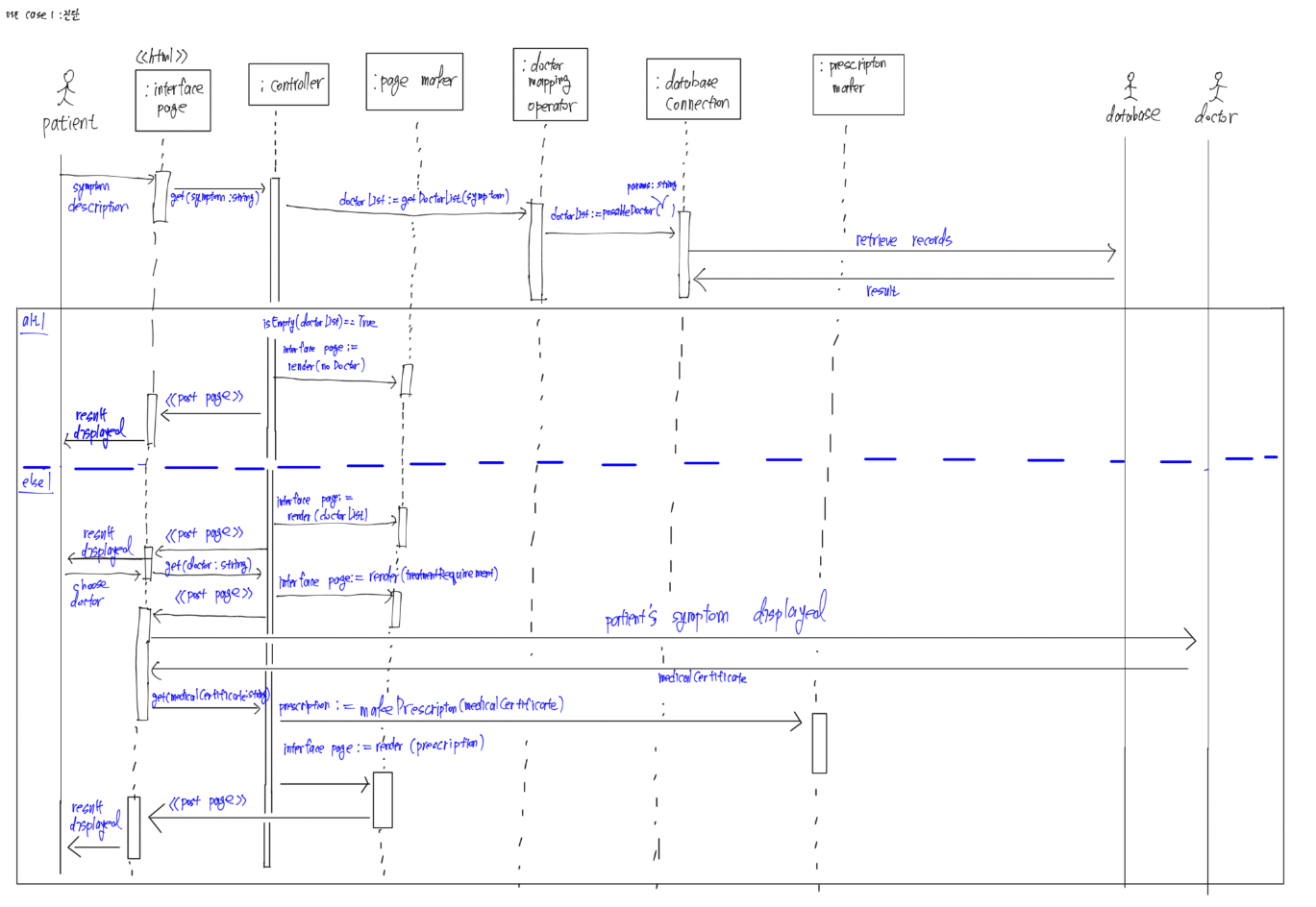
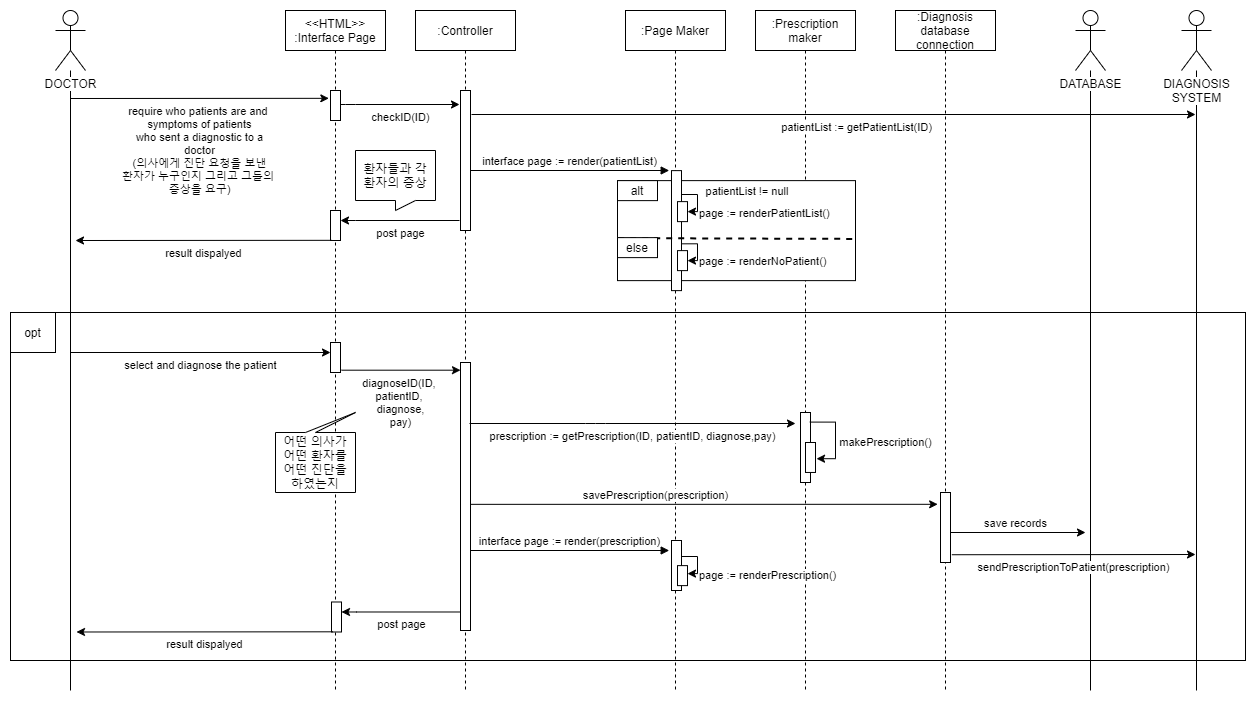
**Doctor Sequence Diagram & Class Diagram**

