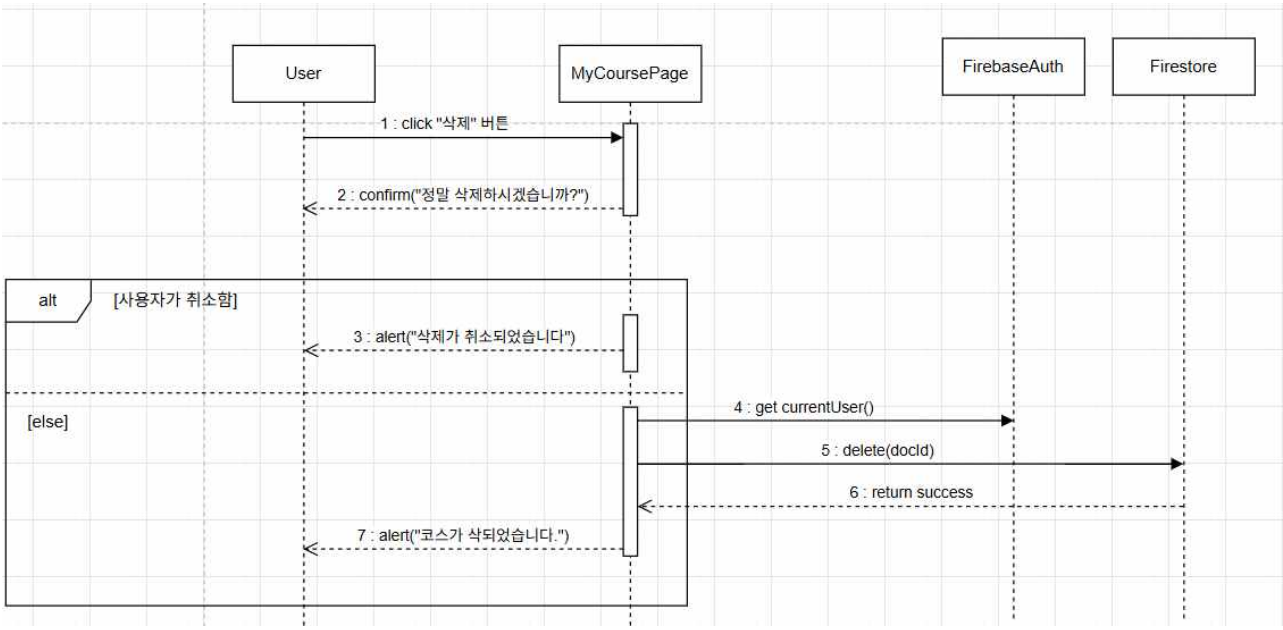



3) 코스 추가

이 시퀀스 다이어그램은 Tripiary 애플리케이션에서 사용자가 새로운 여행 코스를 추가할 때의 과정을 시각적으로 나타낸 것이다. 다이어그램에는 User, MyCoursePage, FirebaseAuth, Firestore 네 객체가 등장하며, 사용자 입력부터 데이터베이스 저장까지의 흐름이 순차적으로 구성되어 있다.

