

Online Pizza Ordering System

<SDD – The 2nd Checkpoint>

소프트웨어공학 02분반

20171105 이민욱

20173875 정용준

20170223 신원준

20172609 여일구

20174438 정종민

20173156 김준기



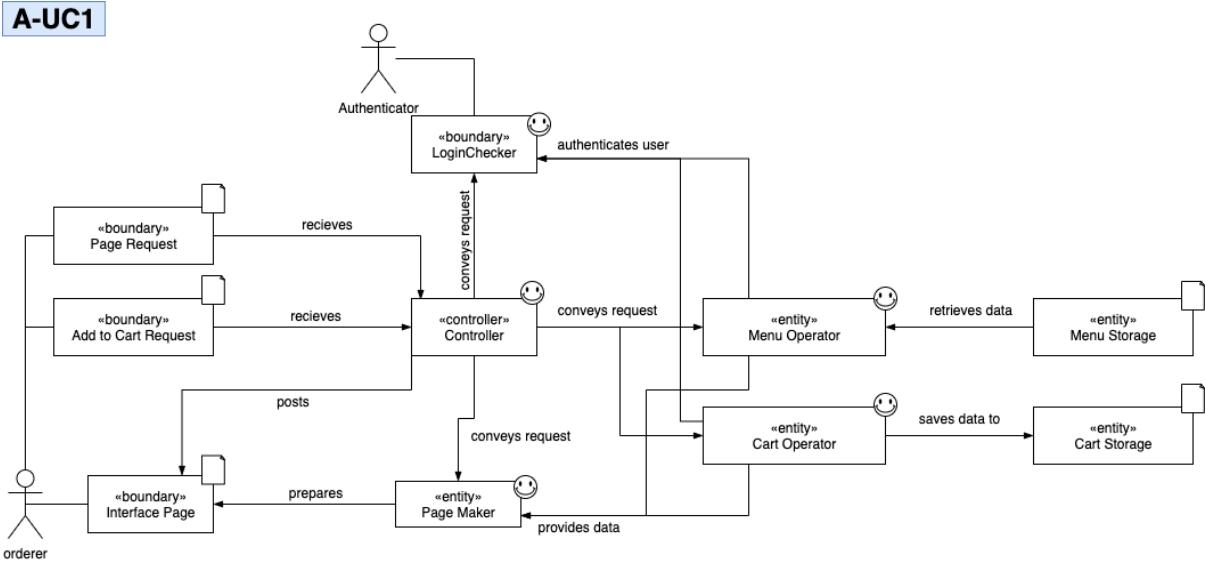
목차

| | |
|---|----|
| Online Pizza Ordering System..... | 1 |
| <SDD – The 2 nd Checkpoint>..... | 1 |
| 1. SubgroupA(PizzaOrder) | 4 |
| 1.1 A-UC-1: Add Menu To Cart | 4 |
| 1.1.1 최종 결과 및 선정 이유 | 6 |
| 1.2 A-UC-4: Remove Cart Item | 8 |
| 1.2.1 최종 결과 및 선정 이유 | 9 |
| 1.3 A-UC-5: Order | 11 |
| 1.3.1 최종 결과 및 선정 이유 | 13 |
| 2. SubgroupB(Management) | 16 |
| 2.1 B-UC-1: AuthenticateUser | 16 |
| 2.1.1 최종 결과 및 선정 이유 | 19 |
| 2.2 B-UC-3: AddPizza | 21 |
| 2.2.1 최종 결과 및 선정 이유 | 25 |
| 2.3 B-UC-9: DeleteUser | 27 |
| 2.3.1 최종 결과 및 선정 이유 | 31 |
| 2.4 DisplayByPeriod | 34 |
| 2.4.1 최종 결과 및 선정 이유 | 37 |
| 3. SubgroupC(OrderStatus) | 39 |
| 3.1 C-UC-1 Accept Order | 39 |
| 3.1.1 최종 결과 및 선정 이유 | 42 |
| 3.2 C-UC-2 Complete Cook..... | 43 |
| 3.2.1 최종 결과 및 선정 이유 | 43 |
| 3.3 C-UC-10 Display Order..... | 45 |

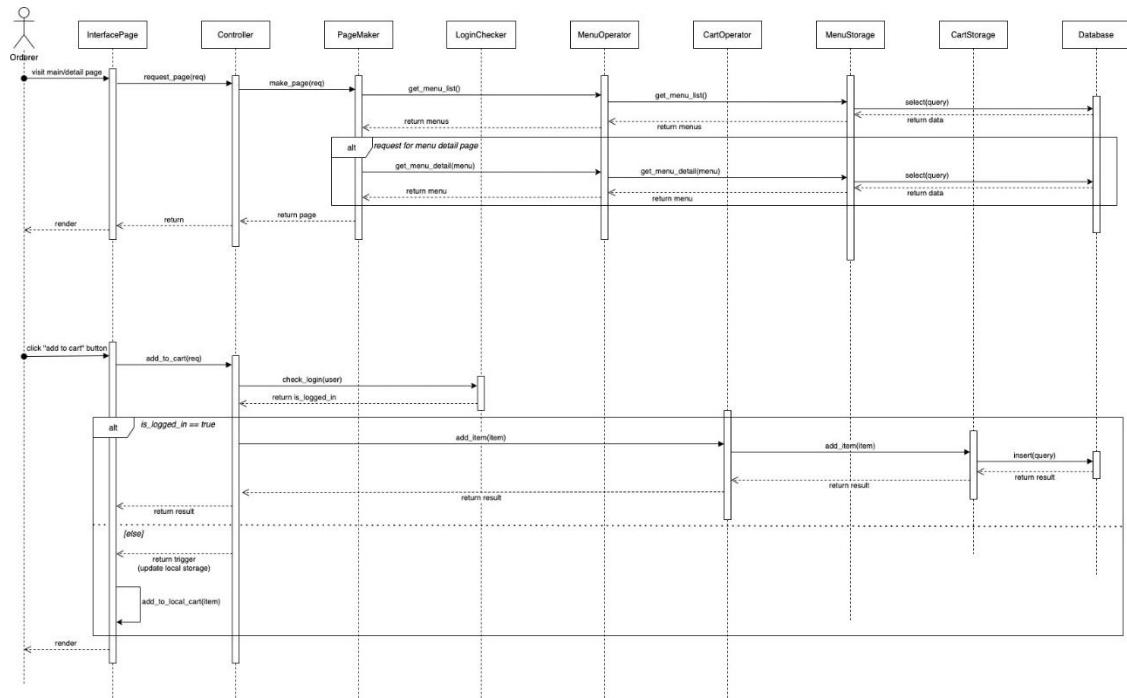
| | | |
|-------|---------------------------|----|
| 3.3.1 | 최종 결과 및 선정 이유 | 48 |
| 3.4 | C-UC-11 Login User | 50 |
| 3.4.1 | 최종 결과 및 선정 이유 | 52 |
| 3.5 | Class Diagram | 54 |

1. SubgroupA(PizzaOrder)

1.1 A-UC-1: Add Menu To Cart



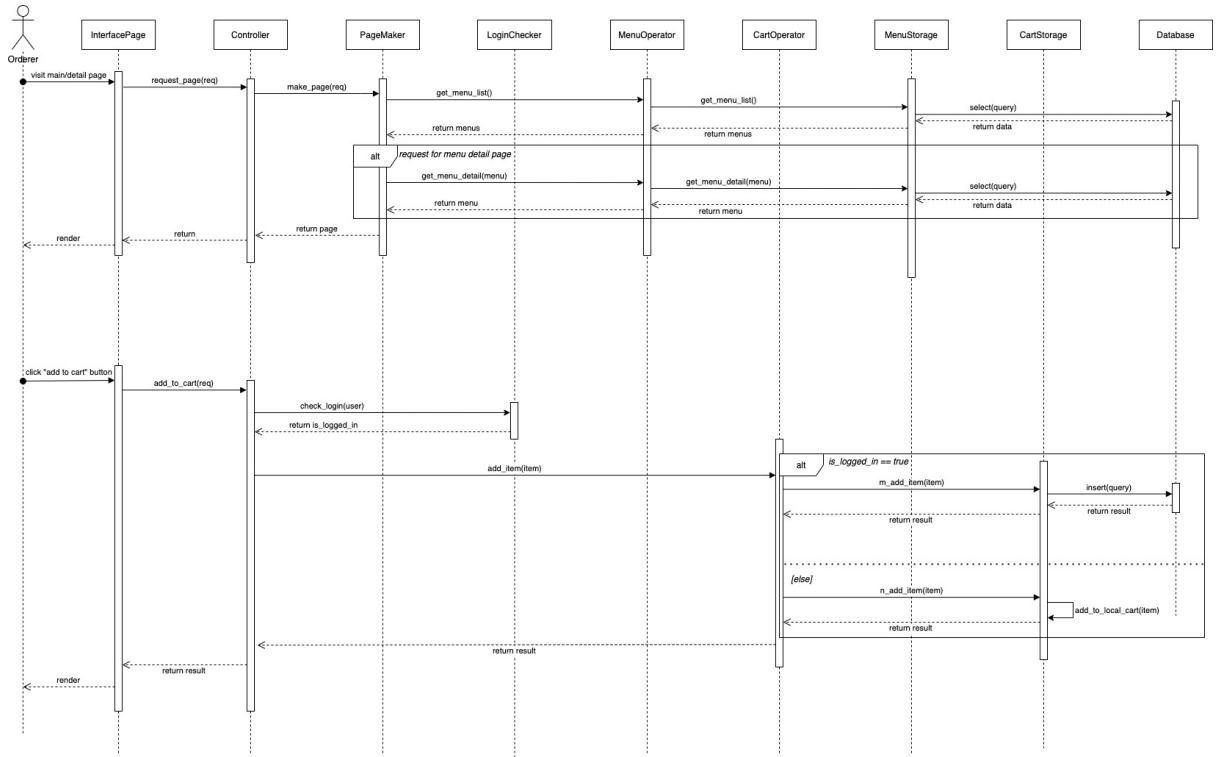
1) Basic



2) Variation 1

Basic과의 차이는 장바구니 데이터를 다룰 때에서 발생한다.

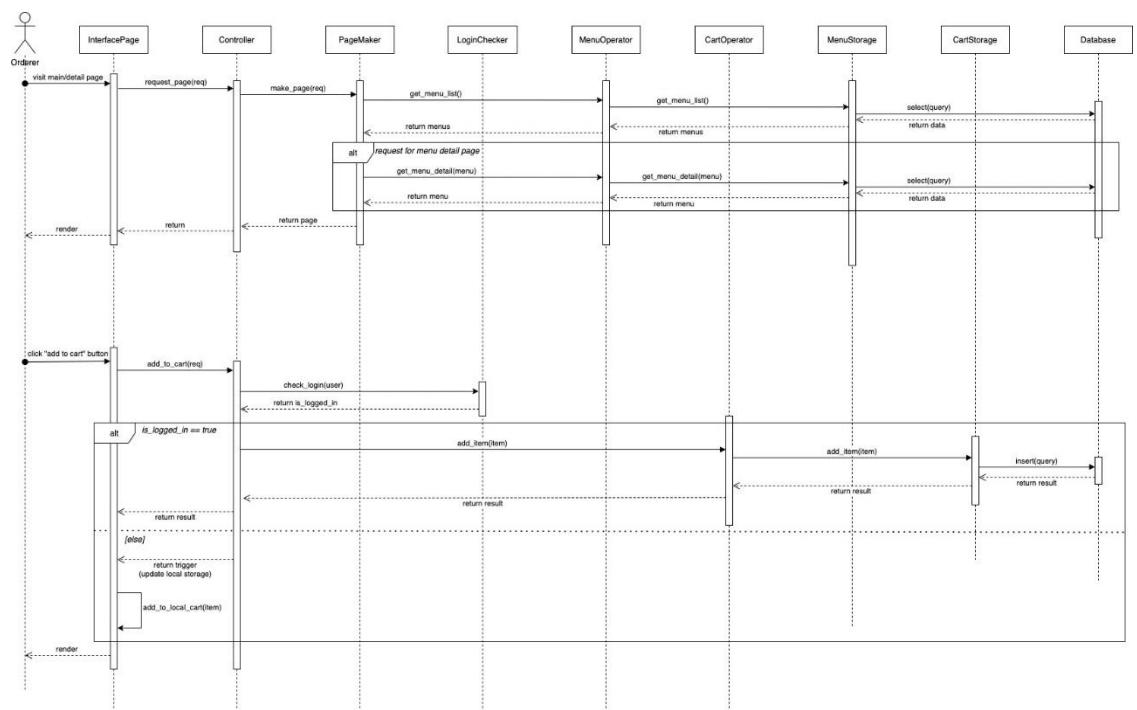
장바구니 데이터의 저장 위치는 유저가 로그인 상태일 때는 DB, 비로그인 상태일 때는 로컬 저장소이다. DB/로컬저장소가 Storage 클래스로 추상화된 형태인 Variation이 새롭게 제안되었다.



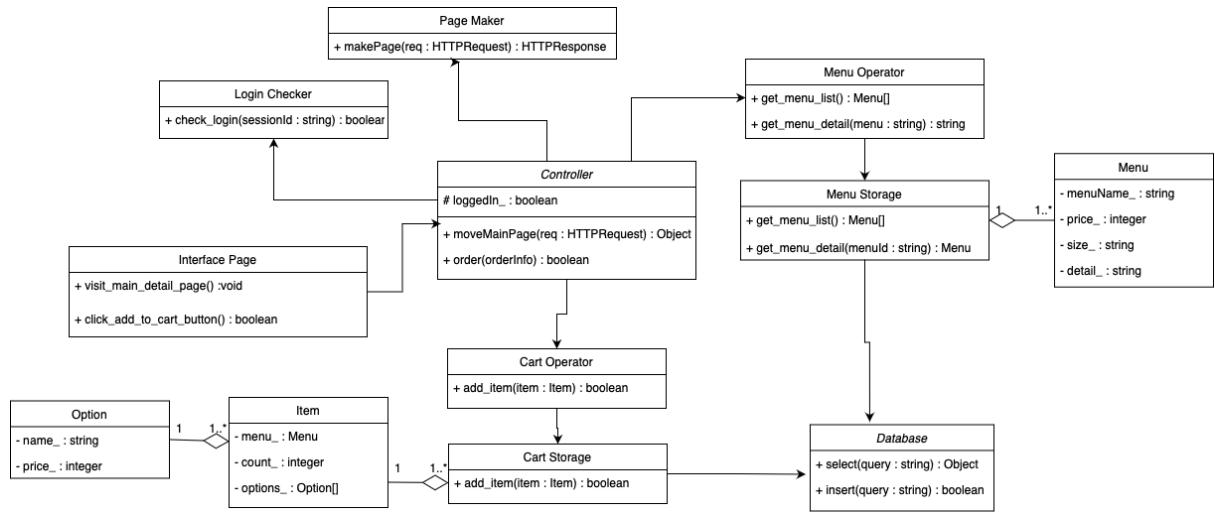
1.1.1 최종 결과 및 선정 이유

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.



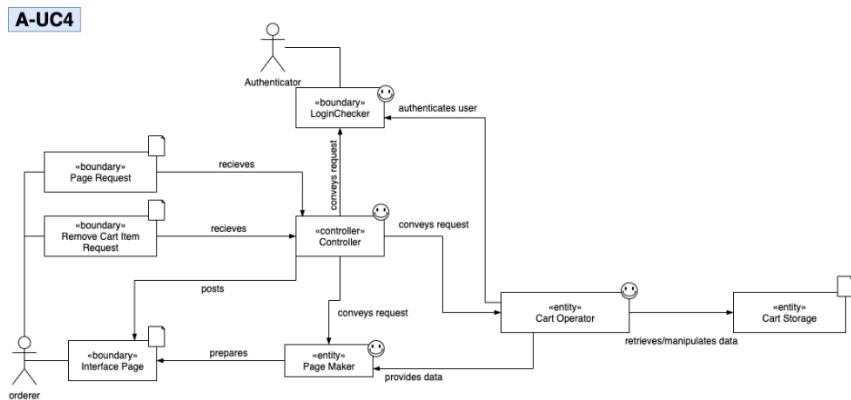
2) Class Diagram



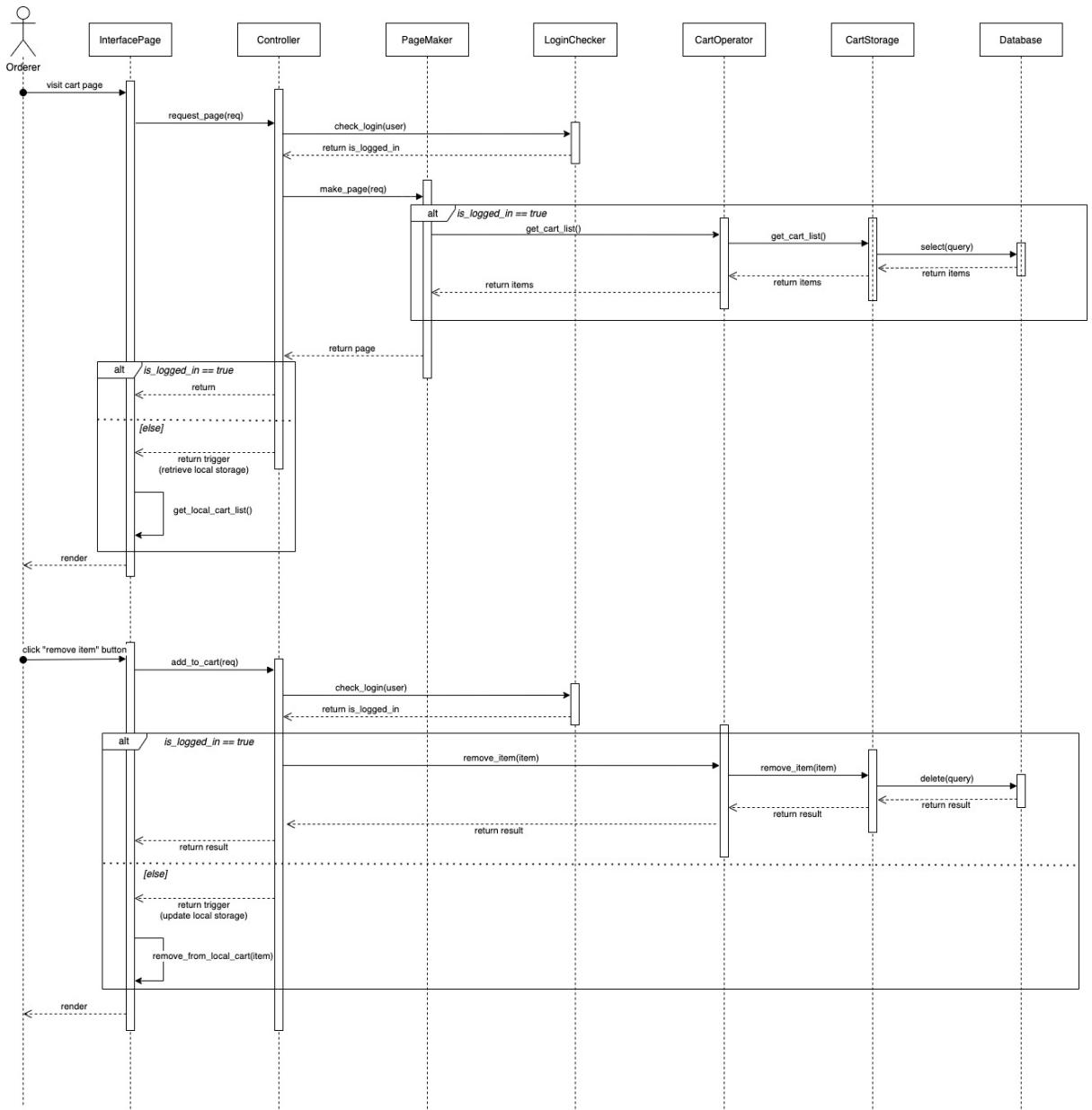
3) Traceability Matrix

| Domain Concept ↴ | Software Classes ↴ | | | | | | | | | | | |
|-----------------------|--------------------|--------------|------------------|-----------------|-----------------|----------------|-----------------|----------------|------------|--------|--------|----------|
| | Controller ↴ | Page Maker ↴ | Interface Page ↴ | Login Checker ↴ | Menu Operator ↴ | Menu Storage ↴ | Cart Operator ↴ | Cart Storage ↴ | Database ↴ | Menu ↴ | Item ↴ | Option ↴ |
| 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 ↴ |

