

캡스톤 디자인1

16.5팀

최종 보고서

정석채

신재현

박상엽

박성환

개요

<< 마이크로 서비스 아키텍처, PWA에 기반한 공동 구매 웹앱 >>

목표

1. 제안 목표

- A. 사용자 회원 가입시 아이디, 이메일 등 기본 정보 외에 위치 정보를 저장한다. (공동 구매 나눔 시 위치 추천 목적)
- B. 분산 서버 환경을 고려하여 JsonWebToken에 기반하여 로그인을 구현한다.
- C. 사용자가 공동 구매 모집 게시글을 등록, 수정 및 삭제할 수 있도록 한다.
- D. 사이트 내에서 사용할 수 있는 코인 기능을 구현하고 가상의 계좌를 통해 충전 가능하도록 한다.
- E. 공동 구매 모집 게시글 별로 채팅방을 생성하여 사용자 간에 자유로운 소통을 가능하도록 한다.
- F. 회원 정보 페이지에서 관심 키워드 수정 기능, 이메일 인증을 이용한 비밀번호 변경 기능을 구현한다.
- G. 공동 구매 희망자가 발생하는 경우, 공동 구매 모집이 완료된 경우, 공동 구매 모집이 파기된 경우, 채팅방에 메세지가 전송된 경우에 대하여 푸시 알림을 구현한다.
- H. 사용자가 PWA로 제작된 공동구매 웹 앱을 디바이스에 설치할 수 있도록 한다.
- I. 마이크로 서비스 아키텍처의 의의에 맞추어 서버간 느슨한 결합을 유지하며 무중단 배포가 가능하도록 한다.

2. 수정 목표

- A. 위치 서비스 기능은 UI의 간결함을 고려하고 채팅 기능으로 충분히 대체될 수 있다고 판단하여 제외하였다.
- B. 구글의 보안 정책 강화로 인해 메일 인증 기능에 차질이 발생하여 향후 개선 방안으로 수정하였다.
- C. 서비스 간 비동기 통신을 위하여 메세지 브로커 카프카를 도입하였다.
- D. 서버의 수평적 확장성을 위하여 푸시 서버 및 이미지 업로드 서버 또한 직접 구현하여 하나의 컨테이너로 배포한다.

3. 성과

- A. 마이크로 서비스 아키텍처 구현을 위한 개별 서비스의 구현 및 통합에 성공하였다. 프로젝트 기획 단계에서 도메인의 기능별로 나눈 각 컨텍스트의 서버들이 컨테이너 형태로 데이터 베이스의 정합성을 유지하며 동작하는 것을 확인하였다.
- B. 하나의 서버에 장애가 발생하여도 다른 서버들은 정상 작동하는 것을 확인하였으며 부하가 집중되는 서버의 경우 추가적으로 컨테이너가 생성되는 클라우드 네이티브 환경을 구축하였다.

- C. 카프카를 통해 푸시 알림이 PWA 환경에서도 정상 작동하는 것을 확인하였다.
- D. PWA의 요건을 충족하여 푸시 알림 및 리소스 캐싱이 동작하고, 디바이스에 설치 가능한 앱이 제공되는 것을 확인하였다. 또, 메인 페이지인 로그인 화면이 오프라인에서 정상적으로 표시되도록 설계하였다.

추진 체계

1. 팀 구성

박성환	채팅 프론트엔드 및 백엔드 개발, Node.js dockerize
	팀장
정석채	회원가입, 로그인, 회원 관리 백엔드 개발 및 스프링 dockerize, 서버 검증
	위키 관리, 문서작업, AWS 계정 관리
박상엽	공동 구매 모집, 페이 기능 백엔드 개발, 스프링 dockerize, DevOps(AWS)
	예산 내역 작성
신재현	프론트엔드 개발, 푸시 서버 개발, PWA 구현, Mock server 구축
	협업 장소 및 멘토 섭외

2. 프로젝트 관리

- A. 깃헙을 이용한 코드 형상 관리 (<https://github.com/park-sy/gonggu-market>)
- B. 구글 드라이브에 회의록 관리
- C. 노션에 개발 관련 주요 개념 정리
- D. (<https://www.notion.so/a730862bc3c94be2aa12d625ae418bc1?v=f0ea20ee87d9455a81efbbc097a23050>)
- E. 화요일, 금요일 저녁에 미팅, 이외에도 항상 강의실을 대여하여 필요 시 추가적으로 미팅 진행
- F. 마이크로 서비스 아키텍쳐를 목표로 개별 서버 단위로 개발하였기에 서버 통합 과정에서의 효율성을 증진하기 위하여 ECR 이미지의 태그 및 카프카의 토픽 등의 작명 방식을 통일하였다.
 - i. 아래 두 사진과 같이 chat 리포지토리와 payment 리포지토리의 tag를 현재 날짜와 시간, 분으로 통일, DevOps로 하여금 최신 이미지 파악을 용이하게 하였다.

