Online Pizza Ordering System

**<SDD – The 2nd Checkpoint>**

소프트웨어공학 02분반

20171105 이민욱

20173875 정용준

20170223 신원준

20172609 여일구

20174438 정종민

20173156 김준기

목차

[Online Pizza Ordering System 1](#_Toc71728231)

[**<SDD – The 2nd Checkpoint>** 1](#_Toc71728232)

[**1.** **SubgroupA(PizzaOrder)** 4](#_Toc71728233)

[1.1 **A-UC-1: Add Menu To Cart** 4](#_Toc71728234)

[1.1.1 Sequence Diagram 4](#_Toc71728235)

[1.1.2 최종 결과 및 선정 이유 5](#_Toc71728236)

[1.2 **A-UC-4: Remove Cart Item** 6](#_Toc71728237)

[1.2.1 Basic Sequence Diagram 6](#_Toc71728238)

[1.2.2 **Variation** 7](#_Toc71728239)

[1.3 **A-UC-5: Order** 7](#_Toc71728240)

[1.3.1 Sequence Diagram 8](#_Toc71728241)

[**2.** **SubgroupB(Management)** 10](#_Toc71728242)

[2.1 **B-UC-1: AuthenticateUser** 10](#_Toc71728243)

[2.1.1 **최종 결과 및 선정 이유** 13](#_Toc71728244)

[2.2 **B-UC-3: AddPizza** 14](#_Toc71728245)

[2.2.1 **최종 결과 및 선정 이유** 14](#_Toc71728246)

[2.3 **B-UC-9: DeleteUser** 15](#_Toc71728247)

[2.3.1 **최종 결과 및 선정 이유** 19](#_Toc71728248)

[2.4 **DisplayByPeriod** 20](#_Toc71728249)

[2.4.1 **최종 결과 및 선정 이유** 21](#_Toc71728250)

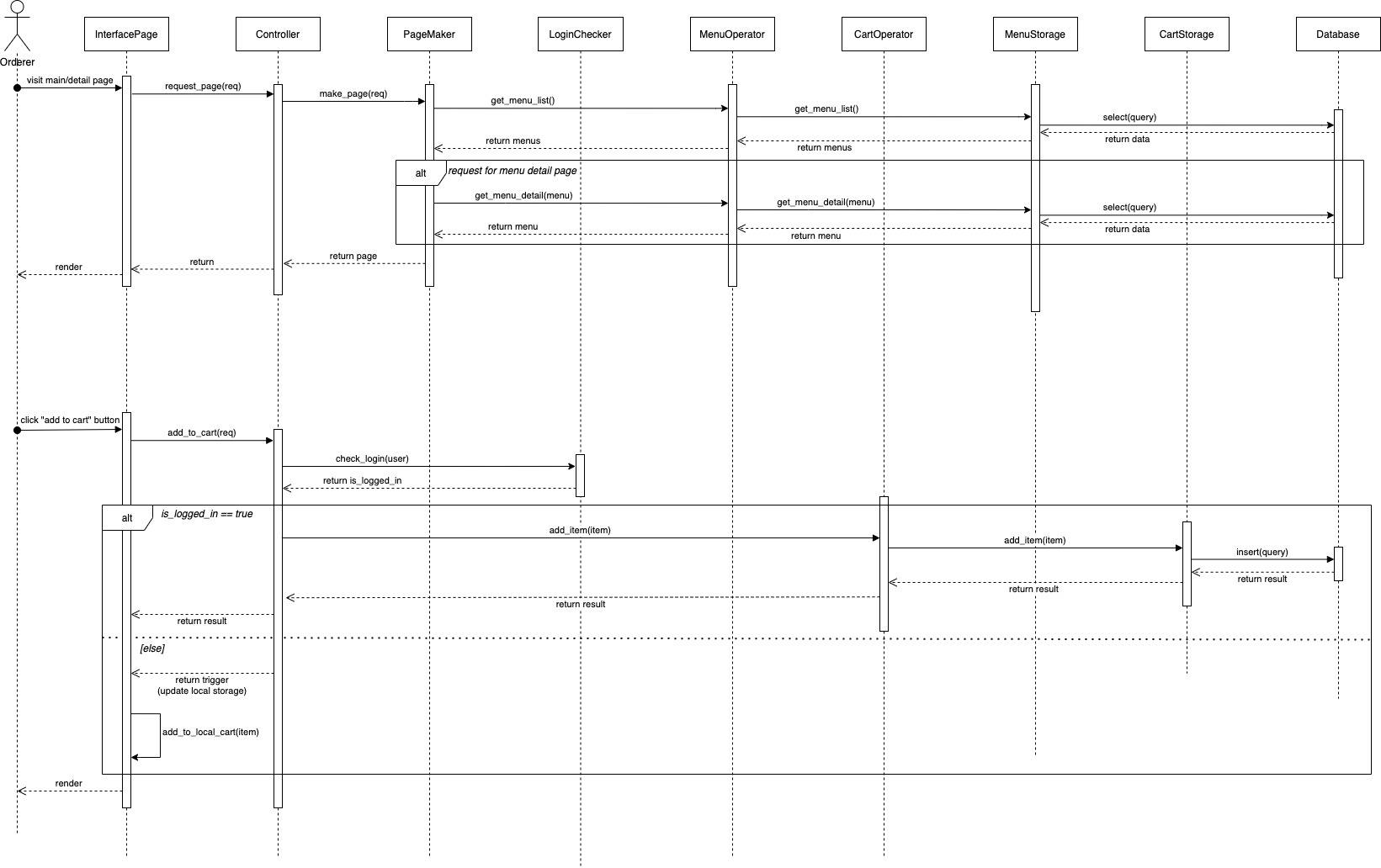
[**3.** **SubgroupC(OrderStatus)** 21](#_Toc71728251)

[3.1 **C-UC-1 blah blah** 21](#_Toc71728252)

# **SubgroupA(PizzaOrder)**

* 1. **A-UC-1: Add Menu To Cart**
     1. Sequence Diagram

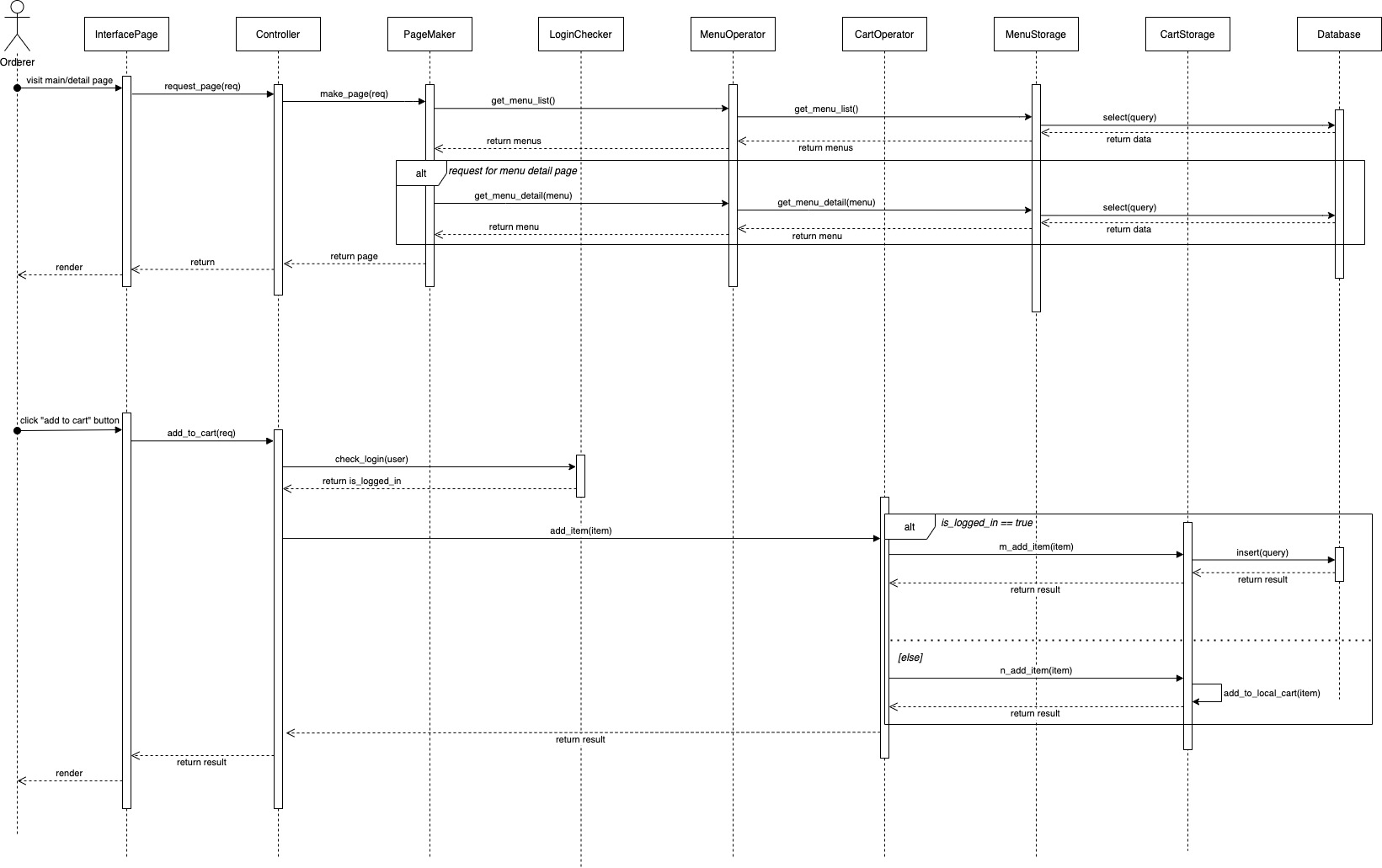
1. Basic



1. Variation

Basic과의 차이는 장바구니 데이터를 다룰 때에서 발생한다.