1.2 A-UC-4: Remove Cart Item



1) Basic



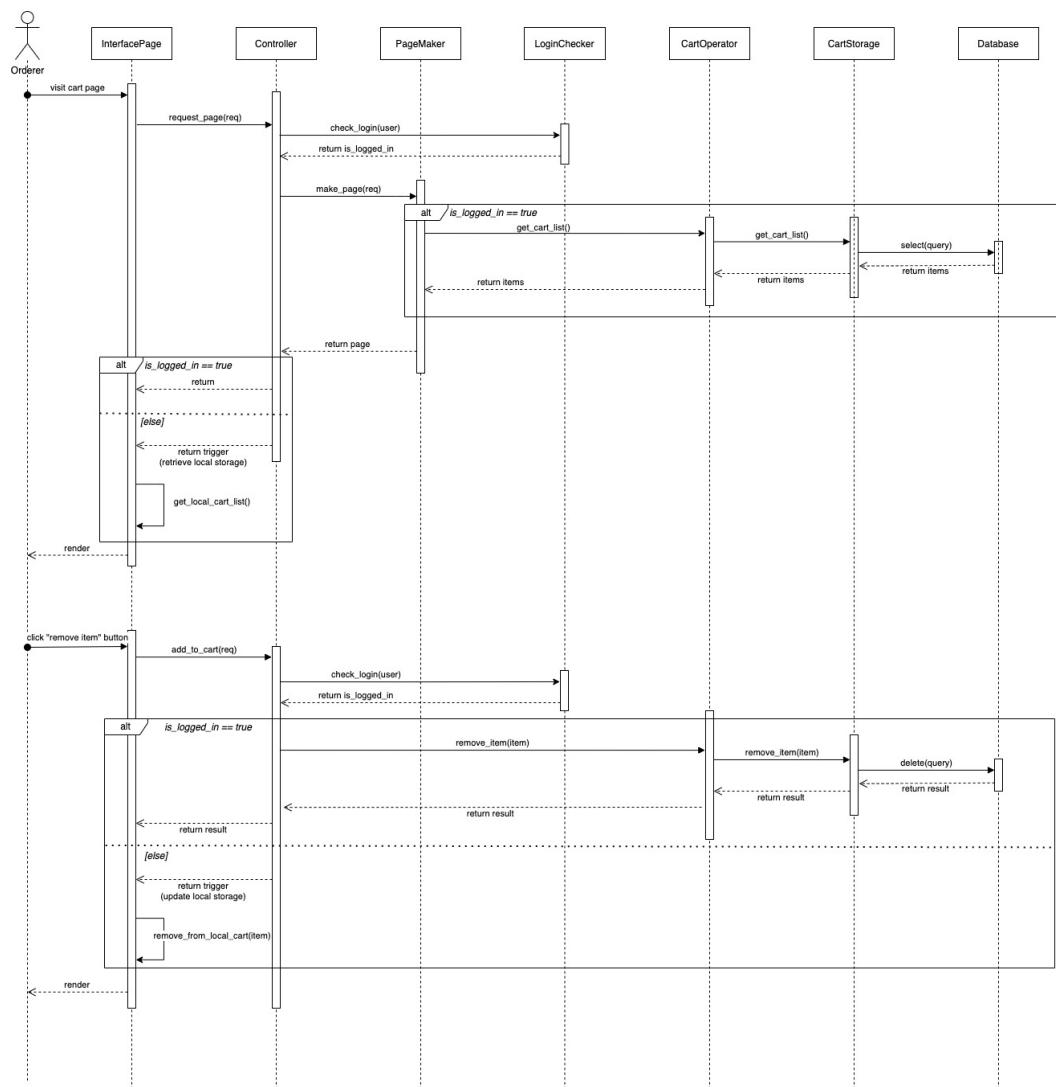
2) Variation

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

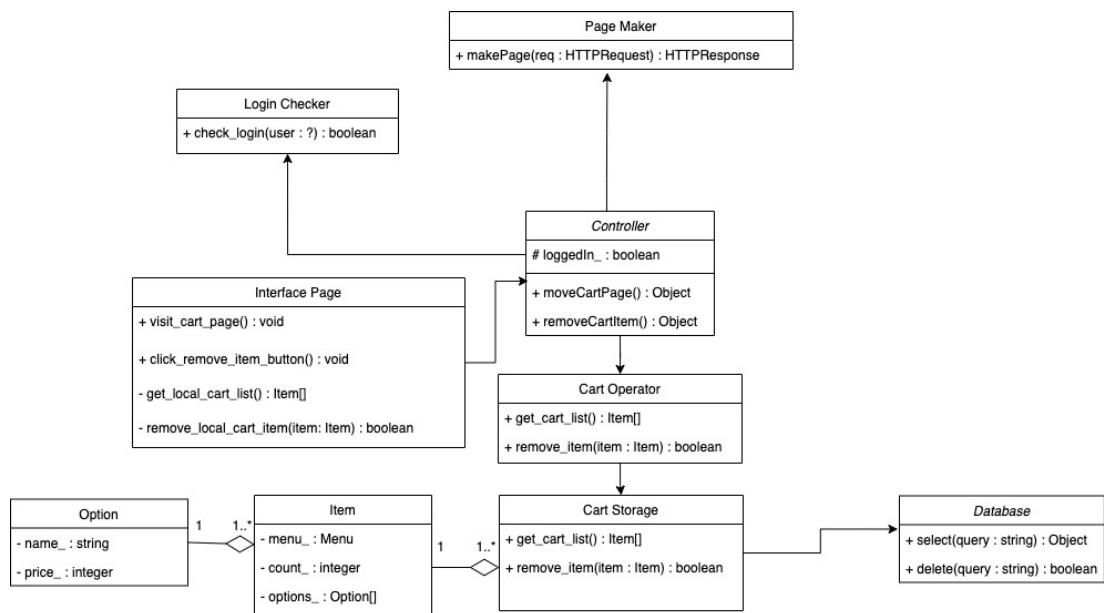
1.2.1 최종 결과 및 선정 이유

1) Object Sequence Diagram

A-UC-1과 A-UC-4는 요청의 의미가 장바구니 Item의 삭제인지 추가인지에 따른 차이 뿐이므로 전반적인 설계는 동일하다. 따라서 Basic Sequence Diagram은 A-UC-1에서 논의된 바와 동일한 과정을 거쳐 나온 것이므로 그대로 채택한다.



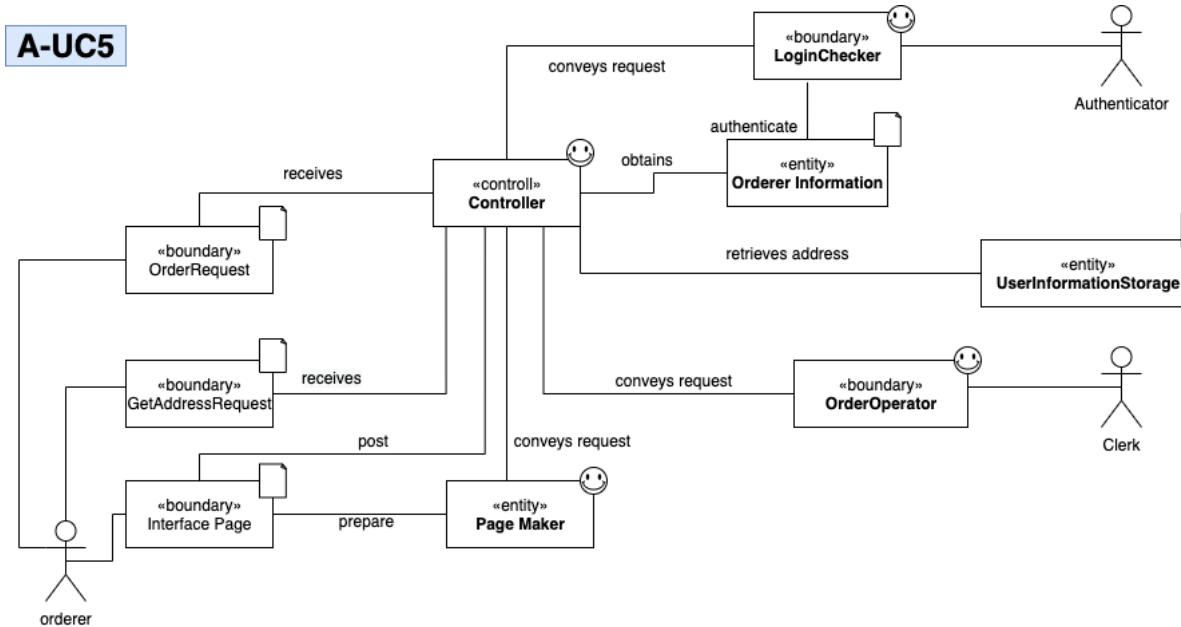
2) Class Diagram



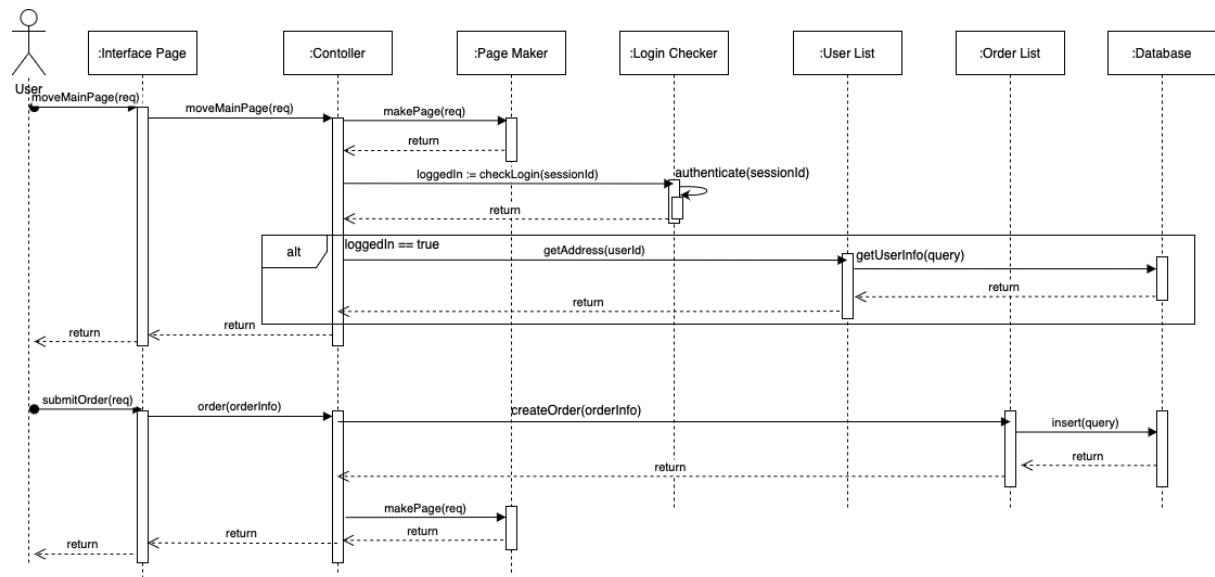
3) Traceability Matrix

| Domain Concept | Software Classes | | | | | | | | | |
|--------------------------|------------------|------------|----------------|---------------|---------------|--------------|----------|------|------|--------|
| | Controller | Page Maker | Interface Page | Login Checker | Cart Operator | Cart Storage | Database | Menu | Item | Option |
| Controller | X | ↔ | ↔ | ↔ | ↔ | ↔ | ↔ | ↔ | ↔ | ↔ |
| PageMaker | ↔ | X | ↔ | ↔ | ↔ | ↔ | ↔ | ↔ | ↔ | ↔ |
| Interface Page | ↔ | ↔ | X | ↔ | ↔ | ↔ | ↔ | ↔ | ↔ | ↔ |
| Login Checker | ↔ | ↔ | ↔ | X | ↔ | ↔ | ↔ | ↔ | ↔ | ↔ |
| Page Request | ↔ | ↔ | X | ↔ | ↔ | ↔ | ↔ | ↔ | ↔ | ↔ |
| Remove Cart Item Request | ↔ | ↔ | X | ↔ | ↔ | ↔ | ↔ | ↔ | ↔ | ↔ |
| Cart Operator | ↔ | ↔ | ↔ | ↔ | X | ↔ | ↔ | ↔ | ↔ | ↔ |
| Cart Storage | ↔ | ↔ | ↔ | ↔ | ↔ | X | X | X | X | X |

1.3 A-UC-5: Order



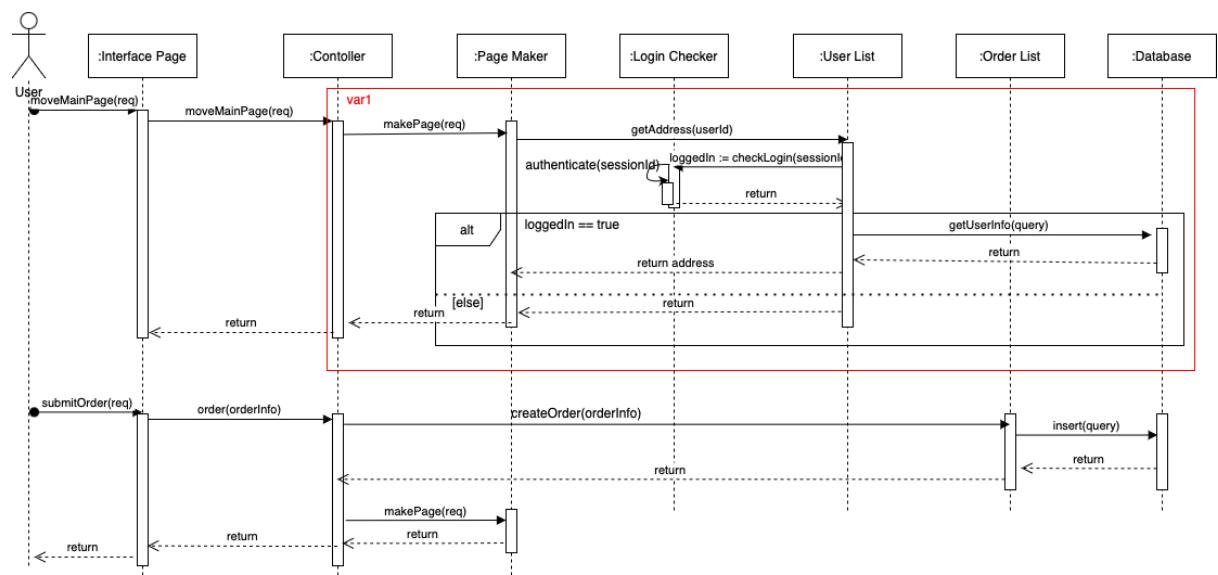
1) Basic



2) Variation 1

초기에는 controller에서 페이지를 호출할 때 로그인 여부에 대한 데이터를 가지고 어디에서 주소를 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.

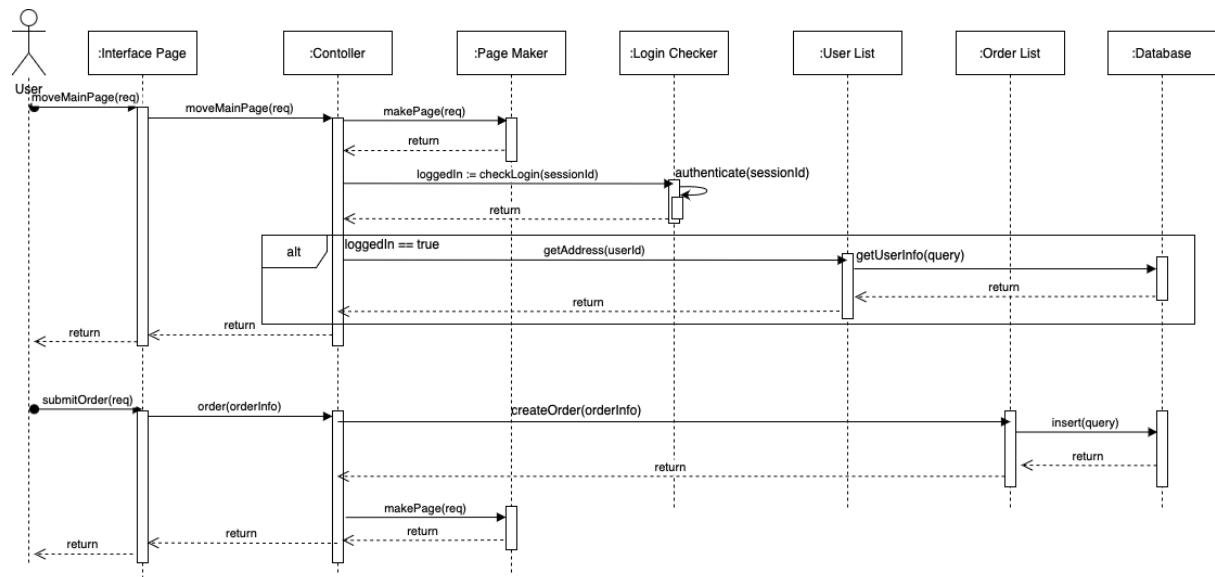
Responsibility 측면에서 봤을 때 controller의 경우 역할 분배가 잘 된 것으로 볼 수 있다. 하지만 Page Maker가 다른 데이터를 호출하고, Database와의 인터페이스 역할을 하는 User List에서 로그인 여부를 판단하기 위해 check login을 호출하는 등 오히려 responsibility 상에서 문제가 생기는 것을 볼 수 있다.



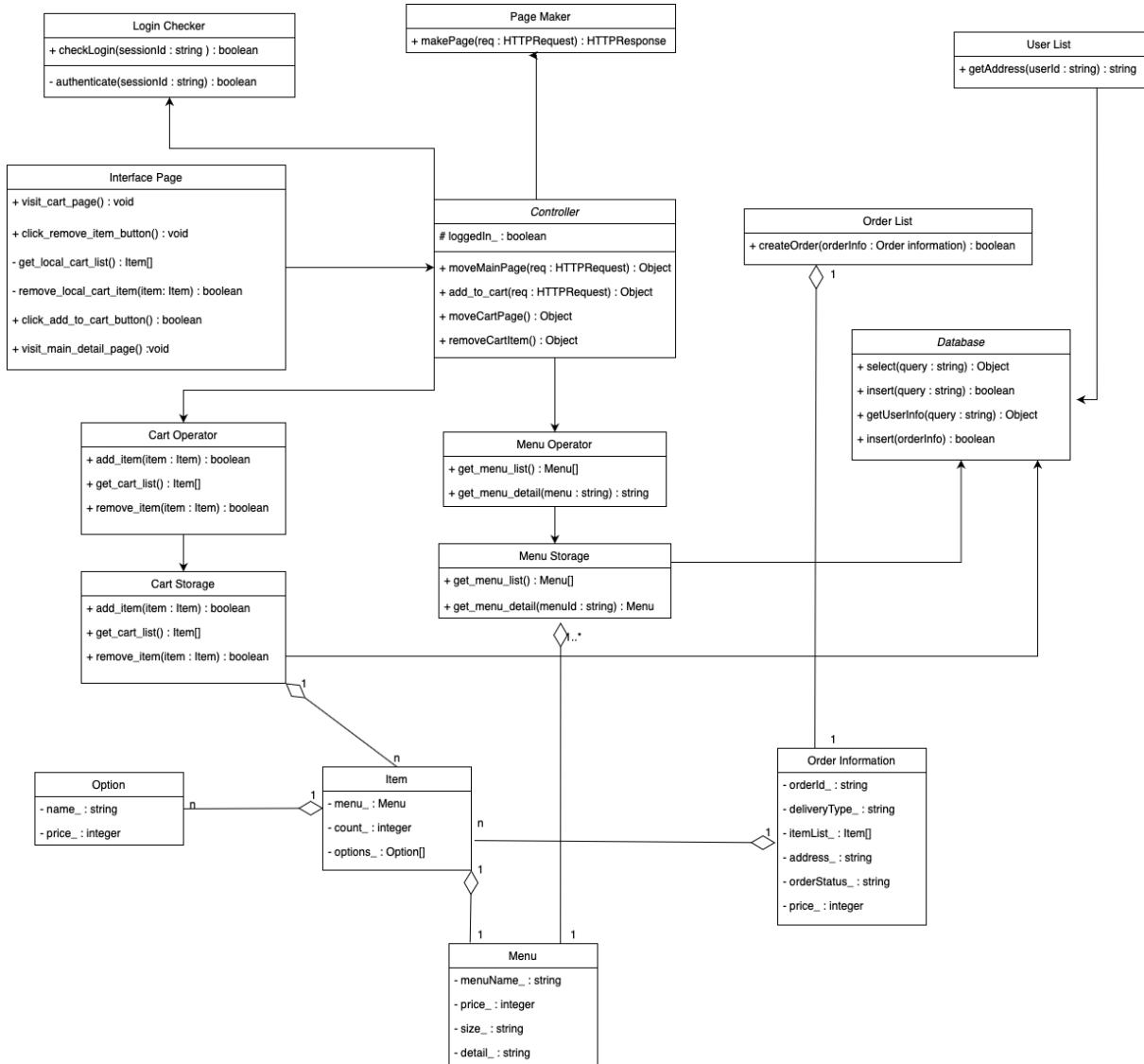
1.3.1 최종 결과 및 선정 이유

1) Object Sequence Diagram

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.