payment					
이미지 (2)					
	이미지 태그	아티팩트 유형	포시 위치	크기 (MB)	이미지 URI
<input type="checkbox"/>	2212142223	Image	2022년 12월 14 일, 22:32:19 (UTC+09)	392.96	URI
<input type="checkbox"/>	2212121149	Image	2022년 12월 12 일, 23:50:19 (UTC+09)	392.96	URI

chat					
이미지 (3)					
	이미지 태그	아티팩트 유형	포시 위치	크기 (MB)	이미지 URI
<input type="checkbox"/>	2212160239	Image	2022년 12월 16 일, 02:42:55 (UTC+09)	392.72	URI
<input type="checkbox"/>	2212152342	Image	2022년 12월 15 일, 23:43:47 (UTC+09)	392.72	URI

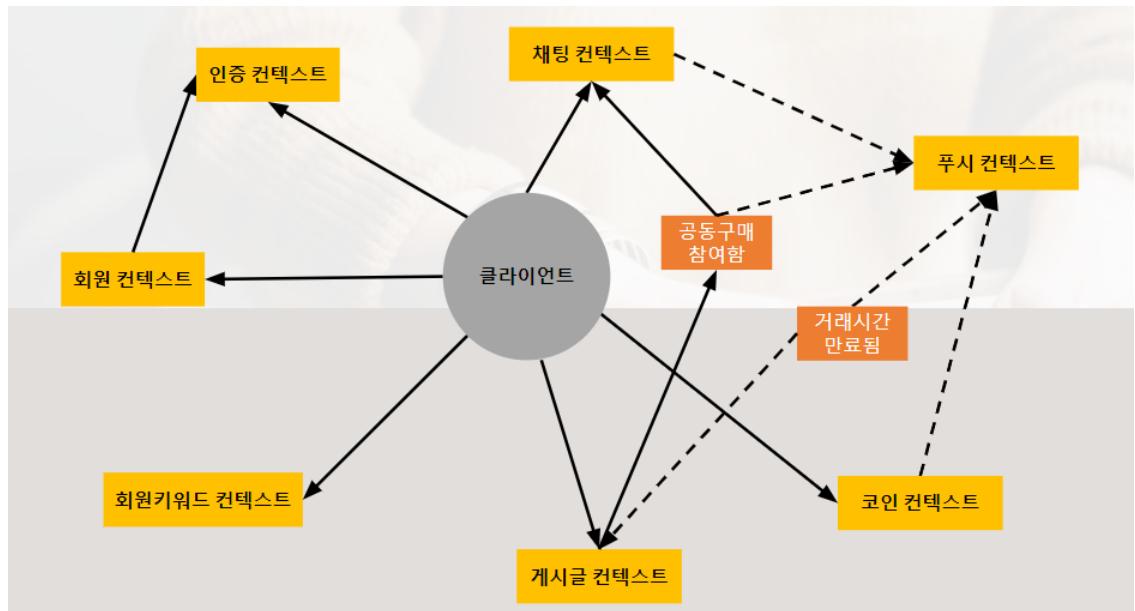
ii. 아래 사진은 Notion의 Kafka 토픽의 produce/consume 형식을 정한 것이다.

Kafka	
• <code>bootstrap-servers: 13.209.40.90:9092</code>	
• <code>group-id : gonggu</code>	
• <code>topic</code>	
◦ <code>from Deal to Push</code>	
▪ <code>dealJoin</code> : 공구 참여	
◦ <code>dealId": 56, "title":"테스트방", "nickname":["테스트유저1","테스트유저2"] }</code>	
◦ <code>{"dealId":7,"title":"이번엔 제발 성공....","nickname":["nick1","nick2","psy"]}</code>	
▪ <code>dealComplete</code> : 공구 완료	
◦ <code>{"dealId": 56, "title":"테스트방", "nickname":["테스트유저1","테스트유저2"] }</code>	
▪ <code>dealDelete</code> : 공구 삭제	
◦ <code>{"dealId": 56, "title":"테스트방", "nickname":["테스트유저1","테스트유저2"] }</code>	
◦ <code>from Deal to Chat</code>	
▪ <code>chatJoin</code> : 채팅 참여(공구 참여)	
◦ <code>{"dealId":56, "title":"테스트방", "nickName":"테스트유저"}</code>	
▪ <code>chatExit</code> : 채팅 나감(공구 취소)	
◦ <code>{"dealId":56, "title":"테스트방", "nickName":"테스트유저"}</code>	
◦ <code>from Chat to Chat</code>	
▪ <code>chatMessage</code> : 메시지 전송/수신	