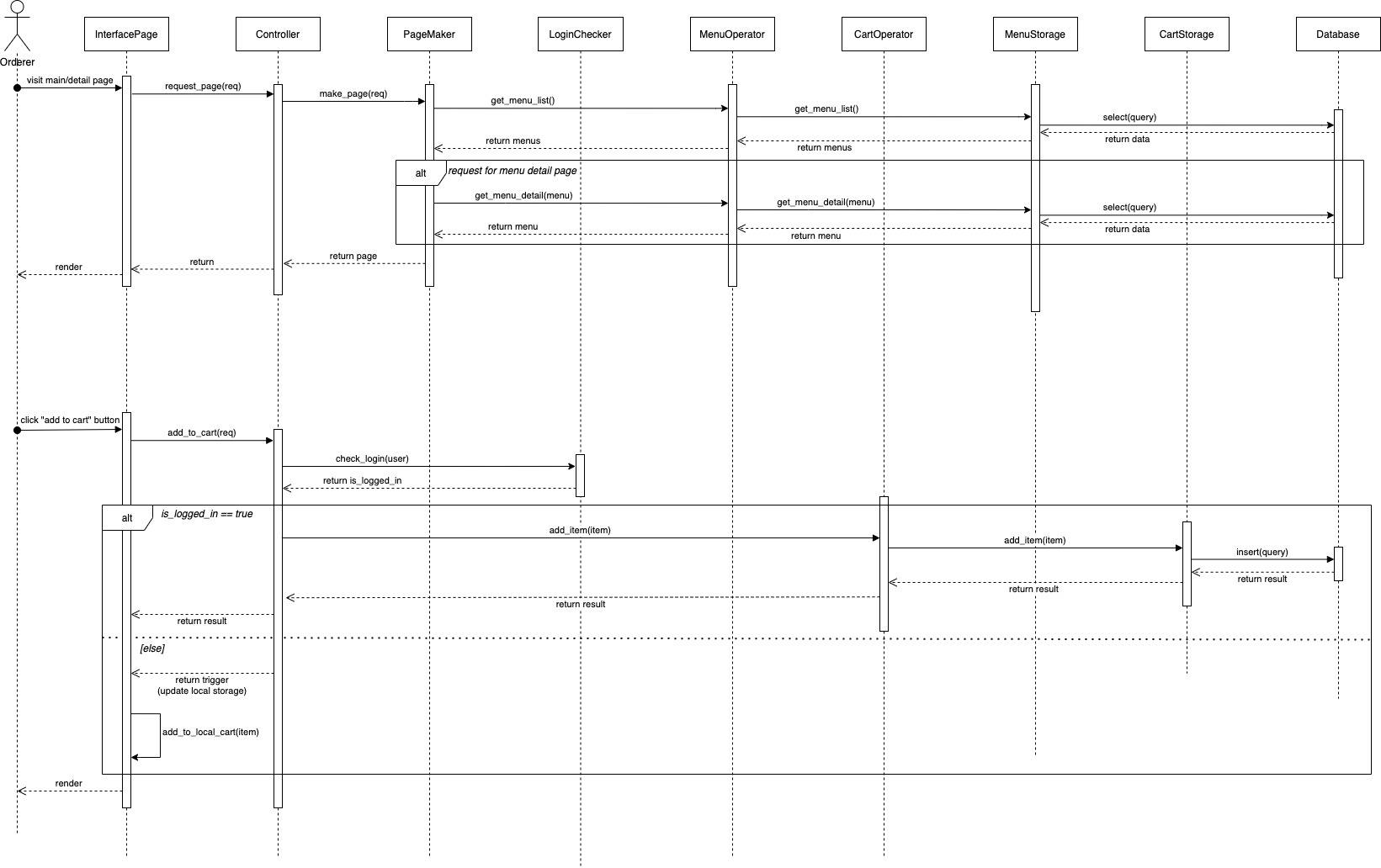
장바구니 데이터의 저장 위치는 유저가 로그인 상태일 때는 DB, 비로그인 상태일 때는 로컬 저장소이다. DB/로컬저장소가 Storage 클래스로 추상화된 형태인 Variation이 새롭게 제안되었다.



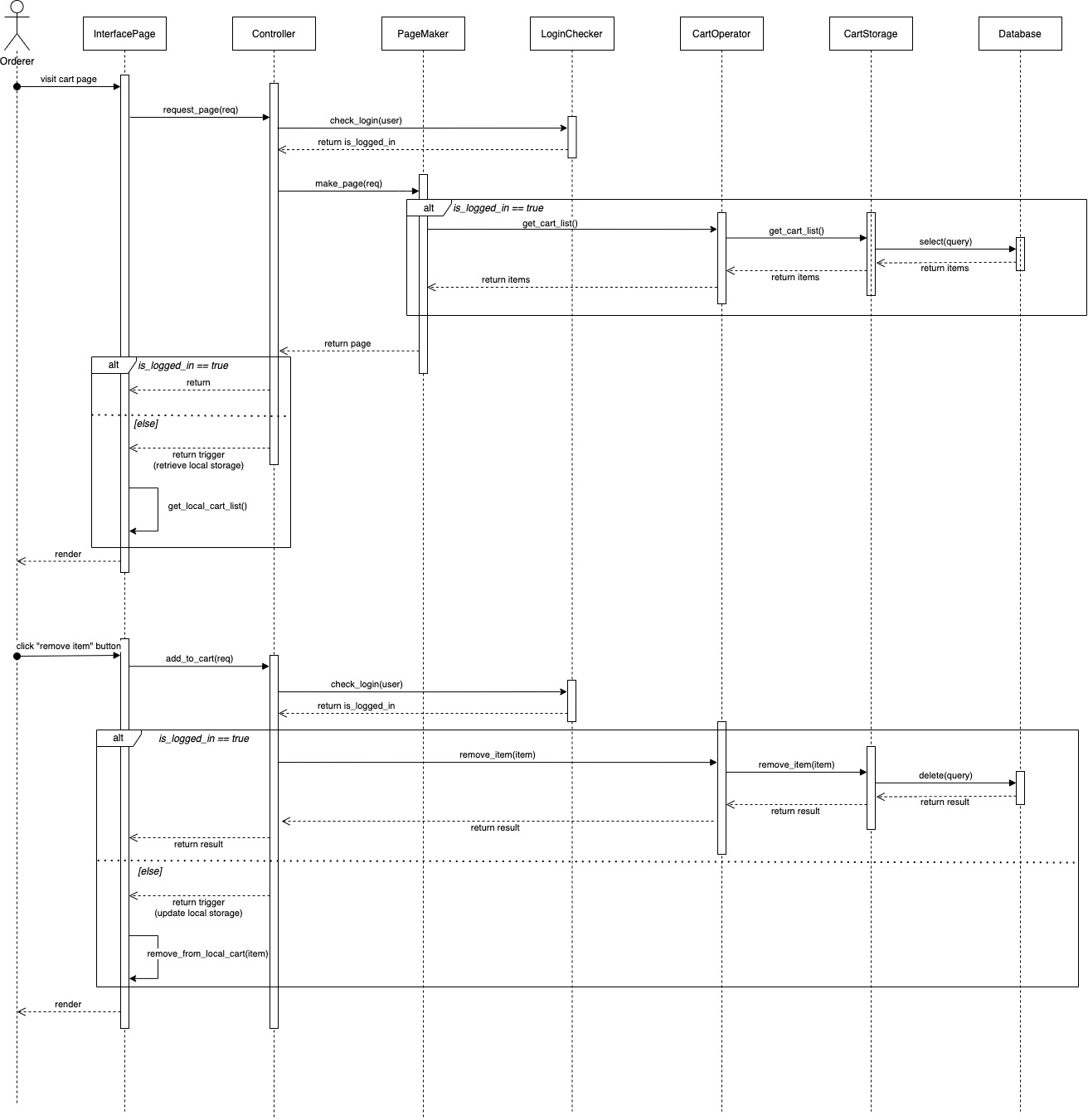
* + 1. 최종 결과 및 선정 이유

1. Object Sequence Diagram

실제 웹 환경에서 로컬저장소는 브라우저에서 JavaScript API로 다뤄야하기 때문에 Storage 클래스는 브라우저보다는 DB와 훨씬 가까운 위치이므로 로컬저장소 처리에 부적절하다는 판단이 들었다. 따라서 InterfacePage에서 로컬 저장소 동작을 처리하는 방식인 **Basic Sequence Diagram**으로 결정하였다. InterfacePage에 책임이 가중된다는 의견도 있으나, 로컬저장소를 관리하는 이슈는 현재 장바구니 데이터 밖에 없으므로 큰 부담이 아니라서 그대로 진행하기로 하였다.