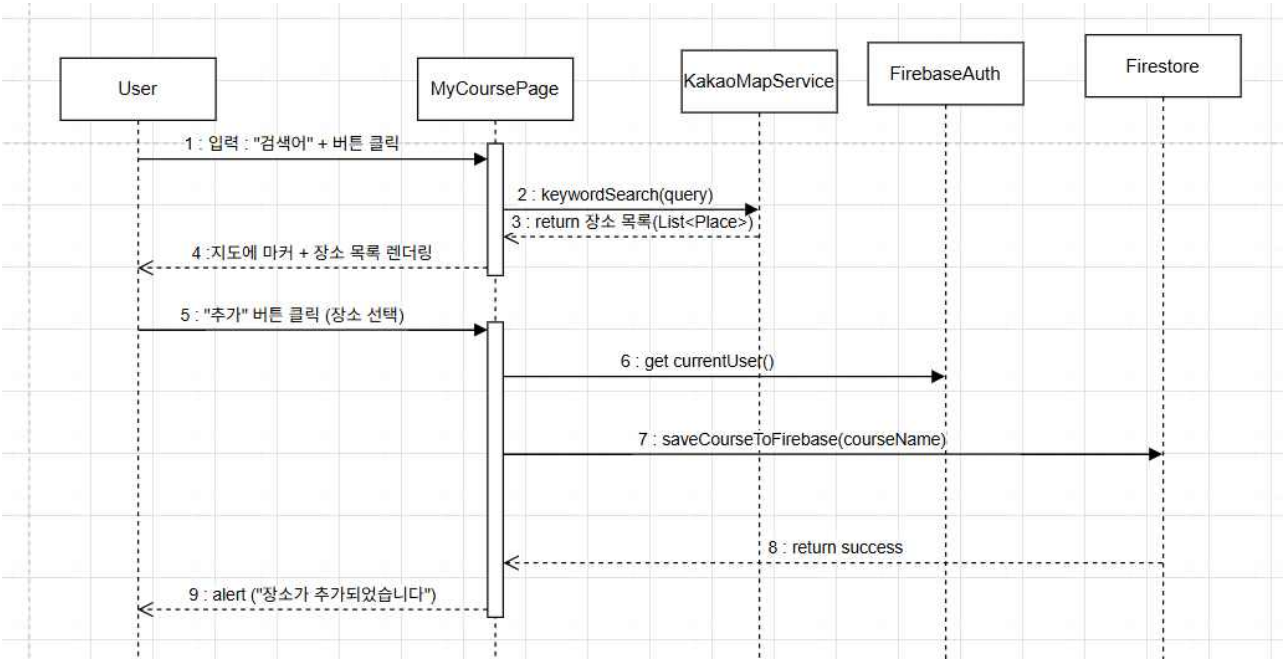
사용자는 "코스 추가" 버튼을 클릭함으로써 다이어그램이 시작되며, MyCoursePage에서는 prompt() 함수를 통해 새로운 코스명을 입력받는다. 이 입력값은 조건 분기(alt)를 통해 먼저 검증되며, 입력이 비어 있거나 이미 존재하는 이름일 경우 "이름이 없거나 이미 있음"이라는 알림창이 표시되고 흐름이 종료된다. 유효한 코스명일 경우, 현재 로그인된 사용자의 정보를 FirebaseAuth를 통해 가져오고, saveCourseToFirebase() 함수가 실행되어 코스 데이터를 Firestore에 저장하게 된다.

이때 저장되는 내용에는 사용자 ID, 코스명, 장소 리스트, 그리고 타임스탬프가 포함되며, Firestore로부터 저장 성공 응답이 돌아오면 사용자에게 "코스가 추가되었습니다"라는 메시지가 출력된다.



