프로젝트 최종 보고서

프로젝트명	MZ세대를 위한 커뮤니티 웹서비스: '절약 왕국'		
작성일	2024-06-16	버전	v1.1
변경 이력	2024-06-17		
팀장	구영민		
팀원	이정우		

< 요 약 서 >

1. 프로젝트명	MZ세대를 위한 커뮤니티 웹서비스		
2. 총수행기간	2024. 5. 1. - 2024. 6. 18.	3. 총 투입인원	총 2 명
4. 프로젝트 목표			

o 프로젝트 소개

- MZ 세대의 경제적 자립과 현명한 소비를 촉진하기 위한 커뮤니티 플랫폼 구축

최근 떠오르고 있는 '거지방' 컨셉을 차용하여, 사용자들이 각자의 소비 내역을 공유하고 서로의 지출에 대해 조언을 할 수 있는 점과 동시에, 최대한 저렴하게 물품을 구입할 수 있는 정보를 공유할 수 있도록 활성화 시키는 프로젝트이다.

5. 주요내용	

o 기획 배경

- '거지방'은 MZ 세대를 중심으로 유행하는 오픈채팅방으로 주로 서로의 소비 내역을 공유하고 절약을 독려한다. 이는 고물가와 취업난에 직면한 젊은 세대들이 경제적 현실에 유머러스하게 대처하는 방식이다. 또한 현대 사회에서 소비 문화는 고객 간의 경험 공유와 추천에 많은 영향을 받는다. 이러한 맥락으로 최대한 저렴하게 물품을 구입하고 싶어하는 젊은 세대들에게 서로 정보를 공유하는 커뮤니티를 구축하고자 한다.

o 사용 대상

- 고물가와 취업난 등 경제적 어려움에 직면한 20~30대 젊은 세대
- 물품을 저렴하게 구입할 수 있도록 정보를 얻고자 하는 이용자

o 주요 기능

- 자유로운 소통 게시판

1. 소통 증진 : 사용자들이 자유롭게 의견을 교환하고 토론할 수 있는 게시판 기능 제공한다.
2. 합리적 소비 습관 형성 : 자신의 소비 내역을 공개하여 사용자들의 피드백을 통해 더 합리적인 소비 습관 형성한다.
3. 커뮤니티 형성: 게시판을 통해 사용자 간 유대감과 소속감 형성한다.

- 물품 최저가 공유 게시판

1. 가격 비교 정보 : 사용자들이 다양한 제품의 최저가 정보를 공유할 수 있는 게시판 제공한다.
2. 합리적 소비 지원 : 사용자들이 최저가 정보를 활용하여 더 저렴하게 물품을 구매할 수 있도록 지원한다.
3. 정보 교류 활성화 : 사용자 간 가격 정보 공유를 통해 정보 교류 증진한다.

- 회원가입 기능

1. 회원 관리 : 회원과 비회원을 구분하여 회원이게는 글 작성 권한을 부여하고 비회원에게는 조회 권한만 제공한다.
2. 사용자 인증 : 로그인 기능을 통해 회원 인증 및 권한 부여한다.

6. 중점사항

o 서비스 개발 시 고려한 중점사항

- 마이크로 서비스 아키텍처 설계

단일 장애 지점을 최소화하여 서버 안정성 향상

각 서비스의 필요 리소스에 맞게 독립적으로 스케일링 가능

- 데이터 가용성 확보

각 서비스별로 적합한 데이터베이스 사용

독립적인 PV(PersistentVolume), PVC(PersistentVolumeClaim) 생성으로 데이터 영구 저장

- 자체 복구 기능 활용

쿠버네티스의 Sefl-healing 기능으로 비정상 컨테이너 자동 교체

안정적인 서비스 유지

- 무중단 배포 구현

쿠버네티스의 Rolling Update 기능 활용

새로운 버전 배포 시 서비스 중단 최소화

- 자동 스케일링 적용

쿠버네티스의 Auto-Scaling 기능 활용

HPA(HorizontalPodAutoscaler)로 CPU 사용량에 따른 레플리카 셋 개수 조절

- 성능 모니터링 및 측정

Jmeter와 Lighthouse를 이용한 반응 속도 및 서비스 성능 측정

지속적인 모니터링을 통한 개선 방향 도출

목 차

표 목차

사진 목차

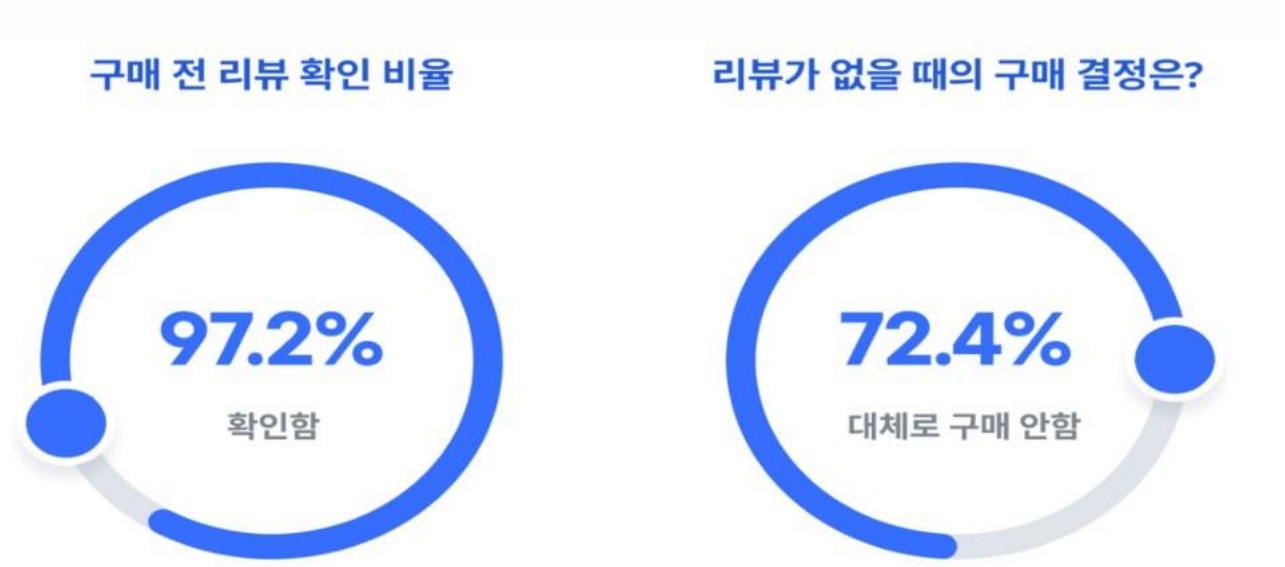
A. SW 개발의 필요성

[사진 1] 거지방 오픈채팅1

[사진 2] 거지방 오픈채팅2

MZ세대의 경제적 어려움 해소

최근 MZ 세대를 중심으로 '거지방' 오픈채팅방이 유행하고 있다. 이는 고물가와 취업난 등 경제적 어려움에 직면한 젊은 세대들이 유머러스하게 대처하는 방식이다. 이러한 상황에서 '절약왕국' 프로젝트가 제안하는 커뮤니티 플랫폼은 MZ 세대의 경제적 자립과 현명한 소비를 촉진할 수 있다.



[사진 3] 구매결정 비율

- **소비문화의 변화 반영**

현대 사회에서 소비문화는 고객 간의 경험 공유와 추천에 많은 영향을 받는다.

저희 프로젝트는 이러한 소비 문화의 변화를 반영하여, 사용자들이 서로 정로를 공유하고 저렴한 물품을 구입할 수 있도록 지원한다.