2) Class Diagram

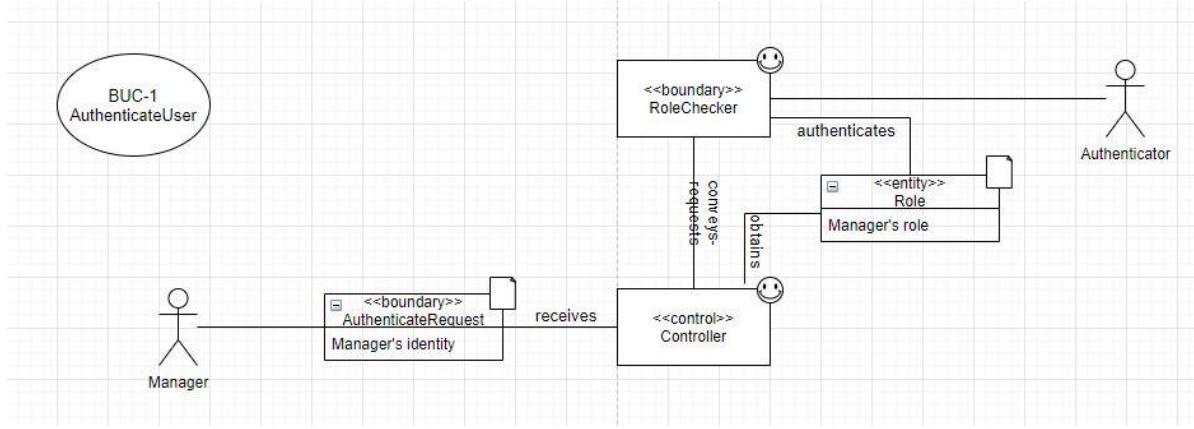


3) Traceability Matrix

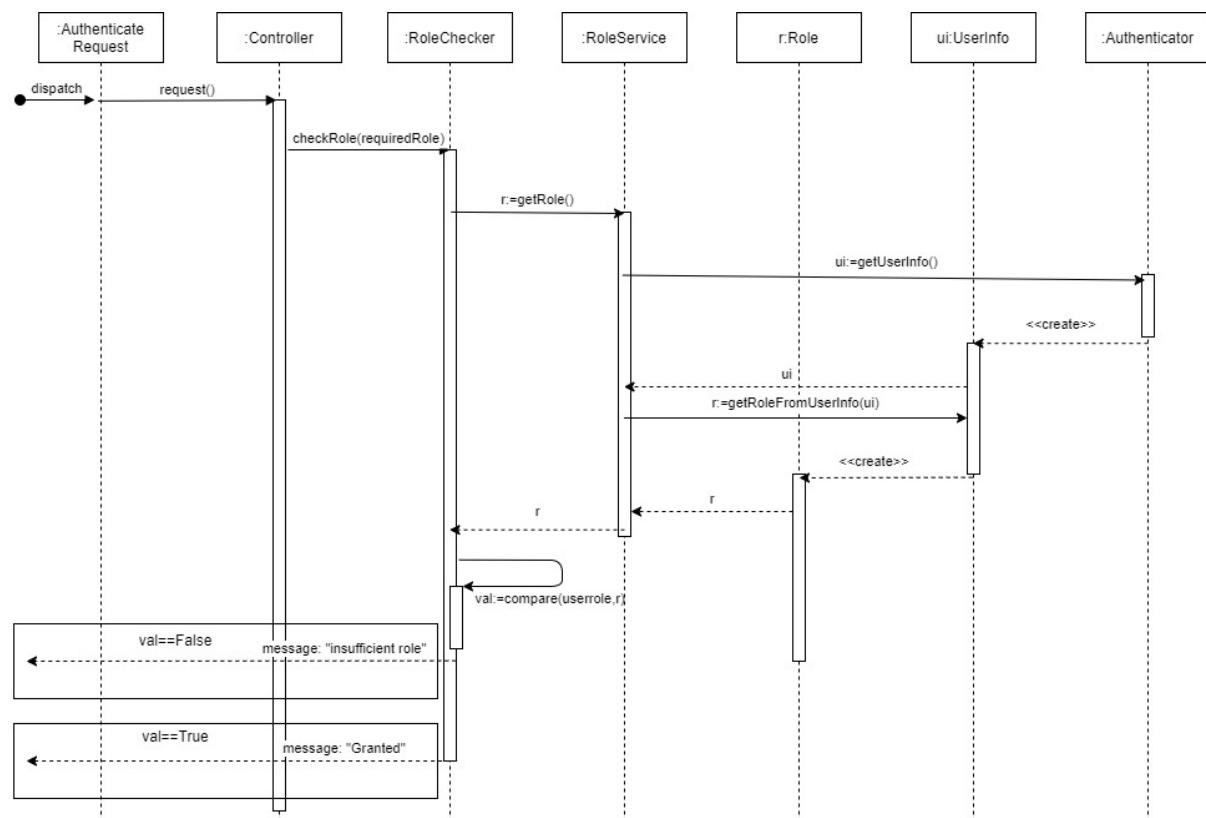
| Domain concepts | Software classes | | | | | | | | | | | | | |
|--------------------------|------------------|------------|----------------|---------------|---------------|--------------|---------------|--------------|-----------|------------|----------|------|------|--------|
| | Controller | Page Maker | Interface Page | Login Checker | Menu Operator | Menu Storage | Cart Operator | Cart Storage | User List | Order List | Database | Menu | Item | Option |
| 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 |

2. SubgroupB(Management)

2.1 B-UC-1: AuthenticateUser



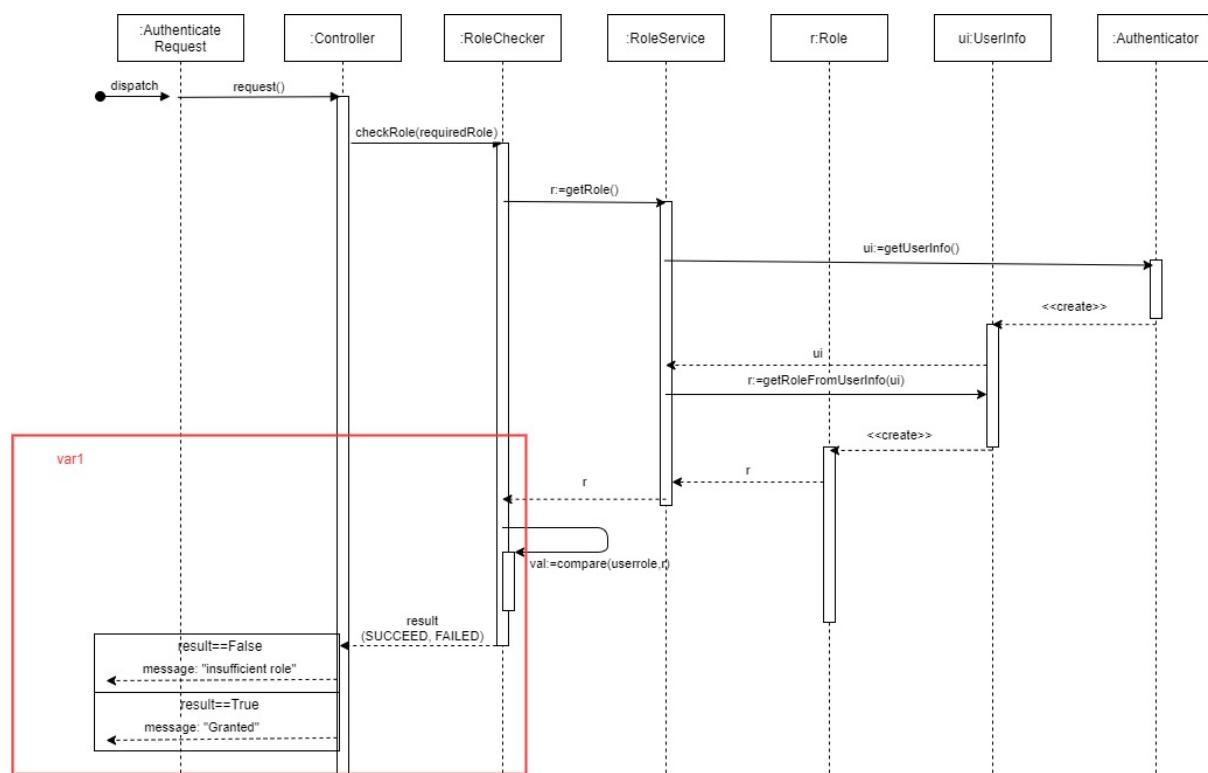
1) Basic



2) 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.

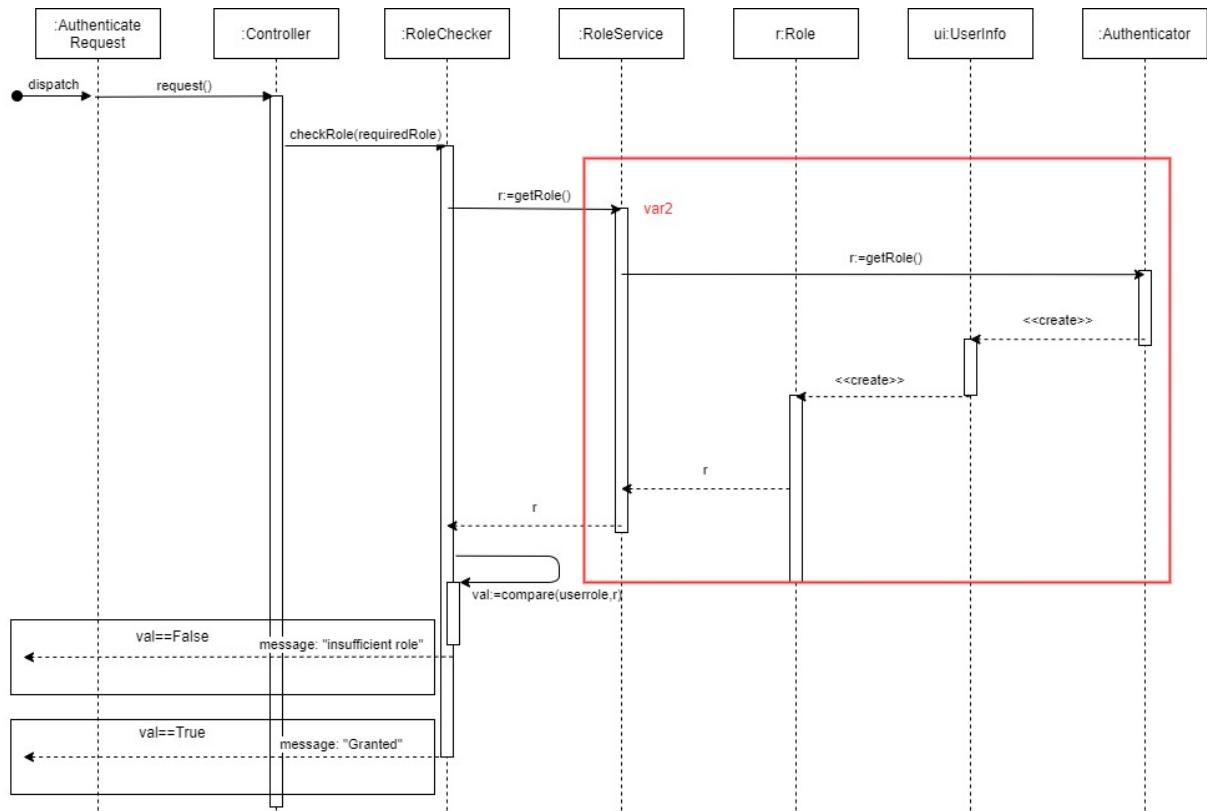
Responsibility 측면에서는 RoleChecker가 역할을 확인하는 역할을 하므로, 역할에 따라 결과 값을 바로 알려주도록 하는 방식이 적절하다. 그러나, Controller에서 checkRole(requiredRole)을 호출하였으므로 이에 대한 결과를 받아보아야 한다는 측면과, Controller가 checkRole에 대한 result를 받아서 단순히 결과에 따른 반환 값만 반환해준다는 것에서는 Variation1 방식이 적절하다.



3) 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 것이 아닌, 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 getter를 이용해 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.

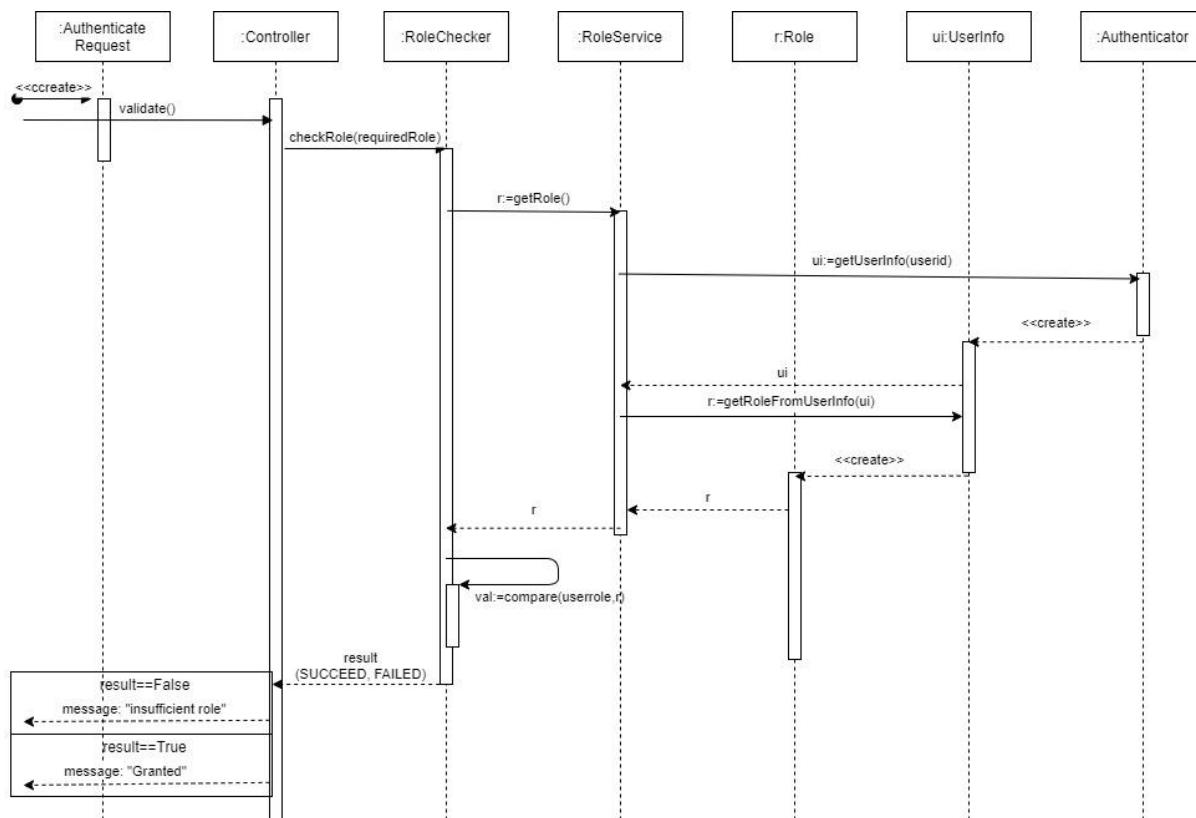


2.1.1 최종 결과 및 선정 이유

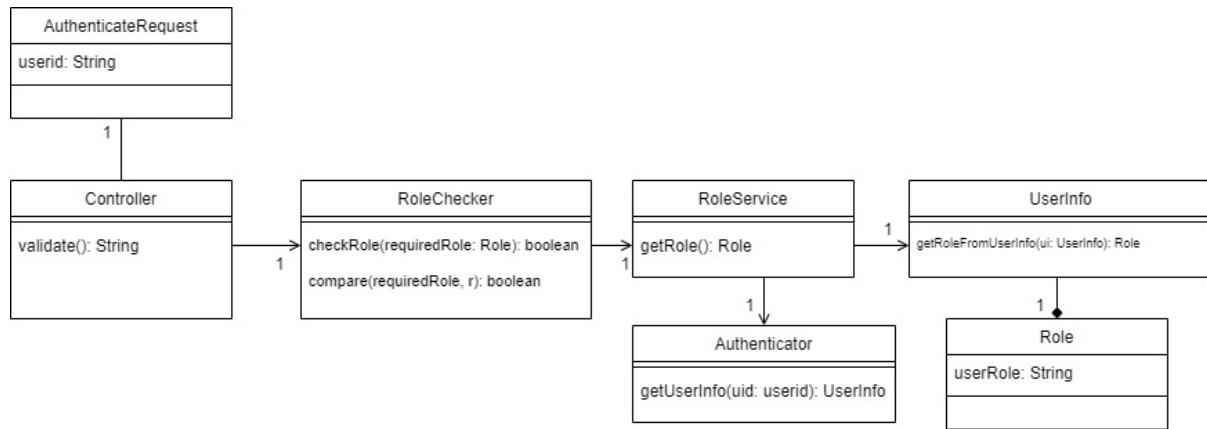
1) Object Sequence Diagram

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 디어그램에서는 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 해치지 않는다고 생각하였고, 다른 variation보다 개발에 있어 용이하기 때문에 선택하였다.



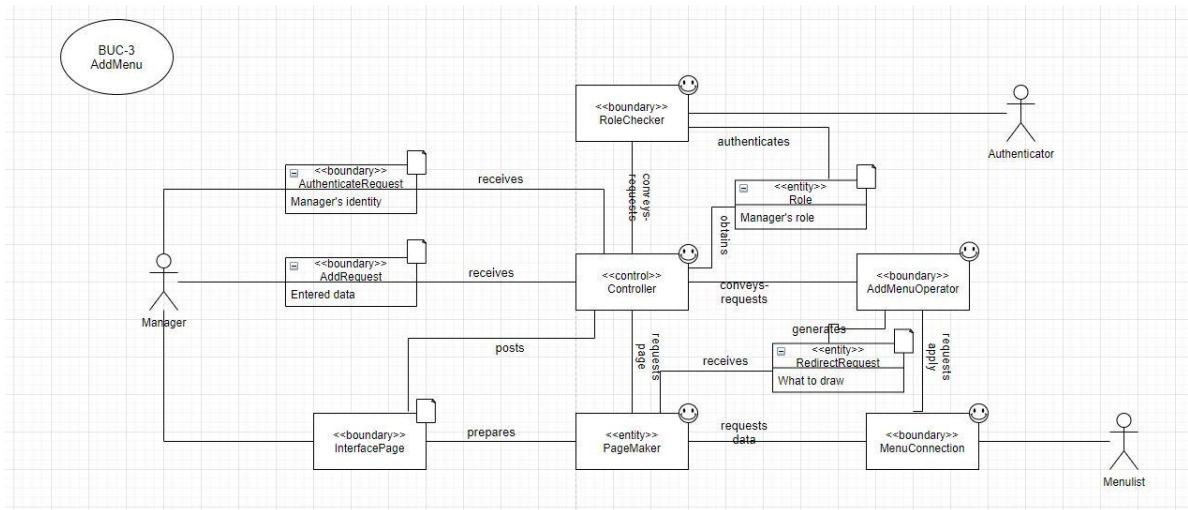
2) Class Diagram



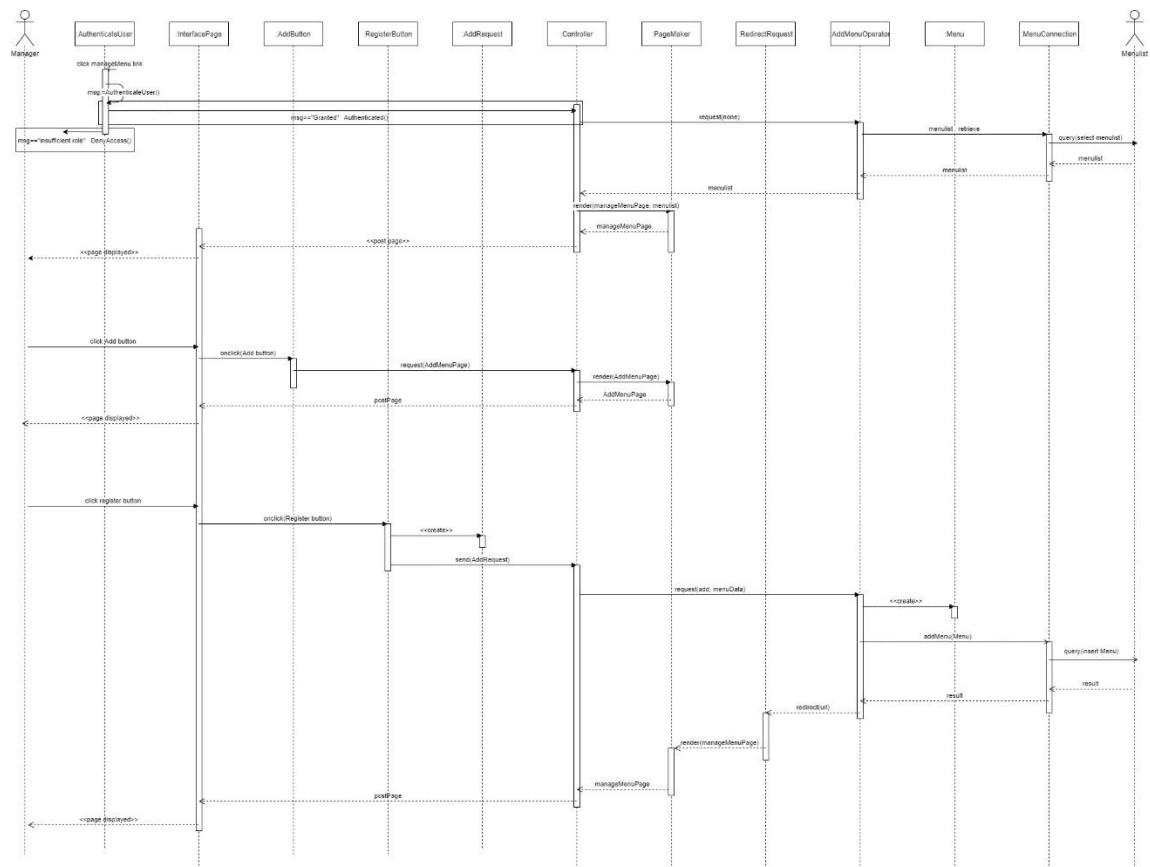
3) Traceability Matrix

| | | Software classes | | | | | |
|---------------------|--|------------------|---------------------|-------------|-------------|------|----------|
| | | Controller | AuthenticateRequest | RoleChecker | RoleService | Role | UserInfo |
| Domain concepts | | O | O | O | O | O | O |
| Controller | | O | O | O | O | O | O |
| AuthenticateRequest | | O | O | O | O | O | O |
| RoleChecker | | O | O | O | O | O | O |
| Role | | O | O | O | O | O | O |