개발 내용

1. 개발 범위 및 내용

A. 마이크로 서비스 아키텍처 설계



- i. 팀원들의 다양한 기술 스택 및 관심사를 반영하여 마이크로 서비스 아키텍처 구현을 목표를 설정하였다.
- ii. 위 그림과 같이 하나의 어플리케이션을 기능을 기준으로 여러 개의 컨텍스트로 구분하여 독립된 개별 서비스로 구현할 것을 목표로 하였다.

B. 정석채

i. 개발 범위 및 내용

- 스프링을 기반으로 회원 가입, 로그인, 유저 정보 조회 관련 API를 구현하였다. 사용자의 닉네임을 기본키로 설정, 회원 가입 시 중복 체크를 할 수 있도록 하였고 서버에서 비밀번호 일치 검사를 하는 등의 예외 처리를 하였다.
- 서버의 유연한 scale-in/out을 위해 JsonWebToken을 세션 관리 기술로 채택하였다.
- 서버는 클라이언트의 로그인 요청 시 access token과 refresh token을 발급하고 이후의 모든 요청에 대하여 토큰의 유효성 검사를 수행한다.
- 서버의 유연한 scale-in/out을 고려하여 토큰을 인-메모리 데이터베이스인 redis에 저장하였다
- 구현한 백엔드 서버를 도커를 통해 이미지화하여 ECR에 업로드하였다.
- 푸시 알림을 위해 데이터베이스에 푸시 알림을 보내는 기기를 등록하는 device 테이블을 구축하고 해당 테이블에 기기를 등록하는 API를 구현하였다.

ii. 이슈

- 프론트엔드 개발자와의 커뮤니케이션을 통해 refresh token을 이용한 access token 재발급 로직을 구축하였다. 추가적으로 토큰을 헤더에 포함하여 전송 할 것으로 결정하였다.
- 보안을 고려하여 access token의 유효기간을 30분으로 짧게 설정, refresh token의 유효기간을 2주로 설정하였다.

C. 박상엽

i. 개발 범위 및 내용

- 스프링을 기반으로 공동 구매 모집 및 페이 기능 백엔드를 구현하였다.
- 게시글 조회, 게시글 상세 조회, 게시글 작성, 게시글 수정, 게시글 삭제, 구매 참여, 탈퇴 등 목표로 한 웹앱의 가장 핵심적인 공동 구매 관련 API를 구현하였으며 적절한 트랜잭션 및 database locking을 이용하여 데이터베이스 레벨에서부터 진행 중인 공구인지, 삭제된 공구인지를 구분할 수 있도록 하여 프론트엔드 개발을 용이하도록 하였다.
- 사용자가 페이 화면에서 현재 잔액 및 본인의 코인 거래 기록을 확인할 수 있는 API를 구현하였으며 잔액이 부족한 경우 등에 대한 예외 처리를 하였다.
- 구현한 공동 구매 모집 및 페이 기능 서버를 도커를 통해 이미지화하여 ECR에 업로드하였다.
- AWS ECR/ECS를 이용하여 이미지를 컨테이너화하였으며 Elastic Load Balancer를 이용하여 서버들의 API 라우팅 환경을 구축하였다.

ii. 이슈

- 공구 마감일과 오늘 날짜가 동일한 경우 사용자에게 ‘종료된 마감’이라 표시되는 문제를 발생하였고 개발 환경과 AWS 배포시에 시간 차이 문제가 원인임을 파악하였다. 공동 구매 만료 여부와 만료시까지 남은 일수를 정확히 계산하기 위해 시간대를 맞춰야 했고 도커 이미지 빌드를 할 때 실행 환경에 독립적으로 시간대를 설정하여 이를 해결하였다.
- 페이 서비스의 송금 기능 등 동시성을 제어하여 데이터 손실을 방지할 필요가 발생하였다. 이를 위해 효율성을 유지하며 특정 로직에 대해 트랜잭션을 격리할 필요가 발생하였고 MySQL의 기능 중 하나인 innodb의 Gap Lock을 활용하여 최소한의 범위에만 Lock을 걸어서 문제를 해결하였다.
- API를 이용해 서버간 동기 통신을 하는 경우 발생하는 비효율성을 해결하기 위해 Kafka를 도입하였다. EC2를 이용하여 Kafka 서버를 구축, 서버 간 비동기 통신을 구현하였다. 그런데 카프카 서버가 다운됐을 때, 클라이언트의 요청은 제대로 처리되었으나 카프카가 연결되지 않아 무한 로딩이 발생, 응답이

