



Software Engineering

Overall Project

Implementation Doc

<Members>

김찬현 / 20172977

장동훈 / 20174353

안재형 / 20171248

이지호 / 20174266

김동욱 / 20173299

김상렬 / 20165020

<Table of Contents>

1. 설계 문서를 기반으로 한 실제 구현

1 – 회원가입 (UC1)

2 – 로그인 (UC2)

3 – 메뉴보기 (UC3)

4 – 주문관리 (UC8)

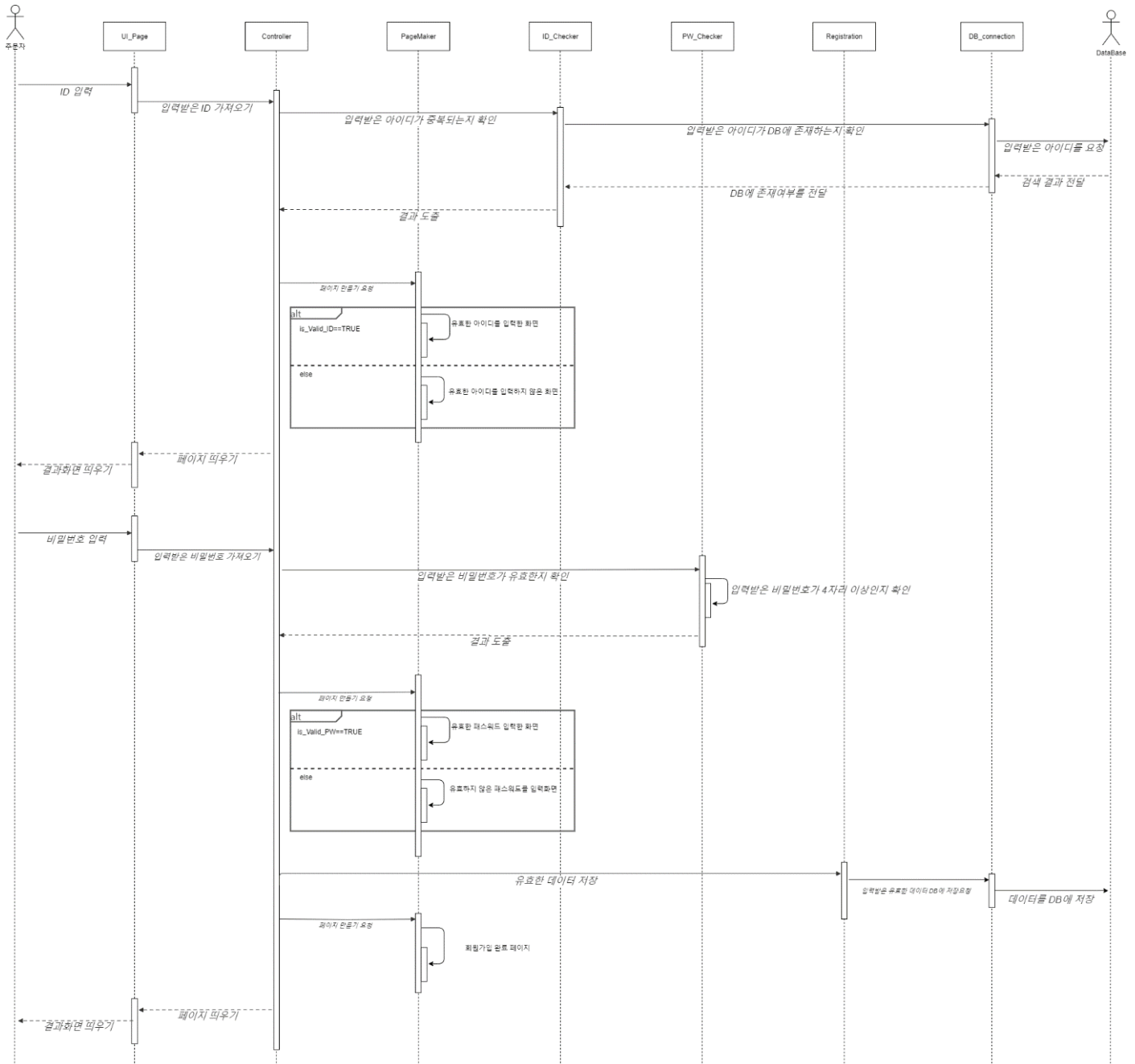
5 – 가게 정보 수정 (UC9)

2. 전체 서비스 구조도 및 설명

3. 팀 프로젝트 협업 방식 및 워크로드

1 – 회원가입 (UC1)

1) System Sequence Diagram



2) 구현

회원가입

A user registration form with three input fields labeled 'name', 'phone', and 'password', and a blue 'Sign Up' button below them.