2.2 B-UC-3: AddPizza

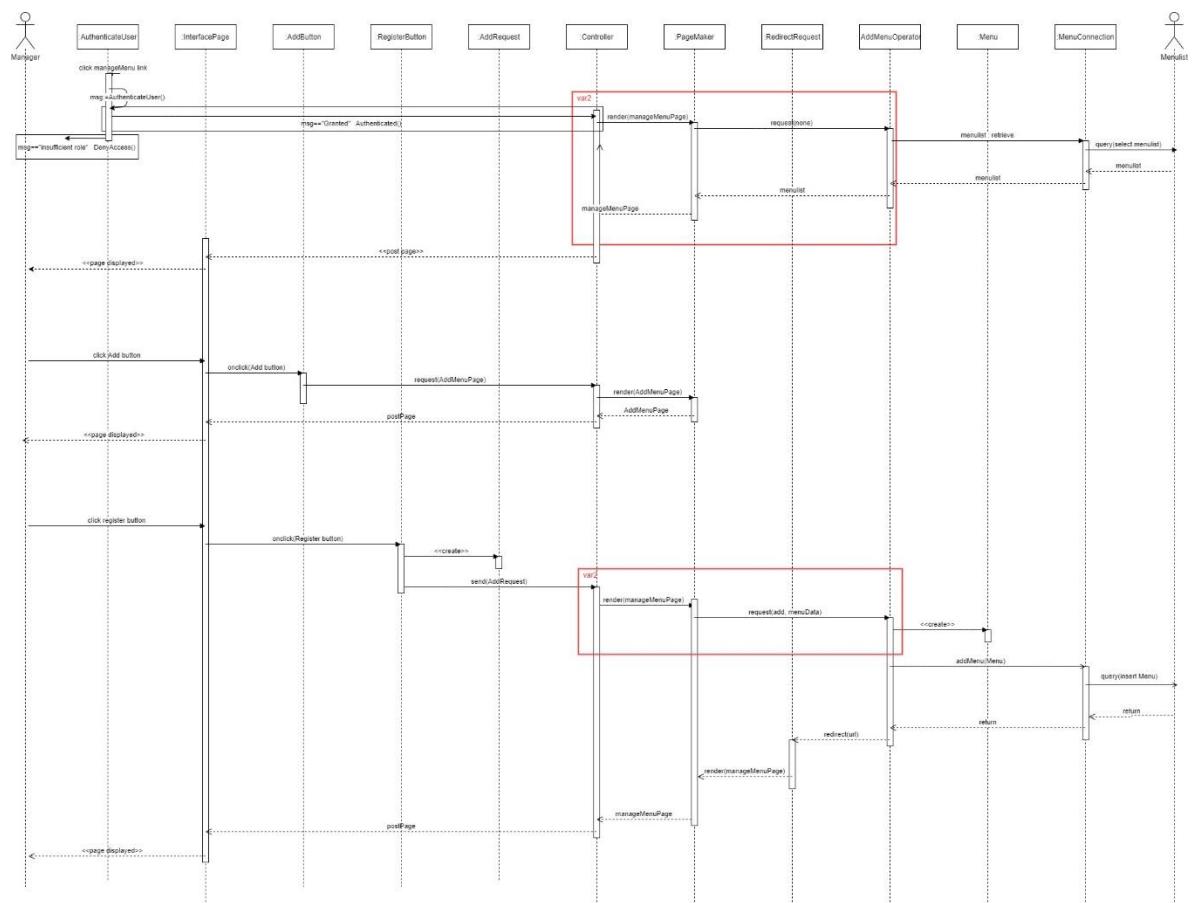


1) Basic



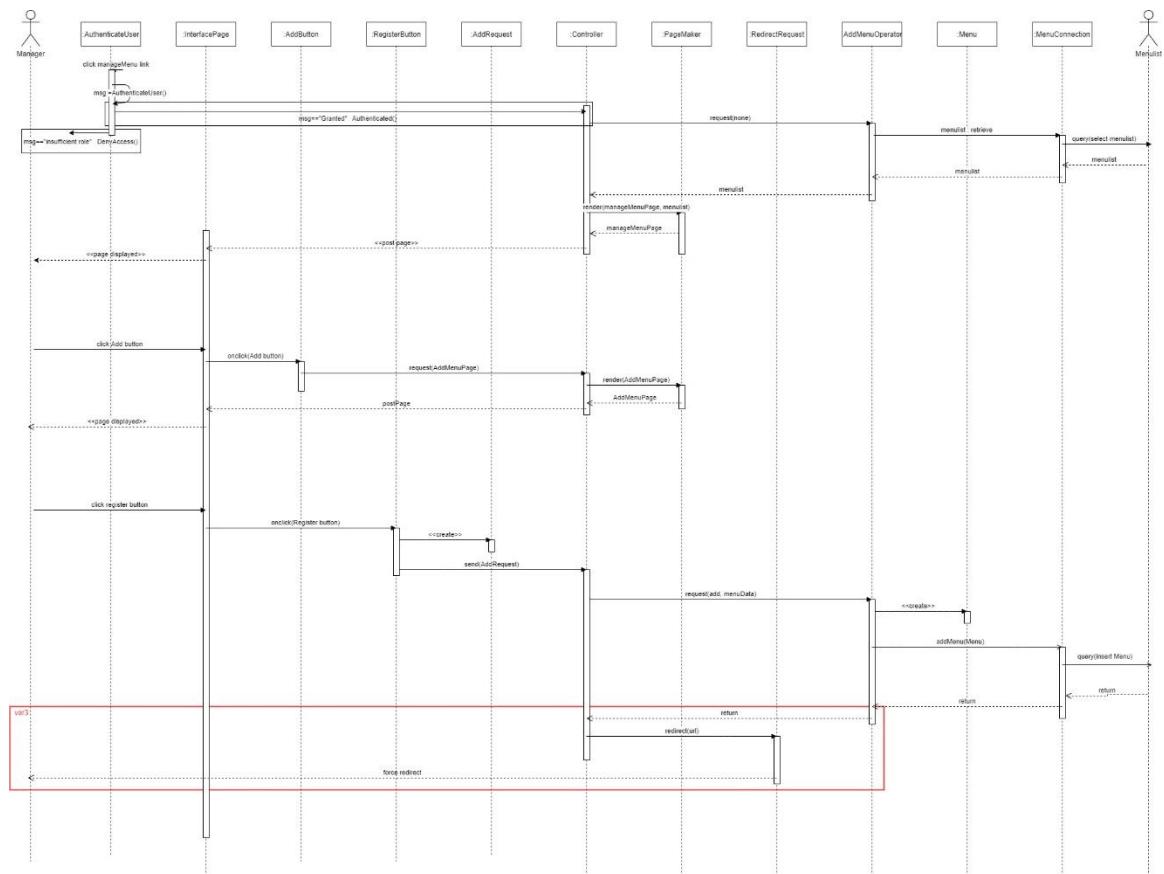
2) Variation1

첫 번째 variation은 PageMaker의 responsibility에 대한 variation이다. Basic sequence에서 데이터를 요청하고 이를 렌더링하는 작업을 각각 Controller와 PageMaker가 나누어 가지는 반면 variation 1에서는 이 작업을 PageMaker가 전담한다. 즉, Controller는 PageMaker에게 렌더링 요청만 하고 이후 렌더링에 관한 모든 작업은 PageMaker에서 처리한다는 것이다. 이 variation은 PageMaker에게 너무 많은 Responsibility를 맡긴다는 문제점이 있다.



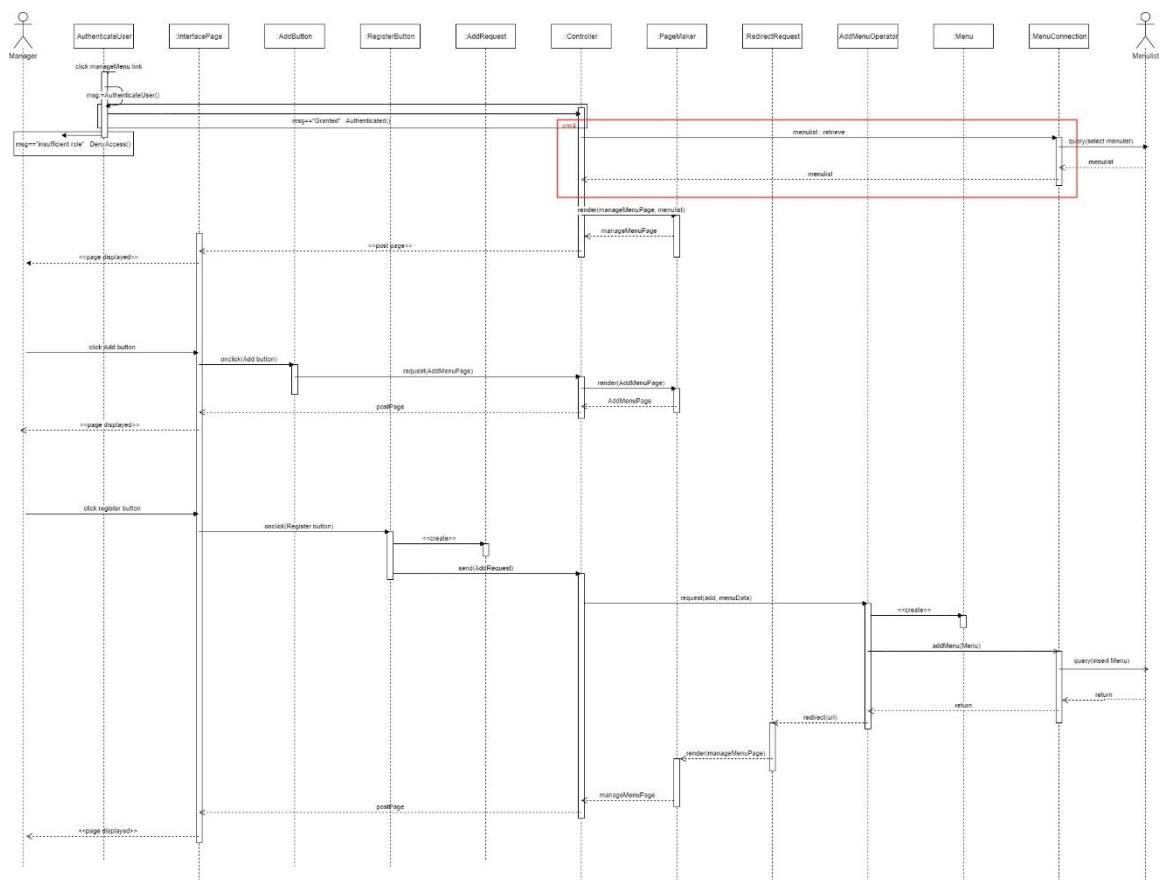
3) Variation 2

두 번째 variation은 리다이렉션의 responsibility에 대한 variation이다. Basic sequence를 보면 AddMenuOperator에서 리다이렉트 요청을 생성하여 PageMaker로 보낸다. PageMaker는 요청을 받아 해당 페이지 생성 후 Controller에게 넘긴다. 반면, variation 2에서는 Controller가 메뉴 추가 요청에 대한 응답이 오면 리다이렉트 요청을 생성해 강제로 리다이렉트를 진행한다. AddMenuOperator의 응집도를 높이며 보다 직관적이라는 장점이 있다.



4) Variation 3

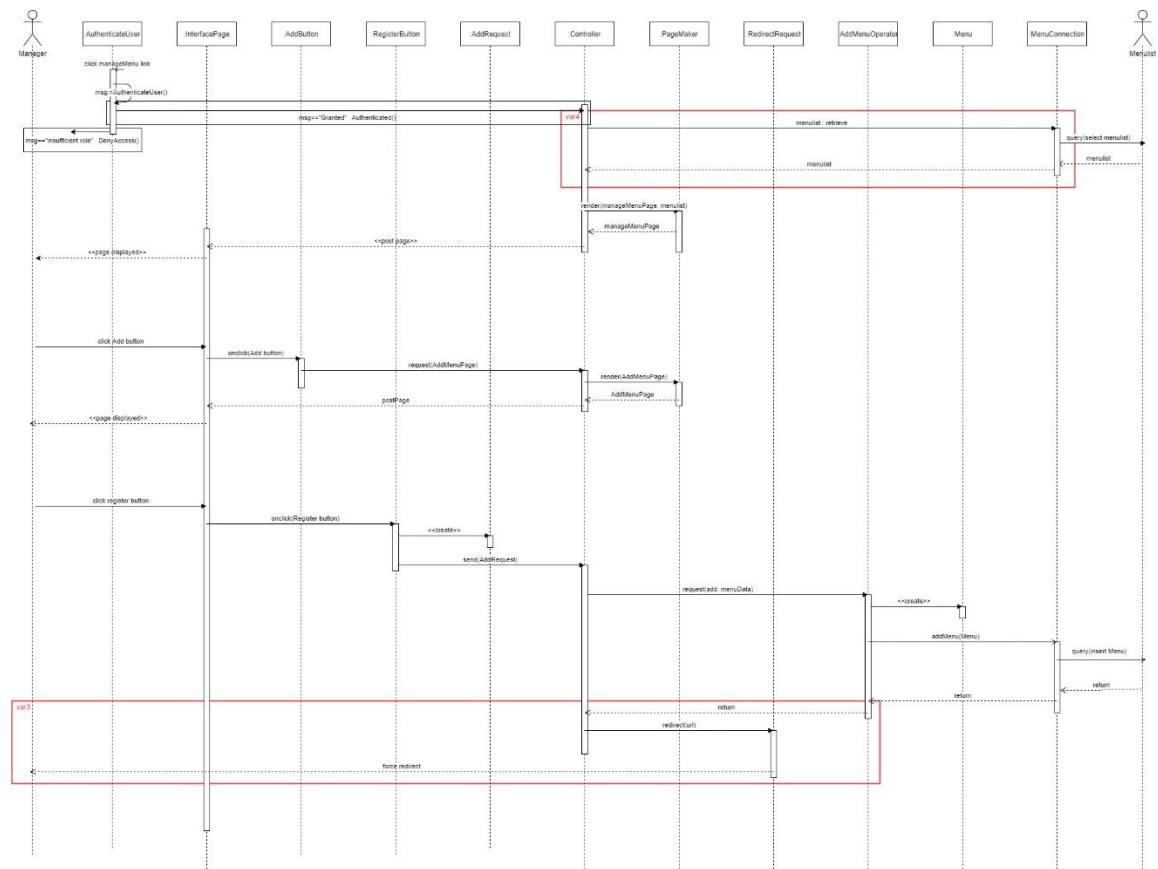
세 번째 variation은 AddMenuOperator의 responsibility에 대한 variation이다. 아래 그림을 보면 처음 메뉴 리스트를 가져오는 과정에서 Controller가 AddMenuOperator를 거치지 않고 바로 MenuConnection으로 요청을 보낸다. AddMenuOperator는 메뉴 추가 요청만 처리하고 단순히 메뉴 리스트를 가져오는 작업은 Controller에게 맡기는 것이다. AddMenuOperator의 응집도를 높인다는 장점이 있다. 다만, ManageMenuOperator라는 상위 클래스가 생긴다면 메뉴 리스트를 불러오는 등 조작에 관한 부분은 이 클래스에게 맡기는 것이 나을 수도 있다.



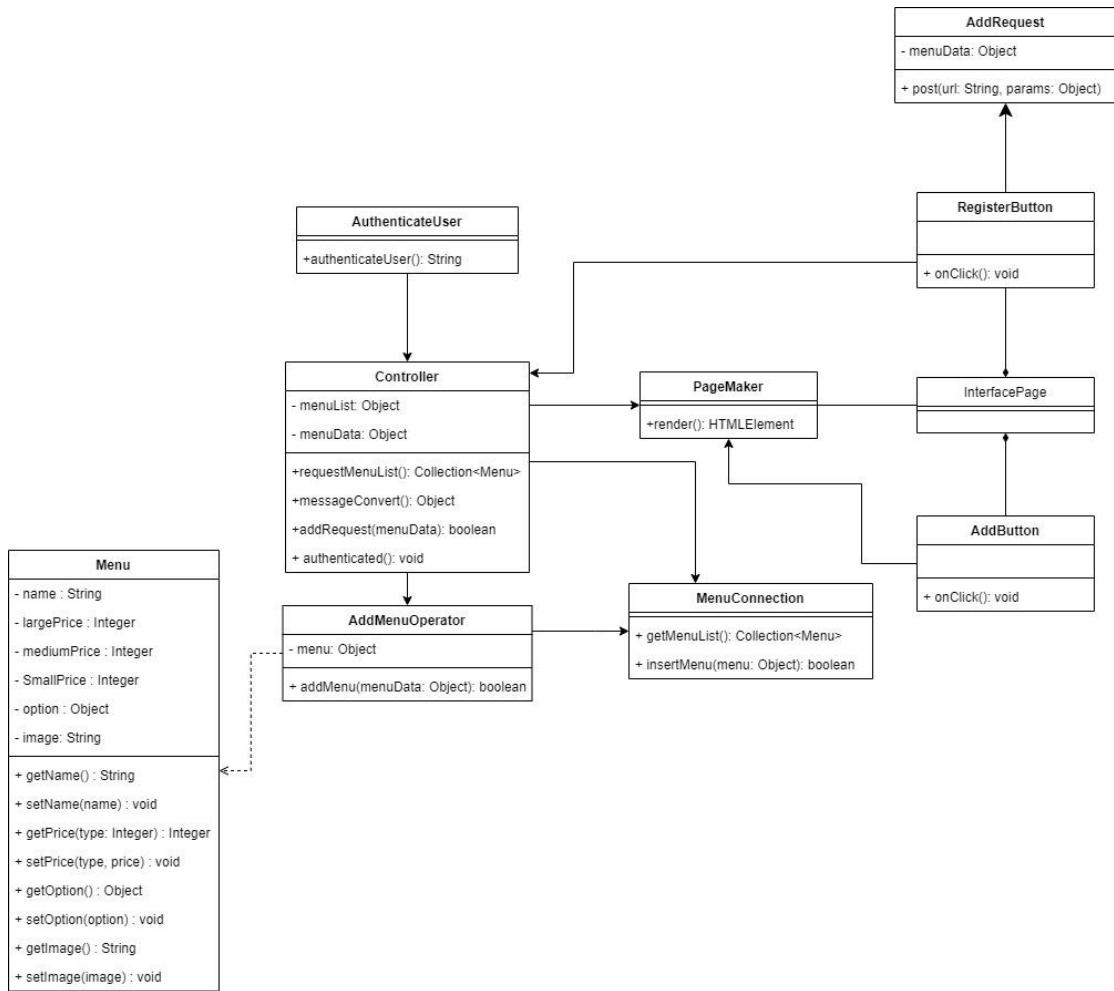
2.2.1 최종 결과 및 선정 이유

1) Object Sequence Diagram

Variation 2와 3를 적용하여 최종 OSD를 완성하였다. 각 모듈의 응집도를 우선적으로 보았다. Controller는 요청과 응답을 전달, AddMenuOperator는 메뉴 추가, PageMaker는 페이지 렌더링만을 담당한다. 또한, 리다이렉트 과정은 요청을 받으면 이후 과정은 초반 메뉴 리스트를 불러오는 과정과 같으므로 variation 2를 적용하여 불필요한 구현을 생략하였다.