보류되는 상황이 발생하였고 해당 요청을 처리는 쓰레드와 카프카 메시지를 처리하는 쓰레드를 분리하여 해결하였다.

D. 신재현

i. 개발 범위 및 내용

- Vue.js를 이용하여 로그인, 회원가입 화면 및 메인 페이지, 공동 구매 관련 페이지, 코인 충전 페이지들을 SPA로 구성, 개발하였다. 토글 버튼을 이용하여 게시한 공고 모집글, 참여한 공구 모집글을 구분할 수 있도록 하는 등 사용자의 UI/UX를 개선하였다.
- 사용자가 이미 참여한 공구에 다시 참여하거나 잔액이 부족한 상황에서 송금을 하려는 등의 예외 케이스를 방지하고 사용자에게 즉시 문제를 알릴 수 있도록 하였다.
- 클라이언트에서 서버와 JWT의 유효성 인증 정보를 주고 받는 로직을 구현하여 access token이 만료된 경우 refresh token을 이용해 access token을 재발급 받아 사용자가 다시 로그인하는 상황을 방지하였다.
- PWA를 구현하여 사용자의 웹앱에 대한 접근성을 높였으며 Lighthouse를 이용하여 PWA 테스팅을 진행하였다.
- Node.js를 이용하여 푸시 서버를 구현하여 공동 구매 모집 완료, 파기, 채팅 전송 등의 이벤트에 대하여 사용자가 푸시 알림을 받을 수 있도록 하였다.

ii. 이슈

- 푸시 기능을 구현하기 위해 웹을 PWA에 맞도록 구현하고, 서비스 워커를 등록했으나 Unit test의 마지막 단계인 '사용자에게 웹 푸시를 시각적으로 보여주는 항목'에서 지속적인 fail이 발생하였다. Logging 등은 정상적으로 이루어져, 브라우저 혹은 디바이스의 설정 상 오류를 찾고자 노력했고 결국 Web Push는 OS의 API를 사용해서 구현되는 것을 확인하고, OS의 Alert 설정을 수정하여 Unit test를 성공적으로 마무리하였다.
- ALB에서 user config를 진행한 뒤 각 서버에 클라이언트의 쿼리를 전달하는 아키텍쳐의 특성으로 인해, 개별 서비스에 대한 실제 작동 테스트는 모든 서버가 완성되고 배포되기 이전에 진행할 수 없었다. 이에 Express를 활용해서 mock server를 제작하여 E2E 테스트와 Integration 테스트를 동시에 진행하였고 이를 통해 배포 단계에서 프론트엔드의 디버깅에 소요되는 시간이 대폭 감소하였다.
- CSS 프레임워크로 Bootstrap 및 Element plus를 채택하였는데 Form validation 및 Modal popup 등 특징적인 요소는 프레임워크에서 제공하는 기능이 불충분하여 필요한 기능 다수를 직접 바닐라 자바스크립트로 구현했음. 이와 같

은 방식으로 구현할 경우 동작의 신뢰성이 보장되지 않기에 Integration Test에 시간을 많이 할애하였고 그 과정에서 구매 제한 개수의 max value가 min value보다 작아서 validation process가 무한 루프에 빠지는 중대한 오류를 발견 및 해결할 수 있었다.