3) 설명

1) UI_Page <-> Controller

설계과정에서는 회원가입시에 ID를 입력한 뒤에 유효성검사를 하고 유효하다면 비밀번호 입력페이지로 넘어가 유효성 검사를 하여 따로 입력을 받는 구조였지만 실제 구현과정에서는 불필요한 페이지 이동을 줄이는 것이 사용자 편의성을 높이는 것이라 생각하여 한 페이지에서 구현하였습니다.

2) Controller <-> Registration(signup)

설계과정에서는 Controller에서 ID_Checker와 PW_Checker로 바로 넘겨 서버에서 input값들의 유효성을 검증하였습지만 실제 구현에서는 Sign Up 버튼을 누르면 받아온 name, phone, password를 signup(Registration) 함수를 실행하여 서버로 input을 넘겨 POST 요청을 하도록 했습니다.

3) Registration(signup) <-> DB_Connection

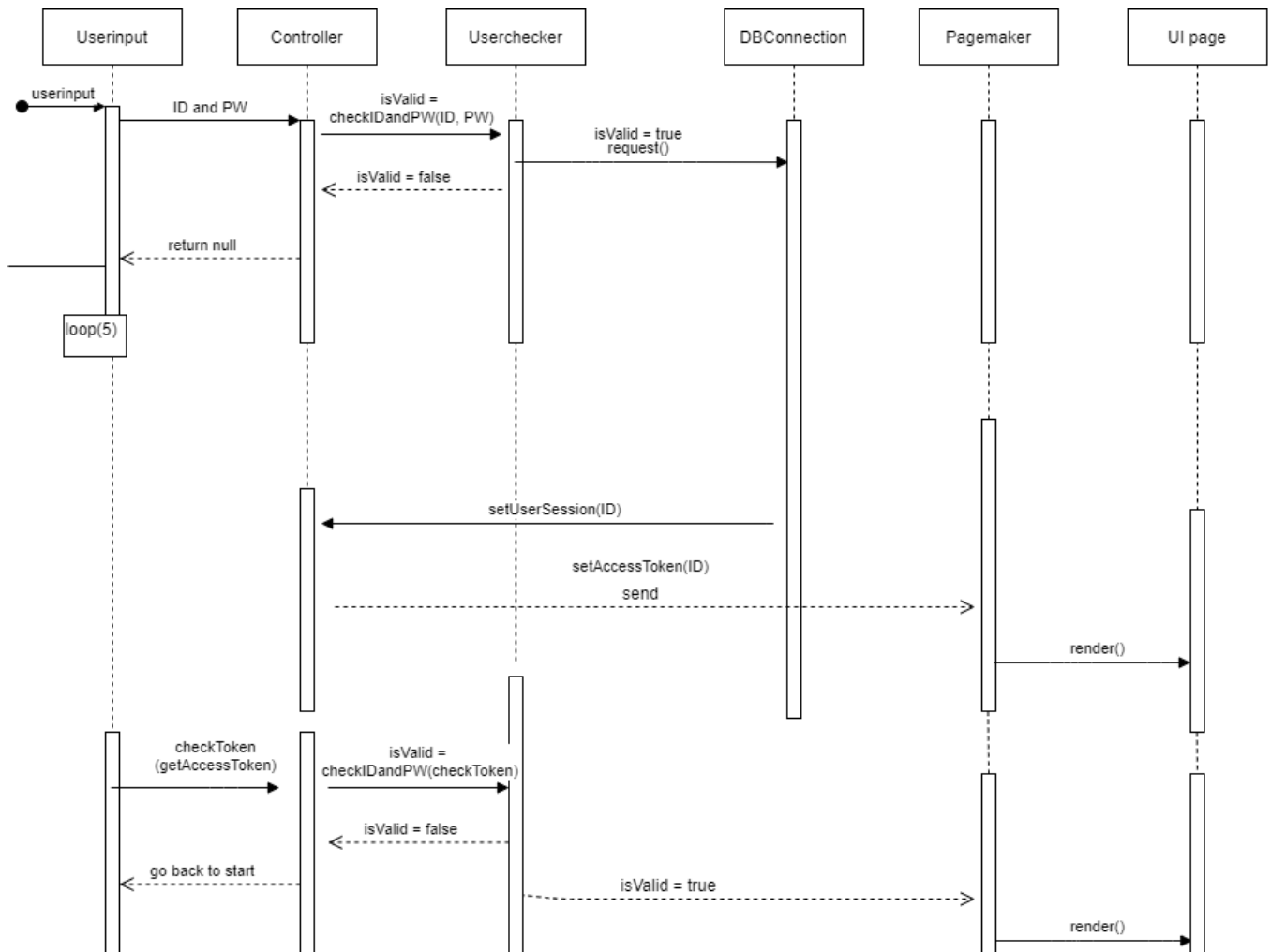
위 과정에서 받아온 input 값을 서버에서 유효성을 판별하고 해당 input이 올바르다면 DB에 게재하였습니다.

