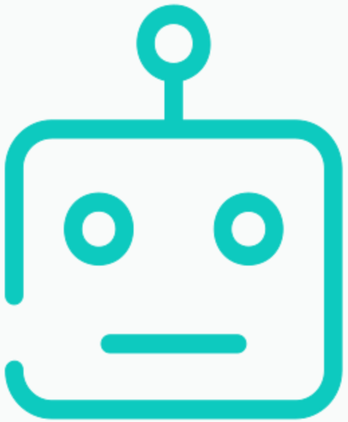
by code receipe

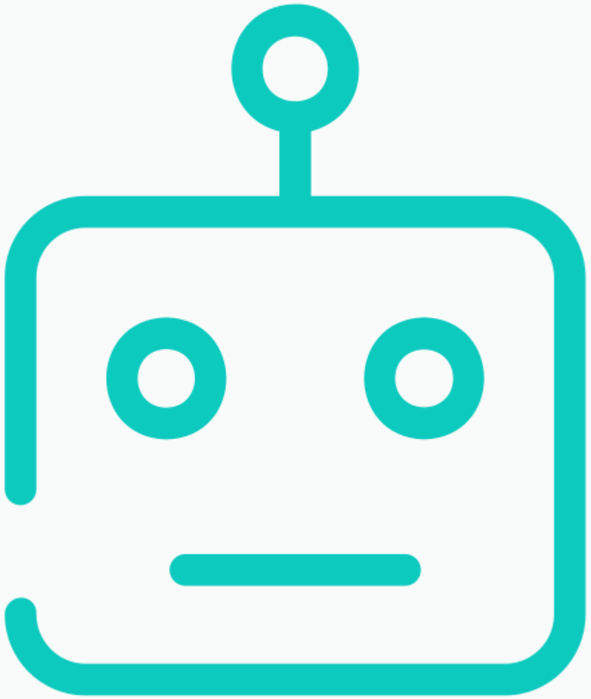
BIBOT

목차

1. 기획 배경



서비스 소개



영수증 처리 자동화 서비스
BiBot

Bill와 Biz을 뜻하는 Bi와
Robot을 뜻하는 Bot을 합친
영수증 비즈니스 로봇 BiBot

서비스 기획 의도

인력의 효율적인 활용과 작업 시간 단축을 위한 자동 경비 처리
OCR을 이용한 경비 처리 자동화 RPA 시스템

01

RPA

B2B RPA 서비스를
기획 및 개발 함으로써
실무 기획 능력 및
비즈니스 역량 향상

02

Micro Service

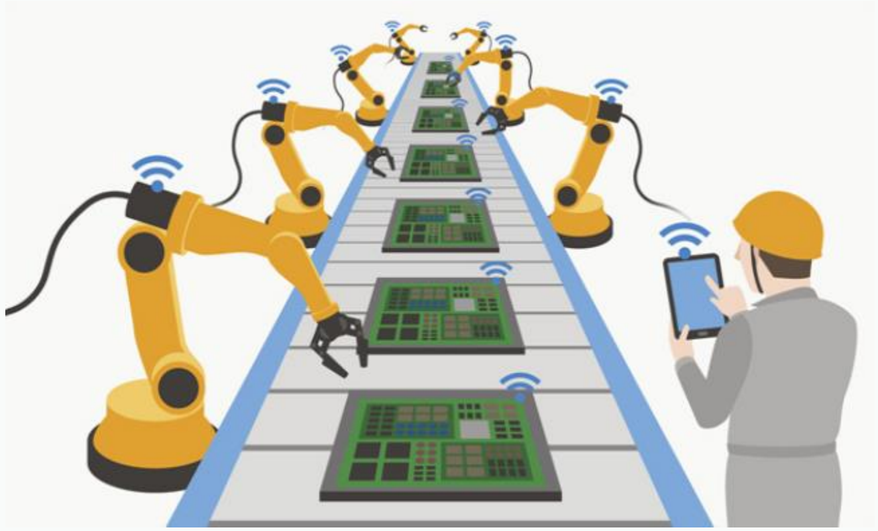
마이크로 서비스를
설계 및 개발 함으로써
개발 능력 및
설계 능력 향상

03

DevOps

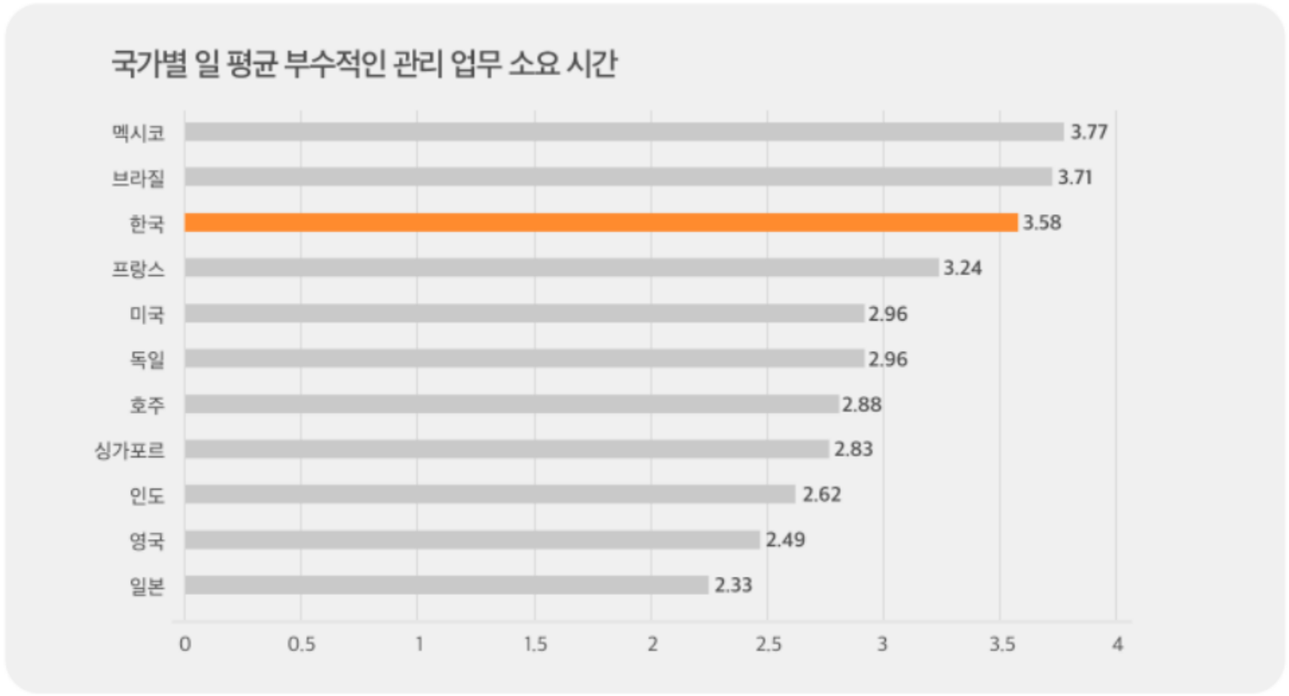
K8s 기반
무중단 CI/CD 체계를
직접 개발 함으로써
개발 운영 역량 향상