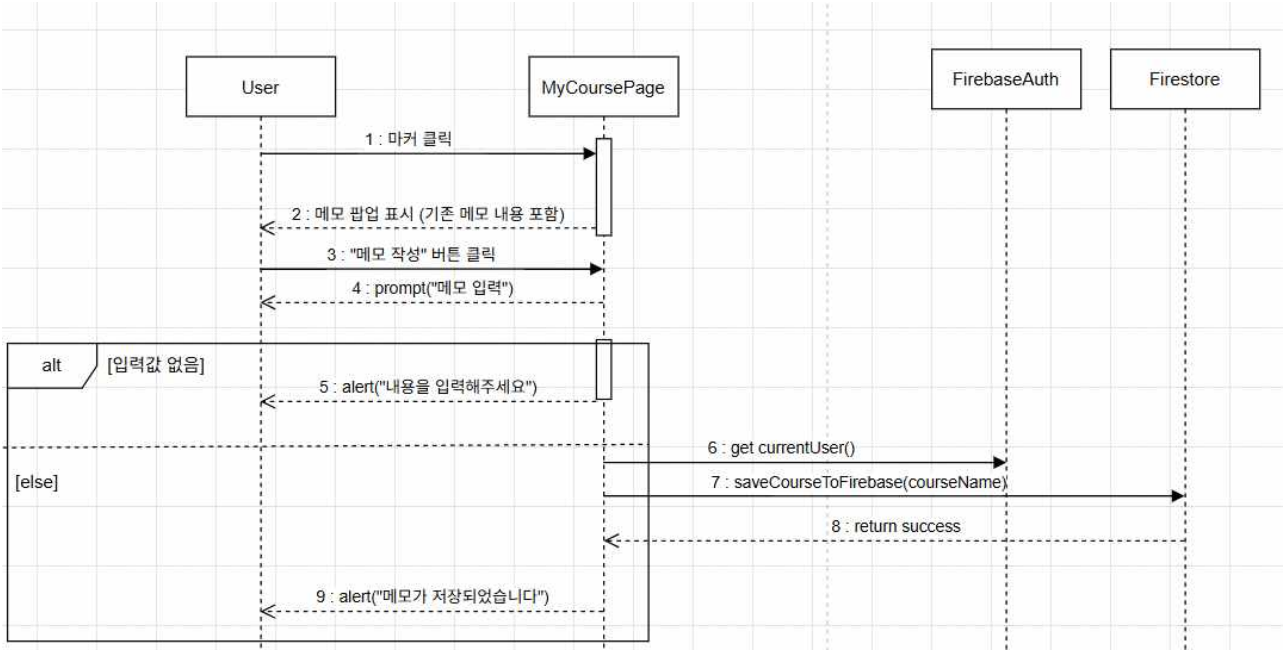
4) 코스 삭제

이 시퀀스 다이어그램은 사용자가 특정 코스를 삭제하는 과정을 나타낸다. 사용자가 "삭제" 버튼을 클릭하면 MyCoursePage에서 삭제 여부를 확인하는 confirm() 창이 표시되며, 사용자가 이를 취소하면 "삭제가 취소되었습니다"라는 알림이 출력되고 흐름이 종료된다. 반면 삭제를 확인한 경우에는 현재 로그인된 사용자 정보를 FirebaseAuth를 통해 확인하고, 해당 사용자의 코스 문서를 Firestore에서 삭제한다. Firestore에서 삭제가 성공적으로 처리되면 "코스가 삭제되었습니다"라는 알림이 사용자에게 표시된다. 조건 분기(alt)를 통해 사용자 취소 여부에 따라 흐름이 갈라지고, Firebase 연동을 통해 실제 데이터 삭제가 수행되는 구조로, 사용자 상호작용과 데이터베이스 반영 흐름을 명확히 표현한 시퀀스이다.