E. 박성환

i. 개발 범위 및 내용

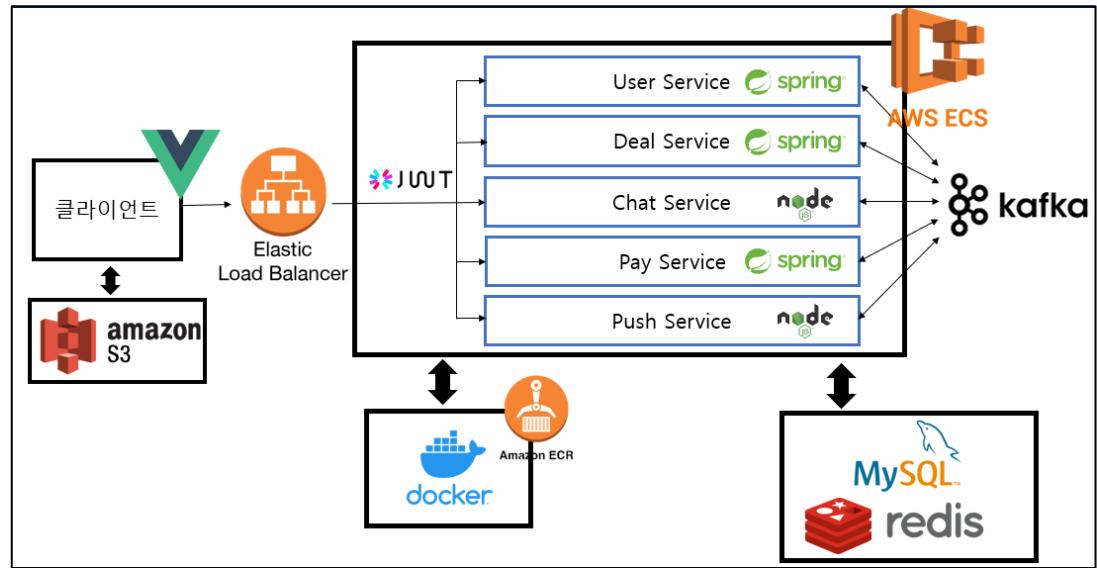
- Node.js, socket.io와 Vue.js를 이용하여 채팅 백엔드, 프론트엔드를 개발하였다. 공구 모집 게시글과 채팅방을 일대일 대응시켜 공구 모집 참여자들끼리 자유로운 의사소통을 할 수 있도록 하였다.
- 이미지 업로드 서버를 EC2에 구현하여 채팅방에서 이미지 전송을 가능하도록 하였고 채팅방 별로 본인이 읽지 않은 메세지 개수를 확인할 수 있는 기능을 추가하였다.
- 실제 카카오톡을 벤치마킹하여 같은 사용자가 연속적으로 메시지를 보내는 경우 닉네임 및 보낸 시간을 생략하고 날짜 구분선을 표시하는 등 채팅 관련 UX 증진을 개선하였다.
- Pinia library를 통해 클라이언트의 상태 관리를 하였고 이를 기반으로 Vue-Router를 이용하여 인증되지 않은 사용자의 주요 페이지 접근을 방지하였다.
- 채팅 서버 및 푸시 서버를 Kafka와 연결한 후 각각을 도커를 이용해 이미지화하여 ECR에 업로드하였다.

ii. 이슈

- 채팅방 별 읽지 않은 메세지 개수 표시 기능을 위하여 사용자가 채팅방을 이동할 때 기존에 있던 방의 소켓 연결을 끊고 새로운 방의 소켓으로 연결 하던 방식에서 채팅방 페이지에 접속 시 사용자가 속해 있는 모든 채팅방에 소켓을 한 번에 연결하고 일괄 관리하는 방식으로 변경하였다. 그리고 SPA의 특성을 이용하여 채팅방 간 이동 시 채팅 대화 내용만 바뀌는 방식으로 구현하였다.
- 데이터베이스 설계 시 공동 구매 모집 게시글 테이블과 채팅방 테이블을 이원화 할 것인지를 고민하였다. 공동 구매 모집글이 생성되면 채팅방을 새로 생성하는 방식은 두 서버에 걸쳐 트랜잭션을 수행해야 하기 때문에 해당 로직을 기간 내에 구현하는 것이 어렵다고 판단하여 두 테이블을 하나로 합치기로 결정하였다.
- MSA의 서버 확장성 때문에 채팅 컨테이너가 추가되어 여러 개의 서버에 하나의 채팅방에 대한 정보가 분산되는 경우, 메세지가 제대로 전송이 되지 않게 된다. 이를 대비하기 위하여 레디스를 이용하였다. 레디스 어댑터를 이

용할 경우 각각의 채팅방에 연결된 소켓의 정보가 key-value 형태로 레디스 서버에 저장된다. 그리고 사용자가 채팅을 보내면 우선 레디스 서버로 전달된 후 알맞는 채팅방으로 전송이 되도록 하였다.