- **종합**

현재 MZ세대는 고물가와 취업난으로 경제적 어려움을 겪고 있다. 하지만 소비는 삶의 필수 부분이기 때문에 저렴하게 물품을 구매하는 것이 중요하다.

그런데 기존 SNS는 주로 소통에 집중하고 있어 이용자들의 경제적 욕구 충족에는 한계가 있다.

이 SW프로젝트는 '거지방' 컨셉을 도입하여 이용자들이 소비내역과 최저가 정보를 자유롭게 공유하고 피드백을 주고받을 수 있는 플랫폼을 제공한다.

특히 소비 패턴 공유와 물품 최저가 정보를 통해 이용자들이 더 저렴하고 합리적으로 소비 할 수 있도록 방향성을 제시한다.

이를 통해 MZ세대의 경제적 부담을 줄이고, 지속 가능한 소비 문화 정착에 필요성이 있기 때문에 'MZ세대를 위한 커뮤니티 웹서비스'가 더욱 필요하다.

B. SW개발의 목표 및 내용

1) 최종 목표

1-1) 최종 목표

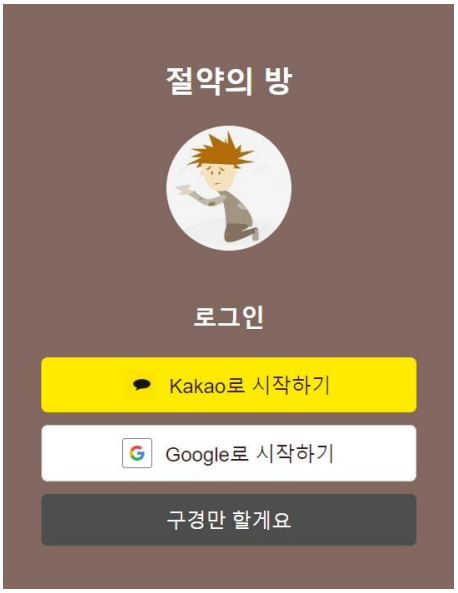
절약왕국 프로젝트는 MSA와 쿠버네티스를 기반으로 구축한 디지털 플랫폼으로, 최저가 공유와 거지방 커뮤니티를 통합하여 경제적이고 환경 친화적인 소비를 촉진하는 것을 목표로 한다. MSA를 통해 각 기능이 독립적으로 개발되고 운영되어 유연성과 확장성이 향상된다. 쿠버네티스를 활용한 컨테이너 오케스트레이션은 높은 가용성을 보장하고, 자동 스케일링 및 롤링 업데이트를 통해 서비스 중단 없이 효율적으로 자원을 관리한다. 이로써 절약왕국은 사용자에게 안정적이고 지속 가능한 서비스를 제공한다.

'절약 왕국' 프로젝트에서 개발할 최종 핵심 기능은 3가지이다.

1-2) 프로젝트 핵심 기능

'절약 왕국' 프로젝트에서 개발할 최종 핵심 기능은 3가지이다.

1. **비회원/회원 가입 기능**



[사진 4] 로그인 창

[사진 5] 로그인 완료 창

2. 거지방 커뮤니티 자유게시판(Bulletin Board) 기능

[사진 6] 자유의 방 게시판

[사진 7] 자유의 방 글 작성

[사진 8] 자유의 방 글 작성 확인

[사진 9] 자유의 방 글 조회

[사진 10] 댓글 작성

[사진 11] 댓글 작성 확인

3. 물품 최저가 공유 게시판(Sharing Board) 기능

[사진 12] 공유의 방

[사진 13] 상품 등록

[사진 14] 상품 등록 확인

[사진 15] 상품 조회

[사진 16] URL이동

[사진 17] 상품 가격 수정

[사진 18] 상품 가격 수정 확인

[사진 19] 상품 삭제 후

2) 프로젝트 내용

2-1) 프로젝트 주차 별 계획

5월 1주차 : 아이디어 계획

5월 2주차 : 프로젝트 계획서 작성

5월 3주차 : 프론트서버, OAuth서버 구현

5월 4주차 : 자유 게시판 서버, 공유 게시판 서버, 댓글 서버 구현

6월 1주차 : 쿠버네티스 기반 컨테이너 오케스트레이션 및 스토리지 관리

6월 2주차 : 성능 테스트

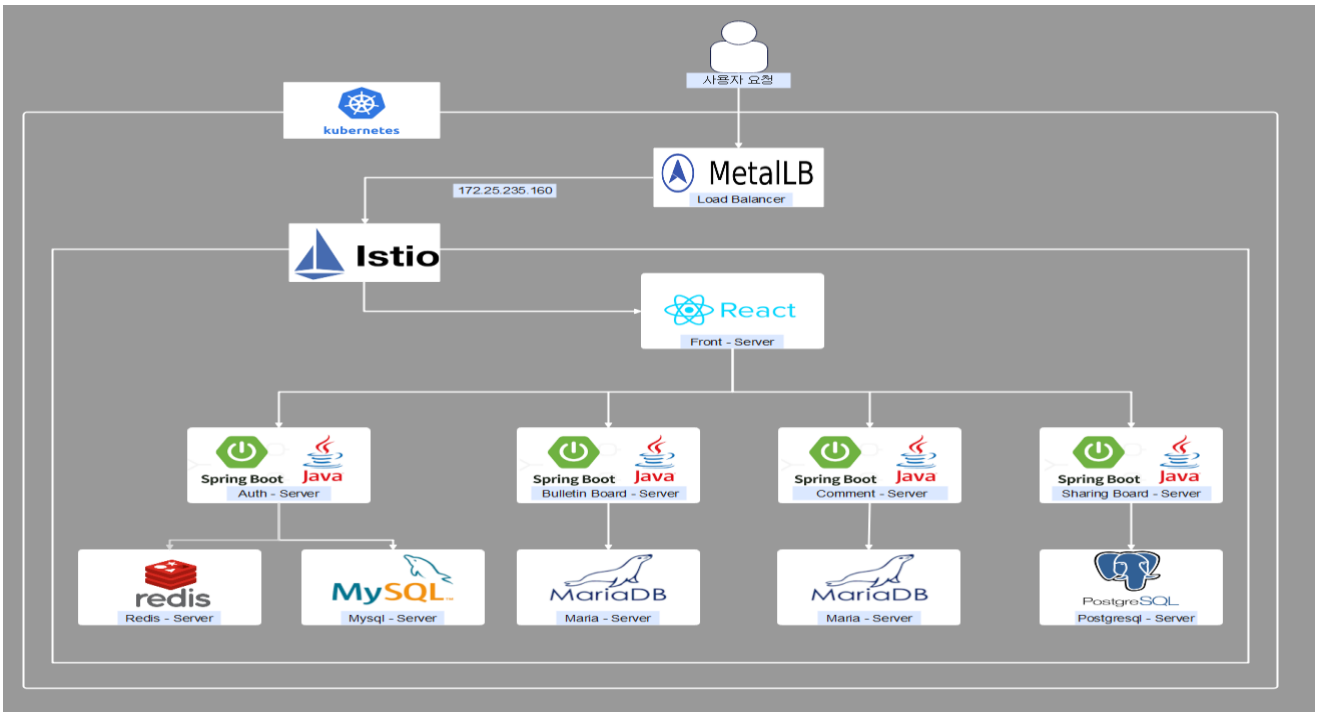
6월 3주차 : 시연영상 제작 및 발표 준비

2-2) 프로젝트 구조

'절약 왕국' 프로젝트는 하나의 큰 어플리케이션을 여러 개의 작은 어플리케이션으로 쪼개어 변경과 조합이 가능하도록 만든 아키텍처인 MSA 방식을 사용한다.