4) Controller <-> PageMaker

위 과정에서 input의 형식이 옳아 DB에 게재되었다면 caupizza.shop/signup 페이지로 바로 넘어가게 구현하였습니다. 만약 형식이 옳지 못하다면 에러 메시지를 띄우도록 구현했습니다.

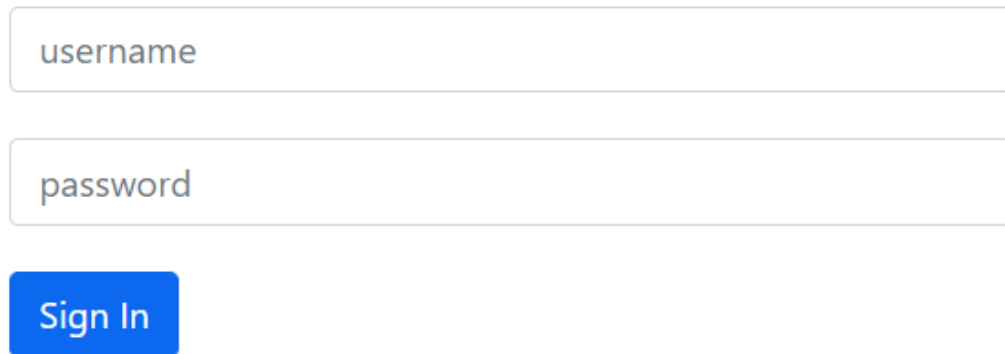
2 – 로그인 (UC2)

1) System Sequence Diagram



2) 구현

Login



A login form with two input fields and a button. The first input field is labeled 'username' and the second is labeled 'password'. Below the input fields is a blue button labeled 'Sign In'.

3) 설명

1) User input->Controller

- username과 password를 입력을 하면 controller로 username과 password가 전달된다.

2) Controller->UserChecker

- Sign In 버튼을 누르면 API 로 입력한 정보가 전달된다.

3) UserChecker<-> DBConnection

- 먼저, 입력받은 아이디가 DB에 저장되어 있는지 쿼리문을 통해 확인한다.
- 아이디가 있을 경우 입력 받은 비밀번호랑 DB에 있는 비밀번호랑 일치하는지 쿼리문을 통해 확인을 한다. DB에 있는 비밀번호는 암호화 되어 있기 때문에 입력받은 비밀번호를 암호화해서 서로 비교한다.

4) UserChecker -> Controller

- 위의 과정이 성공을 하면 UserChecker는 AccessToken을 생성하게 되고

Controller에 전달한다.

- Controller는 쿠키에 AccessToken을 담아 놓는다.