2. 시스템 구조도



3. 구현 결과

A. 로그인 화면



Sign In

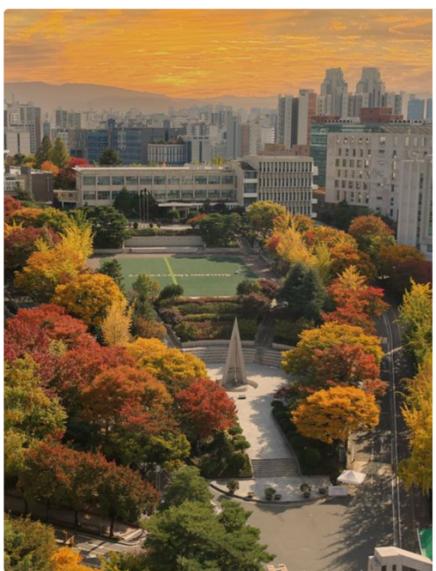
ID

Password

Sign In

Not a member? [Sign Up](#)

B. 회원가입 화면



Registration

ID

 중복검사

EMAIL

PASSWORD

PASSWORD CONFIRM

Be our member

C. 공구 게시글 목록 화면 (메인 페이지)

우리동네 공구마켓 전체 채팅 공구 등록 공구 채팅 공구 페이 nick2 로그아웃

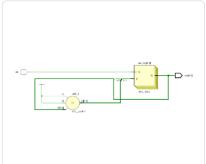
디지털기기



재현신-nick1

생활/수명 2/2명 1명 외 막강

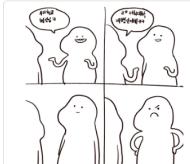
공구 참여 1인당 10,000원



psy게시글

취미/개인/습관 2/2명 오늘 막강

공구 참여 1인당 5,000원



nick2의 12/16마감 공구

생활/수명 4/5명 1명 외 막강

공구 참여 1인당 6,500원



찐막...ㅈㅂㅈㅂ....

가구/인테리어 3/7명 오늘 막강

공구 참여 1인당 7,250원

종료된
공구입니다

종료된
공구입니다

종료된
공구입니다



D. 개별 공동 구매 게시글 화면

우리동네 공구마켓 공구 등록 공구 채팅 공구 페이 shin의 새로운 공구
단가 : 11,000원 | 구매 단위 : 5개
디자인가기 2/13일 오늘 봐요 shin 구매 사이트로 이동
shin의 새로운 공구
구매수량 - 5 + 개
총 상품 금액 11,000원
공구 참여

E. 공동 구매 참여시 사용자의 UX를 고려한 알림

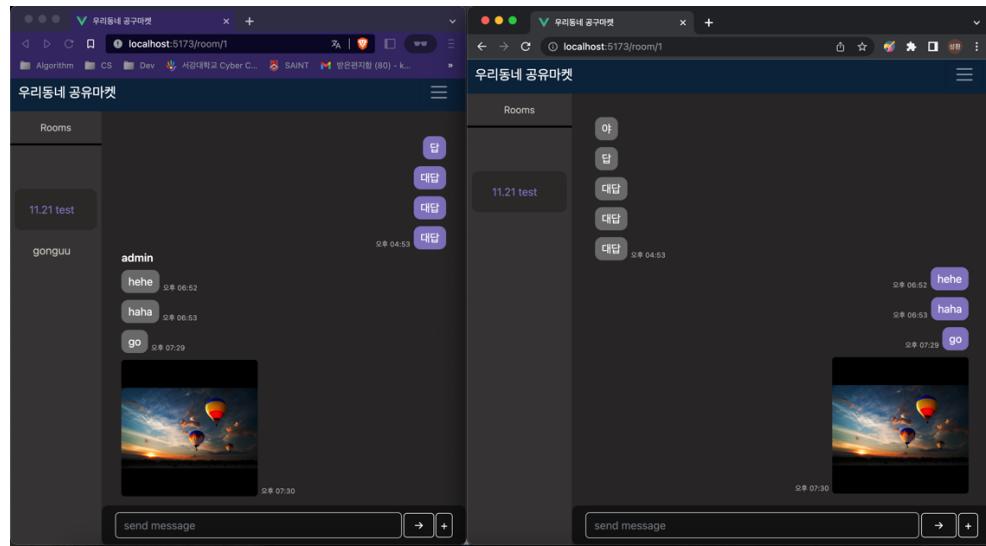
우리동네 공구마켓 공구 등록 공구 채팅 공구 페이 09market.site 내용:
공구에 참여하시겠습니까? 구매 단위를 다시 한번 확인하시기 바랍니다.
구매단위 : 1일위 확인 취소 shin shin의 새로운 공구
구매수량 - 5 + 개
총 상품 금액 11,000원
공구 참여

F. 페이 화면

우리동네 공구마켓 전체 채팅 공구 등록 공구 채팅 공구 페이 nick2 로그아웃
우리동네 공구페이
현재 잔액 37,074원
충전하기 환전하기
송금하기
송금받는 자
금액
송금하기
우리동네 공구마켓은 회원간의 송금에 책임을 지지 않으며, 실물 거래 현장에서 송금할 것을 강력히 권장합니다.