백엔드는 Spring Boot를 사용하여 각각의 기능을 수행하는 독립적인 마이크로 서비스로 구성한다. 이 마이크로 서비스는 서로 통신하고 데이터를 교환하기 위해 RESTful API를 사용한다. 또한 각 마이크로 서비스는 자체적인 데이터베이스를 가지고 있으며, 비즈니스 로직을 처리하고 데이터를 관리한다. 이를 통해 백엔드는 각각의 서비스가 독립적으로 개발, 배포, 확장될 수 있도록 설계한다.

프론트엔드는 리액트(React)를 사용하여 개발한다. 리액트는 사용자 인터페이스를 만들기 위한 자바스크립트 라이브러리로, 컴포넌트 기반 아키텍처를 사용하여 재사용 가능한 UI 요소를 만들고 관리할 수 있다. 사용자는 웹 브라우저를 통해 프론트엔드에 접근한다. 프론트엔드는 백엔드와 통신하여 사용자의 요청을 처리하고 데이터를 가져와 화면에 표시한다. 이를 통해 사용자는 직관적이고 효율적인 방식으로 서비스를 이용할 수 있다.



[사진 20] 프로젝트 구조

1. Front - Server(front-server)

- React 기반 웹 서버

2. 인증 서버 (auth-server)

- Java 기반 Spring Boot 프레임워크
- OAuth 2.0 기반 Kakao, Google 로그인 및 회원가입
- JWT (Json Web Token) 기반 인증 방식 적용
- MySQL 데이터베이스를 PV(Persistent Volume)에 배포하여 사용자의 정보를 관리

3. 자유게시판 서버 (Bulletin Board-server : bb-server)

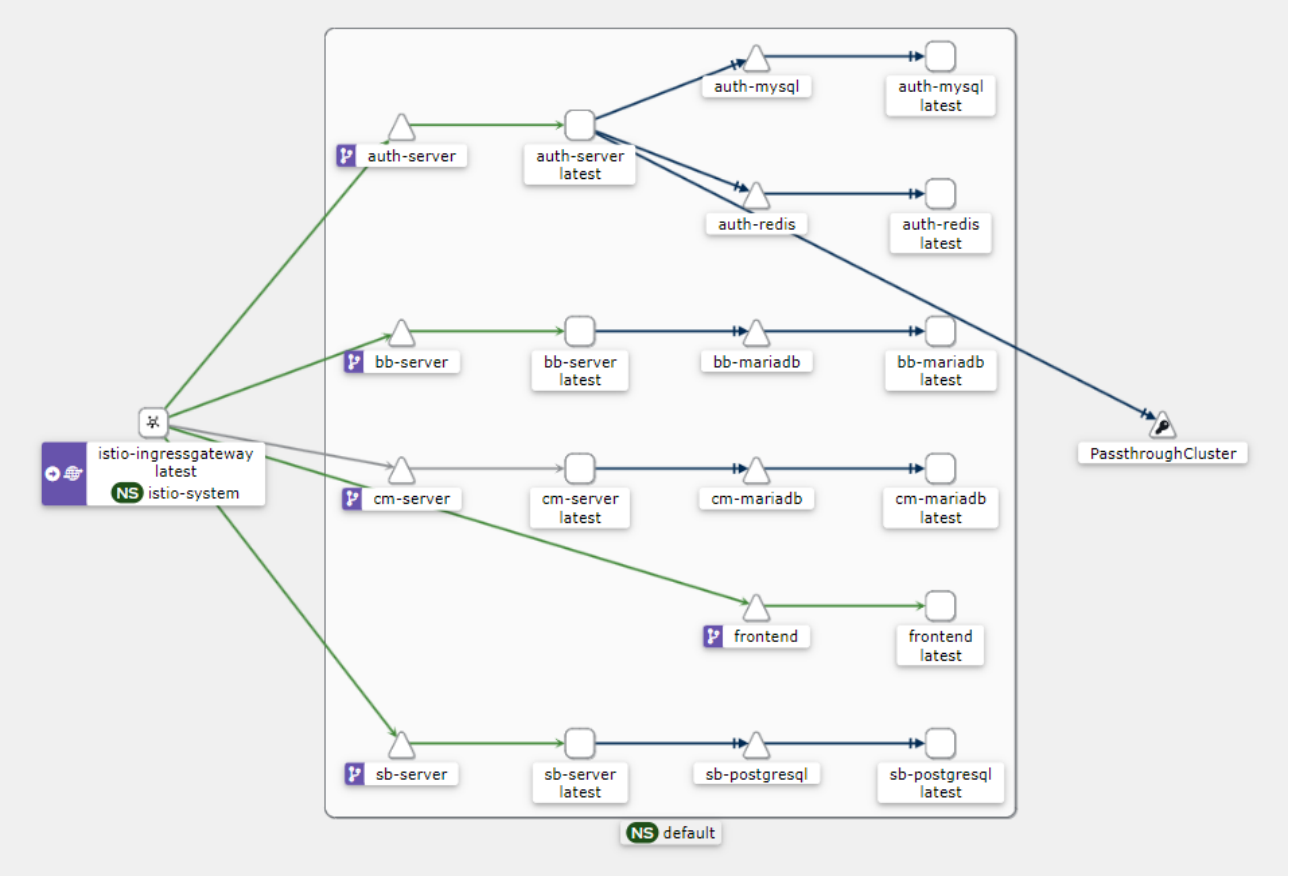
- Java 기반 Spring Boot 프레임워크
- Restful API 기반의 Bulletin Board 글 조회, 등록, 수정, 삭제 기능
- MariaDB 데이터베이스를 PV(Persistent Volume)에 배포하여 거지방 자유게시판 정보를 관리

4. 자유게시판 댓글 서버 (Comment-server : cm-server)

- Java 기반 Spring Boot 프레임워크
- Restful API 기반의 댓글 조회, 등록, 수정, 삭제 기능
- MariaDB 데이터베이스를 PV(Persistent Volume)에 배포하여 거지방 자유게시판 정보를 관리

5. 최저가 공유 서버 (Sharing Board- server : sb-server)

- Java 기반 Spring Boot 프레임워크
- Restful API 기반의 최저가 상품 조회, 등록, 수정, 삭제 기능
- PostgreSQL 데이터베이스를 PV(Persistent Volume)에 배포하여 최저가 공유 게시판 정보를 관리



[사진 21] service mesh 프로젝트 구조

각 서비스가 데이터베이스와 연결되어 있으며, istio-ingressgateway를 통해 외부 트래픽이 내부 서비스로 라우팅됩니다.