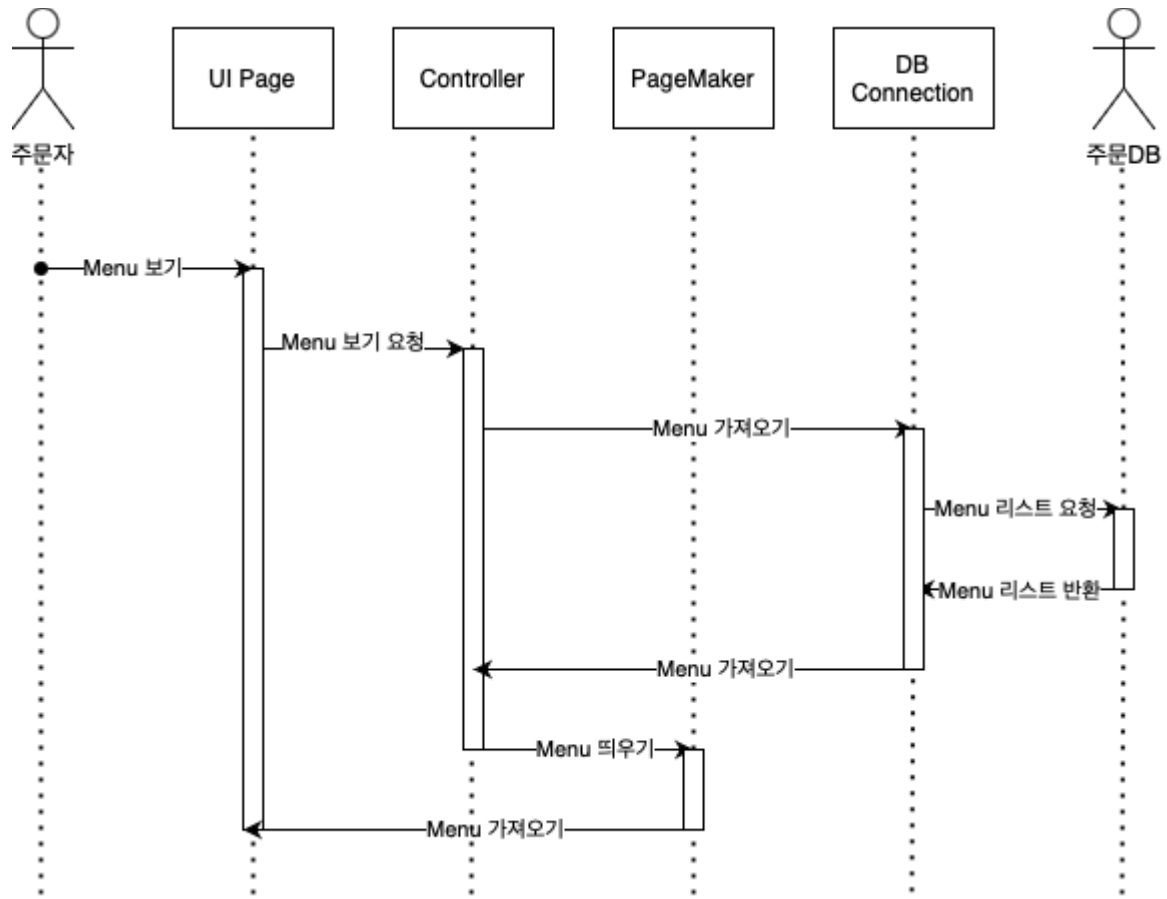
```
▼ 일반  
요청 URL: https://caupizza.shop/menu  
요청 메서드: GET  
상태 코드: 🟢 304 Not Modified  
원격 주소: 13.124.43.11:443  
참조 페이지 정책: strict-origin-when-cross-origin  
  
▶ 응답 헤더 (7)  
▼ 요청 헤더 소스 보기  
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9  
Accept-Encoding: gzip, deflate, br  
Accept-Language: ko,en;q=0.9,en-US;q=0.8  
Cache-Control: max-age=0  
Connection: keep-alive  
Cookie: auth=eYjhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiIyMGZlZWYzM2M2ZGRkM2E3ZjAyMGRkZmIiLCJ1c2VybmFtZSI6ImNhbmdeXVsIiwiaWF0IjOnRydwUSImldCjMTYyMjQyODQ3MywiZXhwIjojNjIwMDMzMjc2Ljpc3MiOiIjYXVwaXp5S5jb20iLCJzdWIiOiIjIc2VySW5mbYj9.3NG0do-BMmdvUeT07unb-R86w7HZaPWuk-yILs7V1VE
```

- 로그인 성공 이후 메뉴페이지에 담겨있는 토큰이다.

5) 설계 과정에서는 AccessToken을 통해 페이지 접근 권한을 주는 방식을 하려고 했으나, api 요청할 때만 Token이 사용되는 방식으로 구현이 되었다.

3 – 메뉴보기 (UC3)

1) System Sequence Diagram



2) 구현

<메뉴 리스트 페이지>

메뉴 페이지

장바구니 바로가기



청년피자
14000원
청년피자



청년피자222
14000원
청년피자

<상세 메뉴 페이지>

청년피자

가격: 14000

설명: 청년피자

장바구니 담기

3) 설명

1) UI Page->Controller

- Page 에 접근하자마자 Controller 에서는 API(DBConnection)를 실행한다.