**홍주표, 이한별**

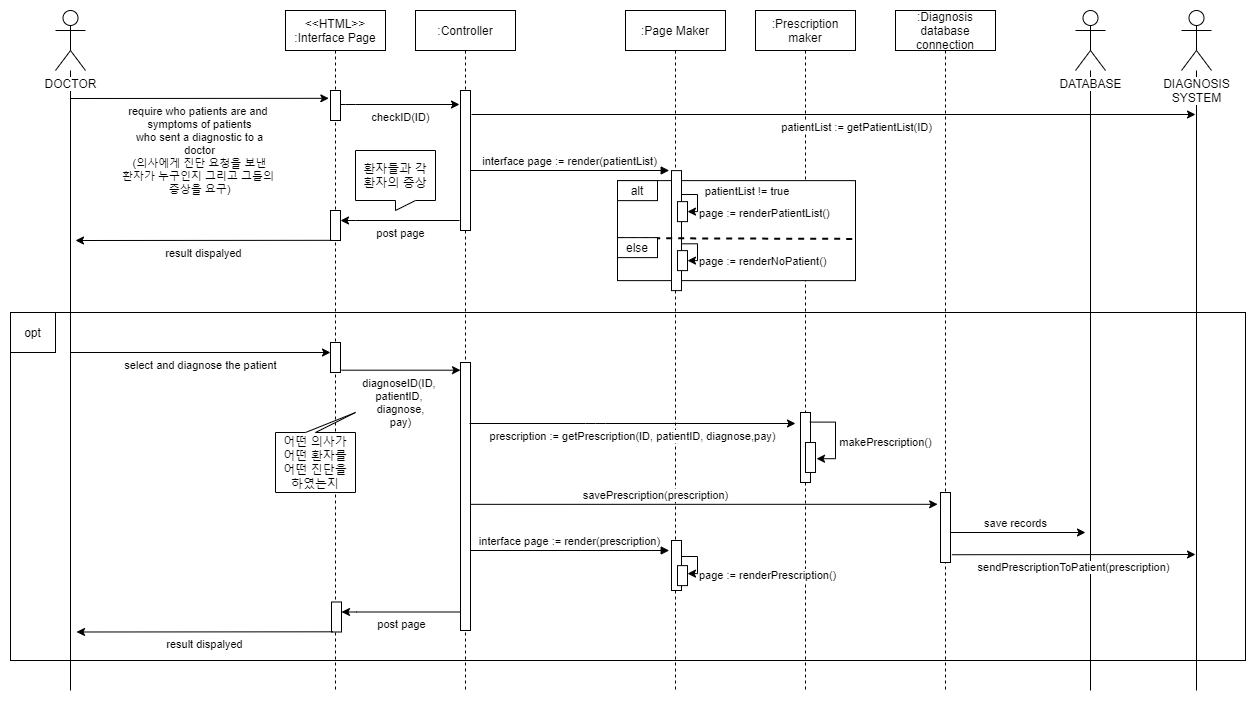
* **UC1-1 ; 진단(의사 부분)**



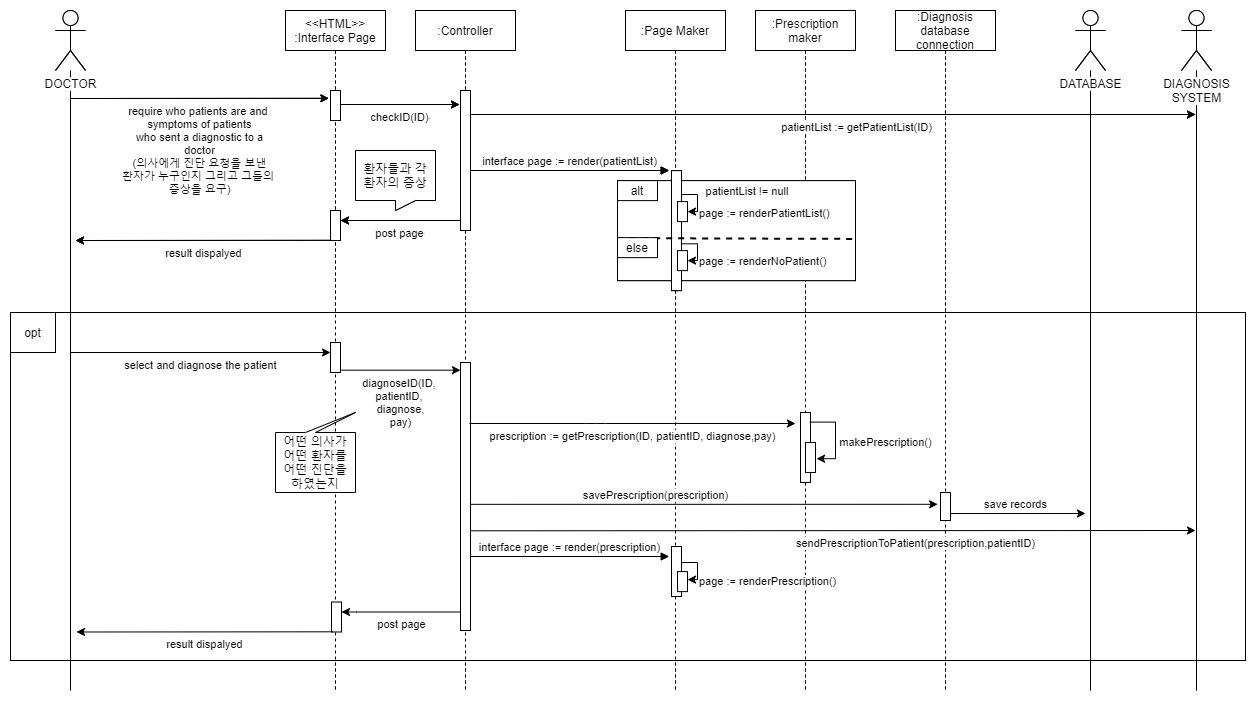
초기 시퀸스 다이어그램은 다음과 같다. 그러나 이러한 다이어그램을 사용하게 된다면 환자 actor가 먼저 시작을 해야 작동하게 된다. 