RPA란? (Robotics Process Automation)



Robot이 공장의 실체적 기계였다면 RPA는 Software로 사람이 하는 단순 반복적인 일을 하는 Robot

RPA 도입 배경



산업	적용 업무 예시	
보험	• 고객 정보(프로파일) 갱신 • 갱신 보험료 자동 생성	• 불편사항 처리 • 정책 관리 및 서비스
교통	• 승장 처리 • 서비스 및 수리 기록 모니터링	• 경로 변경에 대한 고지
	• 예외 처리	• 투자분석을 위한 정보 취합, 분석, 보고서 작성 및 발송
헬스 케어	• 공급자 정보 유효성 검사 • 빌링 및 컴플라이언스 관리 • 승장 발행	• 보험 등록 및 유효성 검사 • 시스템 간 의료 기록 비교 • 예약 정보 고지
인사 등 경영 지원	• 인사복지 관리 • 컴플라이언스 리포팅 • 원천징수 관리	• 주소 등록, 변경 처리 • 데이터 클렌징 • 주문 갱신
물류	• 온라인 주문, 생산 지시서 작성, 제품 발송 시 송장, 포장 목록 작성 • 단순 정보 입력 및 이메일 발송	• 외부 계약자(물류창고, 물류회사 등)에게 출고, 배송 지시서, 해외주문 시 통관서류 작성
소매업	• 고객 불만사항 관리 • 로열티 프로그램 고객 등록	• 배송 고지 • 인벤토리 재정렬
영업/회계	• 신규 고객정보 등록 자동화	• 회계결산 업무 적용하여 오류개선 (수익결산/보고 관련 대량 데이터 정제, 가공)

- 비용 보고서, 급여, 법인카드, 출장비, 매입 세금계산서 처리
- 매출 회계

학벌주의로 인한 고급인력 양산과 저출산, 고령화로 인한 단순 노동 인력 감소

출처 : 오토메이션애니웨어

RPA 시장

RPA
도입 효과

비용 절감 및 만족도 상승

- 반복되는 수작업을 자동화하여 단순작업 인력을 줄이고, 다른 업무로 인력 재배치
  → **직원의 만족도 상승**
- 사업확대시, 인력 증가요구에 능동적으로 대처 가능 (업무 잠재 능력 확보)

인적자원 향상

- 직원이 보다 부가가치가 높은 업무에 집중하여 전문성을 갖고 핵심업무에 집중

오류 감소와 품질 개선

- 사람의 입력 실수 오류를 개선하고 수행로그 기반 업무처리로 투명성 확보

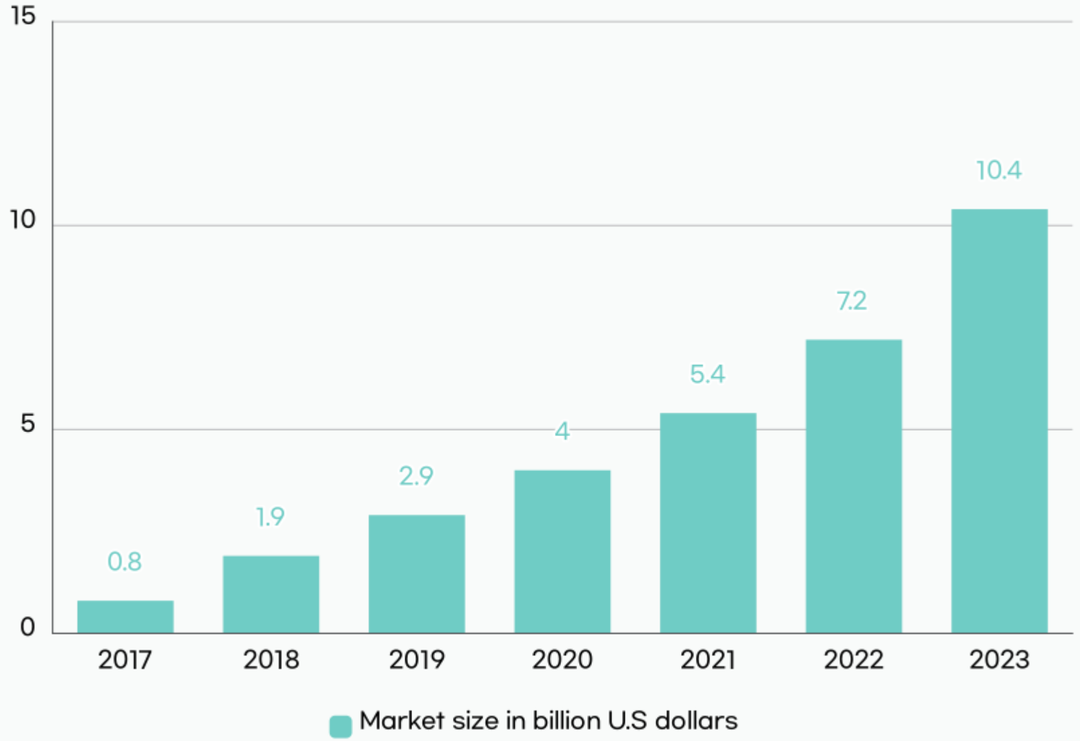
업무 생산성

- 업무 처리에 필요한 시간을 줄여서 목표를 달성할 수 있고, 분산된 시스템과 데이터를 쉽게 통합할 수 있음
- 24시간 X 365일 무중단 업무 수행