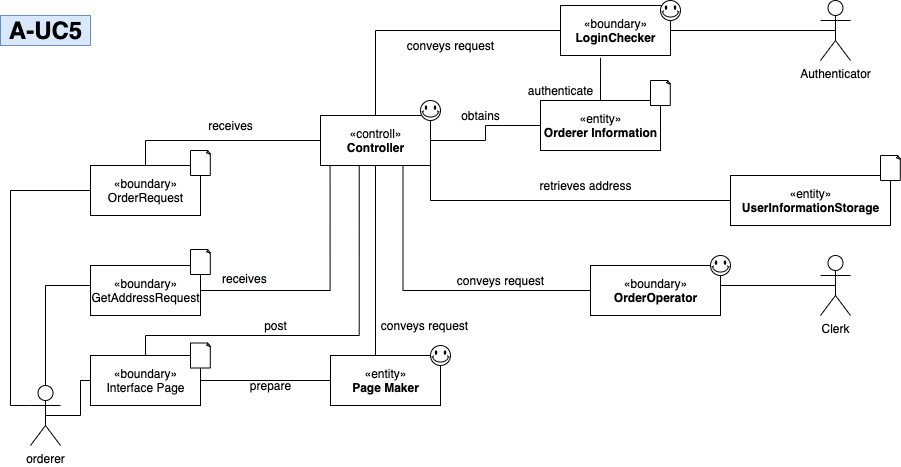
1. Class Diagram
2. Traceability Matrix
   1. **A-UC-4: Remove Cart Item**
      1. Basic Sequence Diagram



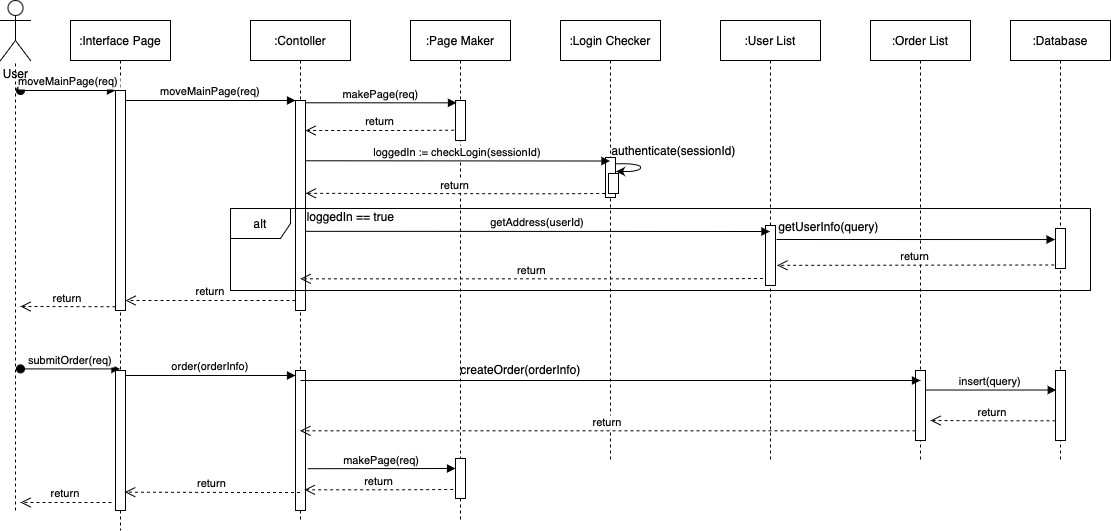
* + 1. **Variation**

A-UC-1과 구조상 큰 차이가 없어서 같은 Variation이 발생하므로 생략한다.

* 1. **A-UC-5: Order**



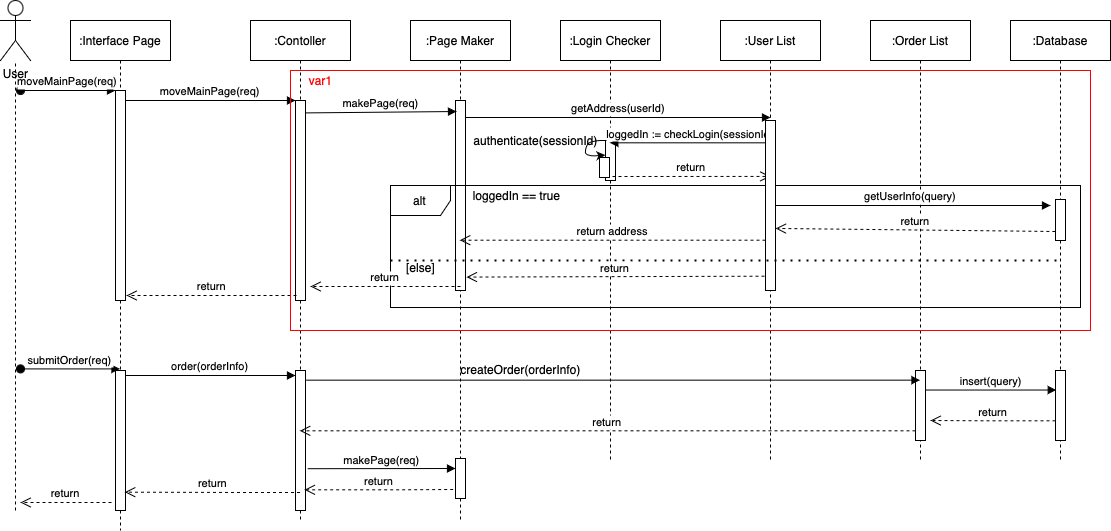
* + 1. Sequence Diagram



1. Variation 1

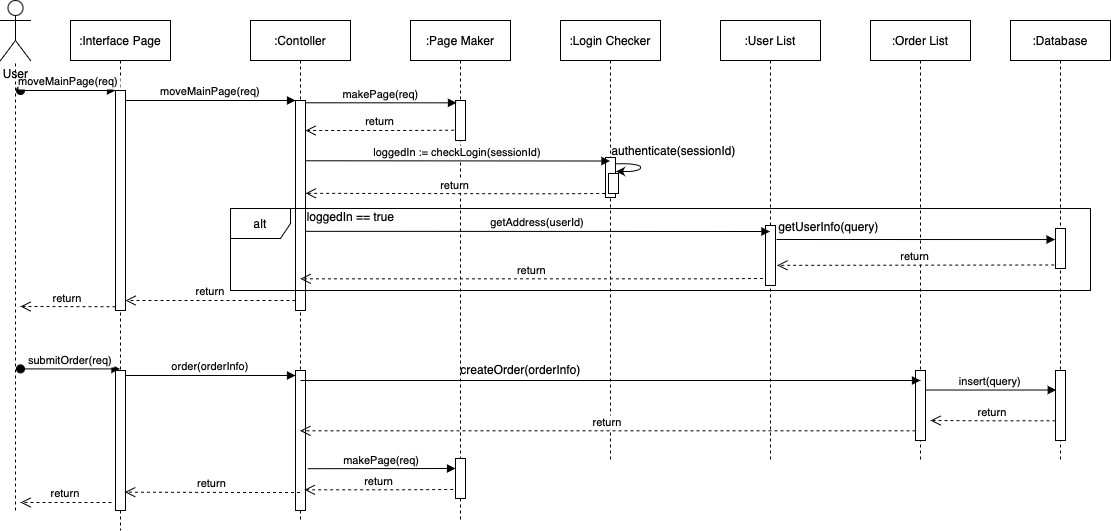
초기에 controller에서 페이지를 호출할 때 로그인 여부에 대한 데이터를 가지고 어디에서 주소 호출을 결정하느냐에 따라 variation이 생길 수 있다. Basic에서는 controller에서 로그인 여부에 대한 데이터를 받아보고 주소를 가져온다. 반면 variation-1에서는 controller가 makePage 함수를 호출하고 그 결과를 그대로 리턴하면 된다.

Responsibility 측면에서 봤을 때 controller의 경우 역할 분배가 잘 된 것으로 볼 수 있다. 하지만 Page Maker가 다른 데이터를 호출하고, Database와의 인터페이스 역할을 하는 User List에서 로그인 여부를 판단하기 위해 check login을 호출하는 등 오히려 responsibility 상에서 문제가 생기는 것을 볼 수 있다.