이 Use Case의 경우 그룹을 나눈 것이 무의미하게 의사 actor가 능동적으로 시작할 수 없다는 것이다.



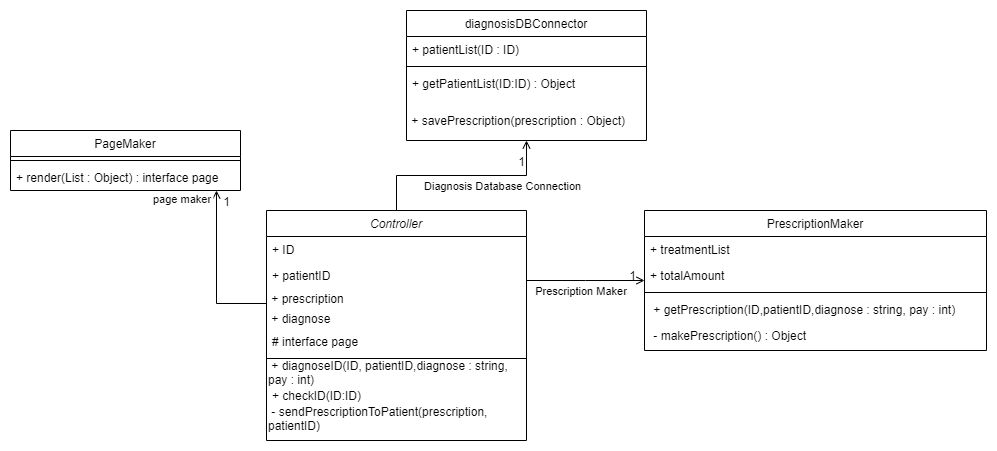
따라서 diagnosis system이라는 자신이 맡은 환자의 증상 목록을 열람하고 환자와 상호작용을 할 수 있는 시스템을 두어 의사 자신에게 진단 요청을 한 환자들을 자발적으로 보고 진단한 수 있도록 하였다.