분석 및 통찰

- 프로세스 데이터 축적으로 개선 기회 발굴

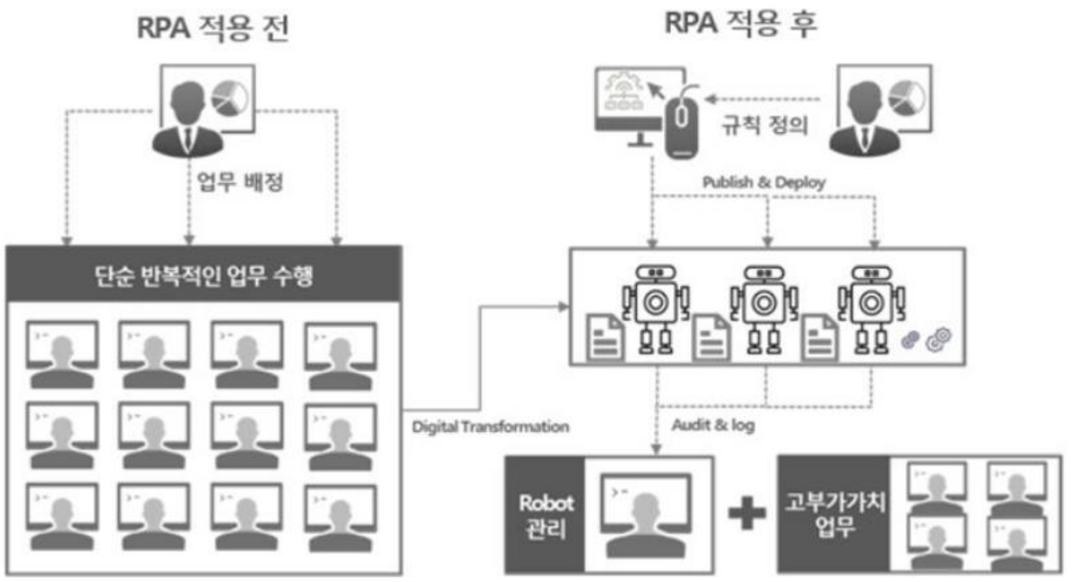
Statista의 글로벌 RPA 소프트웨어 시장 전망 2022



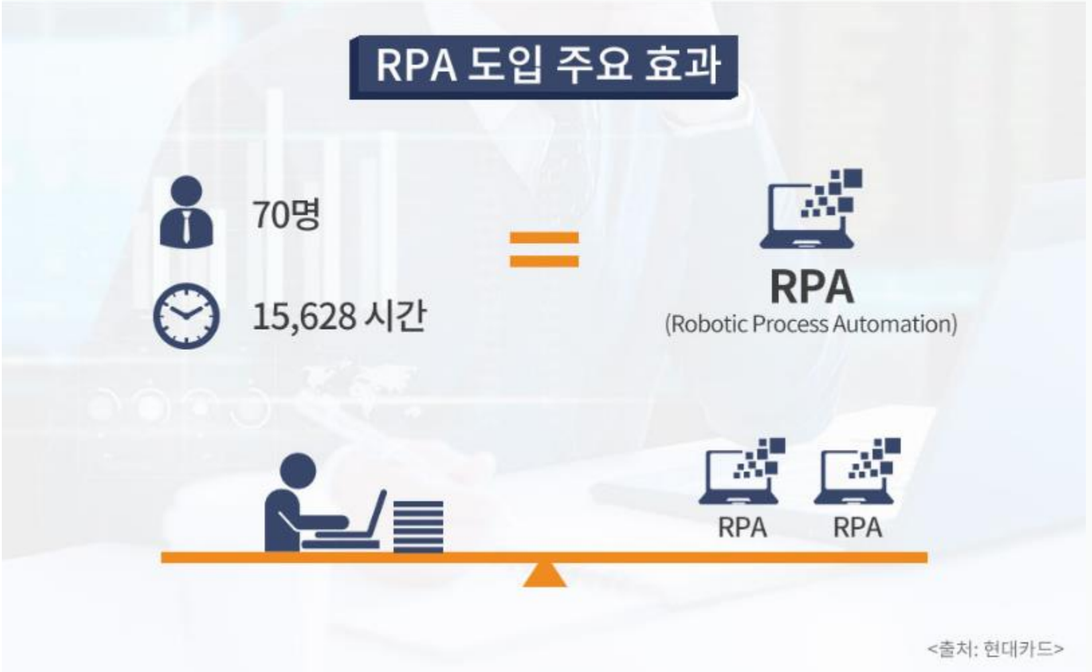
이에 대한 해결방안으로 세계적으로 RPA 도입의 확산

RPA의 장점

그림2 RPA 적용 전/후 비교



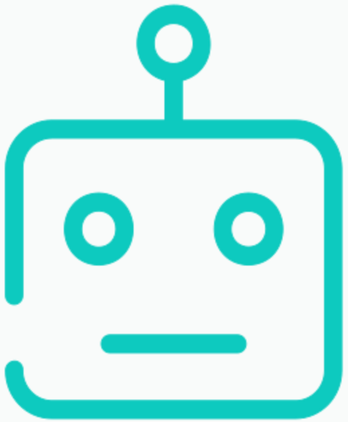
※ 출처: KT DS 공식 홈페이지



<출처: 현대카드>

단순 노동을 로봇으로 대체함으로 임직원은 고부가 업무에 집중 -> 업무 생산성 향상

2. 프로젝트 구성

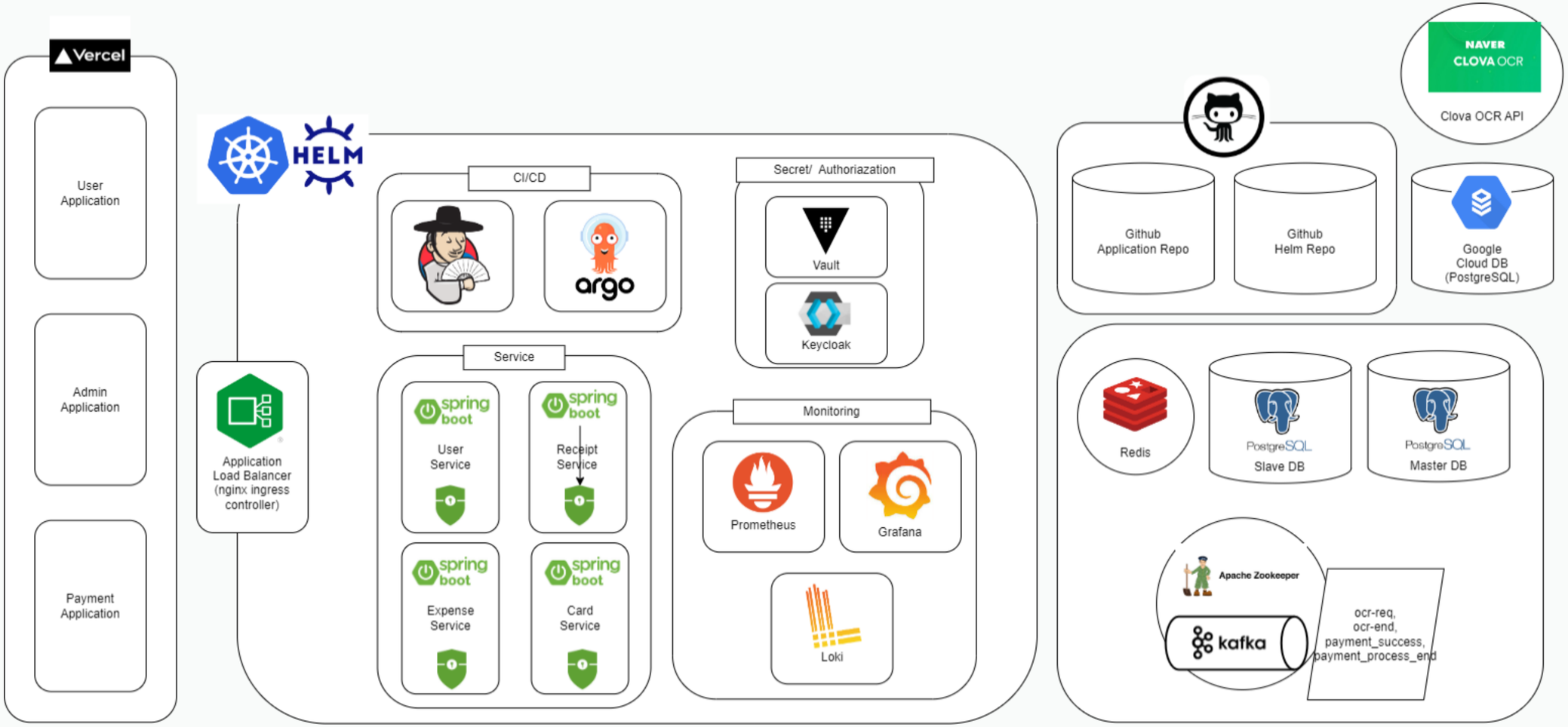


프로젝트 구성 - 기술 스택

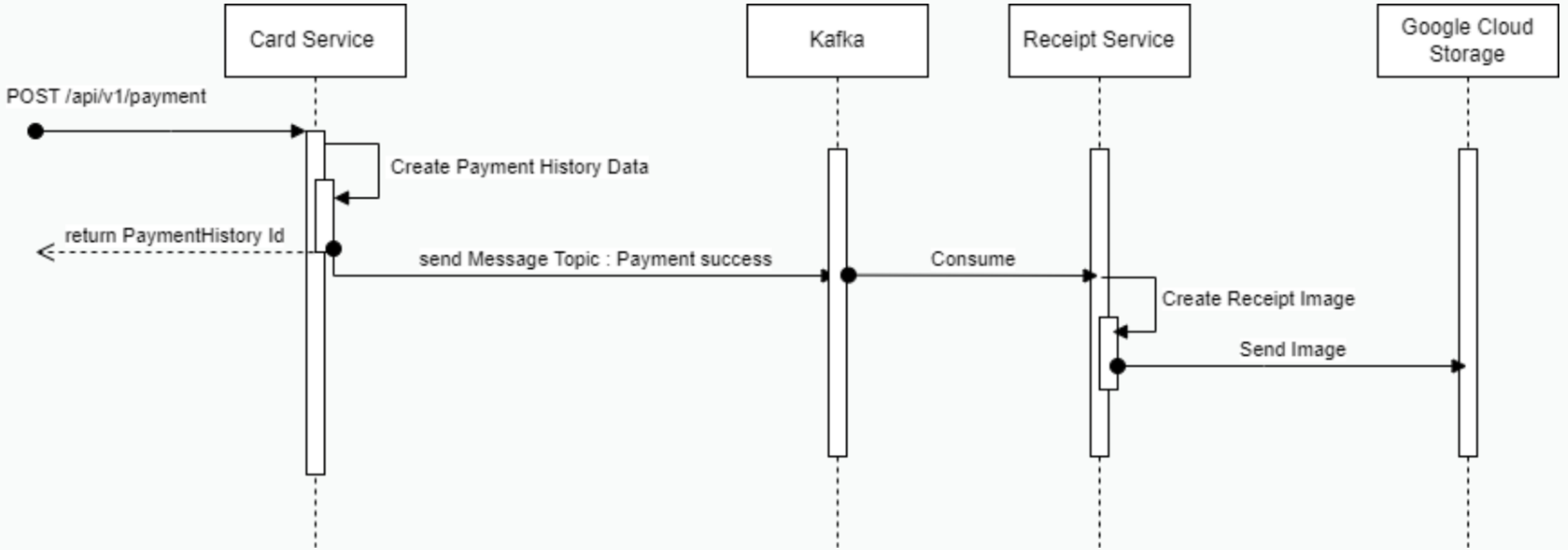