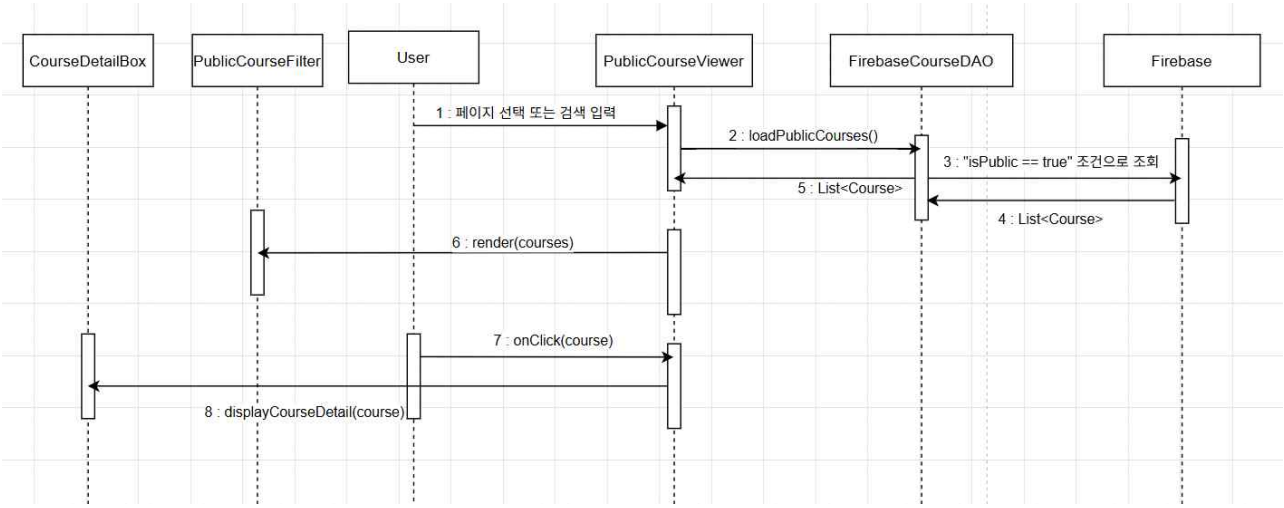

5) 장소 검색 및 추가

이 시퀀스 다이어그램은 사용자가 특정 코스에 장소를 검색하고 추가하는 과정을 나타낸다. 사용자가 검색어를 입력하고 버튼을 클릭하면 MyCoursePage는 카카오맵 API(KakaoMapService)에 keywordSearch()를 요청하고, 장소 목록을 응답받아 지도에 마커를 표시한다. 사용자가 특정 장소의 "추가" 버튼을 누르면, 현재 로그인된 사용자 정보를 FirebaseAuth에서 받아오고, 선택된 장소가 포함된 코스 데이터를 Firestore에 저장한다. 저장이 성공하면 사용자에게 "장소가 추가되었습니다"라는 메시지가 출력된다. 이 흐름은 사용자 입력, 외부 API 호출, 인증 확인, 데이터베이스 저장까지의 구조적 과정을 보여준다.




6) 메모 작성

이 시퀀스 다이어그램은 사용자가 특정 장소에 메모를 작성하고 저장하는 과정을 나타낸다. 사용자가 지도 위의 마커를 클릭하면 해당 장소의 정보와 기존 메모 내용을 포함한 팝업이 표시되며, 이후 "메모 작성" 버튼을 누르면 prompt()를 통해 새로운 메모 내용을 입력받는다. 입력값이 비어 있을 경우에는 "내용을 입력해주세요"라는 경고 메시지가 출력되며 흐름이 종료되고, 유효한 입력인 경우에는 현재 로그인된 사용자 정보를 FirebaseAuth에서 확인한 뒤, 수정된 코스 데이터를 Firestore에 저장한다. 저장 성공 시 "메모가 저장되었습니다"라는 알림이 사용자에게 표시되며, 전체 흐름은 사용자 입력 → 유효성 검사 → Firebase 저장 → 사용자 피드백의 구조로 구성되어 있다.