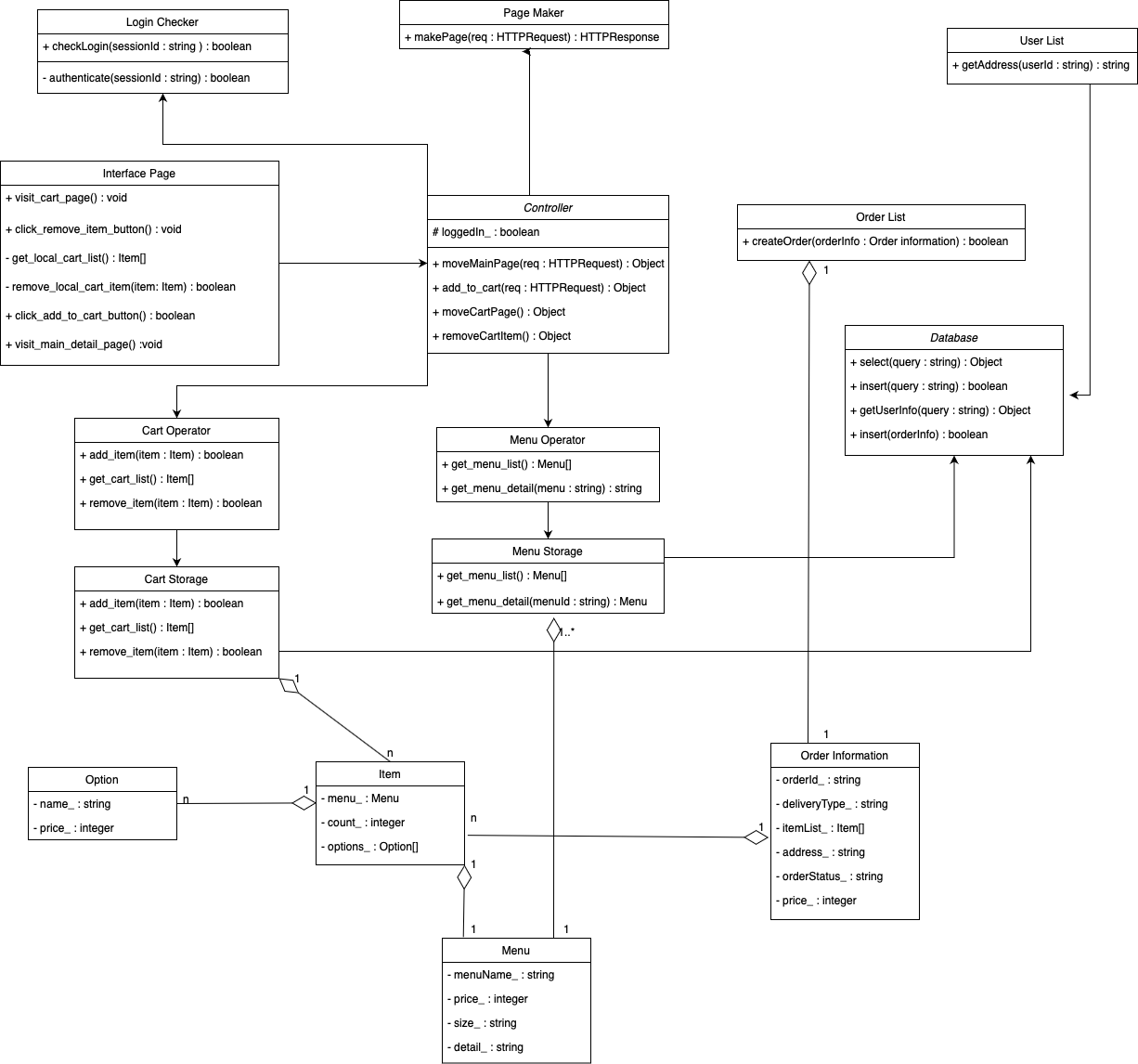


* + 1. 최종 결과 및 선정 이유
* 앞서 설명한 바와 같이 variation-1에는 responsibility 상 문제가 있다. Basic의 경우 controller가 조금 더 많은 역할을 가지고 있지만 받아본 데이터를 이용해 요청들을 각각 잘 분배해 다른 class들의 responsibility가 올바르게 작용하게끔 하고있으며 메서드의 호출 순서가 간단하고 명확하게 표기되어 있는 것으로 보여 Basic을 선택했다.

1. Object Sequence Diagram



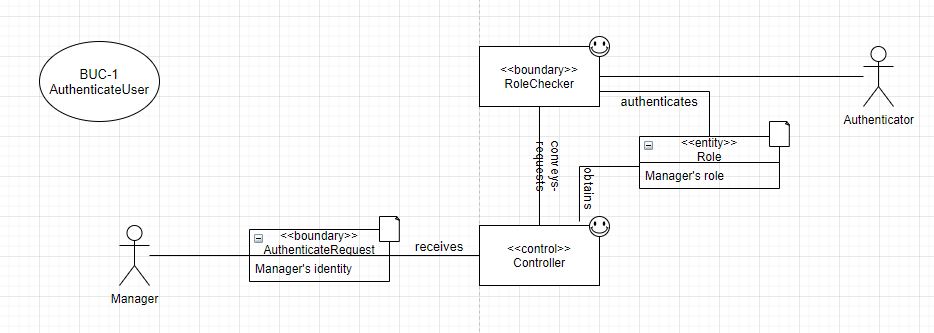
* 1. Class Diagram, Traceability



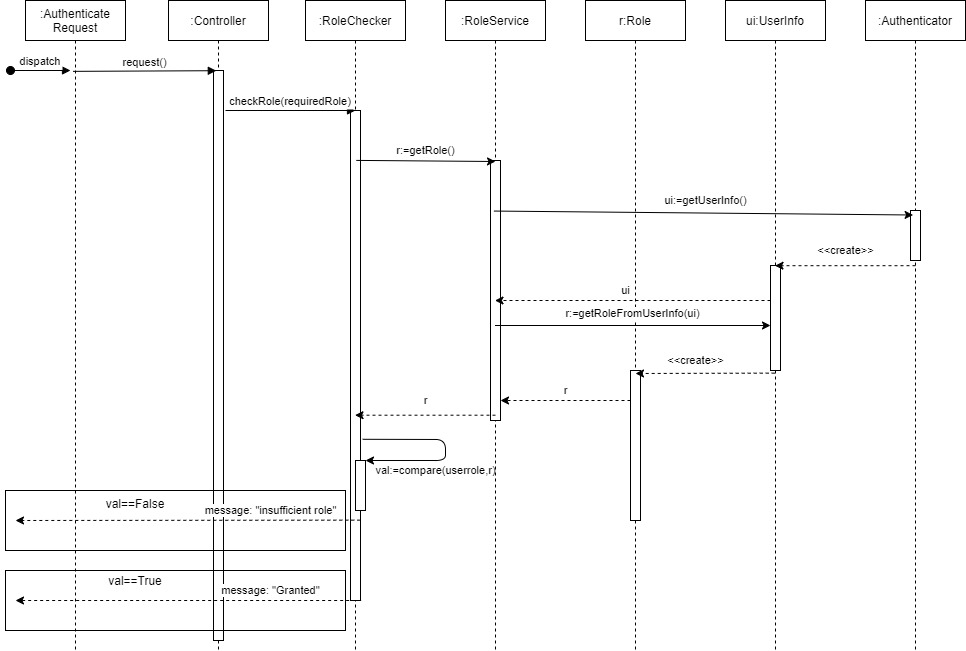
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Domain concepts | Software classes | | | | | | | | | | | | |
| Controller | Page Maker | Interface Page | Login Checker | Menu Operator | Menu Storage | Cart Operator | Cart Storage | User List | Order List | Database | Menu | Item | | Option | Order Information |
| Controller | X |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |
| PageMaker |  | X |  |  |  |  |  |  |  |  |  |  |  | |  |  |
| Interface Page |  |  | X |  |  |  |  |  |  |  |  |  |  | |  |  |
| Login Checker |  |  |  | X |  |  |  |  |  |  |  |  |  | |  |  |
| Page Request |  |  | X |  |  |  |  |  |  |  |  |  |  | |  |  |
| Add to Cart Request |  |  | X |  |  |  |  |  |  |  |  |  |  | |  |  |
| Menu Operator |  |  |  |  | X |  |  |  |  |  |  |  |  | |  |  |
| Menu Storage |  |  |  |  |  | X |  |  |  |  | X | X |  | |  |  |
| Cart Operator |  |  |  |  |  |  | X |  |  |  |  |  |  | |  |  |
| Cart Storage |  |  |  |  |  |  |  | X |  |  | X | X | X | | X |  |
| Remove Cart Item Request |  |  | X |  |  |  |  |  |  |  |  |  |  | |  |  |
| Get Address Request |  |  | X |  |  |  |  |  |  |  |  |  |  | |  |  |
| Order Request |  |  | X |  |  |  |  |  |  |  |  |  |  | |  |  |
| Order Information |  |  |  |  |  |  |  |  |  |  |  |  |  | |  | X |
| User Information Storage |  |  |  |  |  |  |  |  | X |  | X |  |  | |  |  |
| Order Operator |  |  |  |  |  |  |  |  |  | X |  | X | X | | X | X |