Front End

Back End

프로젝트 구성 - 서비스 아키텍쳐 구성

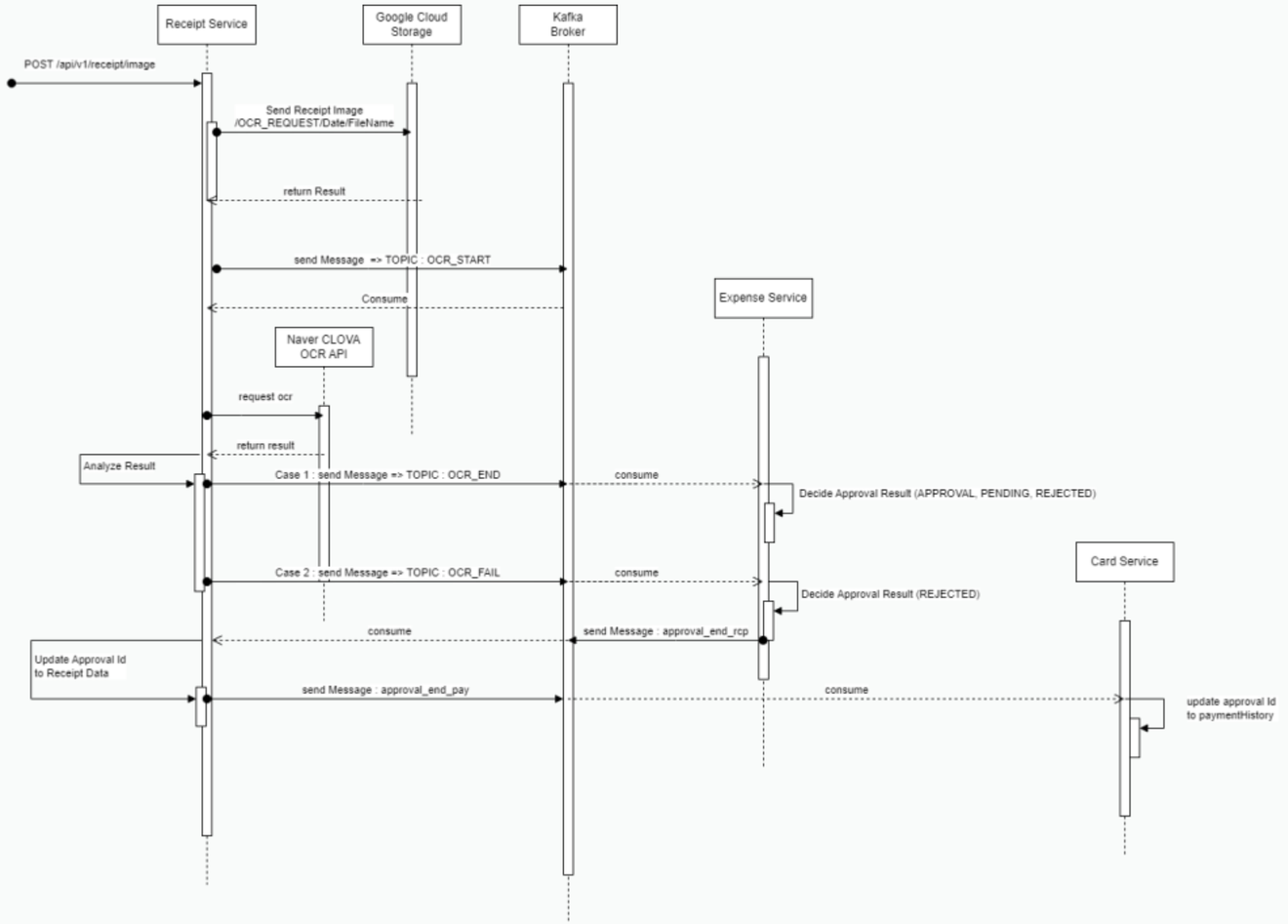


프로젝트 구성 - Sequence Diagram



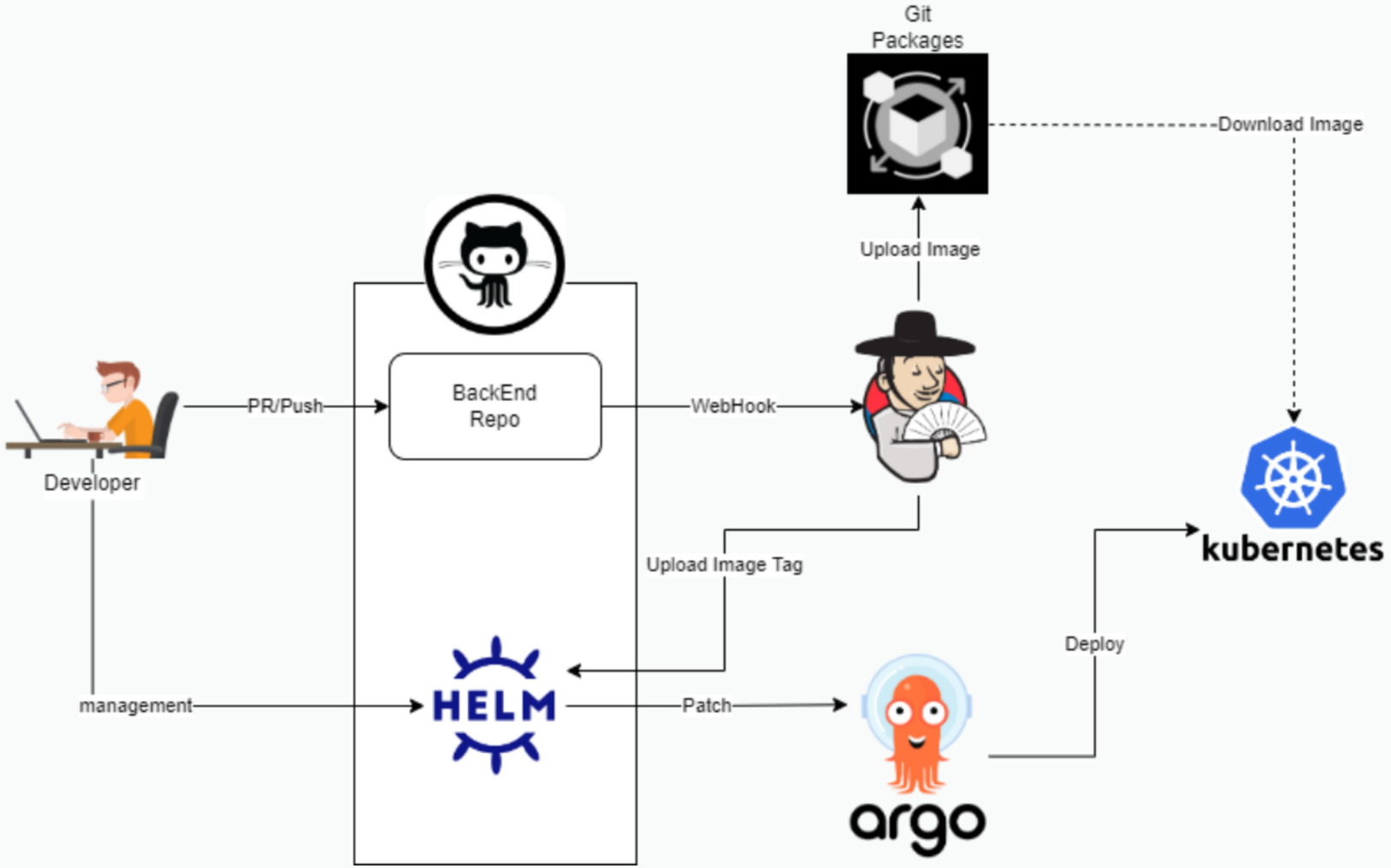
영수증 이미지 생성 로직

프로젝트 구성 - Sequence Diagram

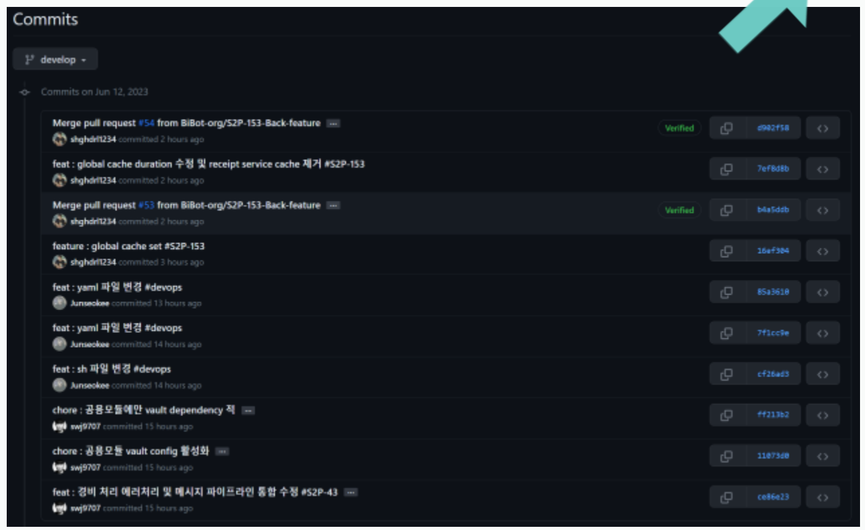


경비처리 로직

프로젝트 구성 - CI/CD 구성도



프로젝트 구성 - CI/CD 구성도



#215 (2023. 6. 11. 오후 3:12:23)

1. chore : application.yml 파일 분리 및 수정 #S2P-43 (commit: f2d0495) — swj1265 / githubweb