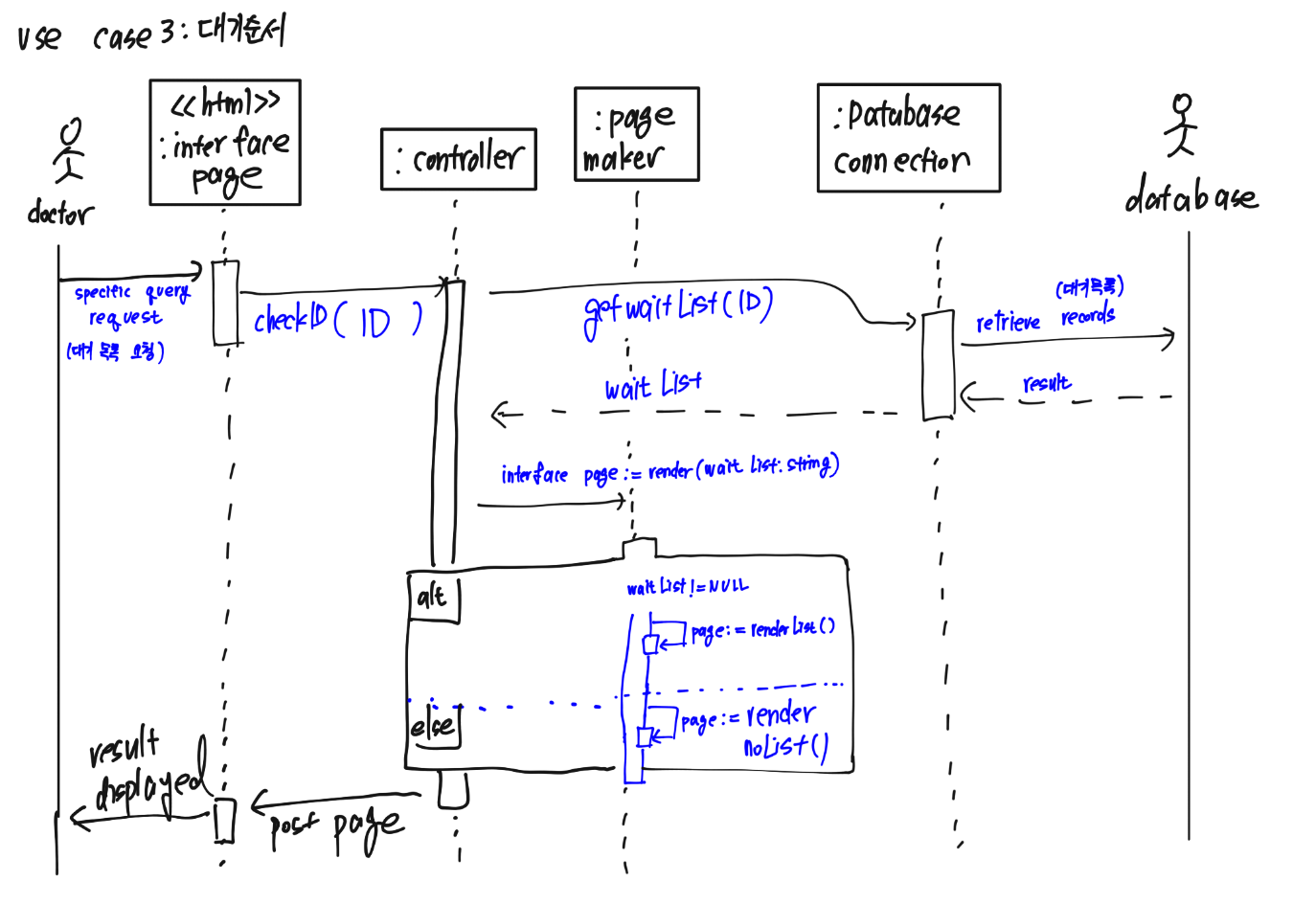
다만 여기서 diagnosis database connection 이 database 이외에 diagnosis system에 직접 처방전을 보내도록 하는 것은 해당 역할과 어긋난다고 생각하여 sendPrescriptionToPatient 을 Controller가 직접 사용할 수 있도록 하였다. 따라서 최종본은 다음과 같다.



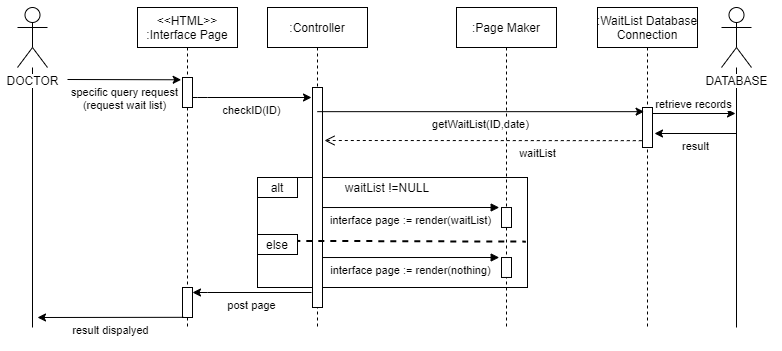
클래스 다이어그램은 다음과 같다.