2-3) 프로젝트 중점 사항

마이크로 서비스 아키텍처 설계

- 단일 장애 지점을 최소화하여 서버 안정성 향상
- 각 서비스의 필요 리소스에 맞게 독립적으로 스케일링 가능

데이터 가용성 확보

- 각 서비스별로 적합한 데이터베이스 사용
- 독립적인 PV(PersistentVolume), PVC(PersistentVolumeClaim) 생성으로 데이터 영구 저장

자체 복구 기능 활용
- 쿠버네티스의 Self-healing 기능으로 비정상 컨테이너 자동 교체
- 안정적인 서비스 유지

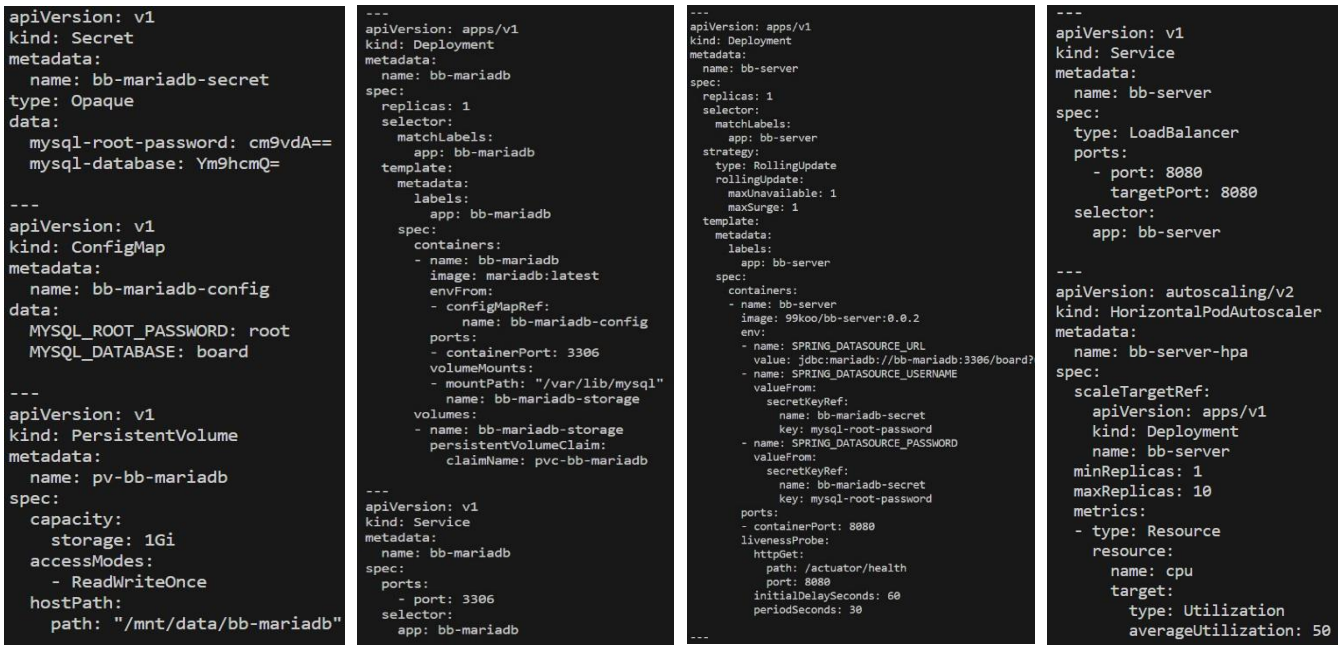
무중단 배포 구현
- 쿠버네티스의 Rolling Update 기능 활용
- 새로운 버전 배포 시 서비스 중단 최소화

자동 스케일링 적용
- 쿠버네티스의 Auto-Scaling 기능 활용
- HPA(HorizontalPodAutoscaler)로 CPU 사용량에 따른 레플리카 셋 개수 조절

성능 모니터링 및 측정
- Jmeter와 Lighthouse를 이용한 반응 속도 및 서비스 성능 측정
- 지속적인 모니터링을 통한 개선 방향 도출

3) 쿠버네티스 기능 시연

저희 프로젝트 서버는 프론트 서버, 로그인 서버, 자유게시판 서버, 댓글 서버, 공유 게시판 서버 총 5개로 이루져 있으며 모든 같은 형식으로 각각의 서버별로 yaml 파일에 정의되어 있습니다. 그 중에서 bb-server 기준으로 시연하겠습니다.



```yaml
apiVersion: v1
kind: Secret
metadata:
  name: bb-mariadb-secret
type: Opaque
data:
  mysql-root-password: cm9vdA==
  mysql-database: Ym9hcmQ=

---
apiVersion: v1
kind: ConfigMap
metadata:
  name: bb-mariadb-config
data:
  MYSQL_ROOT_PASSWORD: root
  MYSQL_DATABASE: board

---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-bb-mariadb
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/mnt/data/bb-mariadb"
```

```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: bb-mariadb
spec:
  replicas: 1
  selector:
    matchLabels:
      app: bb-mariadb
  template:
    metadata:
      labels:
        app: bb-mariadb
    spec:
      containers:
      - name: bb-mariadb
        image: mariadb:latest
        envFrom:
        - configMapRef:
            name: bb-mariadb-config
        ports:
        - containerPort: 3306
        volumeMounts:
        - mountPath: "/var/lib/mysql"
          name: bb-mariadb-storage
      volumes:
      - name: bb-mariadb-storage
        persistentVolumeClaim:
          claimName: pvc-bb-mariadb

---
apiVersion: v1
kind: Service
metadata:
  name: bb-mariadb
spec:
  ports:
    - port: 3306
  selector:
    app: bb-mariadb
```

```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: bb-server
spec:
  replicas: 1
  selector:
    matchLabels:
      app: bb-server
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 1
  template:
    metadata:
      labels:
        app: bb-server
    spec:
      containers:
      - name: bb-server
        image: 99koo/bb-server:0.0.2
        env:
        - name: SPRING_DATASOURCE_URL
          value: jdbc:mariadb://bb-mariadb:3306/board?
        - name: SPRING_DATASOURCE_USERNAME
          valueFrom:
            secretKeyRef:
              name: bb-mariadb-secret
              key: mysql-root-password
        - name: SPRING_DATASOURCE_PASSWORD
          valueFrom:
            secretKeyRef:
              name: bb-mariadb-secret
              key: mysql-root-password
        ports:
        - containerPort: 8080
        livenessProbe:
          httpGet:
            path: /actuator/health
            port: 8080
          initialDelaySeconds: 60
          periodSeconds: 30

---
```

```yaml
---
apiVersion: v1
kind: Service
metadata:
  name: bb-server
spec:
  type: LoadBalancer
  ports:
    - port: 8080
      targetPort: 8080
  selector:
    app: bb-server

---
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: bb-server-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: bb-server
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 50
```

[사진 22] bb-server.yaml 전체 구조

- bb-server.yaml파일의 전체 구조입니다.
- 쿠버네티스의 핵심 기능 6가지를 포함합니다.

3-1) Auto Scale Out/In

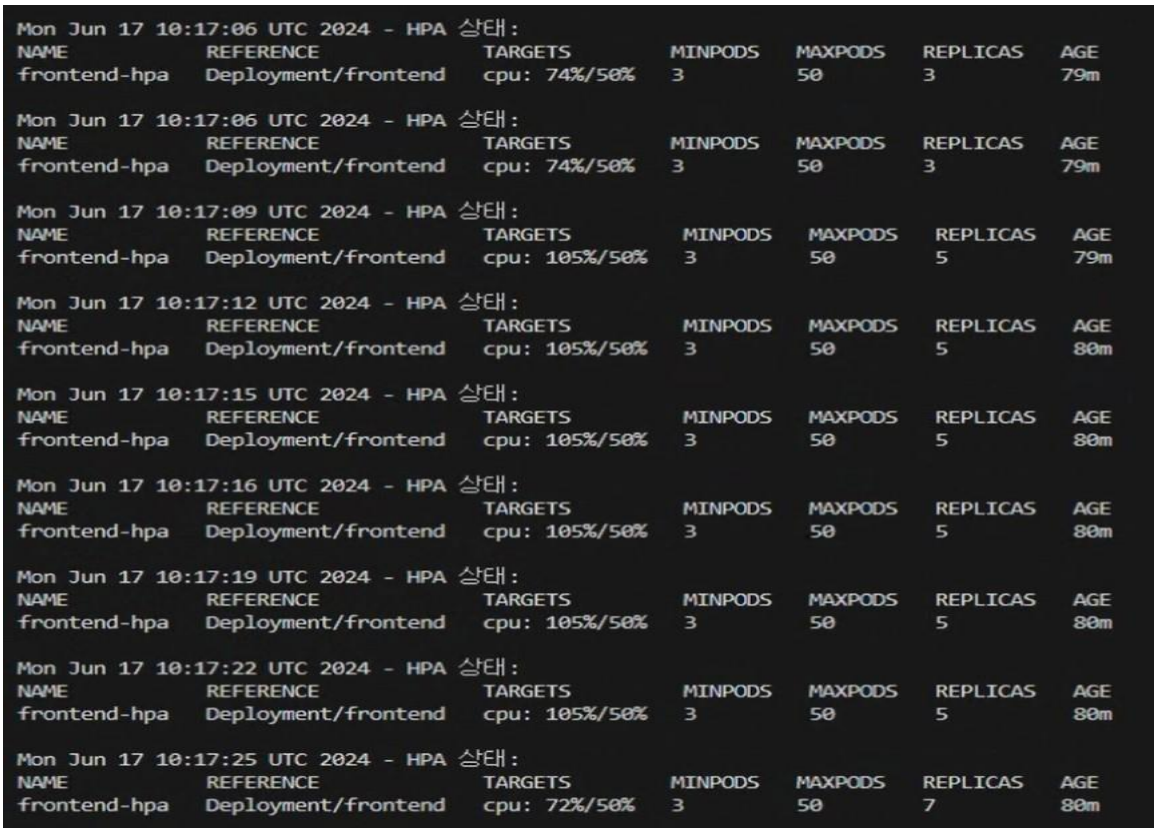
```yaml
---
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: bb-server-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: bb-server
  minReplicas: 3
  maxReplicas: 50
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 50
```