2. feat : 비밀번호 변경 구현 & 캐싱 및 볼트 속성 비활성화 #S2P-43 (commit: 468b657) — swj1265 / githubweb
3. feat : 경비 처리 에러처 #S2P-43 (commit: ce86aea) — swj1265 / githubweb
4. feat : 경비 처리 에러처리 및 메시지 파이프라인 통합 수 #S2P-43 (commit: f892a0c) — swj1265 / githubweb

#214 (2023. 6. 11. 오후 2:47:18)

1. feat : 배포파일 수정 #devops (commit: 0e81298) — 102888442+Junseokee / githubweb

#213 (2023. 6. 11. 오후 1:32:43)

1. feat : 로그인 로직 변경 #devops (commit: e3862bf) — 102888442+Junseokee / githubweb

#212 (2023. 6. 11. 오전 10:03:47)

1. backup : kafka 연동 백업 #S2P-145 (commit: 154a534) — shghdrl1234 / githubweb

#210 (2023. 6. 10. 오후 6:28:39)

1. fix : sh file host 수정 (commit: 8d2ae95) — 102888442+Junseokee / githubweb

#207 (2023. 6. 10. 오후 4:31:54)

1. fix : vault host 수정 #devops (commit: b0aa207) — 102888442+Junseokee / githubweb

#206 (2023. 6. 9. 오전 11:58:44)

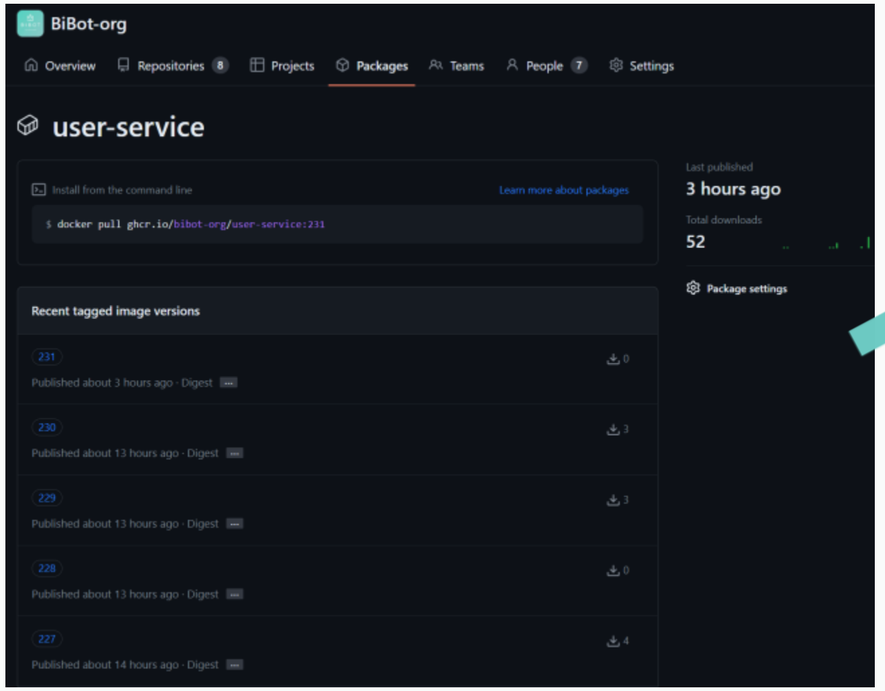
1. 긴급 패치! (commit: c0e9905) — shghdrl1234 / githubweb