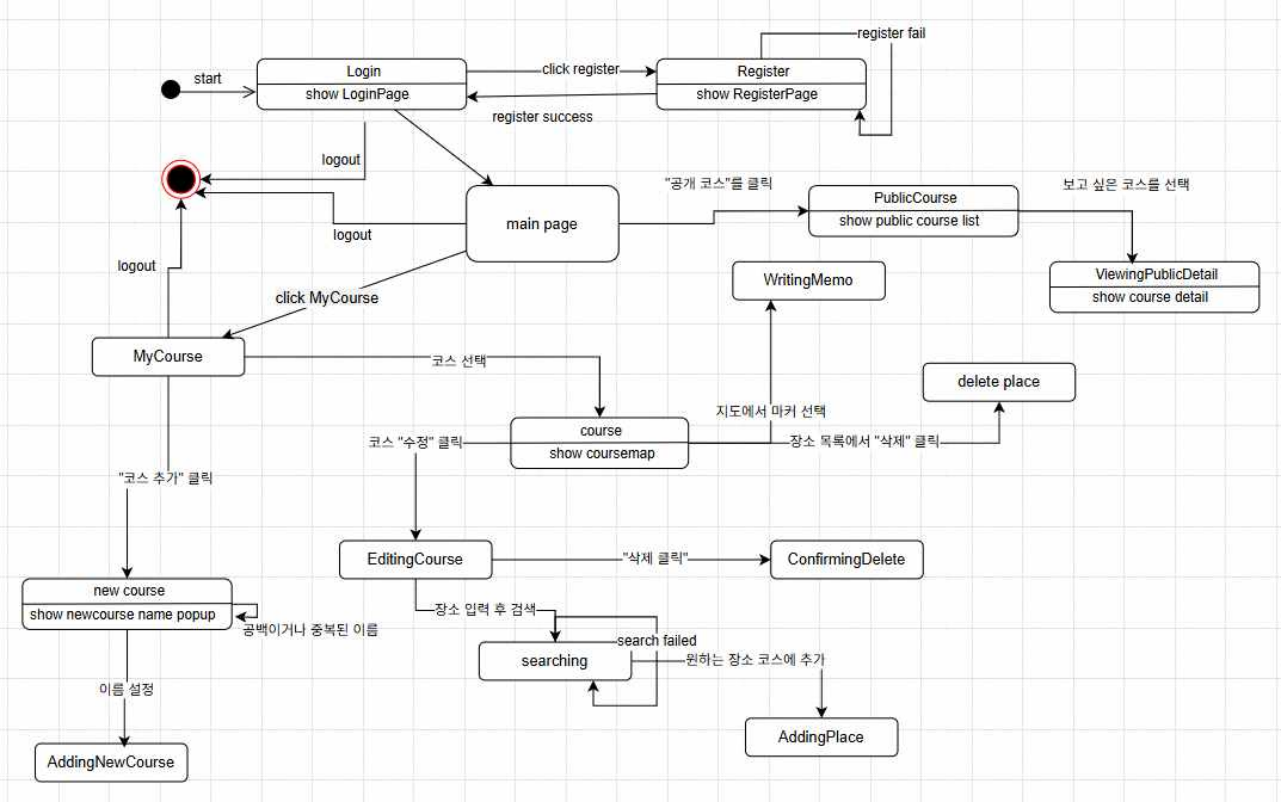

7) 공개 코스

이 시퀀스 다이어그램은 사용자가 Tripiary에서 공개 코스를 열람하는 과정을 나타낸다. 사용자가 공개 코스 보기 페이지를 선택하거나 접근하면, PublicCourseViewer는 내부적으로 FirebaseCourseDAO를 통해 Firestore에서 “isPublic ==true” 조건을 만족하는 코스들만 조회한다. 이때 실제 데이터 요청은 Firebase에 전달되며, 조건에 부합하는 공개 코스 리스트가 반환된다.

조회된 코스 리스트는 PublicCourseViewer에서 render() 함수를 통해 화면에 카드 형태로 표시된다. 이후 사용자가 특정 코스를 클릭하면, onClick() 이벤트가 발생하고, 해당 코스의 정보가 CourseDetailBox에 전달되 세부 정보가 화면에 뜨게된다.

전체 흐름은 사용자 입력 -> 공개 코스 불러오기 -> 화면 렌더링 -> 사용자 클릭 -> 상세 보기로 이어지며, 데이터 로딩과 인터페이스 전환이 유기적으로 연결된 구조로 구성되어있다.