[사진 23] Auto Scale

- Kubernetes 클러스터에서 자동 스케일링을 구현했습니다. 이 설정을 통해 파드 수는 부하에 따라 자동으로 조정됩니다.

- 최소 파드 수는 3개, 최대 파드 수는 50개로 제한되며, CPU 사용률이 평균 50%가 되도록 파드 수를 조정합니다.
- 이를 통해 애플리케이션의 부하에 따라 효율적인 리소스 사용이 보장됩니다.

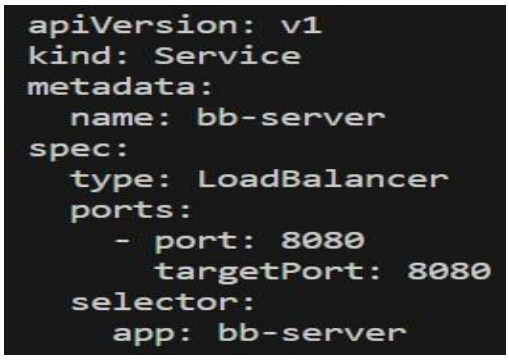
오토스케일링 결과 시연



[사진 24] Auto Scaling 시연

- HPA가 CPU 사용률에 따라 Replicas수를 조정하는 과정 입니다.
- 부하가 증가할 때 Replicas 수가 늘어납니다.

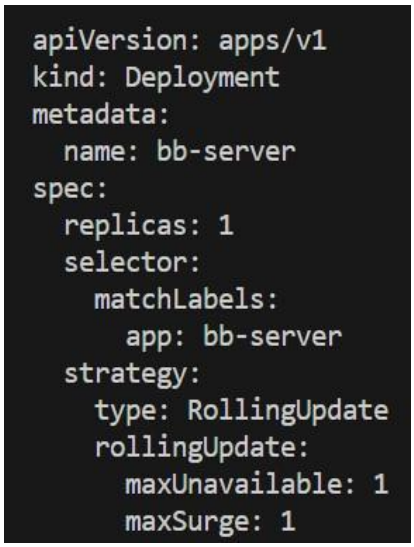
3-2) Load Balancing



[사진 25] Load Balancing

- LoadBalancer로 설정하여, 클라우드 제공자가 관리하는 로드 밸런서를 통해 외부 트래픽을 내부 파드로 분산시킵니다.

3-3) Auto Rolling Update



[사진 26] Auto Rolling Update

- RollingUpdate: Deployment의 업데이트 전략으로 롤링 업데이트를 사용함을 지정합니다.
- maxUnavailable: 업데이트 과정에서 최대 몇 개의 파드가 동시에 중단될 수 있는지를 지정합니다. 여기서는 1개 파드가 중단될 수 있음을 의미합니다.
- maxSurge: 업데이트 과정에서 최대 몇 개의 파드를 추가로 생성할 수 있는지를 지정합니다. 여기서는 1개 파드를 추가로 생성할 수 있음을 의미합니다.
- 이렇게 하면 서비스 중단을 최소화하면서 새로운 버전으로 안전하게 업데이트할 수 있습니다.

3-4) Auto Healing

```
livenessProbe:
  httpGet:
    path: /actuator/health
    port: 8080
  initialDelaySeconds: 60
  periodSeconds: 30
```

[사진 27] Auto Healing

- initialDelaySeconds: 컨테이너가 시작된 후 처음 상태 확인을 시작하기 전 대기 시간을 60초로 설정했습니다.
- periodSeconds: 상태 확인 주기는 30초로 설정했습니다.
- 이 설정을 통해 Kubernetes는 컨테이너가 비정상적인 상태일 때 자동으로 재시작하여 서비스의 가용성과 안정성을 보장합니다.
- 이 설정을 통해 Kubernetes는 컨테이너가 비정상적인 상태일 때 자동으로 재시작하여 서비스의 가용성과 안정성을 보장합니다.

3-5) Persistence Volume

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-bb-mariadb
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/mnt/data/bb-mariadb"
```

[사진 28] PV

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-bb-mariadb
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

[사진 29] PVC

- 데이터의 영속성 보장과 유연한 스토리지 할당을 위해 Persistent Volume (PV)과 Persistent Volume

Claim (PVC)를 사용했습니다.

- 이를 통해 애플리케이션 재시작 시에도 데이터를 안전하게 유지하고, 필요한 스토리지를 동적으로 요청할 수 있습니다.

3-6) Container Orchestration

```
ubuntu@rndudals:~$ kubectl get nodes
NAME              STATUS   ROLES           AGE    VERSION
j-ra1n-worker1    Ready    <none>          4d7h   v1.30.2
j-ra1n-worker2    Ready    <none>          4d7h   v1.30.2
rndudals          Ready    control-plane   4d7h   v1.30.2
```

[사진 30] 마스터노드와 워커노드

- Kubernetes 클러스터를 활용하여 애플리케이션을 배포하고 관리합니다.
- 클러스터는 마스터 노드 1개와 워커 노드 2개로 이루어져 있으며, 이를 통해 애플리케이션의 배포, 확장, 관리, 복구 등의 오케스트레이션을 자동화합니다.