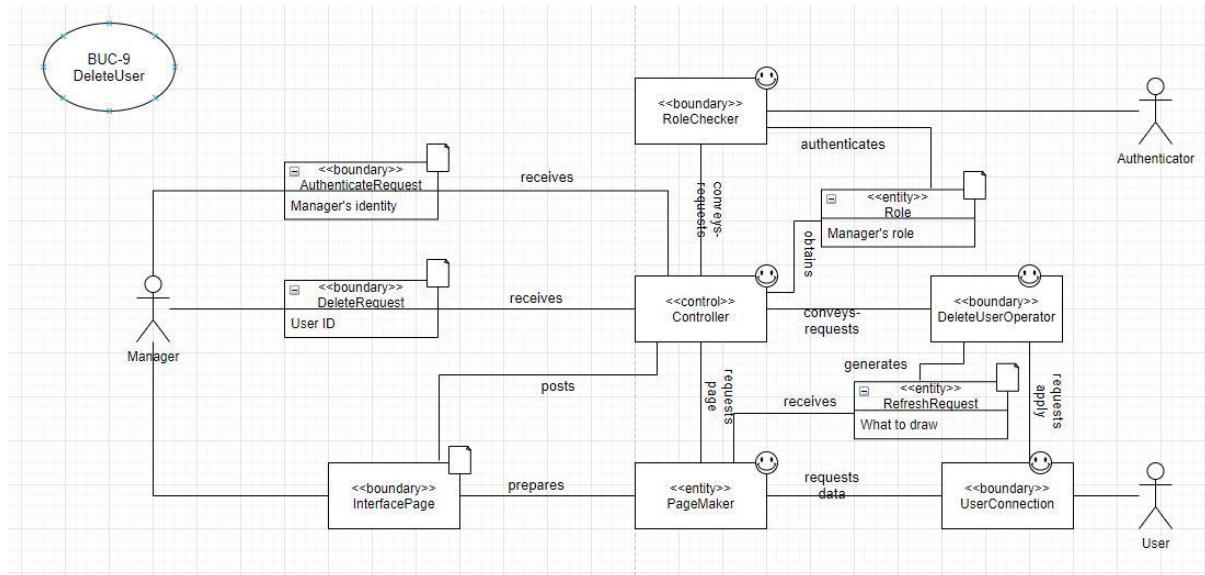
2) Class Diagram



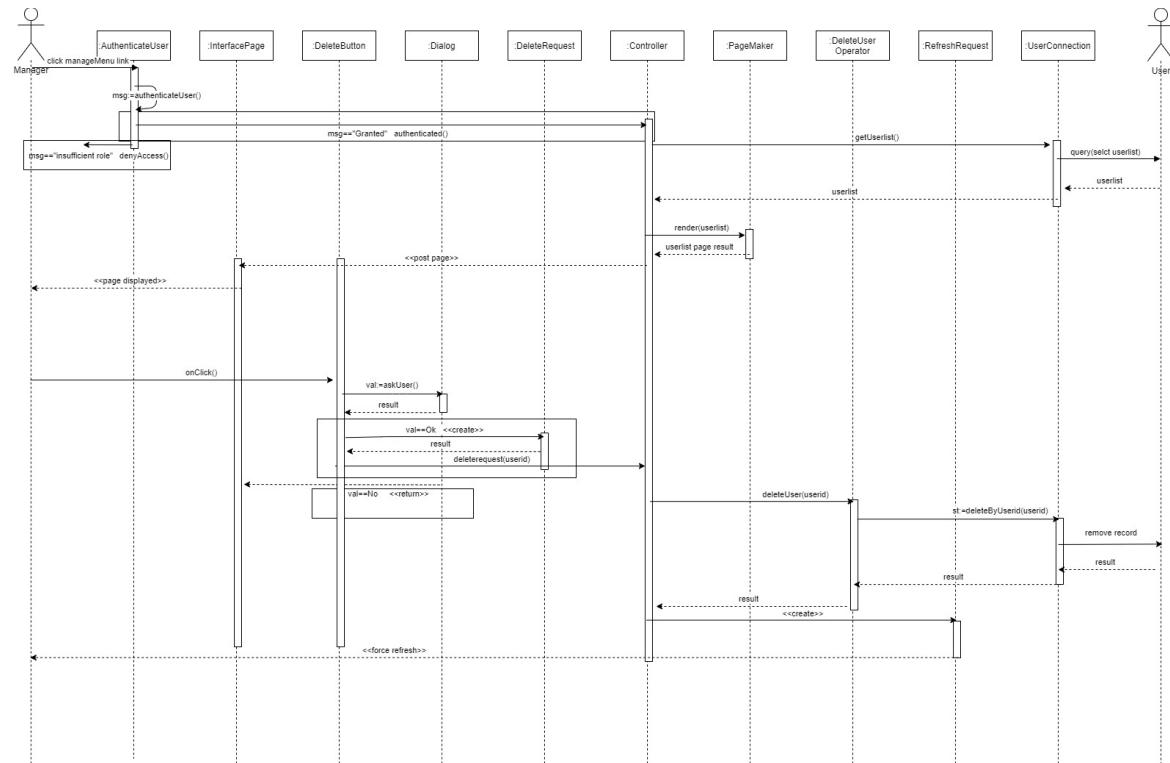
3) Traceability Matrix

| Domain concepts ^② | Software classes ^③ | | | | | | | | | |
|----------------------------------|-------------------------------|-------------------------------|----------------------------|------------------------|------------------------|-----------------------------|-------------------------|------------------------------|-----------------------------|------------------------------|
| | Controller ^④ | AuthenticateUser ^④ | InterfacePage ^④ | PageMaker ^④ | AddButton ^④ | RegisterButton ^④ | AddRequest ^④ | AddMenuOperator ^④ | MenuConnection ^④ | RedirectRequest ^④ |
| Controller ^⑤ | O ^⑥ | O ^⑥ | + | + | + | + | + | + | + | + |
| AuthenticateRequest ^⑤ | + | O ^⑥ | + | + | + | + | + | + | + | + |
| RoleChecker ^⑤ | + | O ^⑥ | + | + | + | + | + | + | + | + |
| Role ^⑤ | + | O ^⑥ | + | + | + | + | + | + | + | + |
| AddRequest ^⑤ | + | + | + | + | + | + | O ^⑥ | + | + | + |
| InterfacePage ^⑤ | + | + | O ^⑥ | + | O ^⑥ | O ^⑥ | + | + | + | + |
| PageMaker ^⑤ | + | + | + | O ^⑥ | + | + | + | + | + | + |
| AddMenuOperator ^⑤ | + | + | + | + | + | + | O ^⑥ | + | + | O ^⑥ |
| MenuConnection ^⑤ | + | + | + | + | + | + | + | O ^⑥ | O ^⑥ | + |
| RedirectRequest ^⑤ | + | + | + | + | + | + | + | O ^⑥ | O ^⑥ | + |

2.3 B-UC-9: DeleteUser

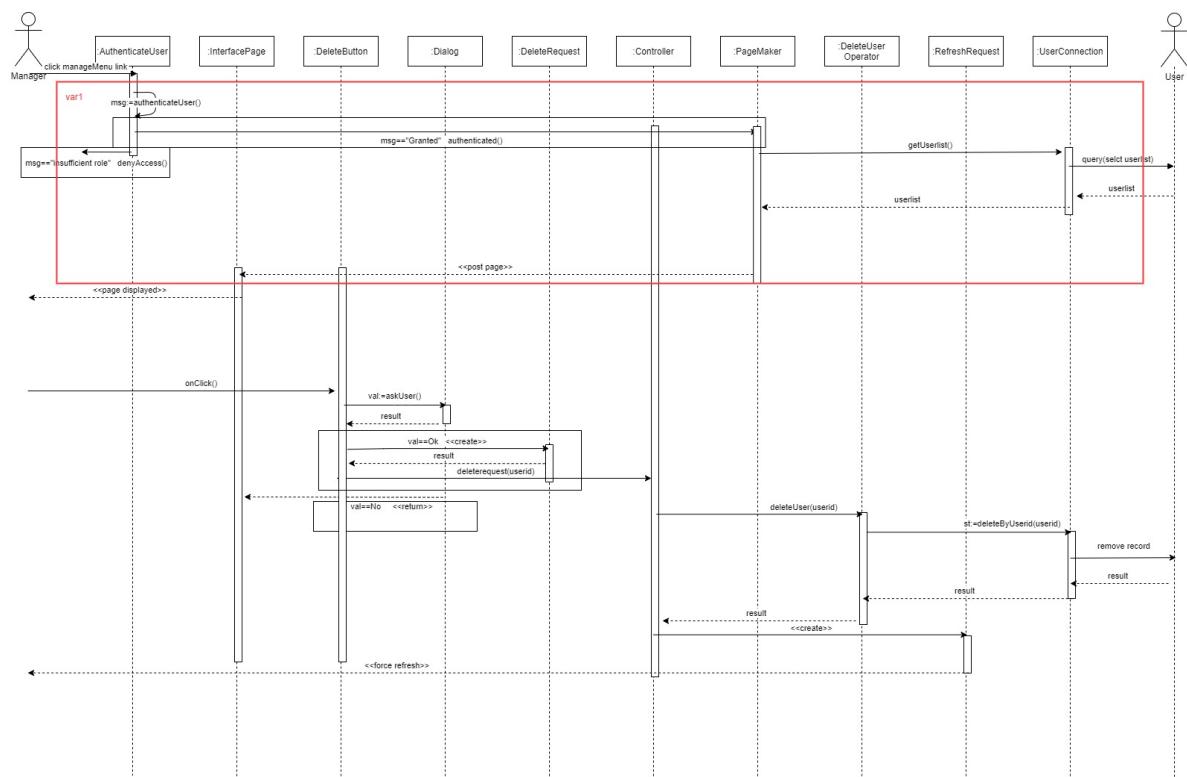


1) Basic



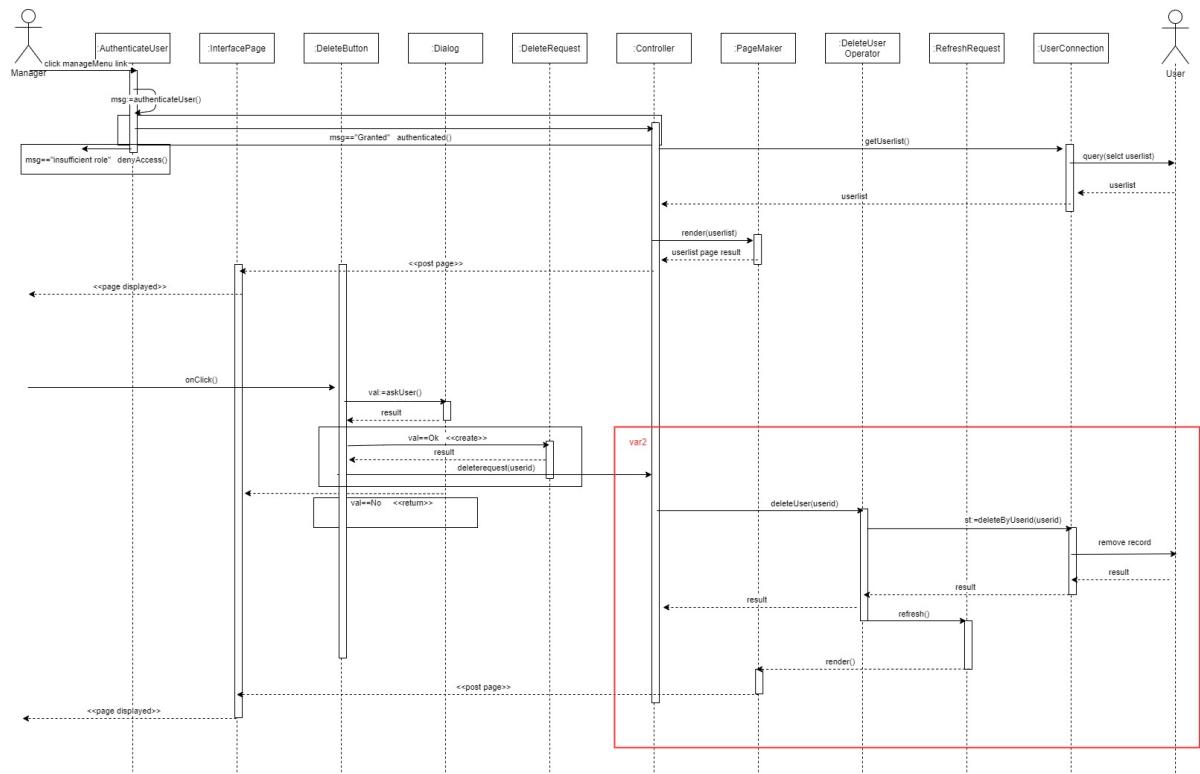
2) Variation1

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.



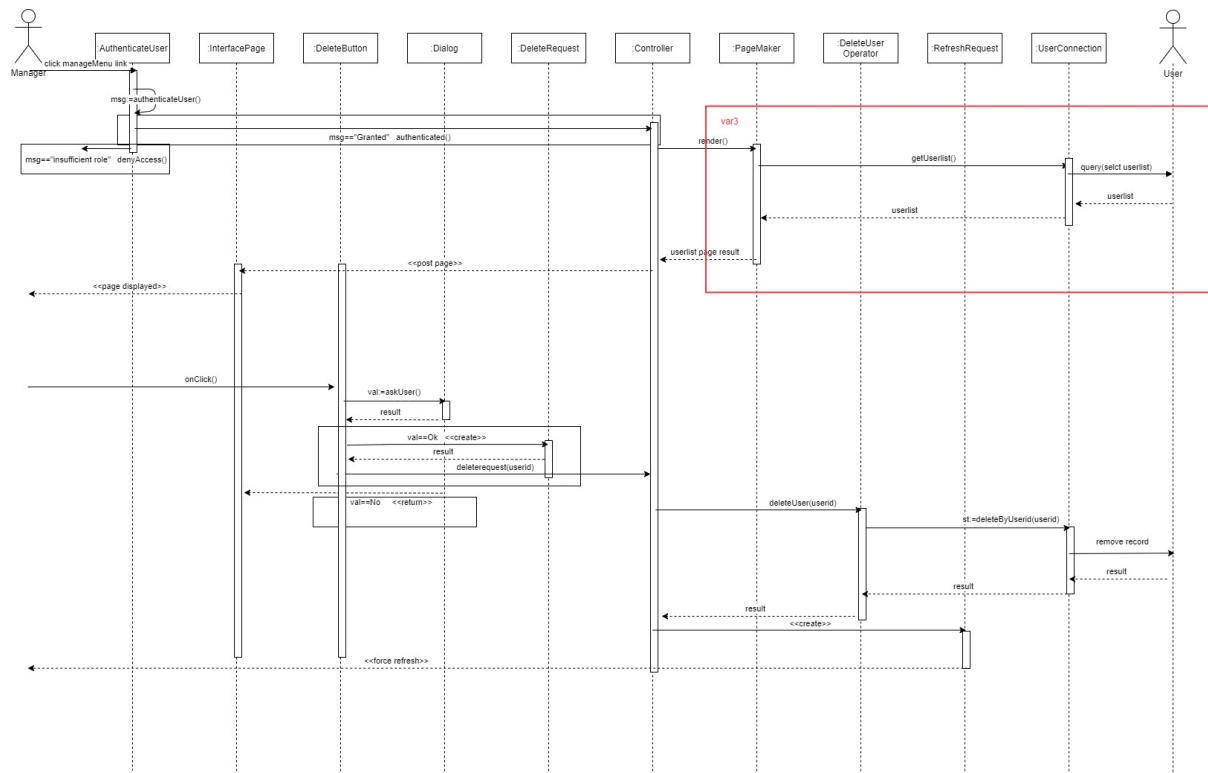
3) Variation 2

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 하도록 한다. Variation에서는 새로고침을해서 화면을 다시 보여주는 것 까지를 해당 diagram에서 해줘야하는 책임으로 보았다. Basic 모델은 보다 직관적이고 간결하다는 장점이 있고, Variation에서는 Delete User를 하고난 이후 상황까지 Controller와 PageMaker사이의 상호작용으로 페이지를 만들게하는 책임을 부여했다는 장점이 있다.



4) 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.

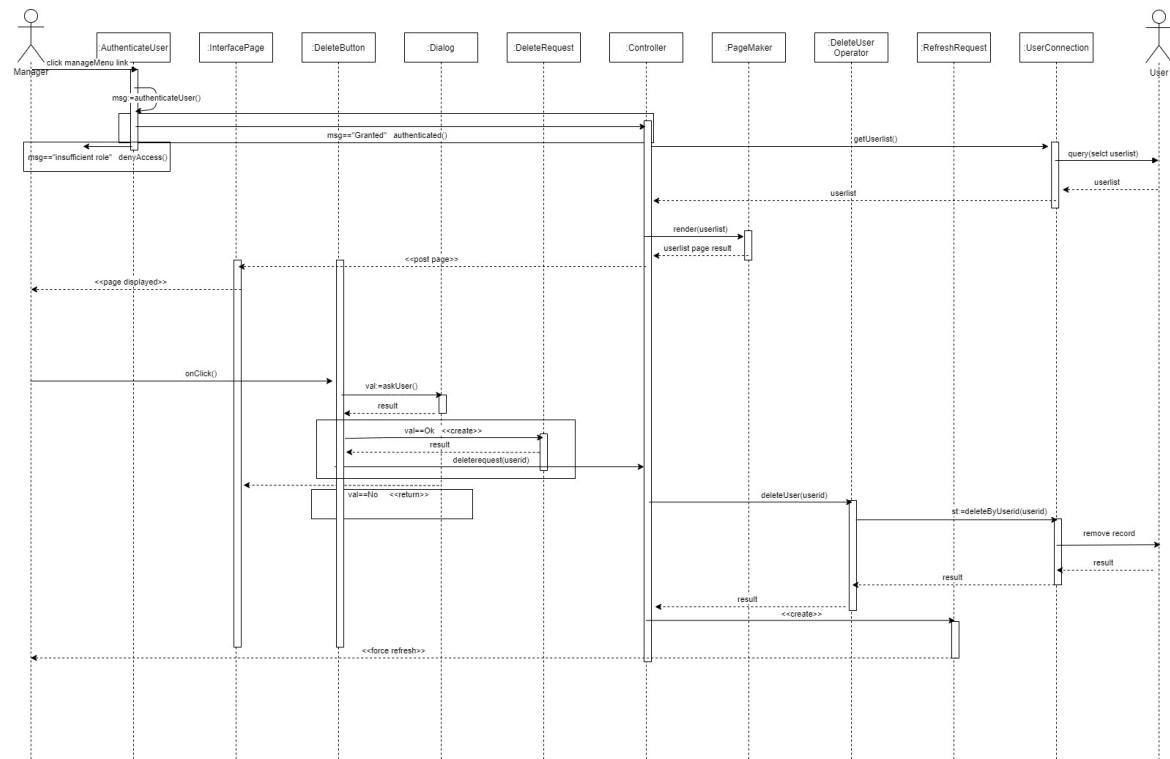


2.3.1 최종 결과 및 선정 이유

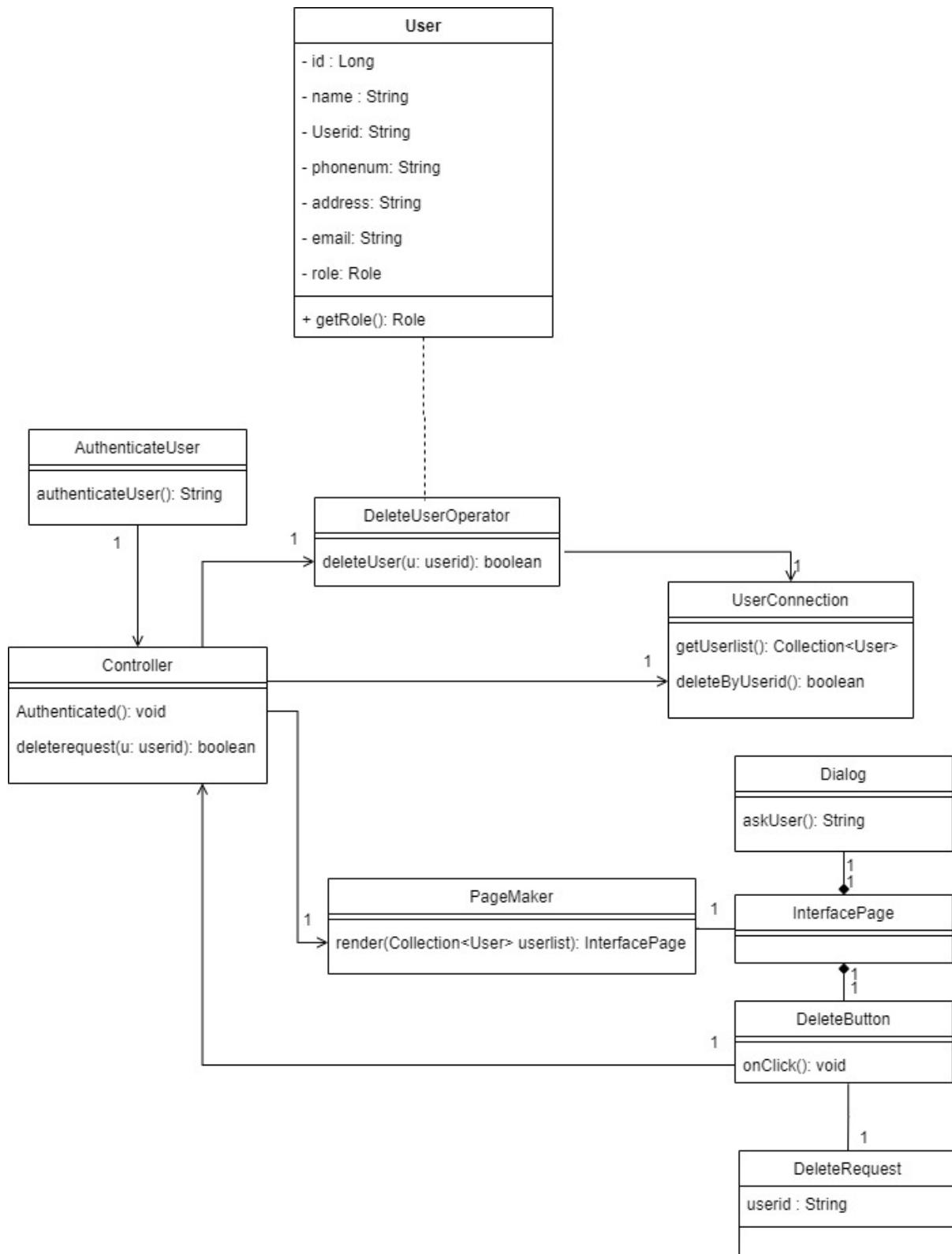
1) Object Sequence Diagram

AuthenticateUser의 인증되었다는 authenticated를 받는 object는 Controller가 하도록하고, Controller가 사용자의 리스트를 받아와서 렌더링하도록 PageMaker에게 요청하는 variation을 선택하였다. Authenticated를 받는 역할은 PageMaker보다는 Controller가 받아서 적절한 요청을 보내주는 것이 맞다고 생각했다. 또한 userlist를 받아오는 부분에서는 해당 방식을 통해 향후 다른 MenuManagement에서 이를 적용할 때 userlist 대신 menulist를 전달만 해주면 되므로 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.