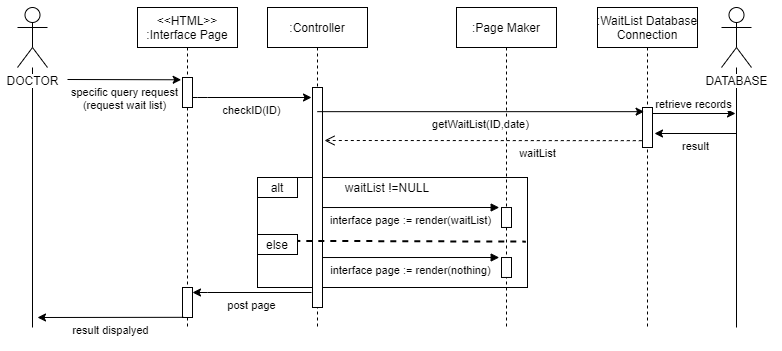
* **UC2 : 예약 및 상담**
* **UC3 : 대기 순서**



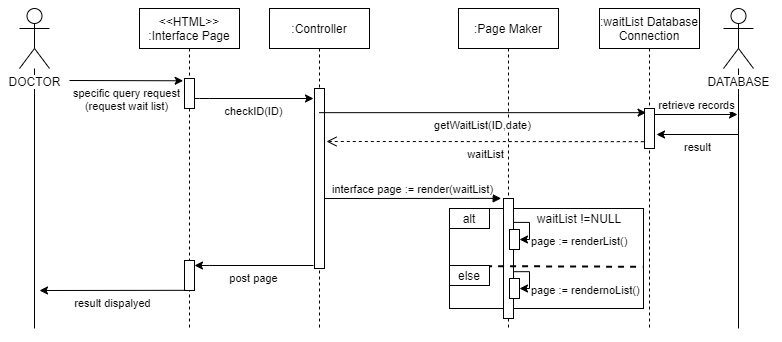
초기 시퀸스 다이어그램은 다음과 같다.



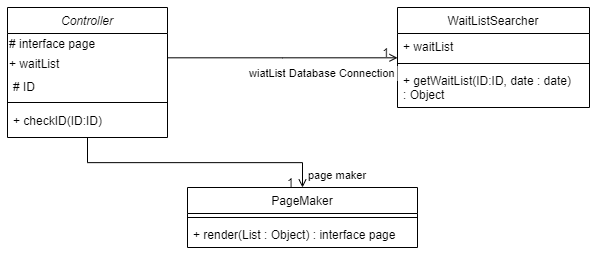
이를 정밀하게 구현하면 다음과 같다.



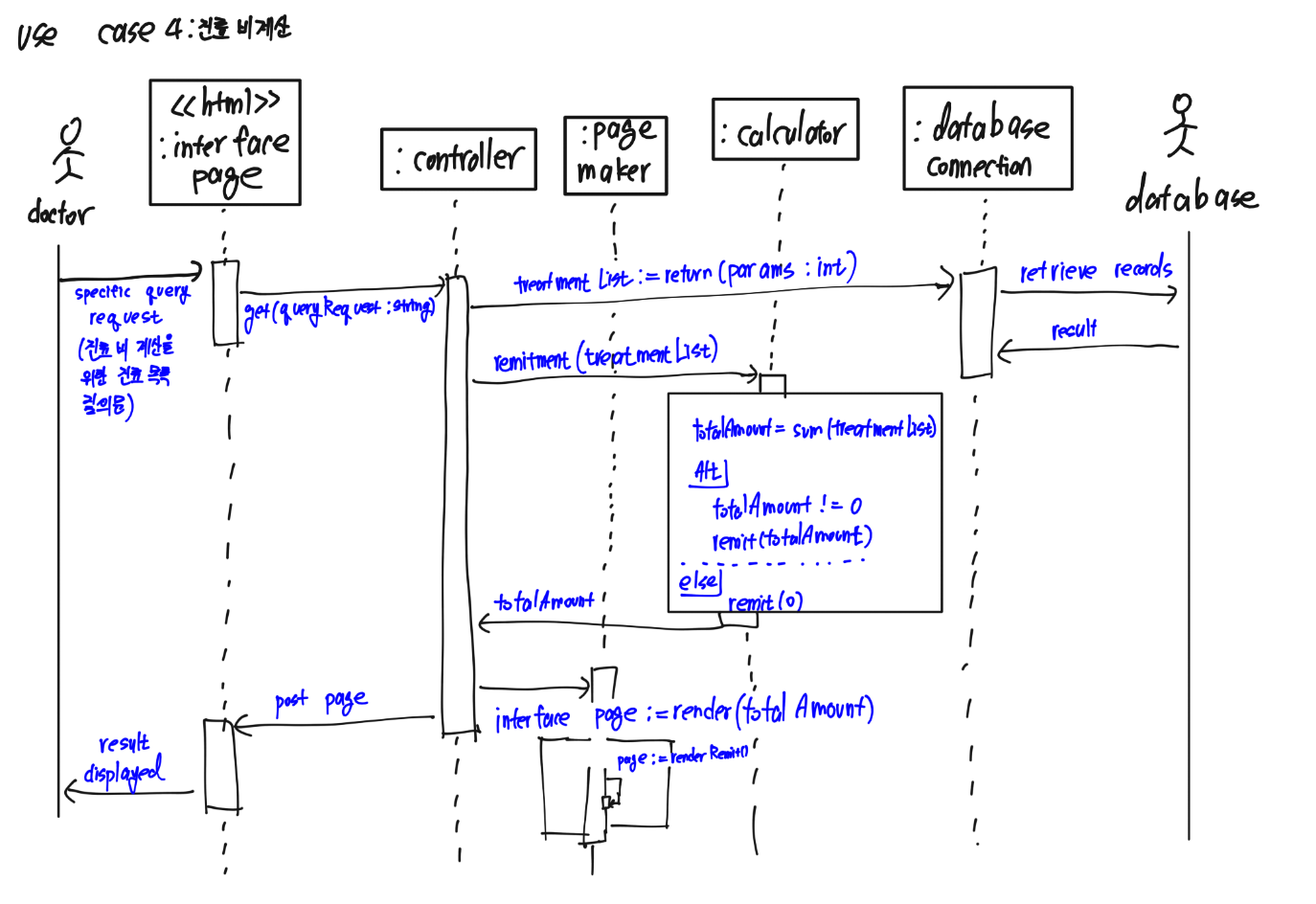
그런데 Controller 부분에서 if-else문을 통해 인터페이스 페이지를 만들게 되는데 이런 사소한 부분까지 Controller가 담당할 필요가 없다고 생각하여 해당 부분은 Page Maker가 처리하도록 하였다. 최종본은 다음과 같다.