```
ubuntu@rndudals:~$ kubectl get pod -A
NAMESPACE      NAME                                       READY   STATUS    RESTARTS   AGE
default        auth-mysql-c5d5bf85-ggvnd                  2/2     Running   0          118s
default        auth-redis-998dd87b6-8xhsw                 2/2     Running   0          118s
default        auth-server-7b889dbc99-t4blk               2/2     Running   0          118s
default        bb-mariadb-6c7c6d59d4-vvhwk                2/2     Running   0          2d6h
default        bb-server-66974889b7-8vng7                 2/2     Running   0          2d6h
default        cm-mariadb-f569d996-gg57m                  2/2     Running   0          2d7h
default        cm-server-64dbbf59dd-4tlds                 2/2     Running   0          2d6h
default        frontend-654b459b58-2gmcr                  2/2     Running   0          46m
default        frontend-654b459b58-5m8m5                  2/2     Running   0          46m
default        frontend-654b459b58-vqq2t                  2/2     Running   0          46m
default        sb-postgresql-658d578984-fmdl7             2/2     Running   0          2d4h
default        sb-server-7968d89b86-n28lx                 2/2     Running   0          2d4h
istio-system   grafana-6d4c849f67-472ks                   1/1     Running   0          9h
istio-system   istio-egressgateway-7cfd5c8676-59jhp       1/1     Running   0          4d7h
istio-system   istio-ingressgateway-86f46c495b-7jlwx      1/1     Running   0          4d7h
istio-system   istiod-f68799b78-fnqbc                     1/1     Running   0          4d7h
istio-system   jaeger-78c77c46bc-jgcm6                    1/1     Running   0          9h
istio-system   kiali-7dffdbdc9d-c4csb                     1/1     Running   0          9h
istio-system   kiali-operator-855fcbfcc5-tvhhk            1/1     Running   0          11h
istio-system   prometheus-777db476b6-j5rnb                2/2     Running   0          10h
kube-system    calico-kube-controllers-5b9b456c66-74lk6   1/1     Running   0          4d7h
kube-system    calico-node-8w98j                          1/1     Running   0          4d7h
kube-system    calico-node-rd6bp                          1/1     Running   0          4d7h
kube-system    calico-node-x4sdv                          1/1     Running   0          4d7h
kube-system    coredns-7db6d8ff4d-9l6ch                   1/1     Running   0          4d7h
kube-system    coredns-7db6d8ff4d-kpgkv                   1/1     Running   0          4d7h
kube-system    etcd-rndudals                              1/1     Running   0          4d7h
kube-system    kube-apiserver-rndudals                    1/1     Running   0          4d7h
kube-system    kube-controller-manager-rndudals           1/1     Running   0          4d7h