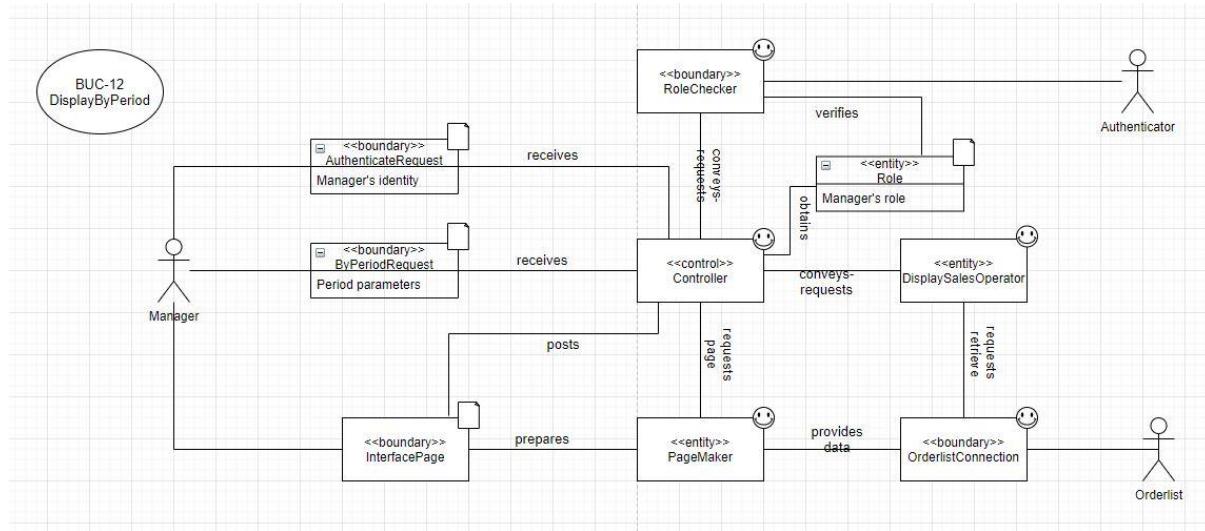
2) Class Diagram



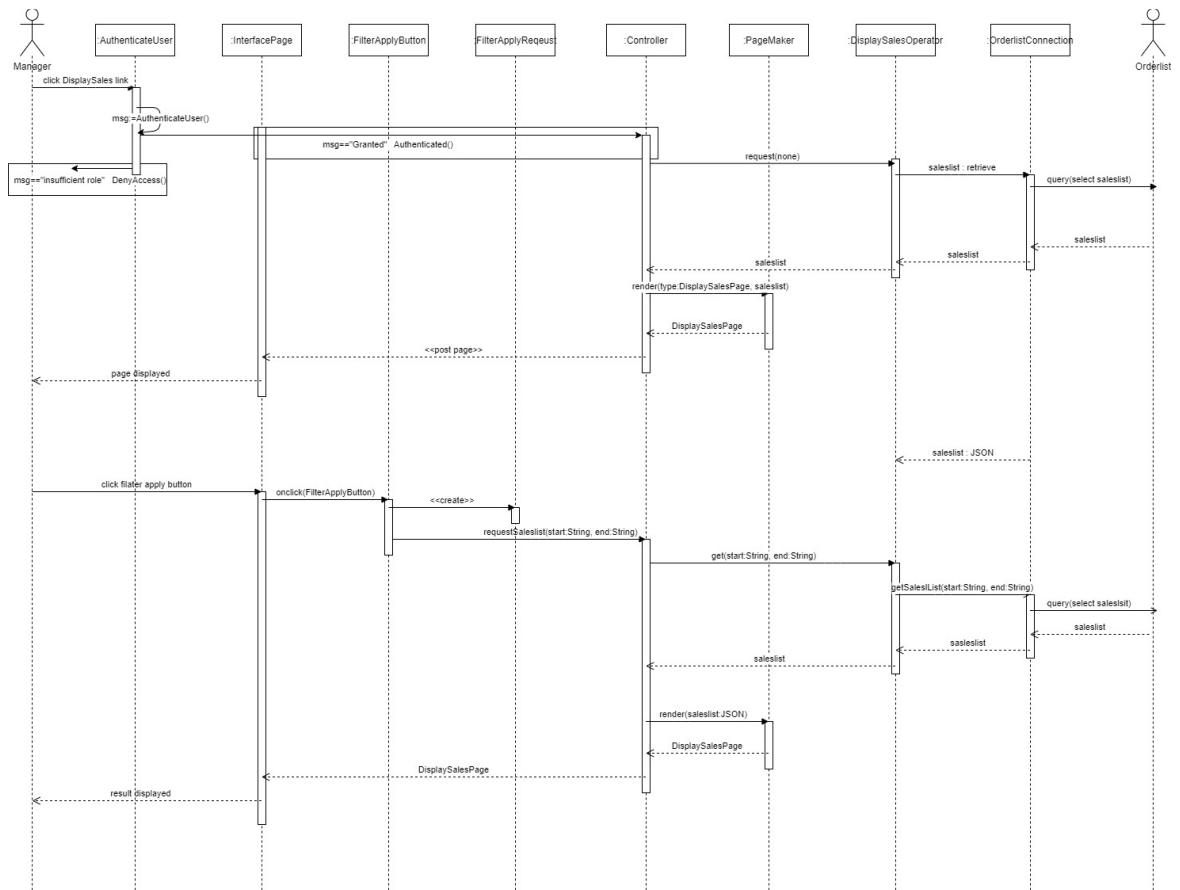
3) Traceability Matrix

| | Software classes ↴ | | | | | | | | | |
|-----------------------|--------------------|--------------------|-----------------|----------------|----------|-----------------|-------------|----------------------|------------------|------------------|
| Domain concepts ↴ | Controller ↴ | AuthenticateUser ↴ | InterfacePage ↴ | DeleteButton ↴ | Dialog ↴ | DeleteRequest ↴ | PageMaker ↴ | DeleteUserOperator ↴ | RefreshRequest ↴ | UserConnection ↴ |
| Controller ↴ | O ↴ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ |
| AuthenticateRequest ↴ | ↑ | O ↴ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ |
| RoleChecker ↴ | ↑ | O ↴ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ |
| Role ↴ | ↑ | O ↴ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ |
| DeleteRequest ↴ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ |
| DeleteUserOperator ↴ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | O ↴ | ↑ | ↑ |
| RefreshRequest ↴ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | O ↴ | ↑ |
| InterfacePage ↴ | ↑ | ↑ | O ↴ | O ↴ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ |
| PageMaker ↴ | ↑ | ↑ | O ↴ | O ↴ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ |
| UserConnection ↴ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | O ↴ | ↑ | ↑ | O ↴ |

2.4 DisplayByPeriod

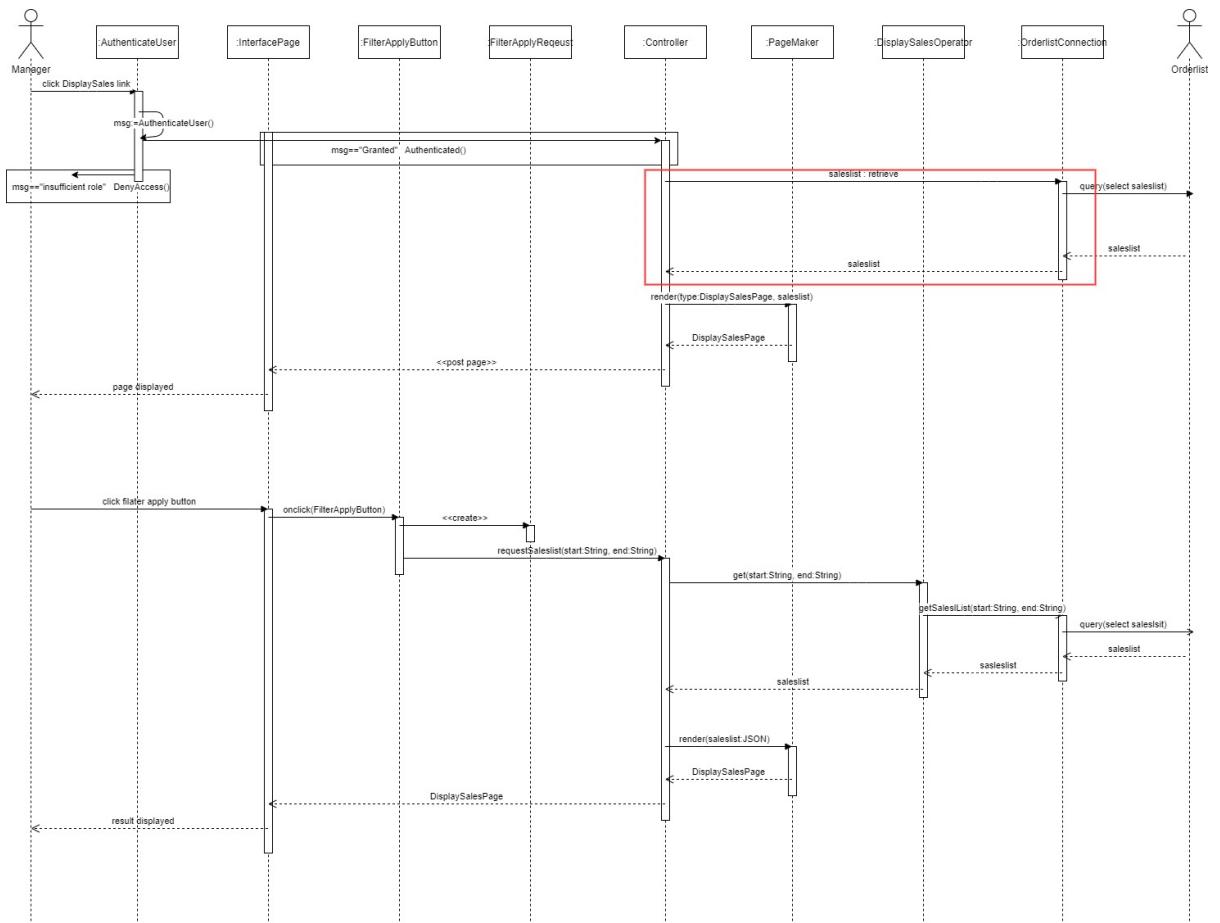


1) Basic



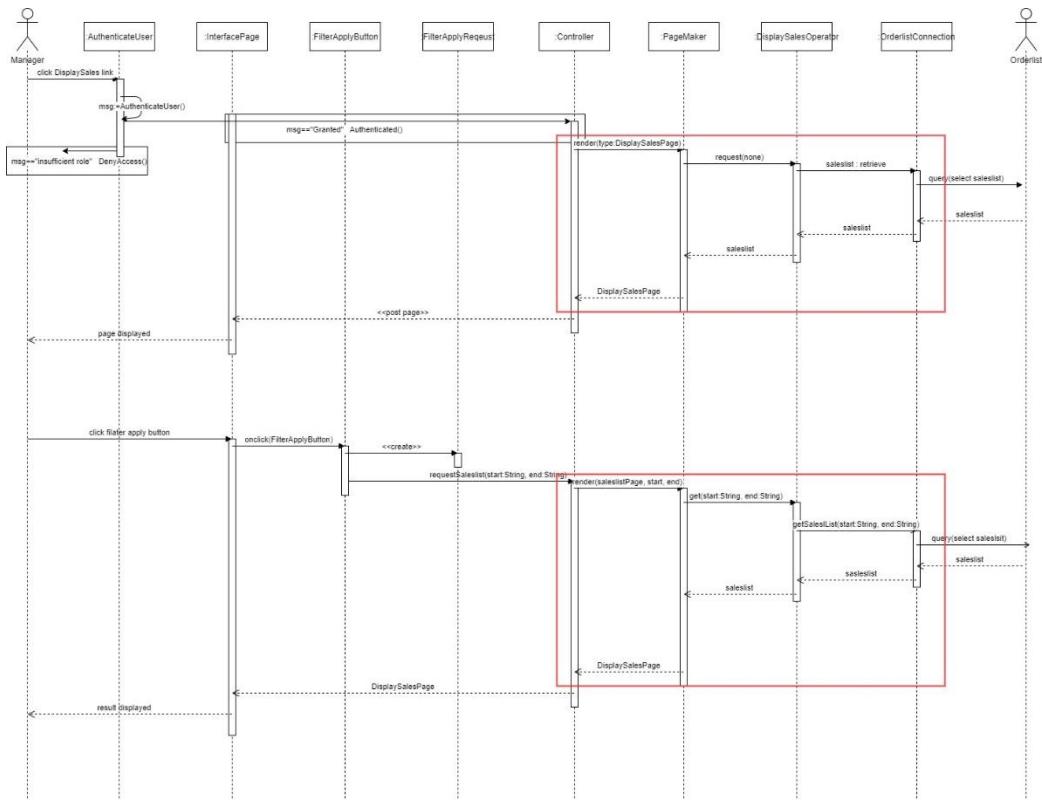
2) Variation 1

메뉴 리스트를 불러오는 것과 마찬가지로 DisplaySalesOperator의 responsibility에 대한 variation이다. 초기에 판매 리스트를 불러오는 과정에서 Controller가 직접적으로 OrderlistConnection에 요청을 한다. 반면, Basic sequence에서는 DisplaySalesOperator를 한 번 거쳐서 요청이 들어간다.



3) Variation 2

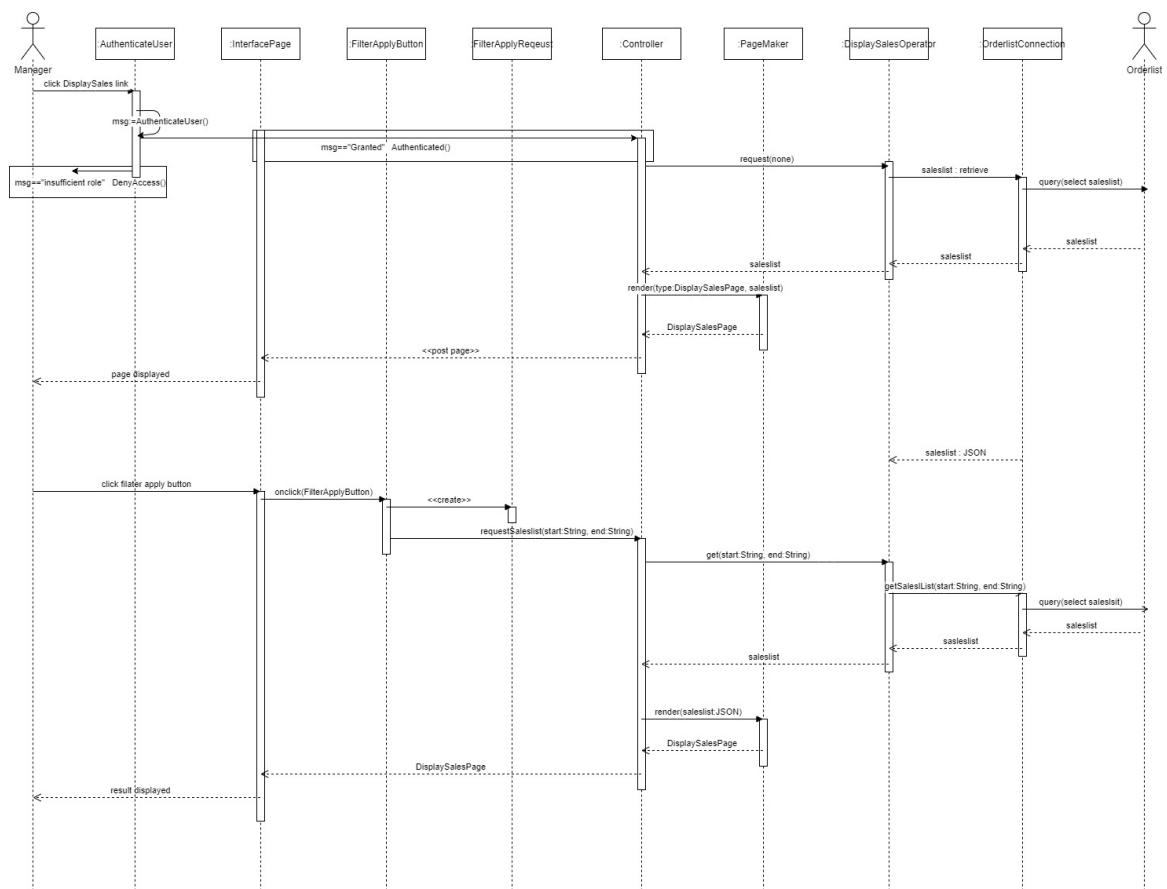
PageMaker의 responsibility에 대한 variation이다. 위에서 언급한 AddMenu variation 1과 마찬가지로 Controller가 PageMaker에게 특정 페이지의 렌더링을 요청하면 PageMaker가 렌더링에 필요한 모든 과정을 맡아서 수행한다.



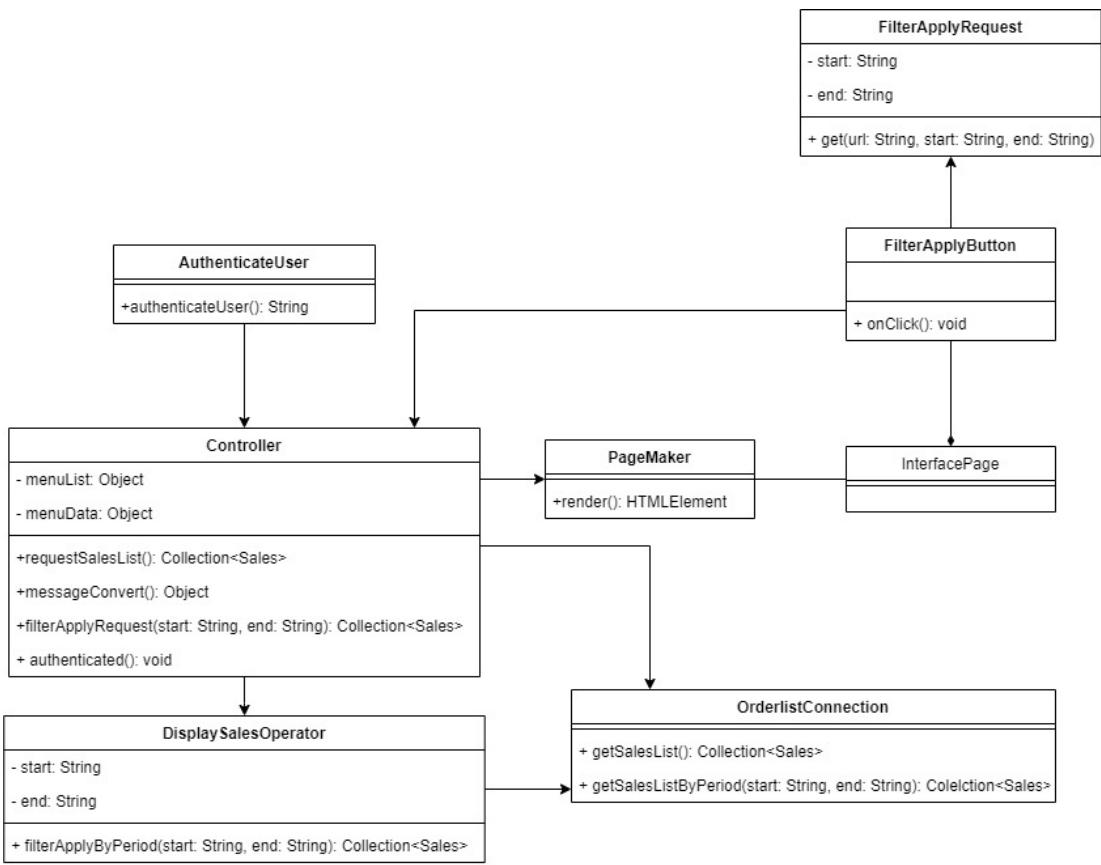
2.4.1 최종 결과 및 선정 이유

1) Object Sequence Diagram

기간에 따라 판매 리스트를 불러오는 작업은 판매 리스트를 불러오는 작업의 확장이다. 따라서, 초기 판매 리스트를 불러오는 작업을 `DisplaySalesOperator`를 거치는 것으로 정하였다. 또한, 렌더링에 필요한 작업을 `PageMaker`에게 맡기면 `PageMaker` 하나에 과도한 responsibility가 들어간다. 따라서, 필요한 데이터는 `Controller`가 받아와 `PageMaker`가 이를 통해 렌더링만을 하는 것으로 구상하였다.