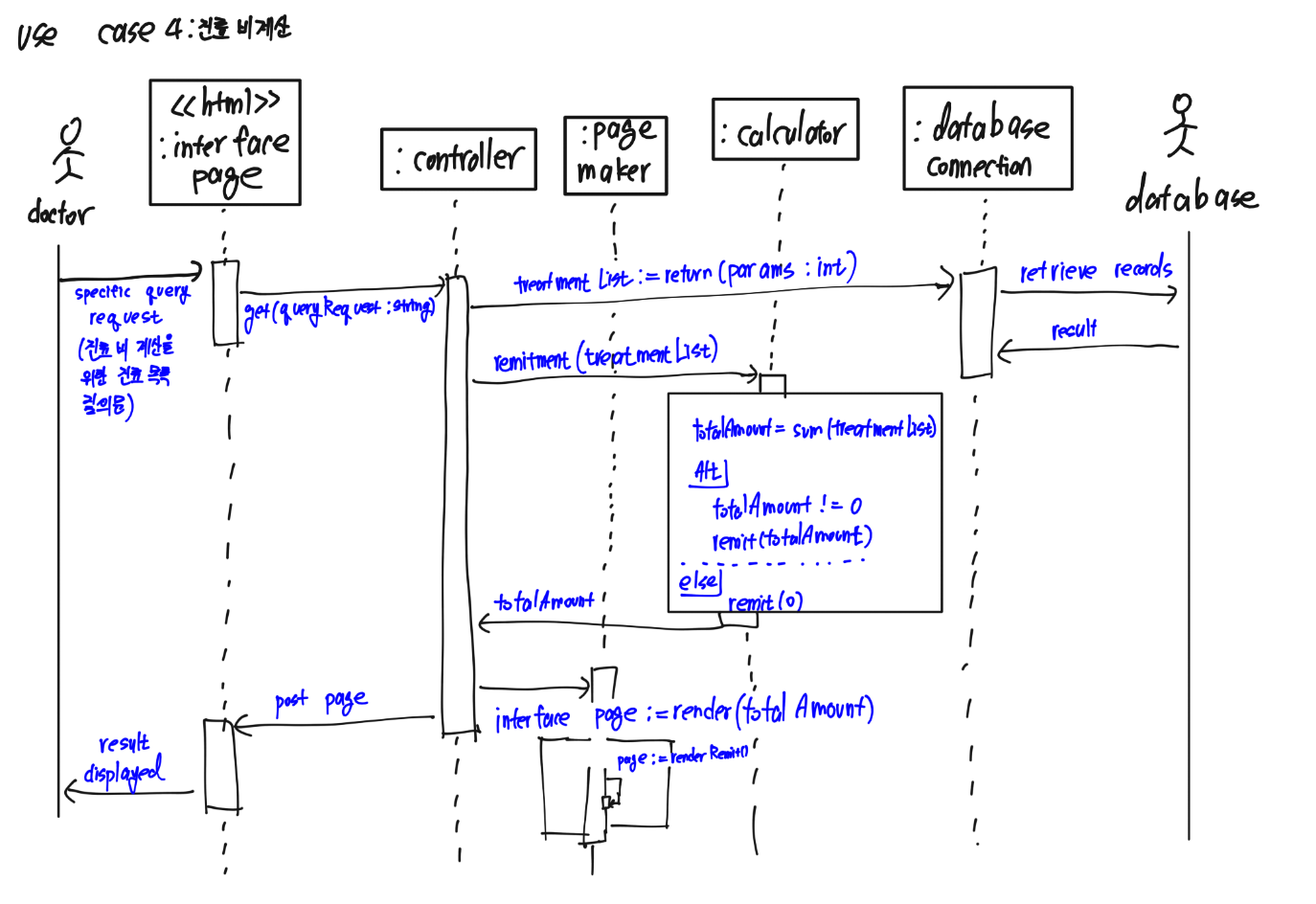
클래스 다이어그램은 다음과 같다.