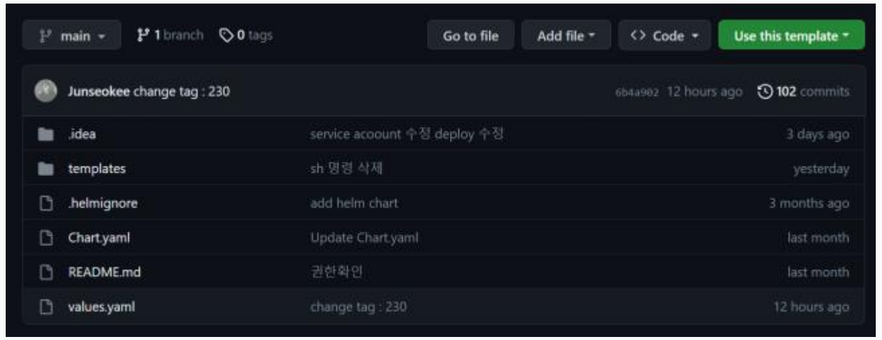
#204 (2023. 6. 9. 오전 1:47:17)

1. backup : user profile CUD API 백업 #S2P-140 (commit: 0f48629) — shghdrl1234 / githubweb
2. refactor : user profile CUD 로직 refactoring #S2P-140 (commit: ef35ff2) — shghdrl1234 / githubweb
3. feat : yaml file secret 설정 #devops (commit: 45b155e) — weq156 / githubweb
4. fix : yaml file #devops (commit: 2399454) — weq156 / githubweb
5. feat : kubernets dockerfile sh #devops (commit: 9980555) — weq156 / githubweb
6. feat : kubernets dockerfile 수정 #devops (commit: 2f1dcea) — weq156 / githubweb
7. backup : global cache 백업 #S2P-141 (commit: 5da7afb) — shghdrl1234 / githubweb
8. feat : 수동 경비처리 구현 #S2P-43 (commit: cdef26f) — swj1265 / githubweb
9. feat : 영수증 경비처리 연 #S2P-43 (commit: 4c7ae68) — swj1265 / githubweb
10. fix : 시크릿 데이터 삭 (commit: a04ba11) — swj1265 / githubweb
11. fix : profiles 환경 변경 # devops (commit: a9a1121) — weq156 / githubweb

개발자가 Git Repo에 Push -> Jenkin에서 Build

프로젝트 구성 - CI/CD 구성도






```yaml
spec:
  containers:
    - image: 'ghcr.io/bibot-org/user-service:230'
      imagePullPolicy: IfNotPresent
      name: bibot
```