# **SubgroupB(Management)**

* 1. **B-UC-1: AuthenticateUser**



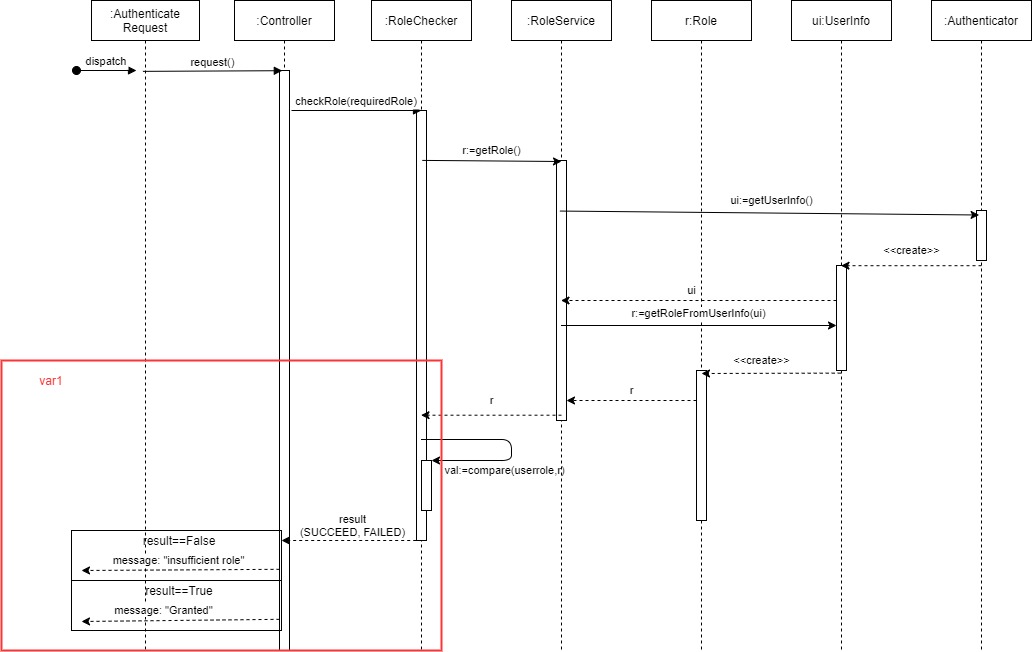
1. Basic



1. Variation1

AuthenticateUser UseCase에서 시스템에 해당 Role이 일치하는지 확인하고 이후 인증에 성공했는지 실패했는지 여부를 Signal해주는 주체에 따라 Variation이 생길 수 있다. Basic 모델에서는 RoleChecker가 Role을 비교한 후 결과 값을 바로 Signal 해준다. Variation 1에서는 Controller가 RoleChekcer에게 역할을 비교하라는 checkRole 메소드의 결과값(성공, 실패)를 받고 이를 토대로 Controller가 적절한 Signal을 반환한다.

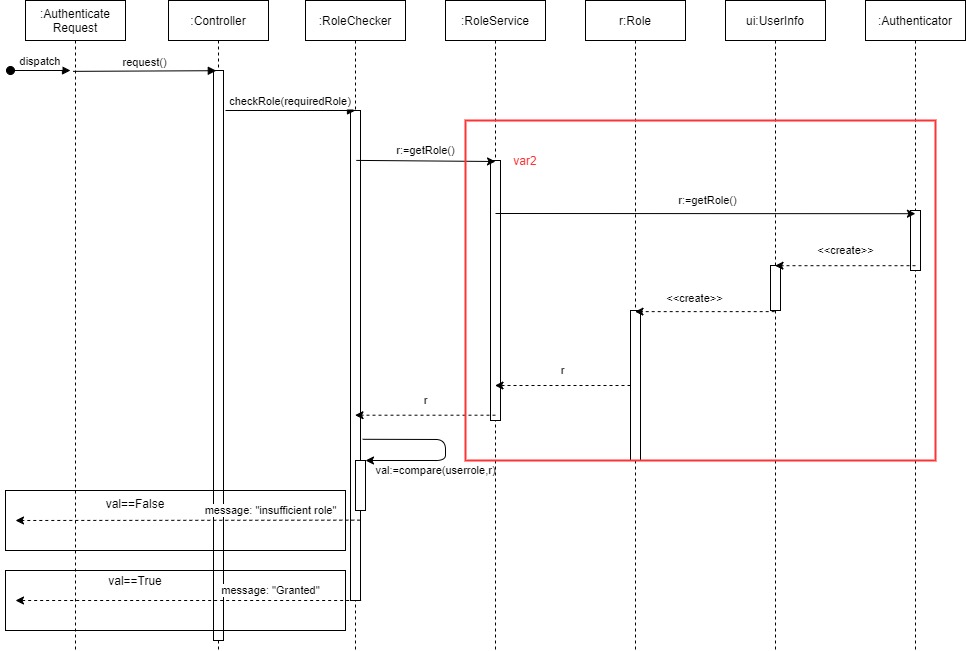
Responsibility 측면에서는 RoleChecker가 역할을 확인하는 역할을 하므로, 역할에 따라 결과 값을 바로 알려주도록 하는 방식이 적절하다. 그러나, Controller에서 checkRole(requiredRole)을 호출하였으므로 이에 대한 결과를 받아보아야 한다는 측면과, Controller가 checkRole에 대한 result를 받아서 단순히 결과에 따른 반환 값만 반환해준다는 것에서는 Variation1 방식이 적절하다.



1. Variation 2

AuthenticateUser UseCase에서 비교에 필요한 User의 Role을 받아오는 과정에서 Variation이 생길 수 있다. UserInfo는 Domain Model에는 없었지만, 개발 설계를 할 때, Role을 바로 가져오는 것이아닌, User id로 UserInfo를 가져오고, 그 안에 Composition된 Role을 꺼내오는 것이 자연스럽다고 생각하여 새로운 UserInfo object를 sequence diagram에 추가하였다. RoleService 또한, RoleCheck는 Role을 비교하는 것에 Responsibility를 집중하고 사용자정보로 Role을 주는 logic을 대신해주는 Responsibility를 가진 object가 필요하다고 생각하여 RoleService를 추가하였다.

Basic 모델에서는 RoleService에게 차례대로 먼저 UserInfo를 달라고 요청한 후, UserInfo와 Composition 관계인 Role을 getter를이용해 가져온다. 이 방식의 장점은 개발을 할 때 더 자연스러운 흐름이라는 것이다. Variation1은 RoleService가 getRole을 요청하면 UserInfo와 Role이 만들어지고 UserInfo안의 Role이 반환되는 형태이다. 여기서 장점은 다이어그램의 흐름이 더 간결해지고, RoleService는 Role을 요청하는 logic에만 집중할 수 있다는 것이다. 단점은, UserInfo에서부터 Role이 반환되는 흐름이 개발을 할 때 모호한 부분이 있다는 것이다.

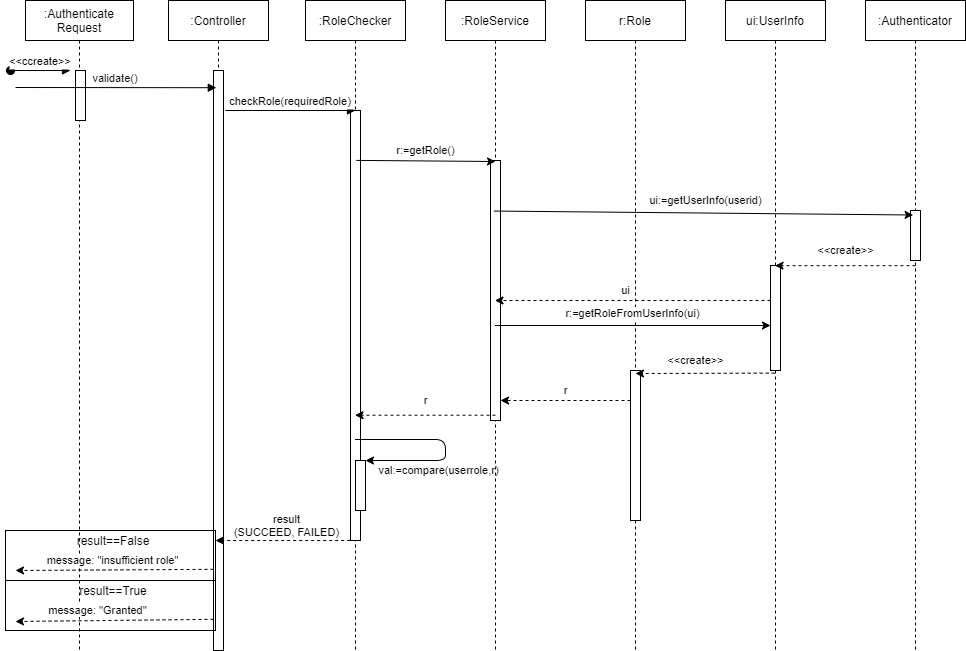


* + 1. **최종 결과 및 선정 이유**

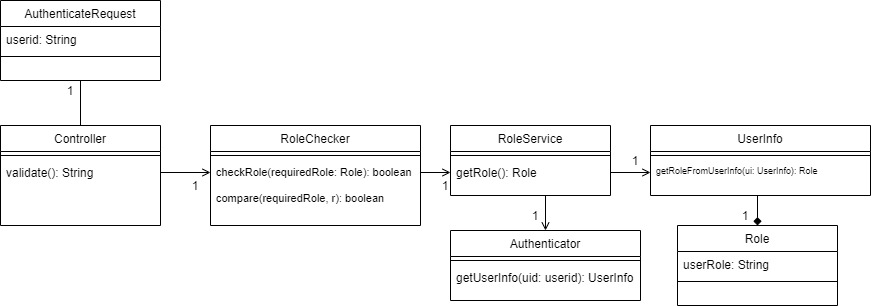
1. Object Sequence Diagram

AuthenticateUser UseCase에서 시스템에 해당 Role이 일치하는지 확인하고 이후 인증에 성공했는지 실패했는지 여부를 Signal해주는 주체에는 Controller가 하도록 선택하였다. 반환을 할 때, 어디로 반환을 할 지도 알아서 이를 돌려주어야 하는데 이런 것을 RoleChecker가 가지고 있는 것보다는 Controller가 가지고 있는 것이 적절하다고 생각하였다. 또한, 해당 다이어그램에서는 단순히 checkRole 이후에 결과를 메시지로 반환하는 형식이지만, 향후에 checkRole 결과를 Logging하거나 Mail로 notify하는 등의 추가 작업이 필요할 때 이 것들을 전부 RoleChecker에게 Responsibility를 부여할 수는 없다고 생각하였다.