거래시각	구분	보낸분/받는분	금액
2022-12-15 22:18:55	임금	psy	6,500원

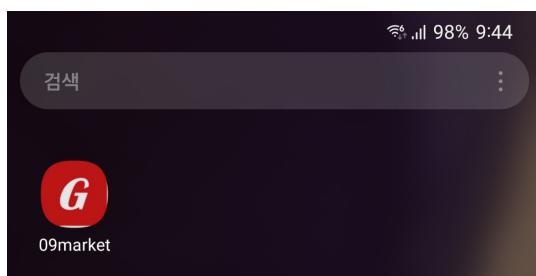
G. 채팅 화면



4. 성과 및 검증

A. PWA

- i. 사용자가 웹을 통해서 서비스를 이용할 수 있을 뿐만 아니라 스마트폰, 태블릿 PC 등의 다양한 디바이스에서 접근할 수 있도록 프로그래시브 웹앱을 도입하였다. 또한 PWA를 사용할 경우 Service Worker를 이용하여 앱을 사용하지 않는 상태에서도 푸시 알림을 받을 수 있기 때문에 사용자들의 앱 접근성을 높일 수 있다.
- ii. Web manifest를 등록해서 사용자의 디바이스 (컴퓨터, 태블릿, 스마트폰 등)에 native application과 같이 설치 가능하도록 제작하였다.



- iii. 최종적으로 구현한 서비스가 다양한 디바이스에서 의도한 UI/UX에 따라 렌더링되는 것을 확인하였으며 푸시 알림 또한 정상적으로 동작하는 것을 확인하였다.
- iv. 아래와 같이 브라우저 개발자 도구의 Lighthouse를 이용하여 PWA 검증을 진행하였다.

http://localhost:4173/login

PWA

설치 가능

- 웹 매티페스트 및 서비스 워커가 설치 가능 요건을 충족함

PWA 최적화됨

- 페이지와 start_url을(를) 제어하는 서비스 워커를 등록함
- 맞춤 스플래시 화면에 맞게 구성됨
- 주소 표시줄의 테마 색상을 설정함
- 콘텐츠의 크기가 표시 영역에 알맞음
- width 또는 initial-scale이(가) 포함된 <meta name="viewport"> 태그가 있음
- 유료한 apple-touch-icon 제공
- 매티페스트에 마스크 가능한 아이콘이 있음

직접 확인해야 하는 추가 항목 (3)

이 검사는 기준 [PWA 체크리스트](#)의 필수 항목이지만 Lighthouse에서 자동으로 확인되지는 않습니다. 절수에 영향을 미치지는 않지만 직접 확인하는 것이 중요합니다.