2) Class Diagram

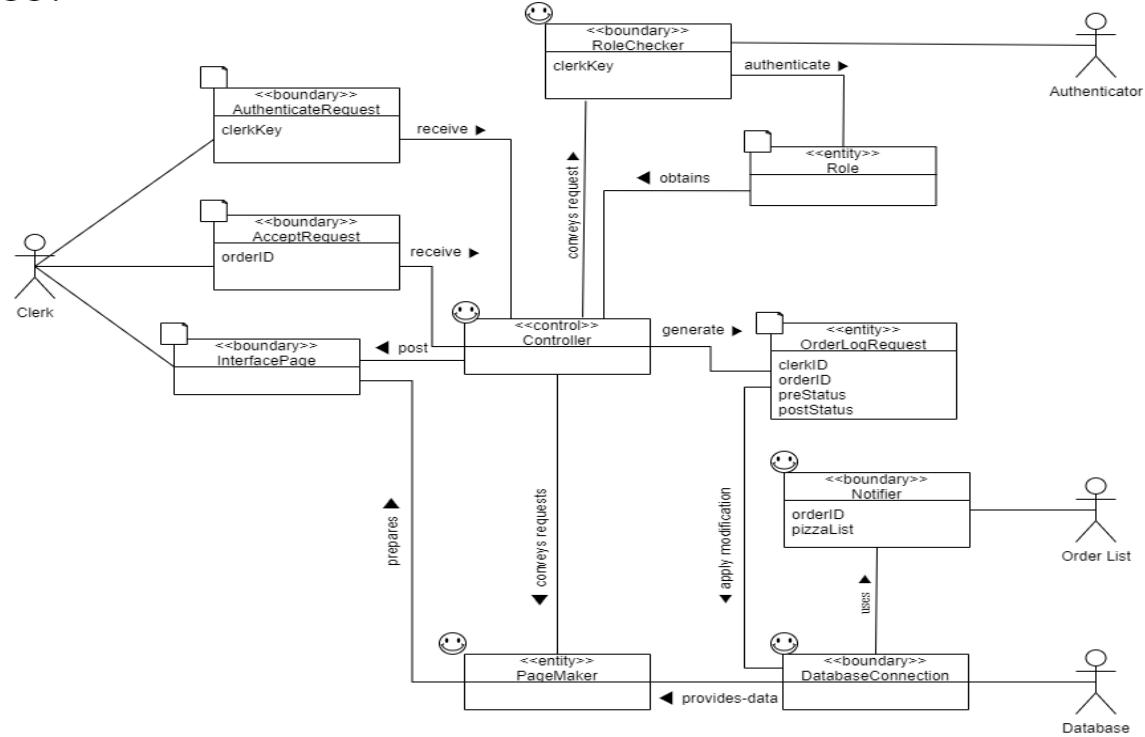


3) Traceability Matrix

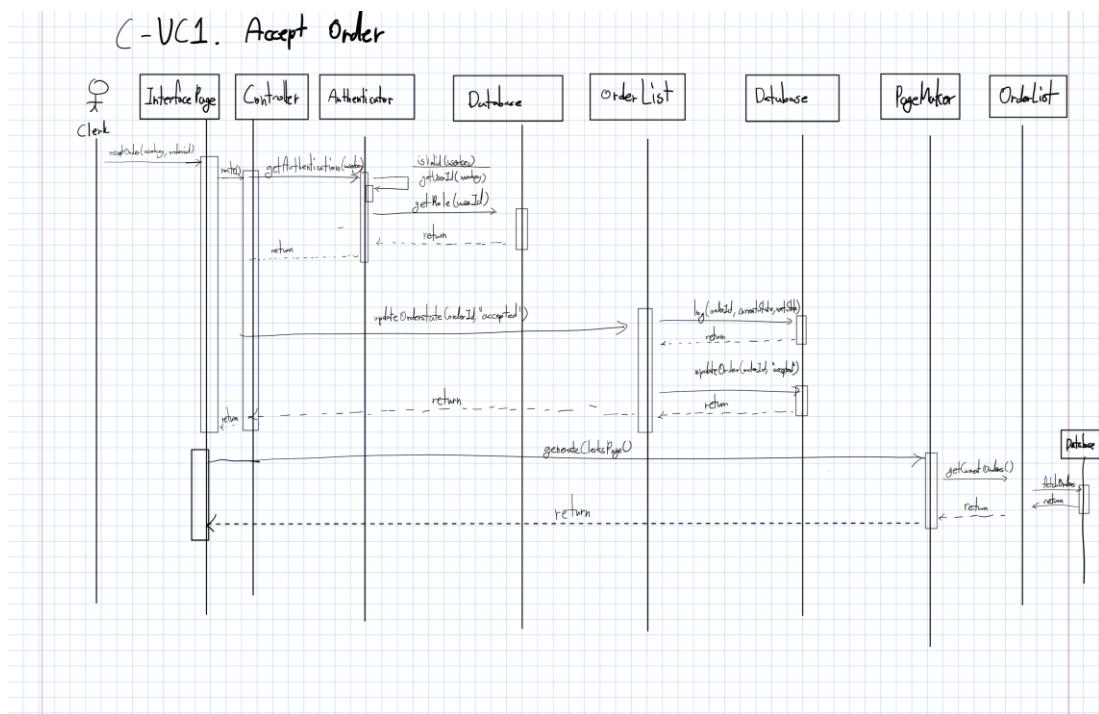
3. SubgroupC(OrderStatus)

3.1 C-UC-1 Accept Order

C-UC1



1) Variation1



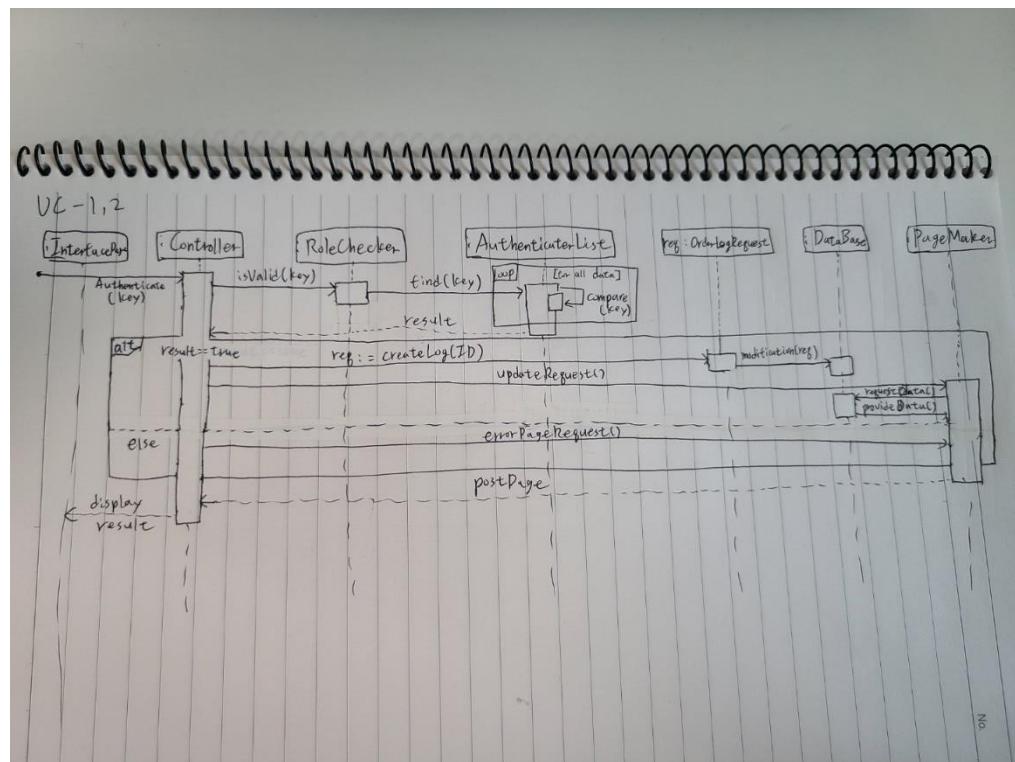
어떤 user로부터 주문을 accept하겠다는 요청이 들어오면, 우선 해당 user의 userkey를 받아서 Authenticator에서 이 user가 Clerk인지를 체크한다. Authenticator 내부에서 userkey를 통해 userID를 찾고, Database에서 해당 userID와 맵핑되는 role을 받아와서 이 role이 Clerk인지를 확인한다.

Clerk이 맞다면, orderID와 이 주문이 accept 되었다는 것을 OrderList에 전해주고, OrderList는 log와 updateOrder를 통해 log를 데이터베이스에 남긴 후에 실질적인 주문현황을 최신화해 준다.

이러한 작업이 모두 끝나면, InterfacePage에서 PageMaker에 새로운 페이지를 요청하고, OrderList, Database를 거쳐 가장 최근에 업데이트 된 주문현황을 받아와 페이지를 재구성한다.

하지만 이는 InterfacePage에 너무 많은 정보가 노출되고, InterfacePage는 Thing에 해당하는 객체이기 때문에 일을 직접적으로 하는 것이 아닌 전달요소로만 쓰여야한다. 따라서 이러한 페이지 요청이 Controller가 요청하는 것으로 되어야 한다.

2) Variation2



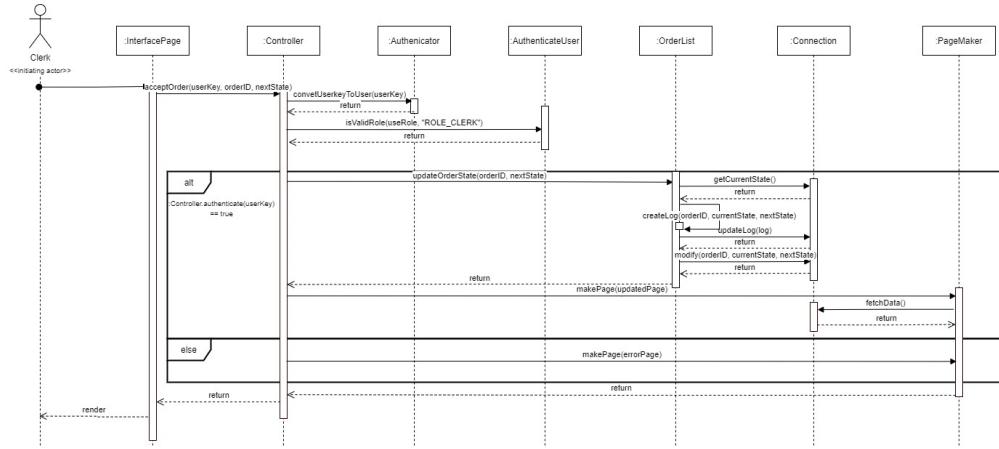
accept 요청이 들어오면, 이때 user의 key를 같이 받아와 RoleChecker에게 전달해준다. RoleChecker는 AuthenticaterList에서 전달받은 key에 맵핑된 role이 accept 작업을 수행할 수 있는지 compare한 후, 이를 True/False로 전달해준다.

True일 경우, Log를 만들어 이를 Database에 업데이트 해준다. 이후 PageMaker에 페이지를 업데이트 해줄 것을 요청하고, PageMaker는 필요한 정보를 Database에서 받아와 업데이트를 해준다. False일 경우, PageMaker는 Database에서 아무것도 받지 않고 에러창을 생성하여 띄워준다.

응답에 따라 분기를 하는 것이 잘 표현되어 있지만, return이 없는 메서드가 존재하는 등의 문제가 있다. 또한 메서드명이 불분명하여 이를 재명명하는 것이 필요하다.

3.1.1 최종 결과 및 선정 이유

1) Object Sequence Diagram



Variation2를 base로 채택하였다.

Variation2에서 isValid()같은 역할이 모호한 함수를 수정하고 return이 없는 함수들에 return 추가하였다.

사용자가 주문을 수락할 권한이 있는가를 판단할 때, Authenticator가 userKey에 대응되는 UserRole을 생성하여 UserRole이 ROLE_CLERK과 일치하는지 비교하도록 변경하였다.

currentState를 Clerk이 상태를 변화시킬 때 전달할 필요가 없기 때문에, 요청에서 이러한 내용이 빠지고, currentState를 Connection로부터 가져오는 부분이 추가되었다. 또한 OrderList가 Log를 남기는 것과 상태를 업데이트 하는 것이 구체화되었다.

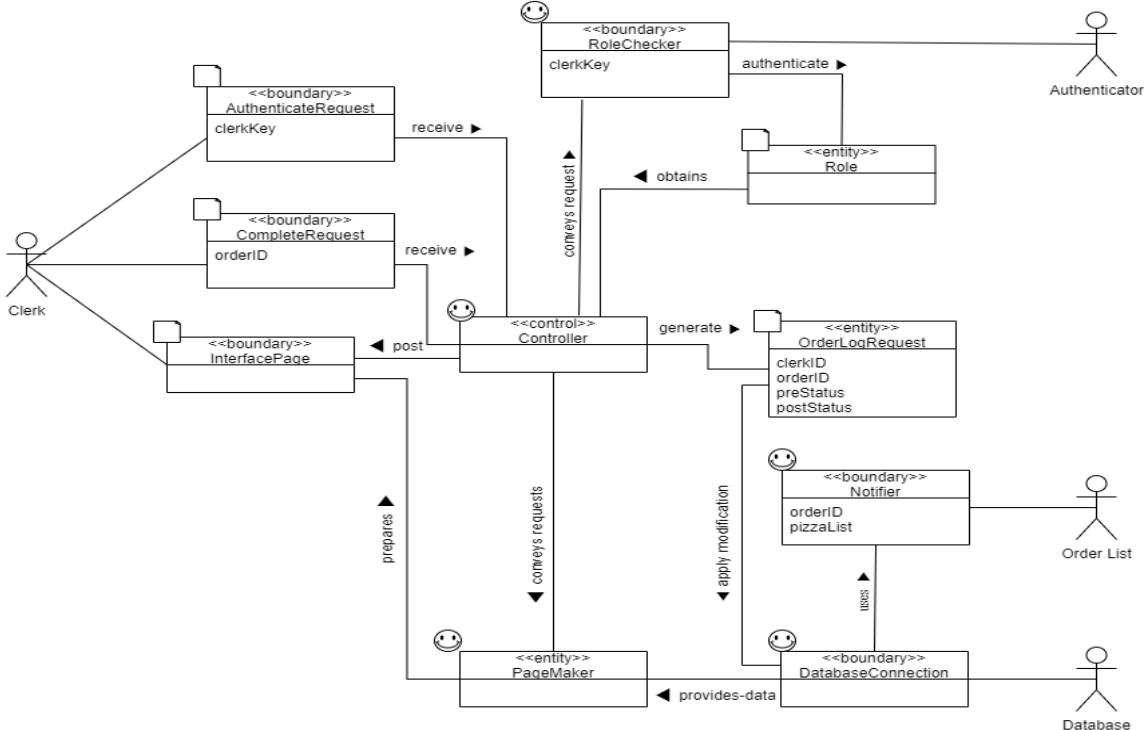
InterfacePage는 정보를 제공만 할 뿐 어떠한 행동을 하는 Responsibility는 Controller로 옮겼다.

2) Traceability Matrix

| Domain Concepts | Software class | | | | | | |
|-----------------|----------------|------------|---------------|------------------|-----------|------------|-----------|
| | InterfacePage | Controller | Authenticator | AuthenticateUser | OrderList | Connection | PageMaker |
| InterfacePage | O | | | | | | |
| Controller | | O | | | | | |
| Authenticator | | | O | O | | | |
| OrderList | | | | | O | O | O |
| PageMaker | | | | | | | O |

3.2 C-UC-2 Complete Cook

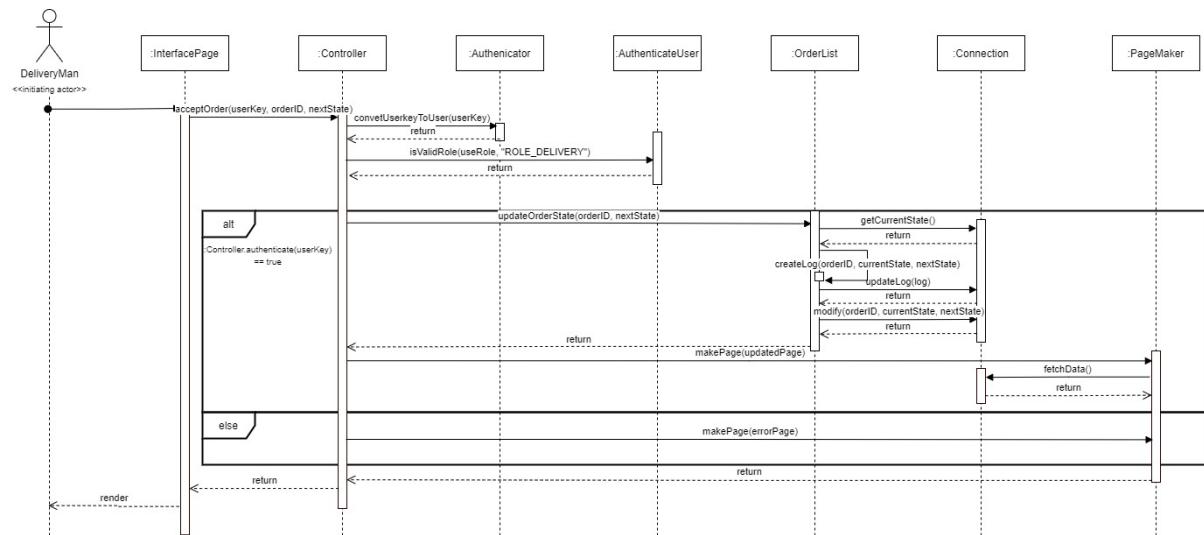
C-UC2



3.2.1 최종 결과 및 선정 이유

1) Object Sequence Diagram

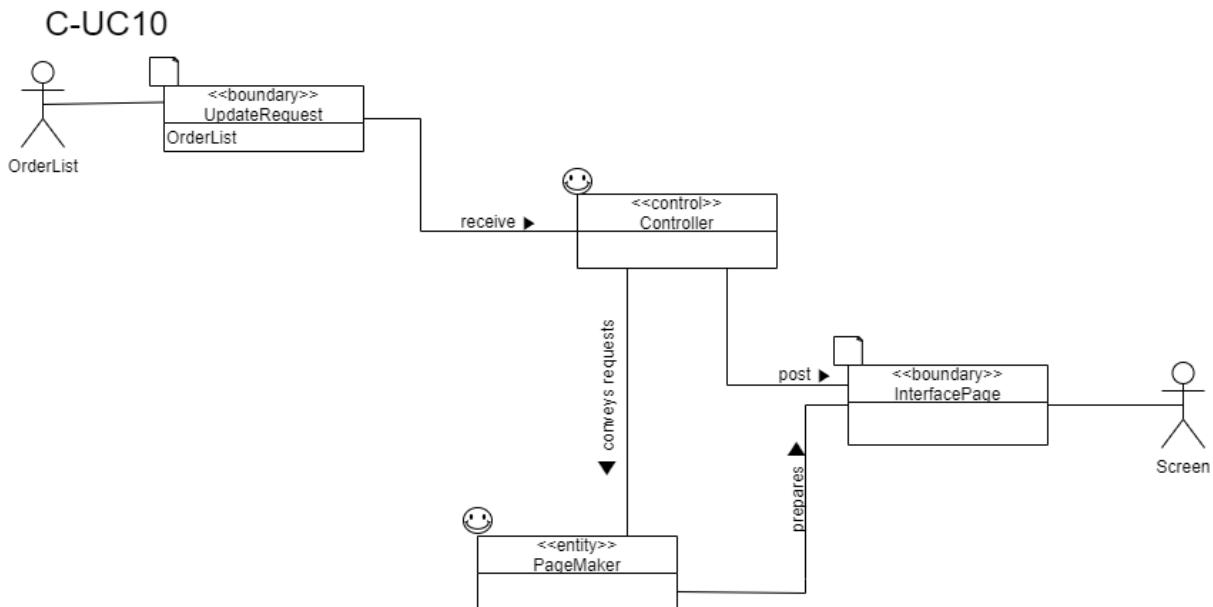
Initiating actor가 Clerk에서 DeliveryMan으로 바뀌는 것을 제외하면, C-UC-1과 전부 동일하다.



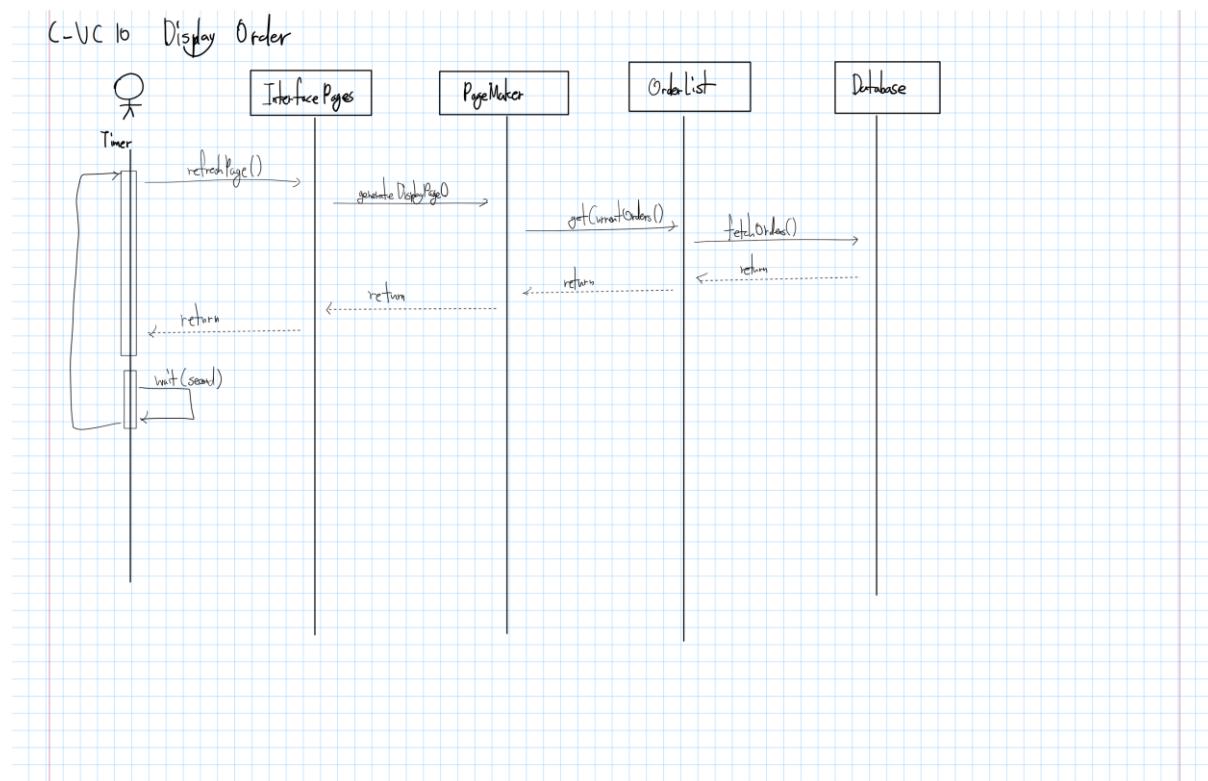
2) Traceability Matrix

| Domain Concepts | Software class | | | | | | |
|-----------------|----------------|------------|---------------|------------------|-----------|------------|-----------|
| | InterfacePage | Controller | Authenticator | AuthenticateUser | OrderList | Connection | PageMaker |
| InterfacePage | O | | | | | | |
| Controller | | O | | | | | |
| Authenticator | | | O | O | | | |
| OrderList | | | | | O | O | O |
| PageMaker | | | | | | | O |

3.3 C-UC-10 Display Order



1) Variation1



매장에 설치되어 있는 Screen에 대한 usecase이므로, 주기적인 업데이트가 필요하다고 판단되어 Initiating actor를 Timer를 만들어 설정하였다.