AuthenticateUser UseCase에서 비교에 필요한 User의 Role을 받아오는 과정에서는 UserInfo를 먼저 받아온 후, UserInfo의 getter를 이용해 Role을 가져오는 것을 선택하였다. Getter는 단순히 composition된 role을 가져오는 것 뿐이기 때문에 RoleService의 Responsibility를 해치치 않는다고 생각하였고, 다른 variation보다 개발에 있어 용이하기 때문에 선택하였다.



1. Class Diagram



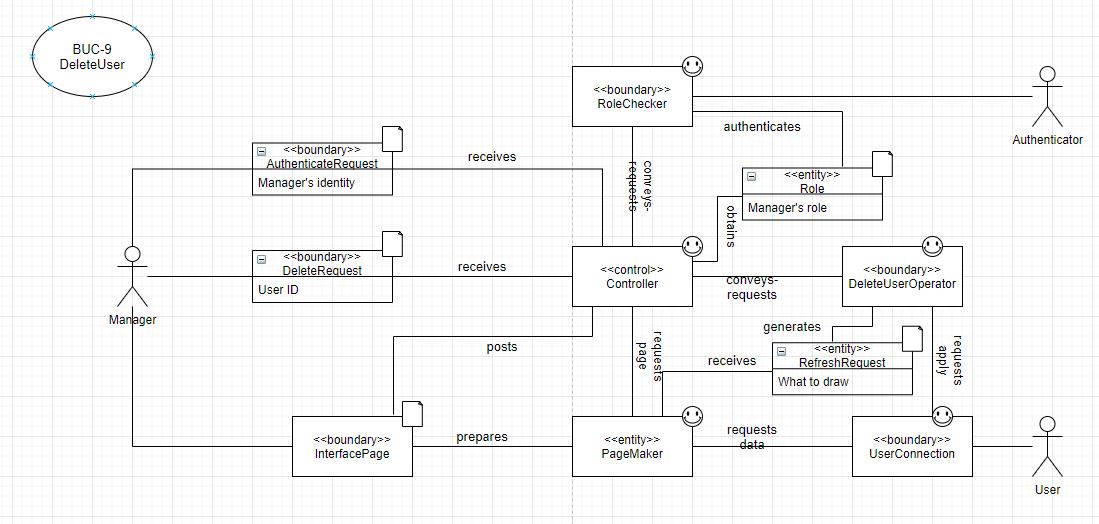
1. Traceability Matrix

테이블이(가) 표시된 사진

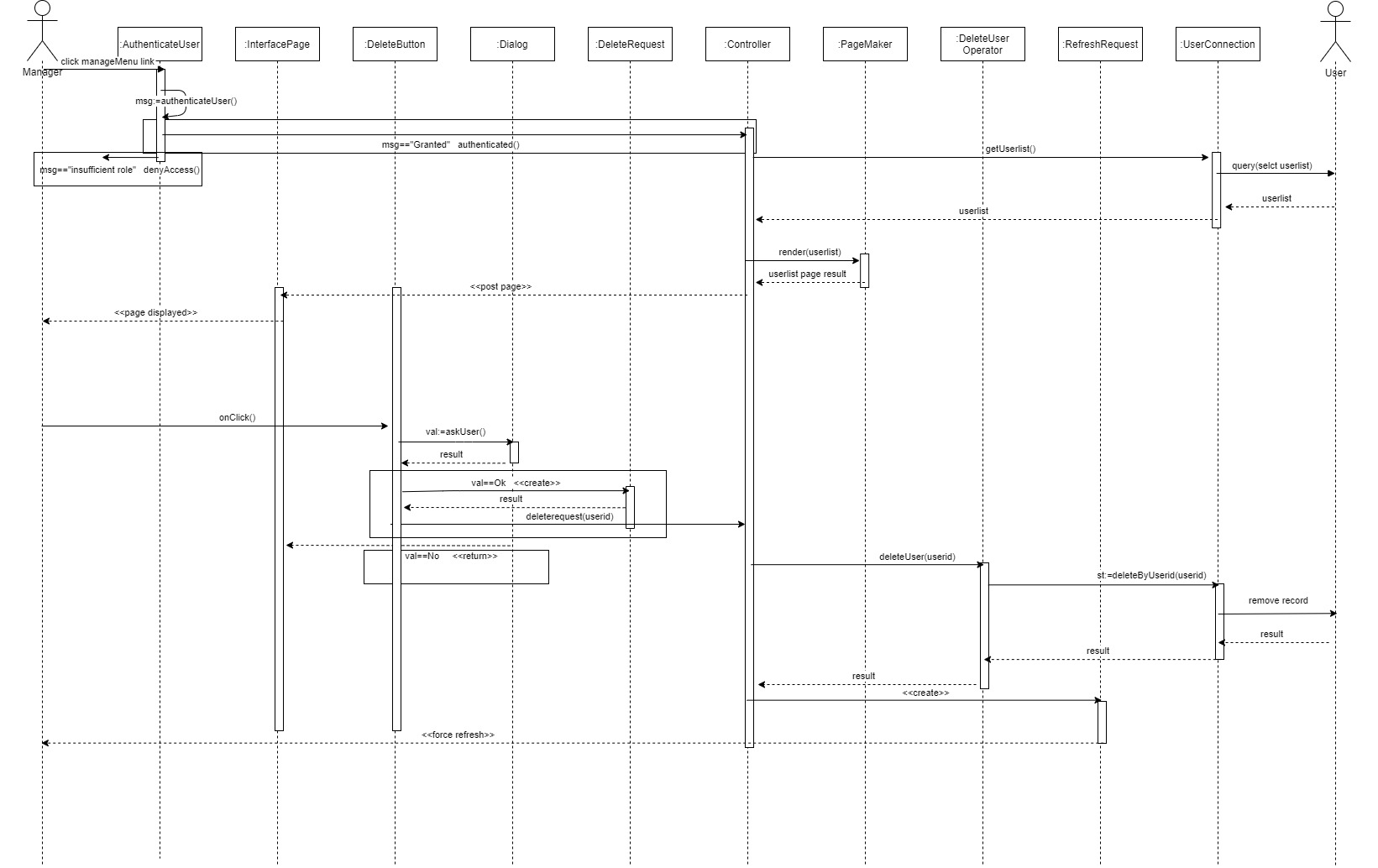
자동 생성된 설명

* 1. **B-UC-3: AddPizza**

1. Basic
2. Variation1
3. Variation 2
4. Variation 3
   * 1. **최종 결과 및 선정 이유**
5. Object Sequence Diagram
6. Class Diagram
7. Traceability Matrix
   1. **B-UC-9: DeleteUser**