kube-system    kube-proxy-2mbh5                           1/1     Running   0          4d7h
kube-system    kube-proxy-lhwxm                           1/1     Running   0          4d7h
kube-system    kube-proxy-ql7zn                           1/1     Running   0          4d7h
kube-system    kube-scheduler-rndudals                    1/1     Running   0          4d7h
metallb-system controller-5484c5f99f-qjbfm                1/1     Running   0          4d7h
metallb-system speaker-8z9q5                              1/1     Running   0          4d7h
metallb-system speaker-m25m7                              1/1     Running   0          4d7h
metallb-system speaker-rxvz5                              1/1     Running   0          4d7h
```

[사진 31] pod 조회

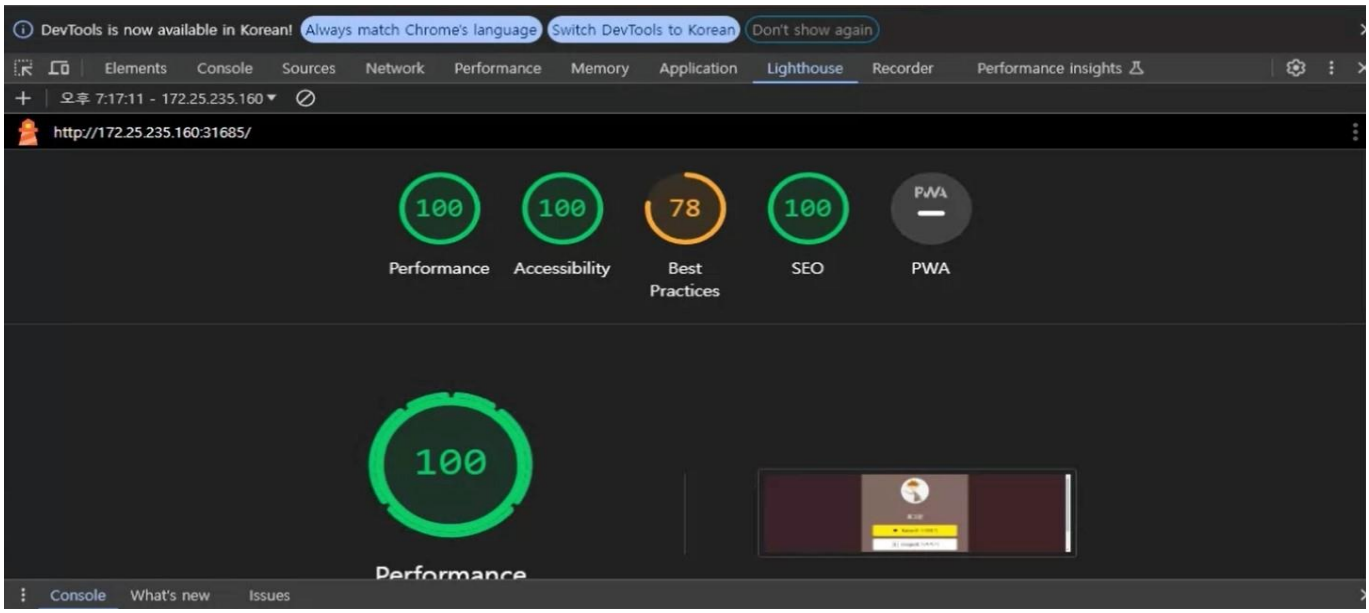
- 클러스터의 전체 파드 목록을 보여줍니다.
- 클러스터 안의 모든 pod의 상태가 Running입니다.

C. 프로젝트 결과물의 평가기준

1) 정량적 평가 기준

평가 항목	정량적 목표	평가 방법
초당 접속자 수	총 500명이 자유게시판, 공유게시판에 동시 접속할 수 있다.	JMeter와 LightHouse를 사용하여 측정
First Contentful Paint (FCP)	총 500명이 자유게시판, 공유게시판에 동시 접속하더라도 FCP 기준 1500ms 이내를 보장한다.	JMeter와 LightHouse를 사용하여 측정
Total Blocking Time (TBT)	총 500명이 자유게시판, 공유게시판에 동시 접속하더라도 TBT 기준 1500ms 이내를 보장한다.	JMeter와 LightHouse를 사용하여 측정
Largest Contentful Paint (LCP)	총 500명이 자유게시판, 공유게시판에 동시 접속하더라도 LCP 기준 1500ms 이내를 보장한다.	JMeter와 LightHouse를 사용하여 측정
Cumulative Layout Shift (CLS)	총 500명이 자유게시판, 공유게시판에 동시 접속하더라도 CLS 기준 1500ms 이내를 보장한다.	JMeter와 LightHouse를 사용하여 측정
Speed Index	총 500명이 자유게시판, 공유게시판에 동시 접속하더라도 Speed Index값이 2000ms 이내로 유지된다.	JMeter와 LightHouse를 사용하여 측정

[표 1] 정량적 평가 기준



[사진 32] 성능 테스트 결과

[사진 33] 성능 테스트 세부 결과

- 500명이 자유게시판과 공유게시판에 동시에 접속했을 때의 성능 평가 결과를 요약한 것입니다. 목표 성취 여부를 평가하기 위해 다양한 지표를 사용했습니다.
- Lighthouse를 통해 측정한 결과, 모든 주요 성능 지표가 목표를 충족했습니다.

First Contentful Paint (FCP)

- 목표 : 800ms
- 결과 : 300ms

Total Blocking Time (TBT)

- 목표 : 800ms
- 결과 : 0ms

Largest Contentful Paint (LCP)

- 목표 : 1500ms
- 결과 : 0.6ms

Cumulative Layout Shift (CLS)

- 목표 : 1500ms
- 결과 : 0

Speed Index

- 목표 : 2000ms
- 결과 : 300ms

모든 평가 항목에서 목표를 달성하였습니다.

2) 정성적 성과 기준

평가 항목	정성적 목표	평가 방법
서비스 배포	쿠버네티스상에 MSA 구조의 서버 배포	쿠버네티스 상에서 Pod가 정상적으로 작동하는 확인, 외부에서 사용자가 접속 가능한지 확인
로그인 서버	로그인 기능의 정상 작동	회원 가입 시 입력한 정보가 MySQL 데이터베이스에 정상적으로 입력되고 로그인이 가능한지 확인
자유게시판 서버	자유게시판 기능의 정상 작동	자유게시판에 게시글을 작성, 수정, 삭제, 조회할 수 있는지 확인
자유 게시판 댓글 서버	자유게시판 댓글 기능의 정상 작동	자유게시판의 글에 댓글을 작성, 수정, 삭제, 조회할 수 있는지 확인
최저가 공유 게시판 서버	최저가 공유 게시판 기능의 정상 작동	저가 정보 게시판에 게시글을 작성, 수정, 삭제, 조회할 수 있는지 확인

[표 2] 정성적 평가 기준

결과 요약

- Istio ingressgateway를 사용하여 외부 트래픽을 라우팅하여 외부 접속이 가능합니다.
- 발표영상과 시연 사진을 보시면 로그인 서버 자유게시판 서버, 댓글 서버, 최저가 공유 게시판 서버는 정상적으로 작동하는 것을 확인할 수 있습니다.

D. 프로젝트 성과의 활용방안 및 기대효과

1) 활용방안

프로젝트의 주요 성과는 MZ 세대를 위한 경제적 자립을 지원하는 커뮤니티 플랫폼 구축입니다. 이 플랫폼을 통해 다양한 활용방안이 기대된다.

1. 교육 및 정보 제공