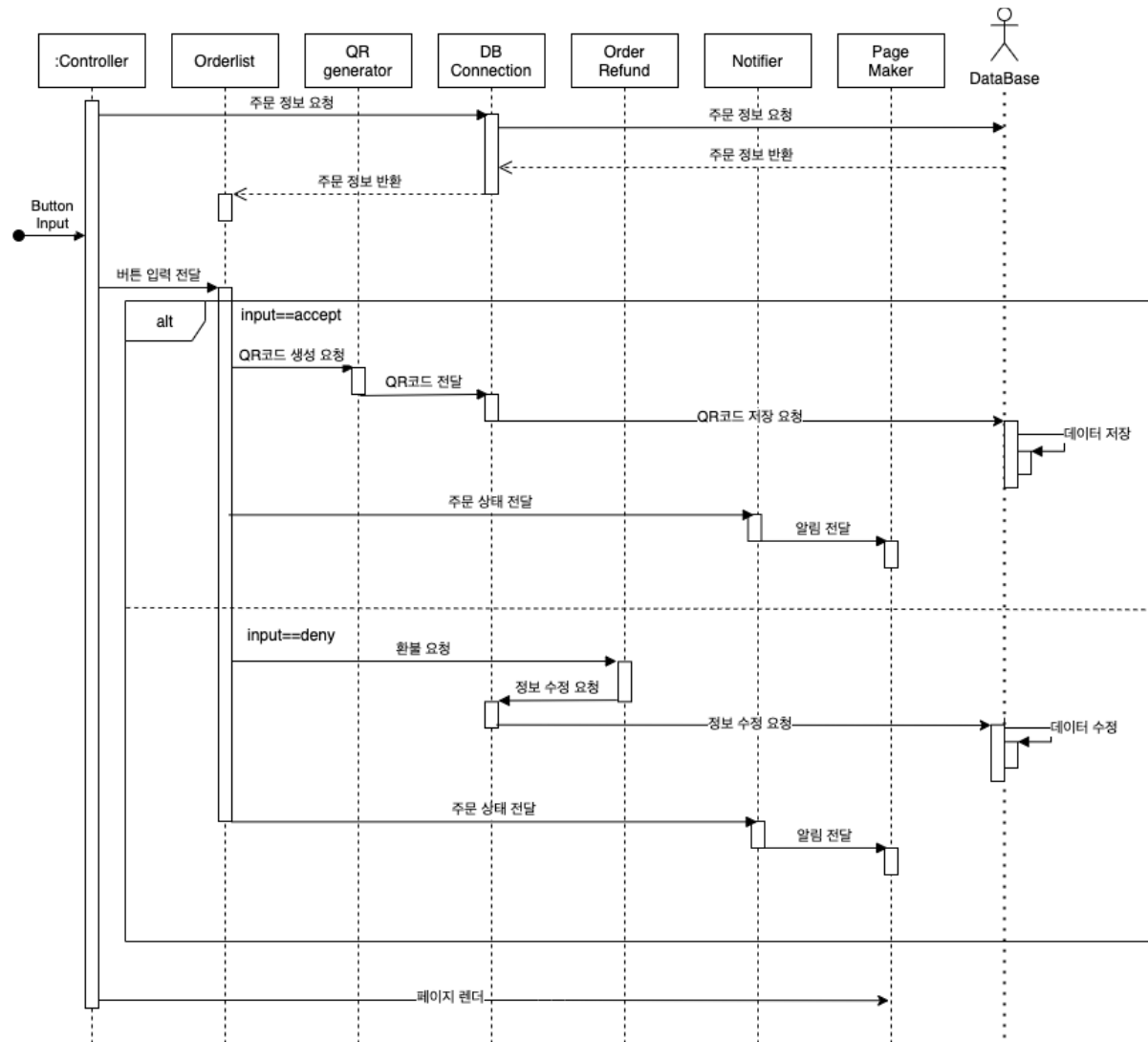
2) Controller<->DBConnection

- Cookie 에서 로그인 토큰을 헤더에 추가한다.
- 서버에서 메뉴를 받아오는 request 를 전송한다.
- 서버는 데이터베이스에 있는 메뉴를 response 해준다.
- response 를 받으면 메뉴 data 를 추출한다.

3) 설계 과정에서 페이지 전달 과정을 상세히 기술하지 않았으나, Controller 에서 url parameter 로 피자 정보를 전달하여 상세정보 페이지로 넘어가게끔 구현하였다.

4 – 주문관리 (UC8)

1) System Sequence Diagram



2) 구현

주문자 : sangryul
60ad3c8b27d1db4b0dc2950f 외 2건
주문시각 : 2021-05-29 05:21:31

주문 요청

수락
거절

주문자 : sangryul
60ad3c8b27d1db4b0dc2950f 외 2건
주문시각 : 2021-05-29 07:32:13

조리중

수령 처리

주문 관리
가게 정보
판매 내역

3) 설명

1) Controller -> DB Connection (주문 정보 요청)

- 주문 정보를 요청하는 api를 호출한다.