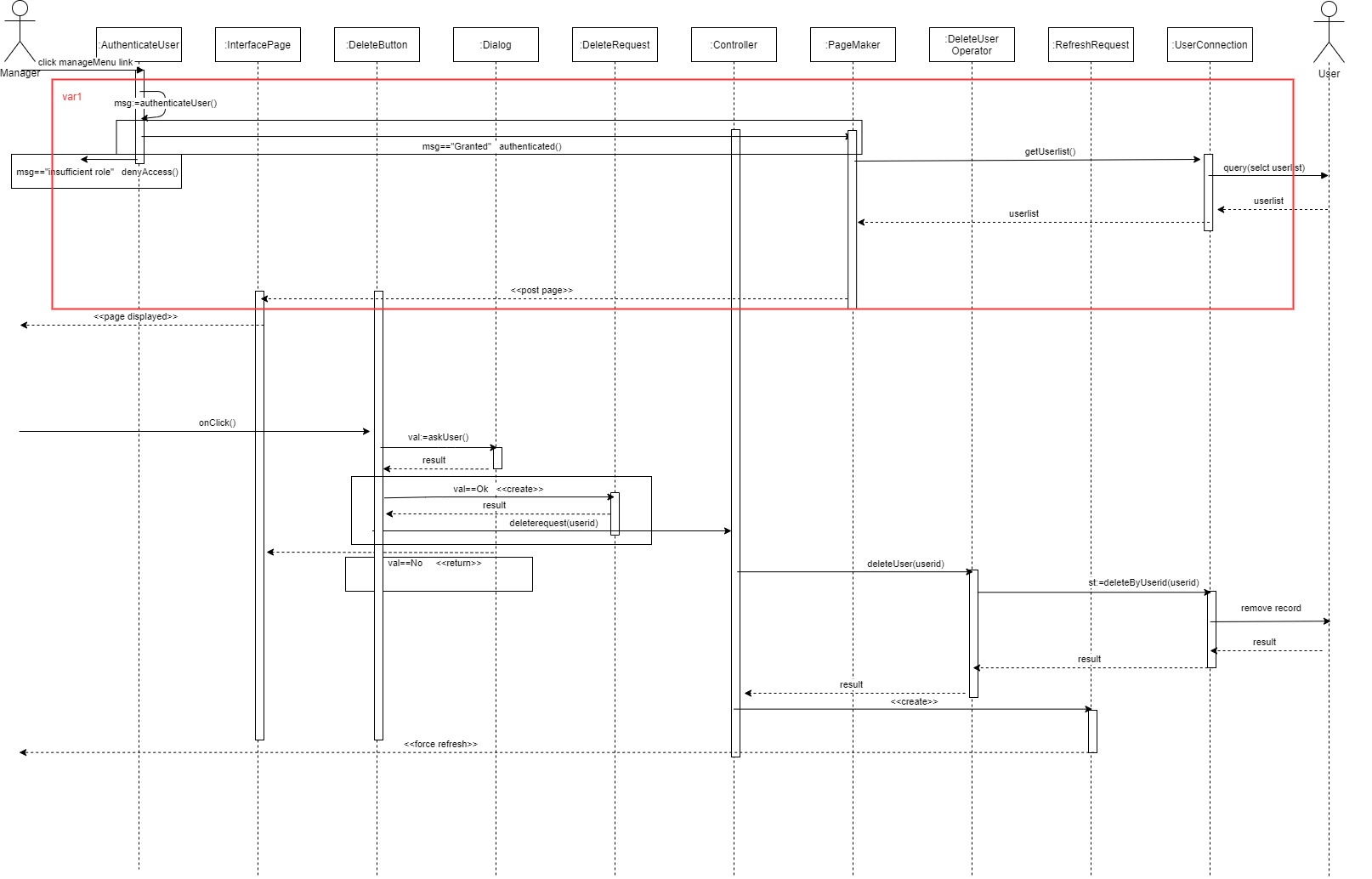


1. Basic



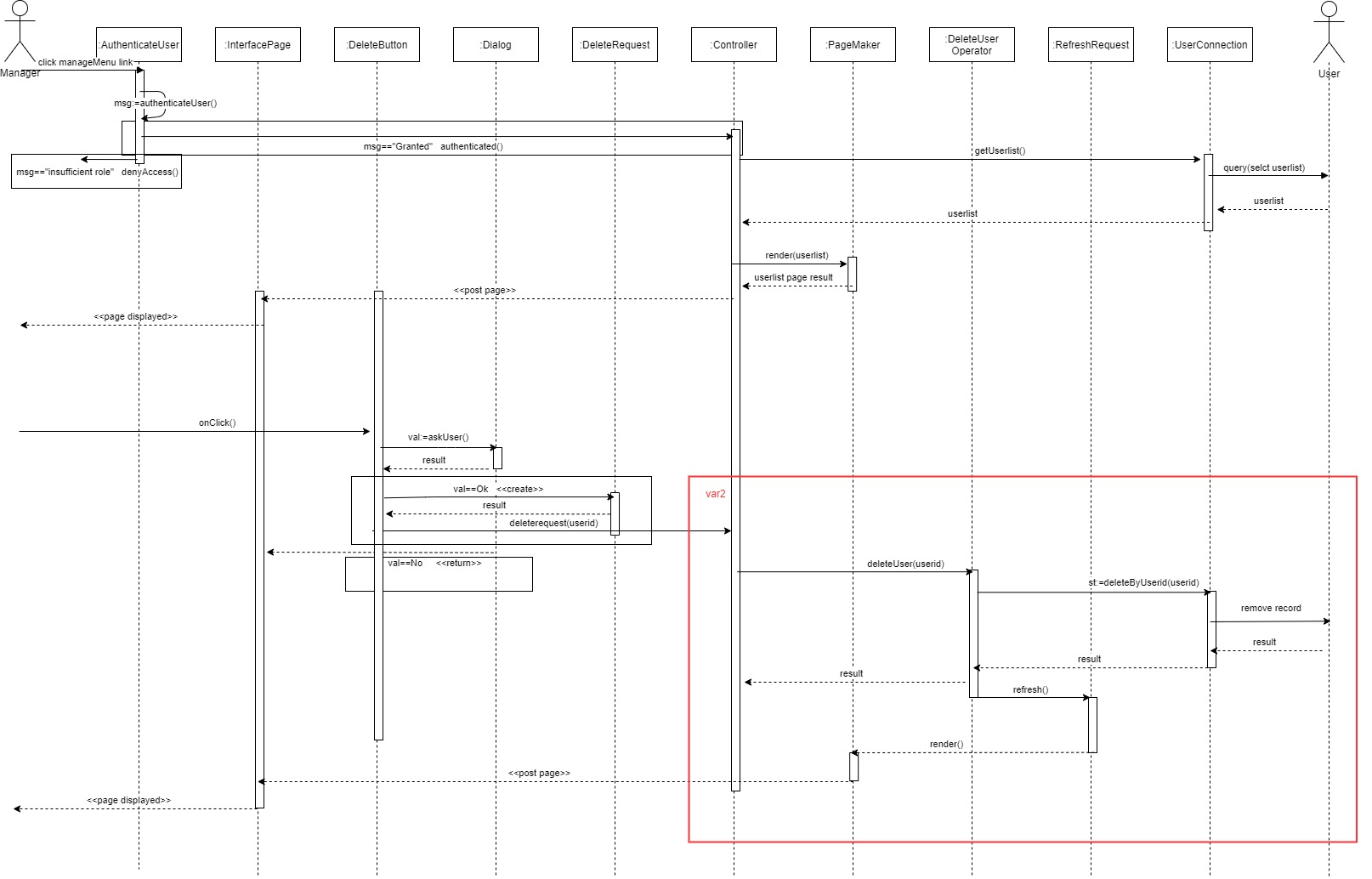
1. Variation1

Basic Model에서는 AuthenticateUser의 인증되었다는 authenticated를 Controller가 받고, UserConnection에게 Userlist를 반환받아서 이를 인자로 PageMaker에게 render하라고 요청한다. variation에서는 인증되었다는 authenticated를 PageMaker가 받고, PageMaker가 userlist를 받아와서 render를 한다. Basic Model의 장점은 처음 화면을 렌더링할 때, Controller가 요청을 받고 이에 대한 InterfacePage를 반환하는 Responsibility를 가질 수 있다는 점이다. 그러나 Page를 만드는 관점에서 생각해보면 Variation에서 Userlist를 받아오는 것은 Pagemaking을 할 때 필요한 것이므로 PageMaker가 요청해서 받아오는 것이 Responsibility를 가질 수 있다.