Captured at 2022년 12월 14 일 오후 3:45 GMT+9

조기 페이지 로드

애플리케이션 Moto G4 with Lighthouse 9.6.6

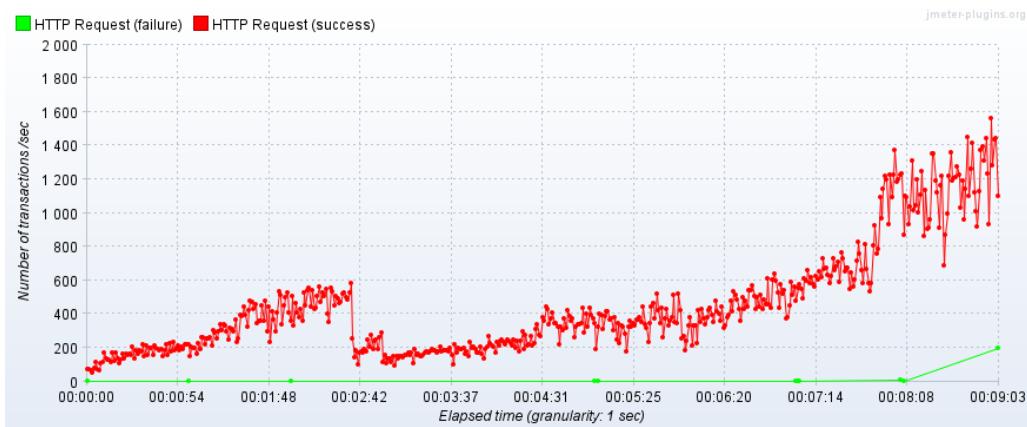
노트: 4G 제한

단일 페이지 로드

Using Chromium 108.0.0.0 with dextools

Generated by Lighthouse 9.6.6 | 문제 신고

B. 서버의 확장성 및 무중단 배포



- i. 위 그림과 같이 서버의 확장성을 테스트하였다.
- ii. 하나의 서버에서 부하를 늘려가며 through-put을 측정하였고 02:42에서 서버가 업데이트되며 무중단 배포되는 것을 확인할 수 있다.
- iii. 서버의 확장성은 발표 시 deal 서버를 중지한 후 채팅 기능이 수행되는 것으로 확인하였다.

5. 멘토 평가 의견 (<https://www.notion.so/41da52f922864354ac94273181bff0f0>)

A. 인상 깊은 부분

- i. 프론트엔드를 네이티브 모바일 앱 대신 PWA로 구성하여 접근성 극대화 (실제로 모바일 앱 설치를 강제하여 접근성을 낮추는 서비스가 많음)
- ii. PWA 검사 도구 사용
- iii. SPOF에 대한 인지
- iv. 매니지드 서비스를 활용한 관리 리소스 절감 및 작동 방식에 대한 이해
- v. MSA 구조 상에서 서비스를 독립적으로 잘 분리하여 코드 레벨이 아닌 인터페이스 레벨의 협업을 수행
- vi. 수치 기반의 throughput, availability 검증

B. 아쉬운 부분

- i. 시간 상 생산성 향상에 대한 고민을 더 하기 어려웠음
 - 배포 자동화 (e.g. GitHub Actions를 활용한 빌드&배포)
- ii. 테스트 자동화
 - unit test
 - integration test: synthetic test like <https://www.datadoghq.com/product/synthetic-monitoring>
- iii. 모니터링
 - e.g. Kafka가 살아있는지 확인하는 health check dashboard
- iv. 협업 도구

- Jira를 사용한 티켓 기반의 업무 처리

향후 개선 방안

1. Search Engine Optimization (SEO)
 - A. 시간 부족으로 검색 엔진 최적화 작업을 수행하지 못하였다. 경쟁력 있는 웹앱이 되기 위해서는 예비 사용자가 공동 구매를 위해 검색창에 공동 구매를 검색할 경우 노출이 되어야 하는데 해당 부분을 향후 개선하면 실제 서비스라 가정하였을 때 소비자 접근성을 높일 수 있다.
 - B. 또, SEO를 위해서는 현재 Client Side Rendering으로만 이루어진 웹 앱의 일부에 대해 Server Side Rendering을 진행할 필요가 있다. Site rendering 작업의 일부를 Server side로 넘김으로써 웹 사이트의 로딩 속도가 개선된다. 이를 구현하기 위해서는 Vue.js의 프레임워크 중 SSR에 강점이 있는 Nuxt.js로 migration을 고려해봄직하다.
2. 위치 기반 서비스
3. 배포 과정에서 디버깅을 진행한 뒤 재배포 하는 과정에서 재배포 된 결과물이 디바이스에 반영되지 않는 문제가 발생, PWA의 서비스 워커의 Cache-first 정책 및 AWS CloudFront의 Cached data가 outdated된 경우, 사용자에게 최신 버전이 제공되지 않는 것을 확인하였다. 이에 PWA에서 updateCacheOnChange 및 github actions를 통한 무효화 자동화를 적용하는 방안을 검토했으나, 프로젝트의 deadline이 임박한 시점이었기에 현상을 충분히 인지하는 수준에서 마무리하고 프로젝트 개선 방안으로 남겨두기로 하였다.
4. 단일 장애점 (Single Point of Failure) 문제
 - A. 현재 AWS ECR/ECS, fargate를 이용하여 각 컨테이너들의 무중단 배포 및 scale-out을 검증하였는데 Redis 서버와 Kafka 서버의 경우 EC2에 설치하여 운영 중이기 때문에 해당 부분에서 SPOF 문제가 발생할 수 있다. 향후 이를 클러스터링 및 이중화를 통해 해결할 방안을 모색할 수 있다.