2) DB Connection -> Orderlist (주문 정보 반환)

- DB로부터 받아온 주문 정보를 Orderlist에 저장한다.

3) Button Input -> Controller -> Orderlist -> DB Connection (주문 상태 관리)

- 수락 / 거절 / 수령 처리 세 가지의 버튼을 만들어서 주문을 관리하도록 설계하였다.
- 세 버튼을 누르면 api가 호출되어 해당 주문의 주문 상태가 바뀐다.
- 시간 제약 상 QR Generator, Order Refund와 Notifier 부분은 구현하지 못하였다.

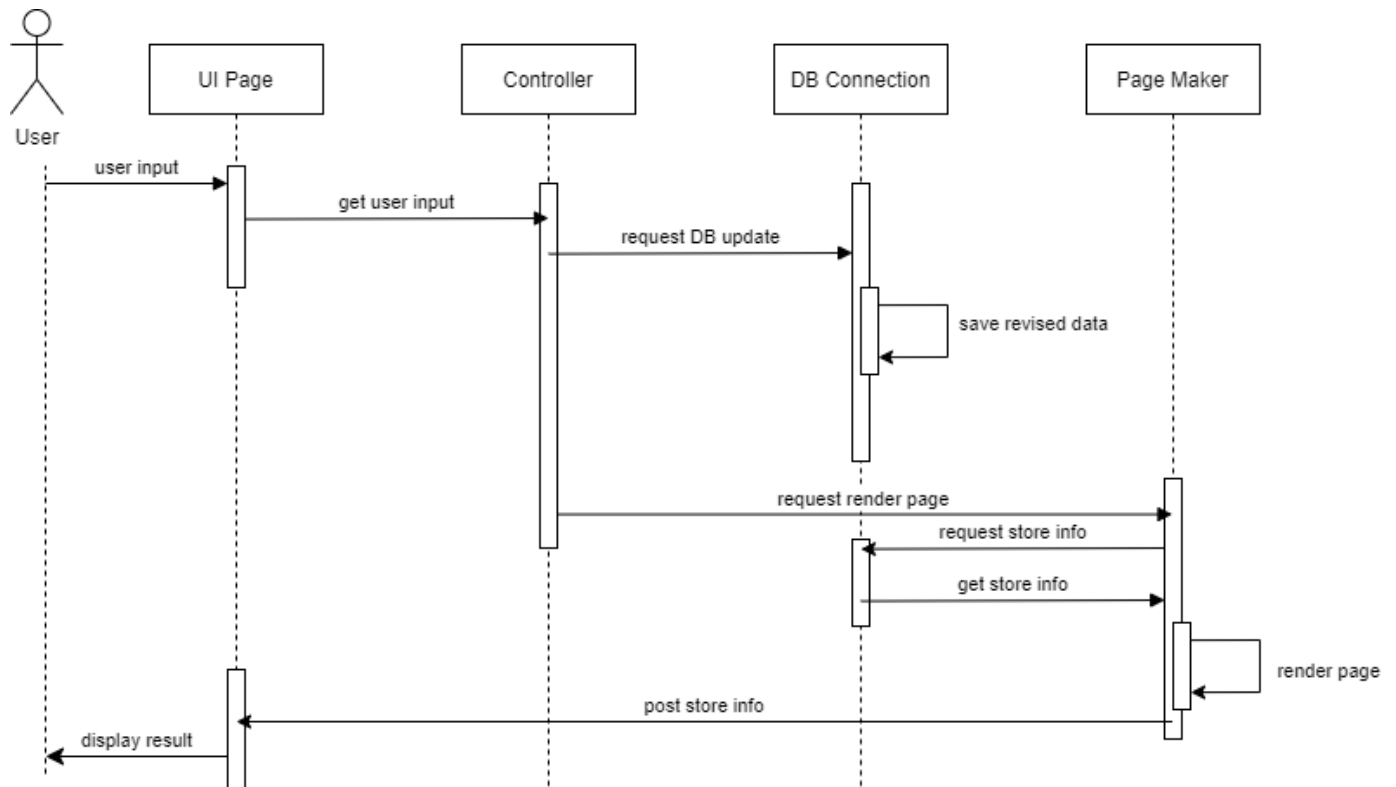
4) Controller -> Page Maker (페이지 렌더)

- 각 주문의 바뀐 주문 상태를 반영하여 페이지가 렌더링된다.