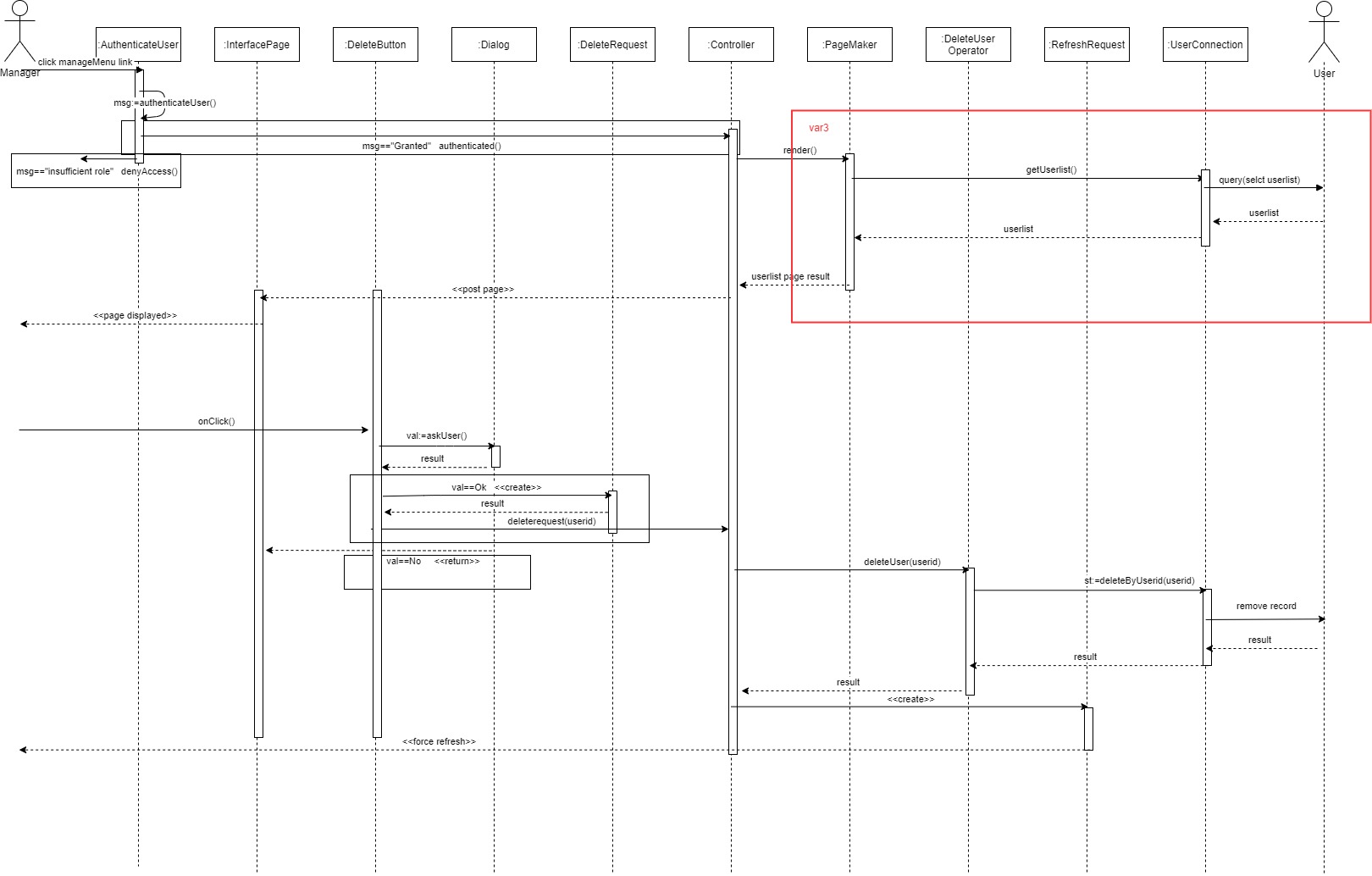
1. Variation 2

DeleteUser Operation을 수행한 후의 작업을 수행하는 것에 있어 어느 object가 responsibility를 가져야하는지에 대한 variation이 있다. Basic 모델에서는 deleteuser에 대한 성공 반환값만을 받아서, 성공 혹은 실패했든 작업이 완료되면 Controller가 RefreshRequest를 생성하여 사용자에게 강제로 새로고침을 하도록한다. Variation에서는 새로고침을해서 화면을 다시 보여주는 것 까지를 해당 diagram에서 해줘야하는 책임으로 보았다. Basic 모델은 보다 직관적이고 간결하다는 장점이 있고, Variaton에서는 Delete User를 하고난 이후 상황까지 Controller와 PageMaker사이의 상호작용으로 페이지를 만들게하는 책임을 부여했다는 장점이 있다.



1. Variation 3

Variation 1과 비슷한 variation이지만, Controller는 render하라는 명령만 내리고, pagemaking에 필요한 컴포넌트들(userlist)를 가져오는 부분은 PageMaker에게 responsibility를 위임한다. Basic Model에서는 PageMaking에 필요한 요소를 Controller가 요청하고 이 요소를 이용해 PageMaking을 해달라고 PageMaker에게 요청한다.

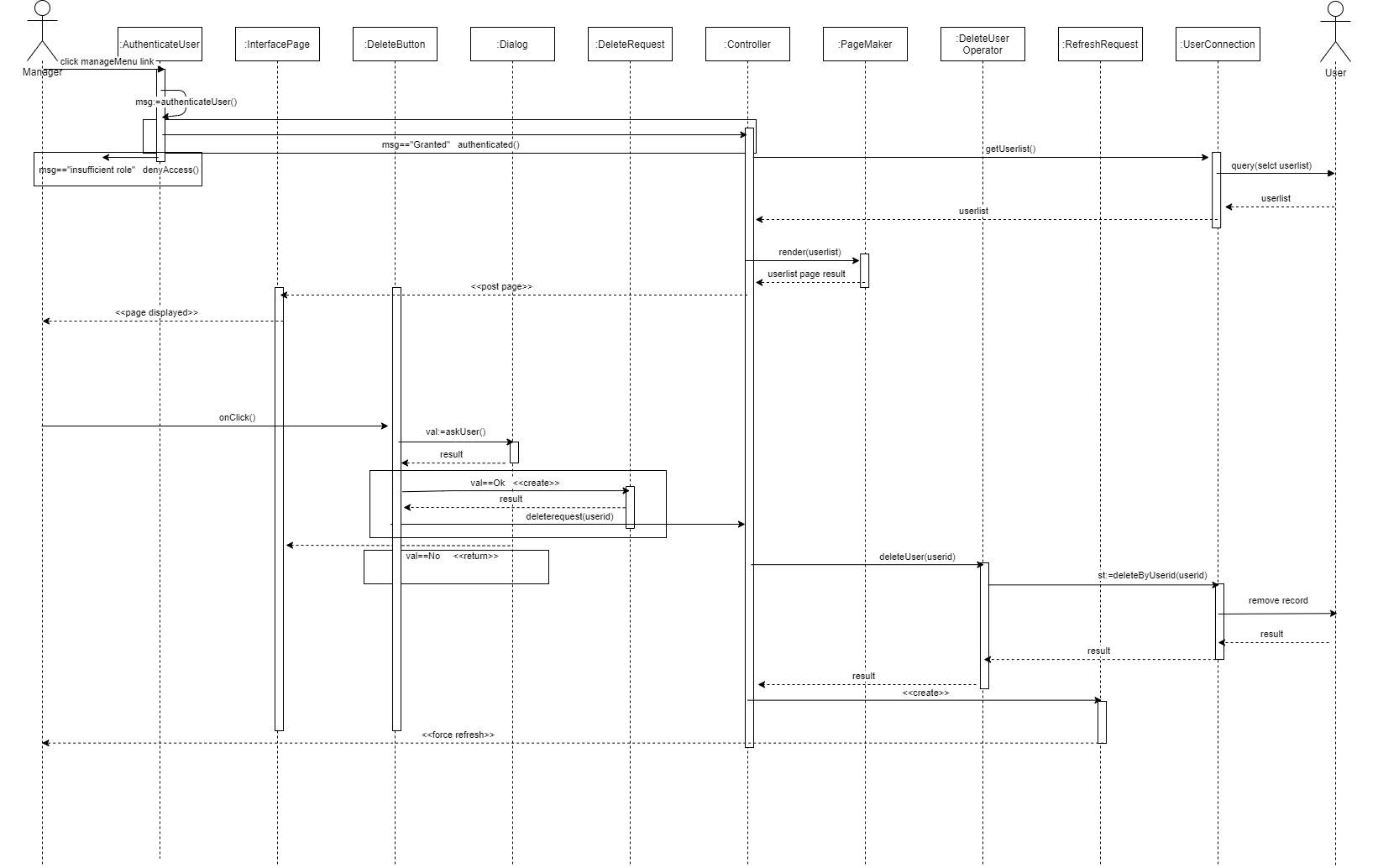


* + 1. **최종 결과 및 선정 이유**

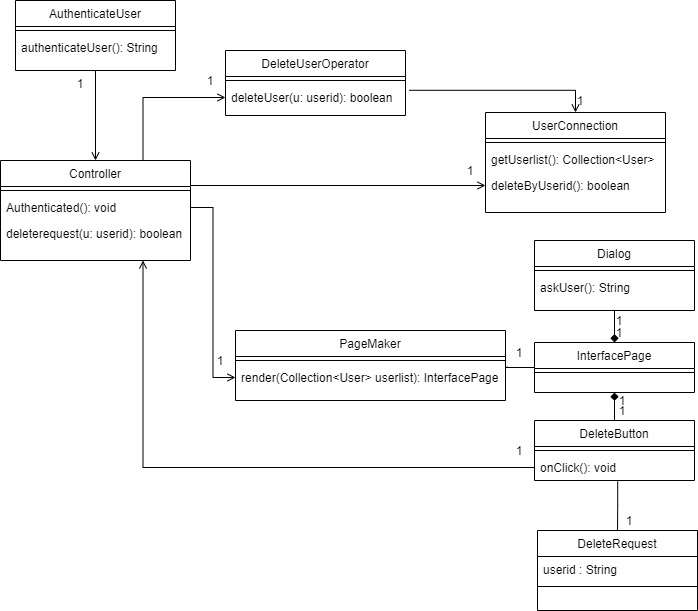
1. Object Sequence Diagram

AuthenticateUser의 인증되었다는 authenticated를 받는 object는 Controller가 하도록하고, Controller가 사용자의 리스트를 받아와서 렌더링하도록 PageMaker에게요청하는 variation을 선택하였다. Authentiated를 받는 역할은 PageMaker보다는 Controller가 받아서 적절한 요청을 보내주는 것이 맞다고 생각했다. 또한 userlist를 받아오는 부분에서는 해당 방식을 통해 향후 다른 MenuManagement에서 이를 적용할 때 userlist대신 menulist를 전달만해주면되므로 더 유연한 개발이 가능할 것이다. 또한, getUserlist도 high-level logic이고 render도 high-level logic이므로 이러한 variation에서도 controller가 과도한 low-level logic을 처리한다고 할 수 없기 때문에 가능하다.

DeleteUser Operation을 수행한 후의 작업을 수행하는 것에 있어서는 다시 페이지를 렌더링하는 로직이 맨 처음 사용자가 인증받았을 때 userlist를 받아와서 렌더링하는 방식과 똑같으므로, 사용자에게 RefreshRequest를 보내 강제로 새로고침을 하게 하면 다시 처음부터 이 Sequence Diagram의 페이지 렌더링 부분이 실행된다. 따라서 중복되게 PageMaker가 렌더링하지 않아도 강제로 새로고침하도록 요청을 보내는 것이 로직 이해에도 쉽고, 개발시에도 중복된 개발을 막을 수 있다.



1. Class Diagram



1. Traceability Matrix

테이블이(가) 표시된 사진

자동 생성된 설명

* 1. **DisplayByPeriod**

1. Basic
2. Variation1
3. Variation 2
4. Variation 3
   * 1. **최종 결과 및 선정 이유**
5. Object Sequence Diagram
6. Class Diagram
7. Traceability Matrix

# **SubgroupC(OrderStatus)**

* 1. **C-UC-1 blah blah**