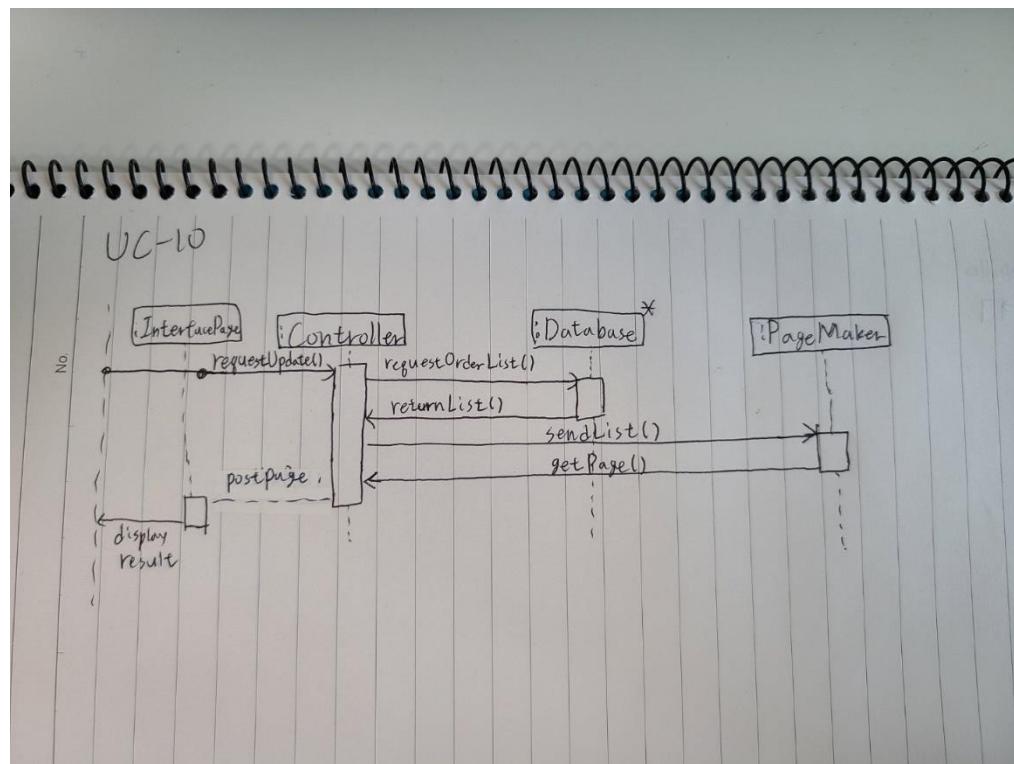
Timer가 refreshPage()를 호출하면, PageMaker에서 OrderList로, OrderList에서 Database로 이동하며 화면을 만드는데 필요한 정보들을 요청하고, return된 정보들을 통해 PageMaker에서

새로운 페이지를 만들어 업데이트해준다.

이 작업이 끝나면, 주기적으로 업데이트를 해주기 위해, Timer에서 약간의 시간을 wait()한 이후 위의 과정을 반복한다.

이 그림에서 Timer는 어떻게 시작되며, 언제 종료되는지가 불명확하다. 또한 이렇게 생성된 InterfacePage를 최종적으로 누구에게 제공하는지가 없다.

2) Variation2



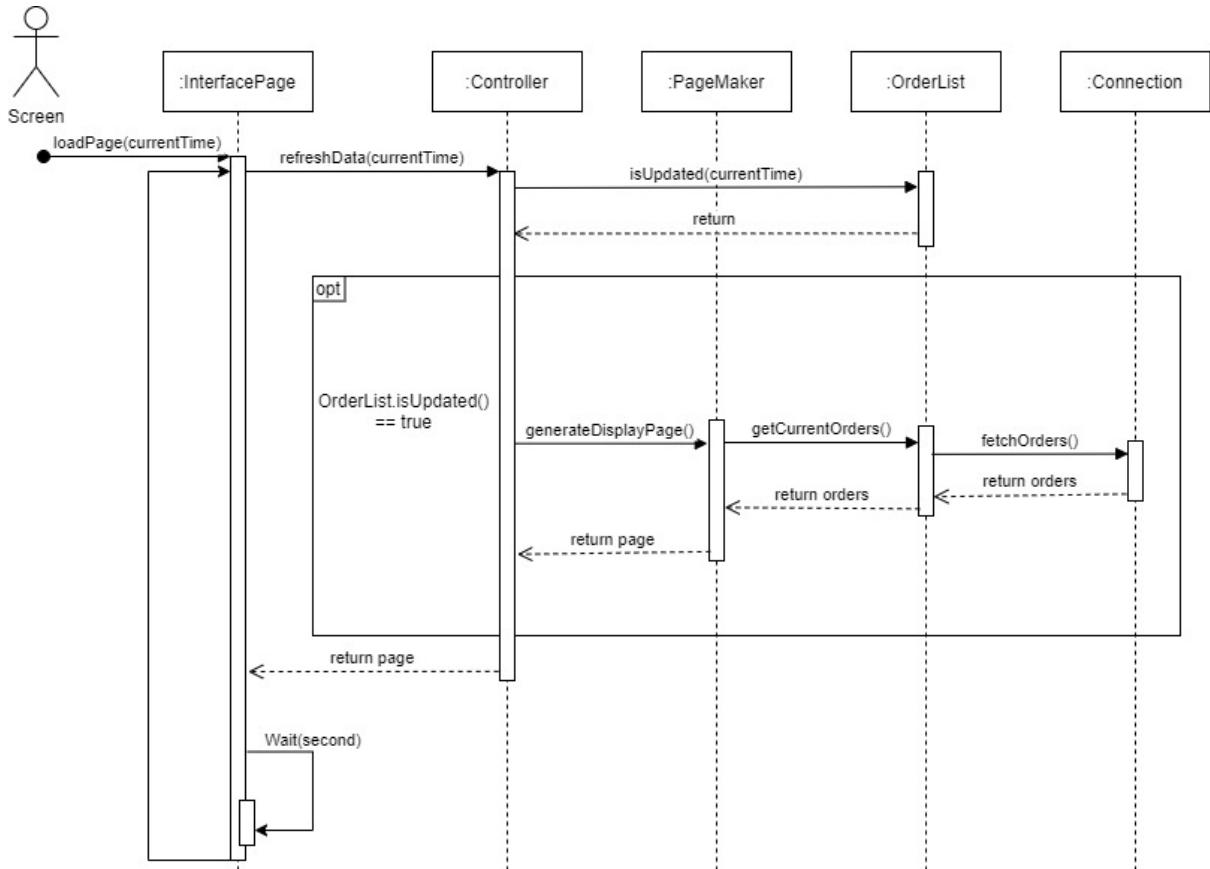
Screen이 Controller에게 페이지를 업데이트할 것을 요청하고, Controller는 페이지를 구성하는데 필요한 정보를 Database에서 가져와 이를 PageMaker에 전달한다.

PageMaker는 이를 바탕으로 새로운 페이지를 생성하여 InterfacePage에 전달해준다.

이를 Screen에 띄우는 것으로 추정되는데 이것이 명확하게 표현되지 않고 있고, 주기적으로 update가 되어야하는 페이지임에도 불구하고 이를 명시한 흐름을 찾아보기 어렵다.

3.3.1 최종 결과 및 선정 이유

1) Object Sequence Diagram



Variation1을 base로 채택하였다.

`orderList`가 업데이트 될 때마다 혹은 주기적인 시간마다 `Screen`이 업데이트 되어야 하는데, Variation2에서는 이러한 사항이 배제되어 있다.

또한 Variation2는 cohesion을 과도하게 의식하여 전체적인 flow가 비효율적이고 `Controller`가 unrelated한 responsibility를 가지고 있다고 판단하였다.

Variation1의 전체적인 flow를 `OrderList.isUpdated() == true`인 부분에 사용하였고, 기존에 있던 usecase 설계를 반영하여 actor를 `Screen`으로 변경하고 `Controller object`를 추가하였다.

`Controller`는 `refreshData()`를 통해 `currentTime`을 받아오고, `OrderList`는 `isUpdated()`를 통해 `currentTime` 이후에 `orderList`에 변동이 생겨 페이지 업데이트가 필요한지 여부를 판단한다.

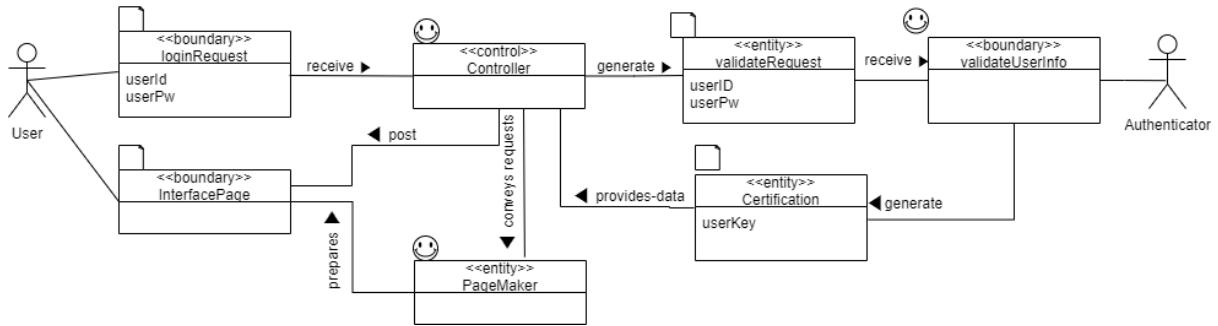
`Controller`가 `returnPage()`를 하면 `InterfacePage`는 `wait()`을 한 후, 다시 `loadPage()`를 호출하도록 하여 주기적으로 화면 업데이트를 요청하도록 하였다.

2) Traceability Matrix

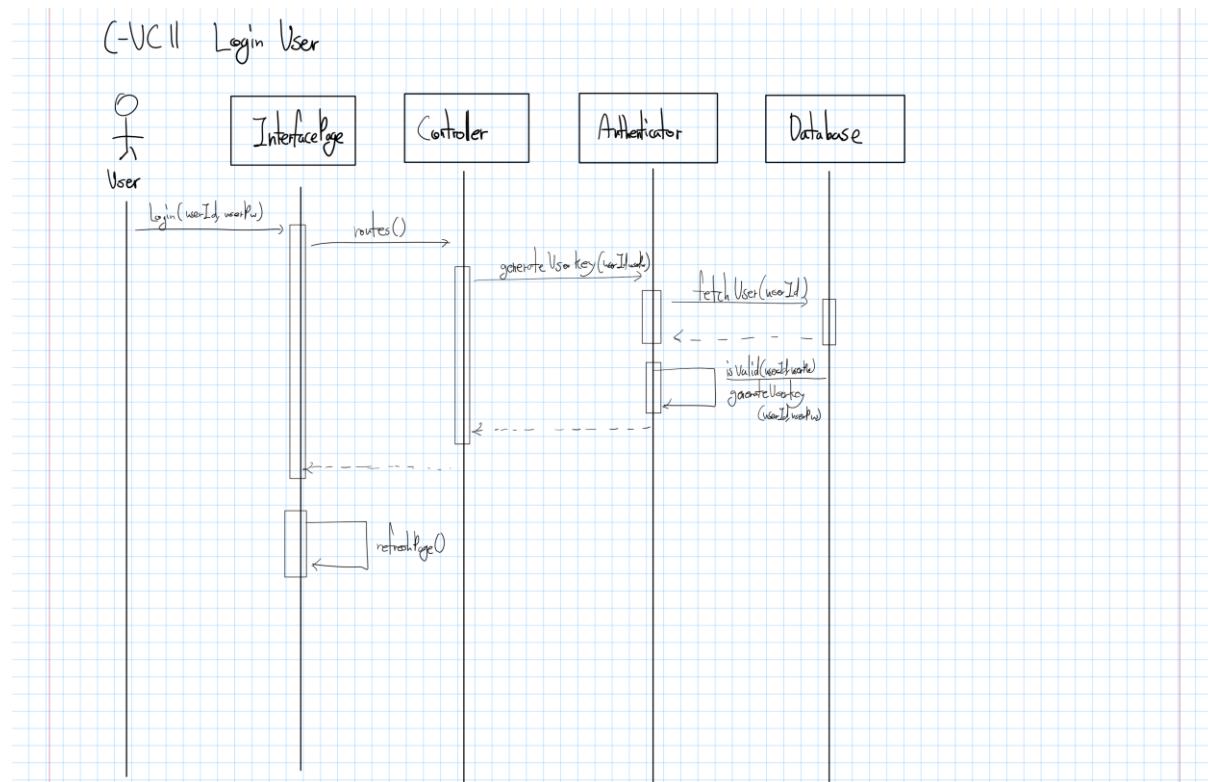
| Domain Concepts | Software class | | | | |
|-----------------|----------------|------------|-----------|------------|-----------|
| | InterfacePage | Controller | OrderList | Connection | PageMaker |
| InterfacePage | O | | | | |
| Controller | | O | | | |
| OrderList | | | O | O | |
| PageMaker | | | | | O |

3.4 C-UC-11 Login User

C-UC11



1) Variation1

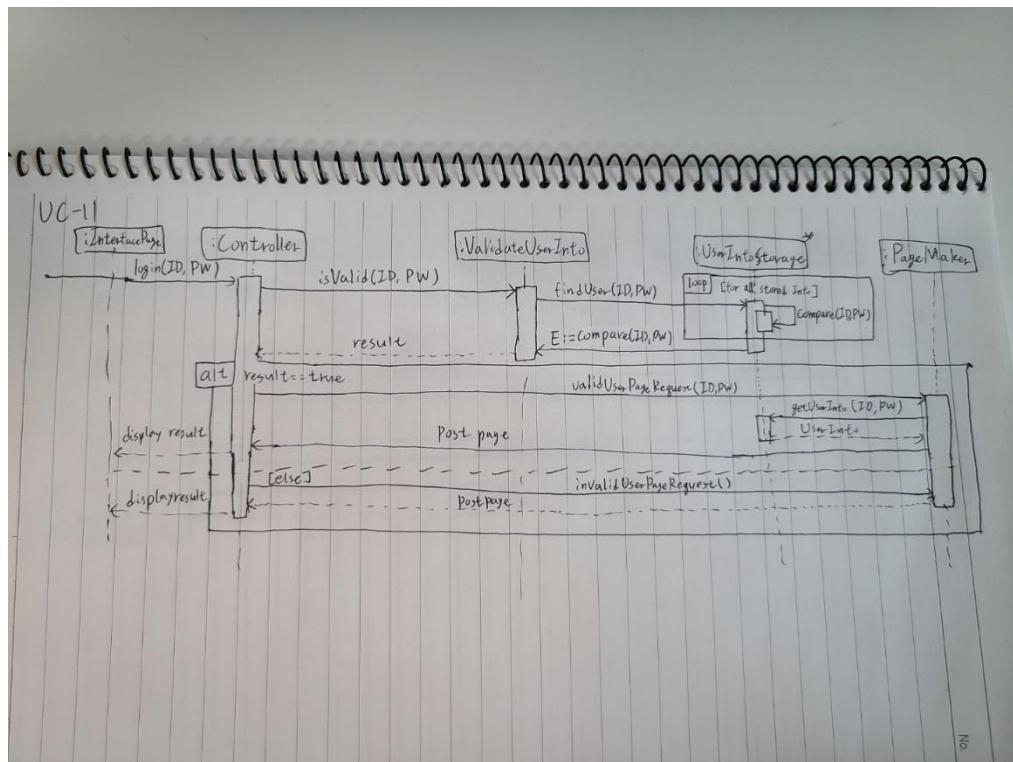


User의 로그인 시도로부터 userID와 userPW를 받아 이를 Controller에 전달한다. Authenticator는 generateUserKey()를 통해 userID와 userPW를 Controller로부터 받아오고, fetchUser()를 통해 Database에서 userID에 대한 userPW를 받아온다.

입력받은 userID, userPW와 Database에서 가져온 userPW를 통해 isValid()를 호출하여 valid한 로그인 시도인지 판단하고, valid할 경우 generateUserKey()로 userKey를 만들고 이를 return해준다.

return0| InterfacePages까지 전달되면, refreshPage()를 통해 로그인 시도 성공 여부를 창에 표시해준다.

2) Variation2



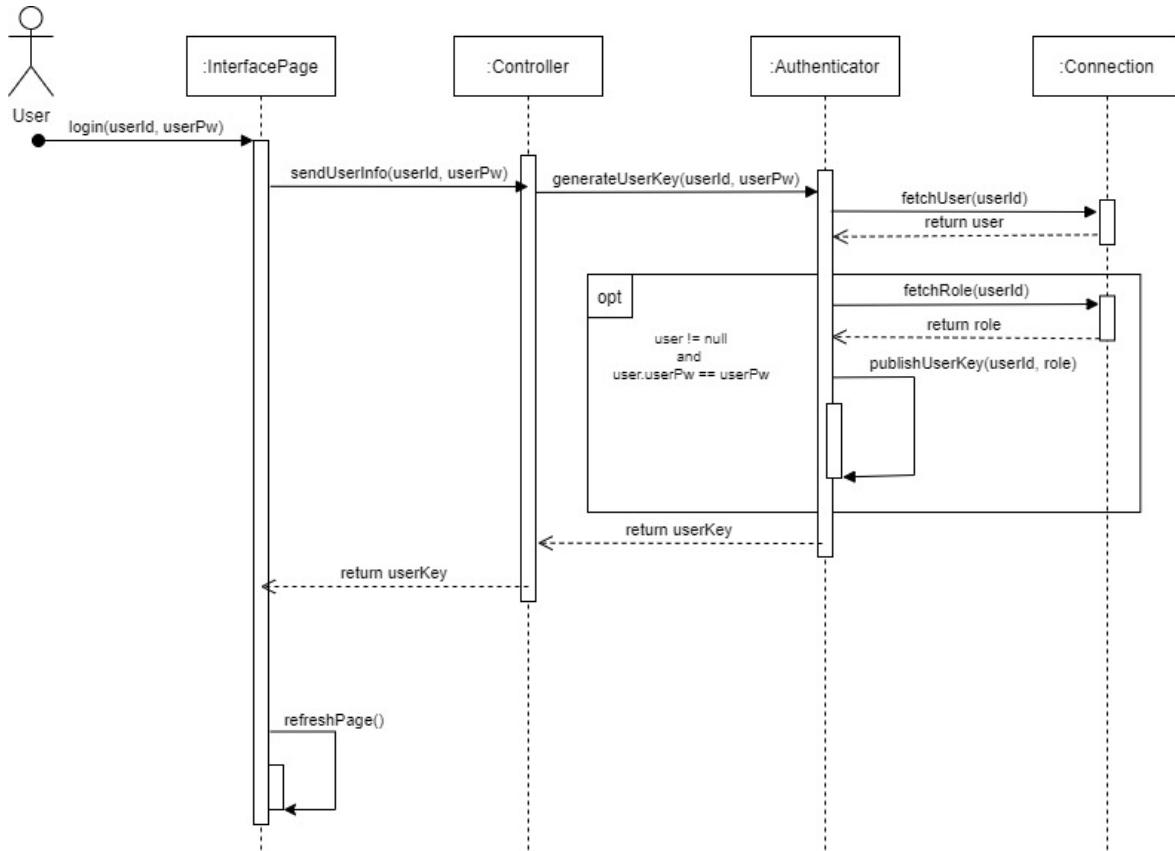
User가 로그인을 하면 Controller가 ID와 PW를 ValidateUserInfo로 전달한다. ValidateUserInfo는 UserInfoStorage에서 전달받은 ID, PW가 있는지 확인하여 로그인 가능 여부를 판단한다.

만약 compare()의 결과가 True라면, PageMaker는 getUserInfo()를 통해 UserInfoStorage에서 해당 유저의 정보를 받아와 페이지를 만들어준다.

False라면 PageMaker는 바로 에러창을 만들어준다.

3.4.1 최종 결과 및 선정 이유

1) Object Sequence Diagram



Variation1을 base로 채택하였다.

일반적인 웹페이지에서 로그인에 성공 또는 실패한 경우를 생각해봤을 때, PageMaker라는 object를 넣어 새로운 페이지를 만드는 것이 불필요하다고 생각되었다.

그리고 Variation2처럼 `userID`와 `userPW`를 변환하지 않고 object들이 계속 가지고 있는 것보다 Variation1처럼 `userID`와 `userPW`를 `userKey`로 변환하여 로그인의 valid 여부를 판단하는 것이 domain model을 더 잘 반영한다고 판단하였다.

2) Traceability Matrix

| Domain Concepts | Software class | | | |
|-----------------|----------------|------------|---------------|------------|
| | InterfacePage | Controller | Authenticator | Connection |
| InterfacePage | O | | | |
| Controller | | O | | |
| Authenticator | | | O | |
| PageMaker | | | | O |

3.5 Class Diagram