기존 설계에서는 주문 정보를 한번 반환하고 주문 상태를 변경한 뒤 페이지를 렌더링 하는 방식으로 계획하였지만 React를 사용하여 구현한 결과 주문 정보는 DB에서 주문의 정보가 수정되면 실시간으로 페이지에 반영되어 렌더링이 되도록 하였다.

5- 가게 정보 수정 (UC9)

1) System Sequence Diagram



2) 구현

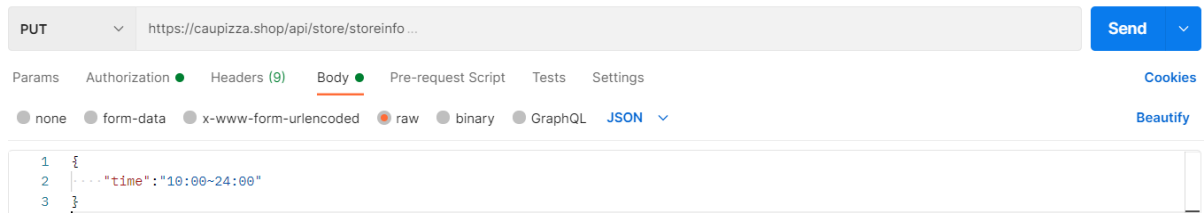
수정 전

CAU Pizza

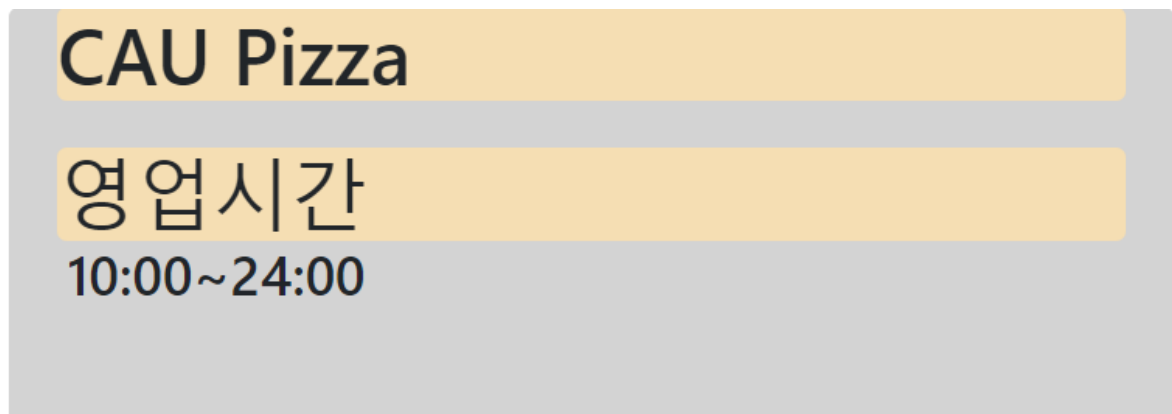
영업시간

12:00~24:00

시간 수정 요청



수정 후

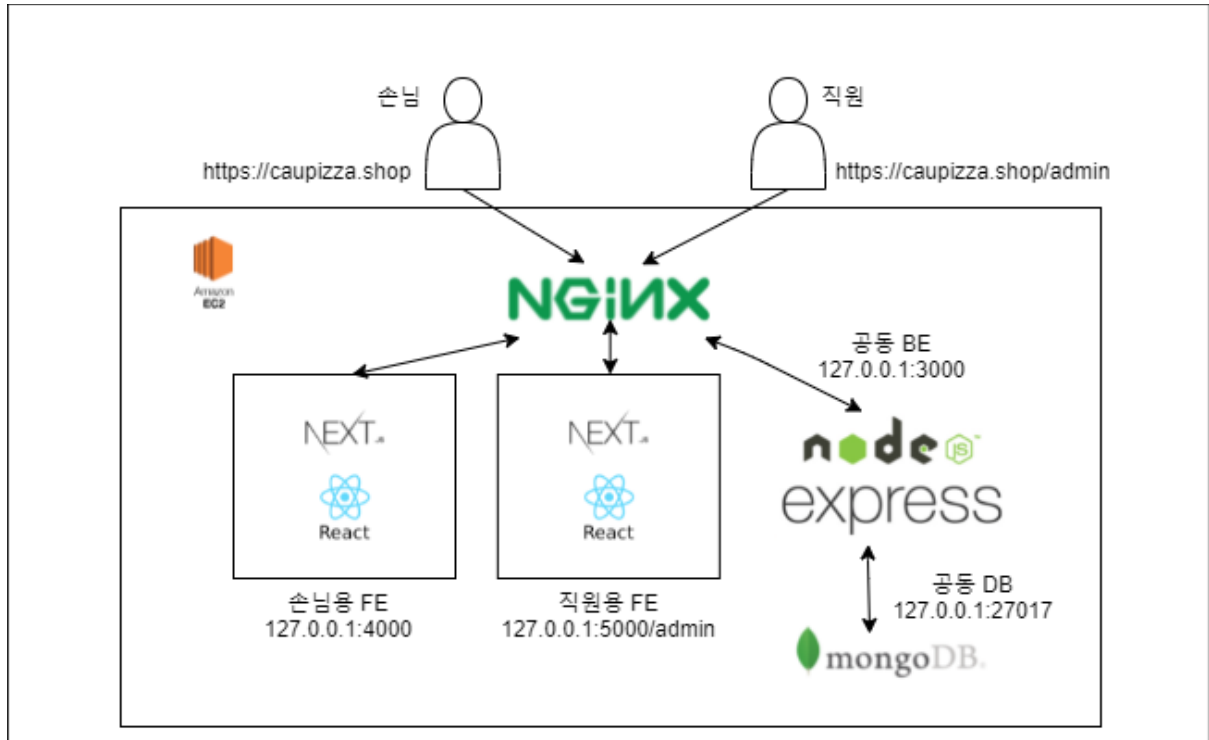


3) 설명

- 1) UI-Page -> Controller -> DB Connection
 - 구현하지 않았다. Postman을 통해 직접 요청하였다. 영업 시간만 수정한다.
- 2) DB connection -> Pagemaker
 - 수정된 시간을 받는다.

2. 전체 서비스 구조도 및 설명

1) 서비스 구조도



2) 설명

EC2 위에 NGINX 를 웹서버로 올려놓고 Route35 서비스를 이용해서 caupizza.shop 도메인과 연결해두었다. EC2 안에는 Next.js 로 작성된 손님용 FE, 직원용 BE 가 존재하고 mongoDB 를 DB 로 하는 express.js 로 작성된 BE 가 존재한다. 각 인스턴스들의 통신은 nginx 를 통해서 이루어지며, 각각 독립적으로 동작하며 API 명세에 따라서 동작한다.

3. 팀 프로젝트 협업 방식 및 워크로드

1) 코드 형상관리

손님용 FE

<https://github.com/causwe-bts/BTS-Front>

직원용 FE

<https://github.com/causwe-bts/BTS-Front-Admin>