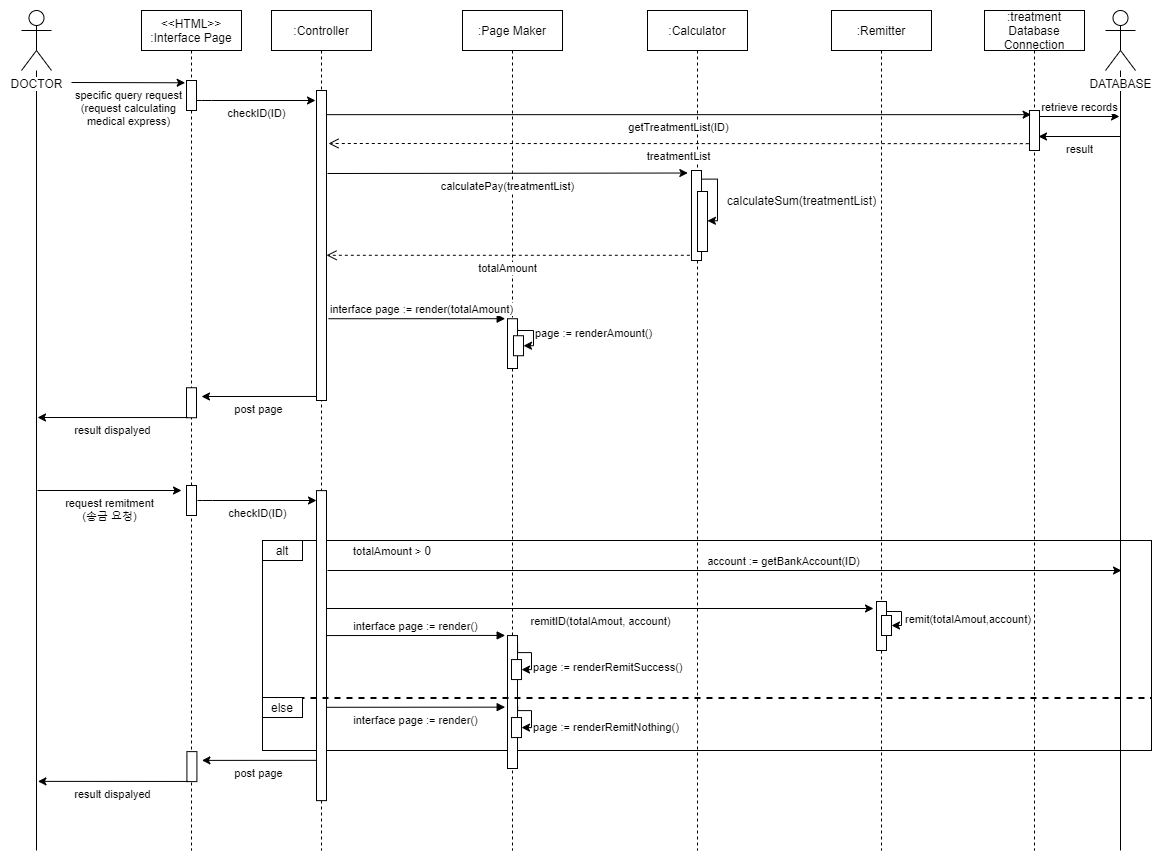
* **UC4 : 진료비 계산&송금**



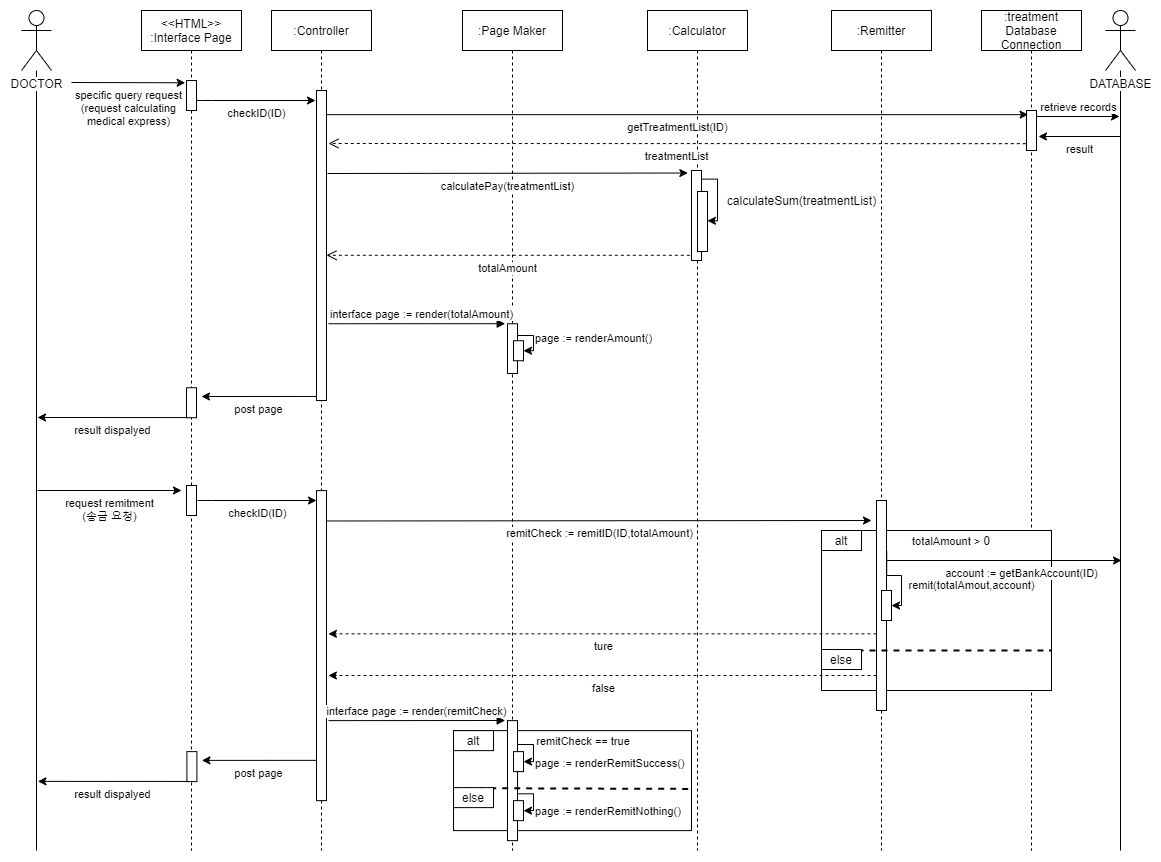
초기 시퀸스 모델은 다음과 같다. 그런데 여기서는



Calculator가 진료비 정산 뿐만 아니라 송금까지 담당하고 있다. 이는 Calculator의 역할에서 벗어났다고 생각하여 Remitter를 새로 만들었다.

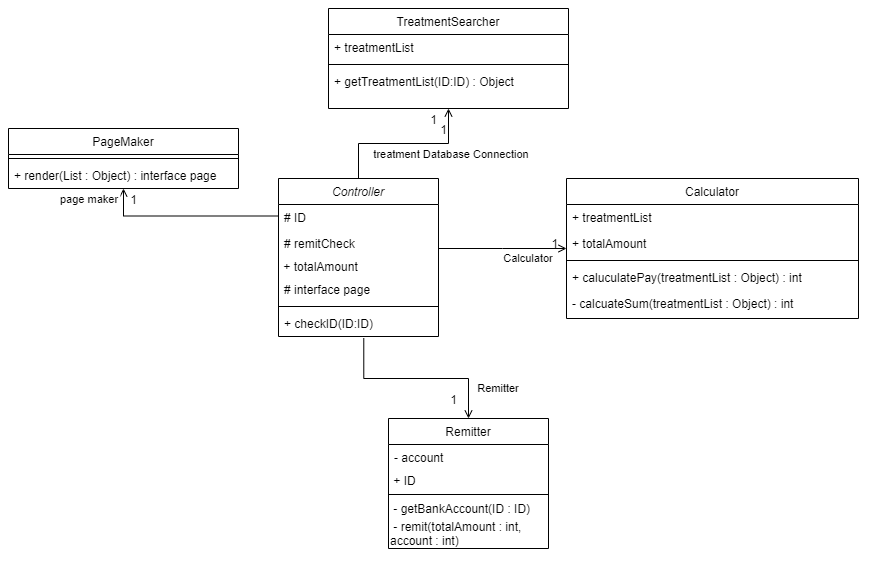


따라서 Remitter를 추가한 시퀸스 모델은 다음과 같다. Remitter가 Controller로부터 넘어와 계좌번호와 송금액을 받아 진료비를 송금하게 된다. 그런데 Controller가 데이터베이스에서 계좌 번호를 가지고 와서 금액에 따라 송금을 할지 말지 결정하는 것이 Controller의 역할에서 벗어난다고 생각하여 해당 부분은 Remitter로 이동시켰다. 따라서 최종본은 다음과 같다.



Remitter에서 송금할 금액이 있다면 계좌번호를 데이터베이스에서 찾아와 송금을 하고 송금을 하였는지에 대한 Boolean 값을 리턴하도록 하였다.

클래스 다이어그램은 다음과 같다.



* **UC11 : 사이트 피드백**