플랫폼은 사용자들에게 경제 교육과 소비 습관 개선에 필요한 정보를 제공합니다. 예를 들어, 재정 관리 방법, 절약 팁, 최저가 상품 정보 등이 포함될 수 있다.

2. 소셜 네트워킹

사용자들은 경험과 정보를 공유하며 서로를 지원하는 강력한 커뮤니티를 형성합니다. 이 네트워크는 직업 정보, 경제 뉴스, 할인 정보 등을 공유하는 플랫폼으로 활용될 수 있다.

3. 이용자 이익

서비스 이용자들은 소비정보 공유, 최저가 비교, 커뮤니케이션으로 경제적 이익을 얻을 수 있다.

4. 데이터 활용

플랫폼 이용 데이터는 MZ세대 소비 트렌드 분석 등에 활용될 수 있습니다. 수집된 데이터를 분석하여 정부나 기업들이 젊은 세대의 소비 패턴과 경제 상황에 대한 인사이트를 제공할 수 있다.

5. 기술 이전

개발 기술과 아키텍처는 이후 다른 분야의 플랫폼 서비스 개발에 활용될 수 있다.

2) 기대효과

'절약 왕국' 프로젝트의 성과물이 과학기술적, 경제적, 사회적 측면에서 미치는 기대효과는 다음과 같다.

과학기술적 측면

1. 기술 혁신과 발전

이 프로젝트는 최신 IT 기술과 통합 데이터 관리 시스템을 활용하여 개발된다. 특히 마이크로서비스 아키텍처(MSA)와 RESTful API의 적용은 개별 서비스의 독립적 운영을 가능하게 하여 유지보수와 업데이터가 용이하며, 이는 소프트웨어 엔지니어링 분야에서의 모범 사례로 자리 잡을 잠재력을 가지고 있다.

경제·산업적 측면

1. 시장 기회 창출

이 플랫폼은 저렴한 가격으로 상품을 구매하기 원하는 소비자들과 저비용 마케팅 채널을 필요로 하는 소규모 제조업체 및 소매업자들 간의 연결 고리를 제공한다. 이는 새로운 시장 기회를 창출하고, 경제 활동을 촉진시키는 데 기여할 수 있다.

2. 산업 경쟁력 강화

최저가 정보 공유와 소비자 리뷰는 시장의 투명성을 높이고, 공정 경쟁을 촉진한다. 이는 전체 산업의 경쟁력 강화에 기여하며, 효율적인 가격 결정 및 소비자 만족도 향상을 이끈다.

사회적 측면

1. 소비자 교육 및 정보 접근성 향상

이 플랫폼은 사용자들에게 재정 관리, 합리적 소비 방법 등에 대한 교육을 제공한다. 이는 사회 전반의 금융 리터러시 향상에 기여하며, 정보 접근성 증가는 더 많은 사람들이 경제적 자립을 달성하는 데 도움을 준다.

2. 사회적 연대감 및 네트워킹 강화

공통의 경제적 목표를 가진 사용자들 간의 상호작용은 사회적 연대감을 강화한다. 이는 특히 경제적으로 어려운 시기에 개인들이 서로를 지원하고 격려하는 강력한 네트워크를 형성하는 데 중요하다.

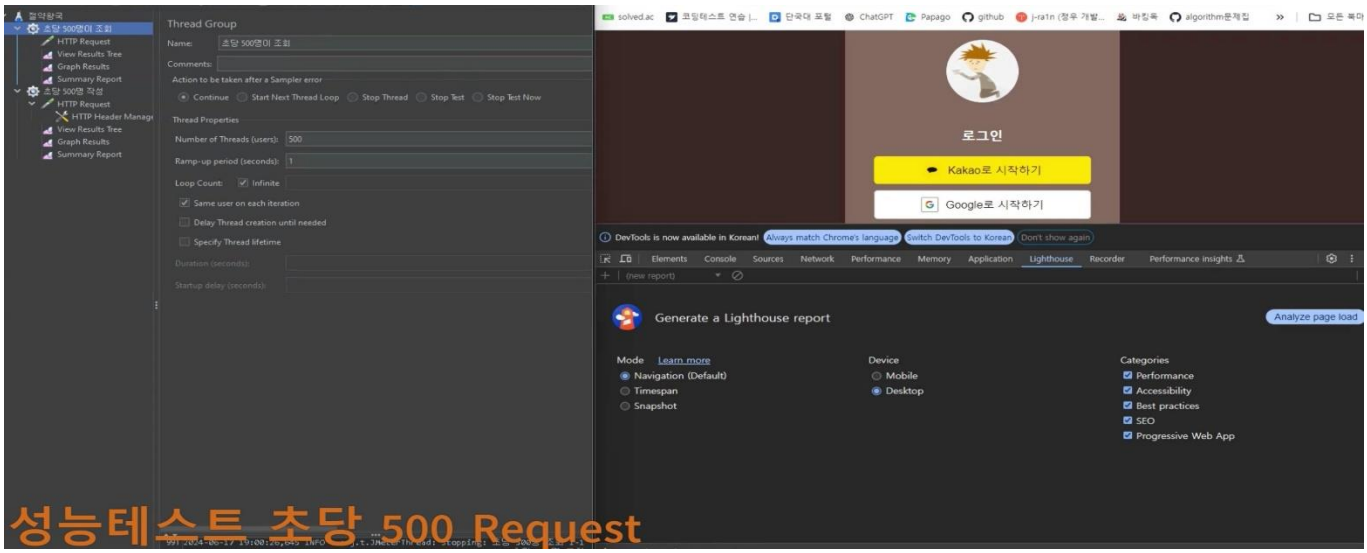
파급효과

1. 경제적 안정성 증대: 소비자의 합리적 소비 습관 형성은 장기적인 경제적 안정성을 증대시킨다. 이는 가계 경제를 강화하고, 경제적 불확실성 시대에 안정적인 소비 기반을 마련한다.
2. 문화적 변화 유도: 이 플랫폼을 통한 적극적인 정보 공유 및 교류는 새로운 소비 문화를 형성한다. 소비자들이 보다 의식적이고 정보에 기반한 결정을 내리게 되며, 이는 사회 전반의 소비 문화를 변화시키는 동력이 된다.

이러한 기대효과와 파급효과는 프로젝트가 성공적으로 실행되고 지속적으로 관리될 때 극대화될 수 있다. 따라서 '절약 왕국' 프로젝트는 MZ 세대뿐만 아니라 사회 전반에 긍정적인 영향을 미칠 잠재력을 가지고 있다.

E. 기타

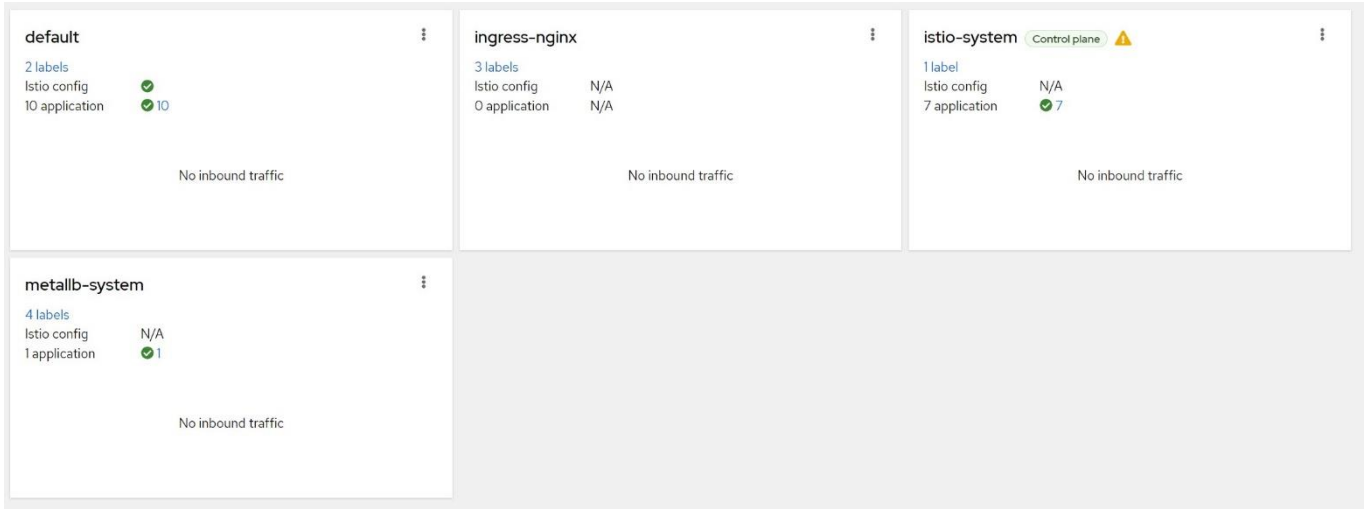
JMeter



[사진 34] JMeter

JMeter는 주로 웹 애플리케이션의 성능을 평가하는 데 사용되는 도구이다. 기능적 측면에서 주요 목적은 부하 테스트, 스트레스 테스트, 그리고 안정성 테스트를 수행하여 애플리케이션의 성능을 측정하는 것이다.

Istio

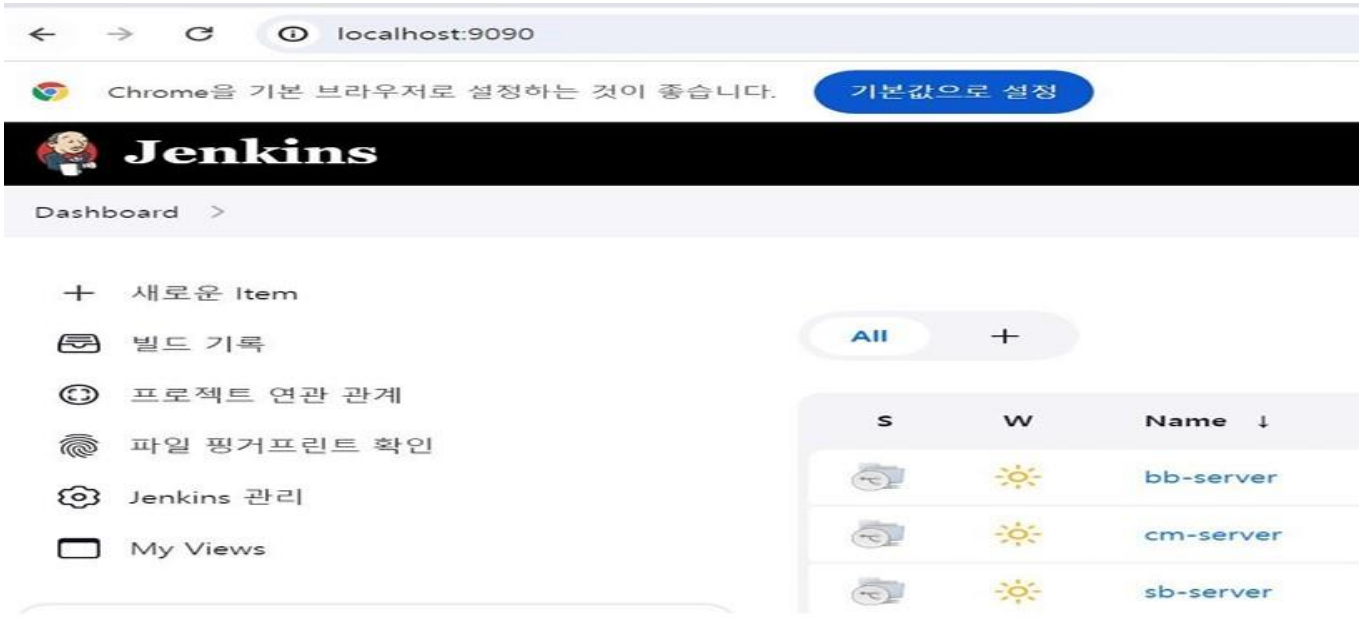


[사진 35] istio의 kiali

Istio는 컨테이너화된 애플리케이션을 위한 오픈 소스 서비스 메쉬(service mesh)이다. MSA 프로젝트를 진행하며

같은 컨테이너 오케스트레이션 플랫폼에서 실행되는 마이크로서비스 간의 통신을 관리, 보안 및 모니터링하는 데 사용하였다. 마이크로서비스 아키텍처에서 통신 복잡성을 추상화하고, 애플리케이션 로직에 더 집중할 수 있도록 도움을 주었다.

Jenkins



[사진 36] jenkins

Jenkins는 자동화 서버로 널리 사용되며, 주로 소프트웨어 개발의 지속적 통합(CI)과 지속적 배포(CD) 프로세스를 자동화하는 데 사용된다. Jenkins를 사용하면 빌드, 테스트, 배포 과정을 자동으로 실행하여 개발 워크플로우를 효율화할 수 있다. 플러그인으로 확장 가능하며, 여러 언어와 플랫폼에 걸쳐 광범위하게 지원된다.

- 참고문헌(Reference)

[1] Oracle, https://www.oracle.com/cloud/what-is-cloud-computing/

[2] BrowserStack, https://www.browserstack.com/guide/performance-testing-tools

[3] Istio, Service Mesh, https://istio.io

[4] Kubernetes, https://kubernetes.io/docs/concepts/workloads/autoscaling/

[5] Apache JMeter, https://jmeter.apache.org/

[6] Jenkins, https://www.jenkins.io/doc/book/pipeline/jenkinsfile/

[6] MSA, https://learn.microsoft.com/ko-kr/azure/architecture/guide/architecture-styles/