BE

<https://github.com/causwe-bts/BTS-Backend>

API DOCS

<https://github.com/causwe-bts/Project-APIDOC>

Project DOCS

<https://github.com/causwe-bts/Project-Docs>

2) 협업 도구

코드 기록 : git/github

토론 기록 : ZOOM, Kakao Talk, Slack

문서 관리 기록 : Google Drive DOCS, Git/Github, Postman

3) 프로젝트 설계

공동으로 필요한 중심 API 에 대해서는 사전에 만든 여러 Diagram 을 토대로 필요한 항목들의 리스트를 만든 뒤 RESTFUL API 명세에 맞추어서 개발을 진행을 했다. API 명세를 정의할 때 프로젝트 전체의 팀인원의 회의를 통해서 URL,

method, req/res data 를 정의하고 서버에 합치기 전까지는 위 설계를 토대로 프로젝트 인스턴스 별 독립적으로 개발을 했다. 최종적 합의된 사항을 맞춰서 실서버에 migration 을 해서 협업에 대한 overhead 를 크게 줄일 수 있었다.

4) 역할 담당 및 전체 협업 진행 방식

기본적으로 Agile 방법론에 따라 주 2 회의 고정적인 회의와 각 파트 별 소회의를 통해서 프로젝트의 방향과 현재 진행하고 있는 개발의 방향이 맞는지 지속적으로 확인하였다. 프로젝트 문서 설계에 따라서 각 파트 FE, BE, INFA 등의 역할 분배를 했고 공통적으로 API 문서만 공유하고 나머지는 API 명세와 프로젝트 설계 문서를 토대로 개발을 진행했기 때문에 목표점에 크게 벗어나지 않고 개발을 할 수 있었다. 일정 관리와 예상된 bottleneck 부분에 대해서도 위 방식을 채택함으로써 크게 줄일 수 있었다.