4. State machine diagram




사용자가 Tripiary 서비스를 이용하면서 겪게 되는 전체 시스템의 상태 전이 흐름을 나타낸다.
서비스를 이용하기 위해 고객은 먼저 Login 상태를 거쳐야 하며, 아직 회원이 아닐 경우 Register 상태로 진입하여 회원가입 절차를 진행하게 된다. 회원가입에 성공하면 다시 Login 상태로 되돌아가며, 로그인까지 성공하게 되면 main page 상태로 진입하게 된다.
메인 화면에서는 사용자의 선택에 따라 두 가지 방향으로 흐름이 나뉜다.
첫 번째는 MyCourse 상태로의 진입니다. 사용자가 '나의 코스'를 선택하면 현재 가지고 있는 코스를 조회하거나 새로운 코스를 생성할 수 있다. "코스 추가" 버튼을 클릭하면 new course 상태로 전이되며, 이때 사용자에게 코스 이름 입력을 요청한다. 이름이 비어있거나 중복되는 경우에는 다시 입력하도록 안내되며, 유효한 이름이 입력되면 AddingNewCourse 상태로 전환되어 신규 코스가 저장된다.
기존 코스를 선택하여 편집을 진행할 경우에는 EditingCourse 상태로 진입하게 되며, 이 상태에서는 장소 추가, 메모 작성, 삭제 기능 등을 수행할 수 있다. 장소를 추가하려면 먼저 검색어를 입력하여 searching 상태로 이동하고, 검색 결과가 없으면 search failed 상태로 분기된다. 검색이 성공하면 AddingPlace 상태로 전이되어 사용자가 선택한 장소가 현재 코스에 반영된다. 또한, 마커를 클릭하면 WritingMemo 상태로 진입해 메모를 입력할 수 있고, 장소 목록에서 삭제를 선택할 경우에는 ConfirmingDelete 상태를 거쳐 deleteplace 상태로 이어진다. 이 모든 흐름은 실제 Tripiary의 지도 기반 인ㅌ페이스와 밀접하게 연결되어 있다.
두 번째 흐름은 공개 코스 기능으로, 사용자가 메인 페이지에서 '공개 코스' 메뉴를 선택하면 PublicCourse 상태로 전이한다. 이 상태에서는 다른 사용자들의 공유한 공개 코스 목록이 리스트 형태로 표시되며, 사용자를 이를 열람할 수 있다. 특정 코스를 클릭하면 ViewingPublicDetail 상태로 이동하여 해당 코스의 전체 장소 구성 및 설명을 확인할 수 있다. 공개 코스는 읽기 전용 상태로 제공되며,

사용자 간 코스 아이디어를 공유하고 참고하는 기능을 중심으로 구성된다. 이 흐름은 기존의 나만의 코스를 관리하는 흐름과는 독립적으로 동작하며, Tripiary의 정보 공유 기능을 구현하는 핵심 구성 요소이다.
이 외에도 사용자는 어느 상태에서든 logout 동작을 통해 시스템을 종료할 수 있다. Logout이 실행되면 시스템은 로그아웃 상태로 전이되며, 다시 로그인 또는 회원가입 과정ㅇ을 통해 서비스를 재이용할 수 있다.
전체적으로 Tripiary의 상태 흐름은 사용자 인증부터 시작하여 코스 생성, 편집, 장소 검색 및 추가, 메모 작성, 삭제, 공개 코스 열람, 그리고 로그아웃에 이르기까지의 기능 흐름을 state machine diagram으로 구조화하였다. 이는 사용자 경험 기반으로 각 기능이 어떻게 유기적으로 연결되어 있는지를 시스템 수준에서 시각적으로 설명해주며, 실제 구현 흐름과도 밀접하게 대응된다.

## 5. Implementation requirements

S/W 요구사항
(1) 최신 웹 표준을 지원하는 웹 브라우저 (Chrome, Firefox, Edge 등 ES6+ 지원 브라우저)
(2) HTML5, CSS3, JavaScript (ES6) 실행 환경
(3) Firebase SDK (Authentication, Firestore 기능 포함)
(4) Kakao Maps JavaScript API 사용 가능 환경

H/W 요구사항
(1) Windows 10 이상, macOS, 또는 GUI 기반 Linux 환경의 데스크탑 또는 노트북
(2) 최소 4GB 이상의 RAM을 가진 일반 사무용 PC 또는 노트북
(3) 모바일 테스트 환경용 Android 8.0 이상 스마트폰
(4) 인터넷 연결이 가능한 디바이스 (Firebase 및 Kakao API 통신 필요)

## 6. Glossary

| term | description |
|---|---|
| api | 소프트웨어 애플리케이션 간의 상호작용을 위한 일련의 규칙 |
| firebase | 앱 개발 플랫폼 |
| sequence | 연속된 하나의 장면 설정 |

## 7. References

https://brownbears.tistory.com/511
https://sabarada.tistory.com/192