Build가 완료되면 Image Push -> HelmChart Tag 변경 -> ArgoCD에서 Patch

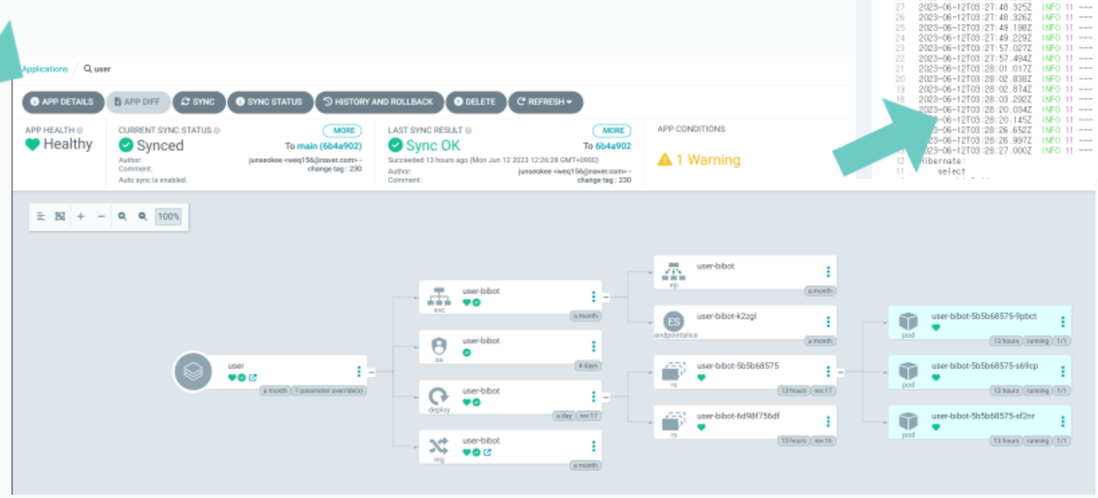
프로젝트 구성 - CI/CD 구성도

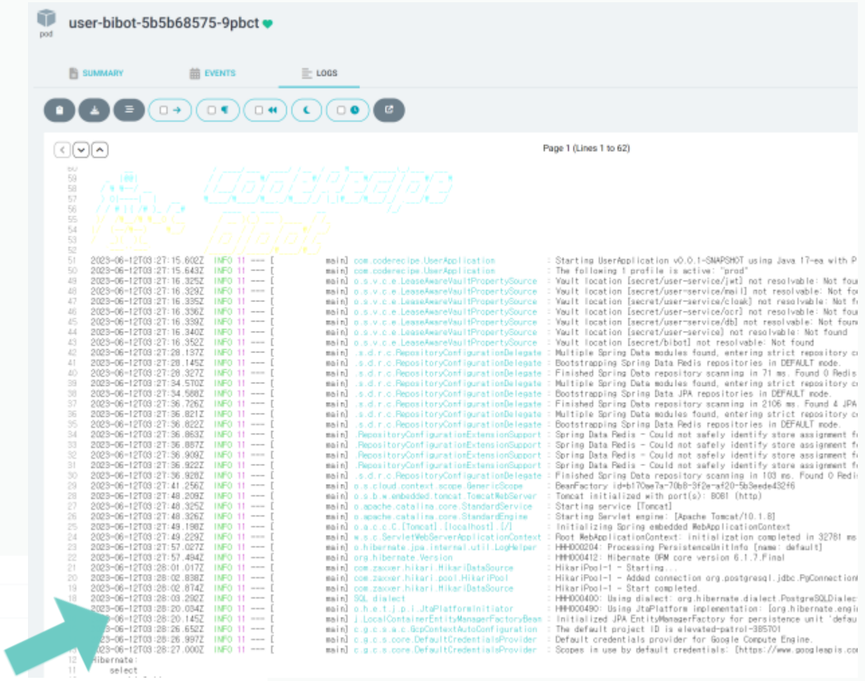
argoCD / templates / deployment.yaml

Junseokee sh 명령 삭제

Code Blame 66 lines (65 loc) · 2 KB

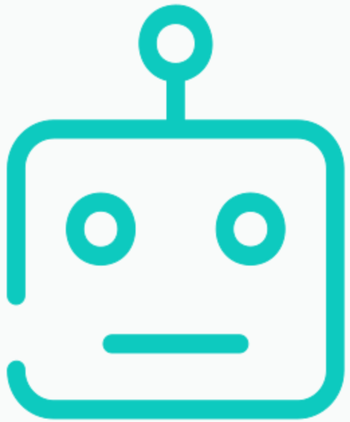
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ include "bibot.fullname" . }}
  labels:
    {{- include "bibot.labels" . | nindent 4 }}

spec:
  {{- if not .Values.autoscaling.enabled }}
  replicas: {{ .Values.replicaCount }}
  {{- end }}
  selector:
    matchLabels:
      {{- include "bibot.selectorLabels" . | nindent 6 }}
  template:
    metadata:
      {{- with .Values.podAnnotations }}
      annotations:
        {{- toYaml . | nindent 8 }}
        vault.hashicorp.com/agent-inject: "true"
        vault.hashicorp.com/role: "secret-read"
        vault.hashicorp.com/address: "http://34.22.82.97:8200"
        vault.hashicorp.com/token: $VAULT_TOKEN
      {{- end }}
      labels:
        {{- include "bibot.selectorLabels" . | nindent 8 }}
    spec:
      {{- with .Values.imagePullSecrets }}
      imagePullSecrets:
        {{- toYaml . | nindent 8 }}
      {{- end }}
```



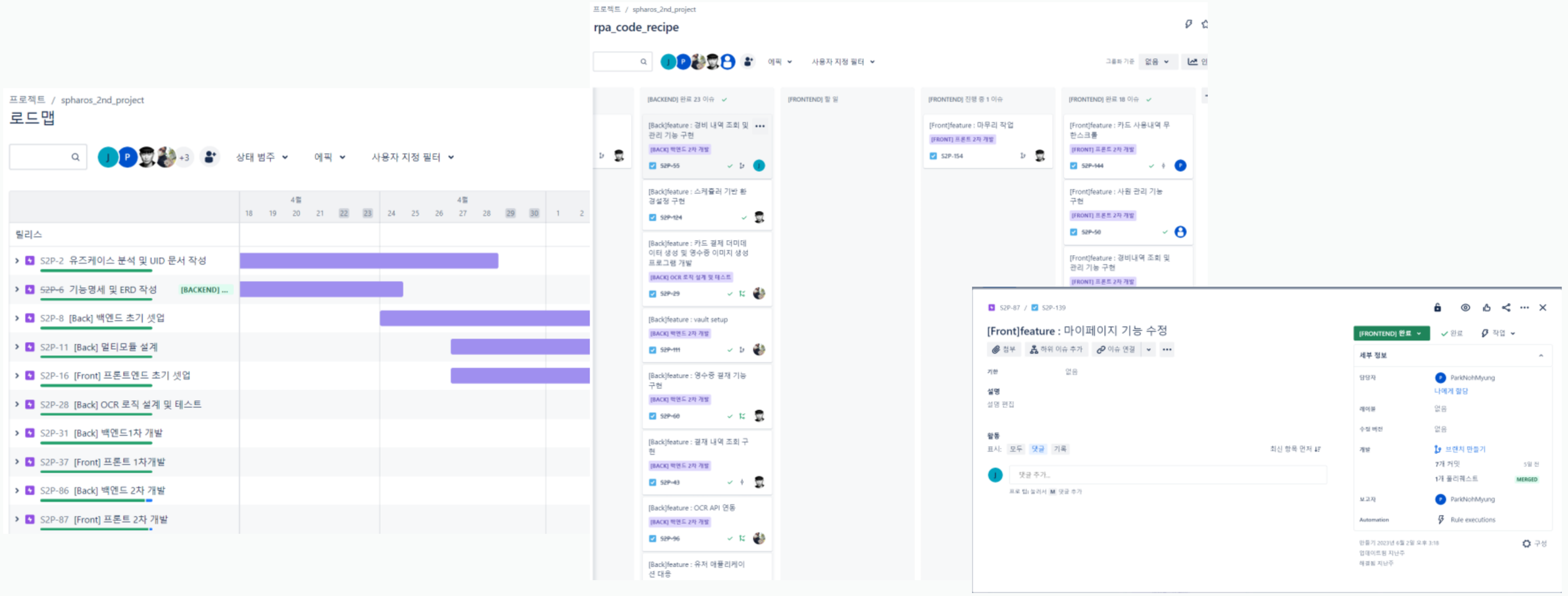


Helm 템플릿 바탕으로 Pod 배포 -> Kubernetes에서 Application 실행

3. 프로젝트 진행

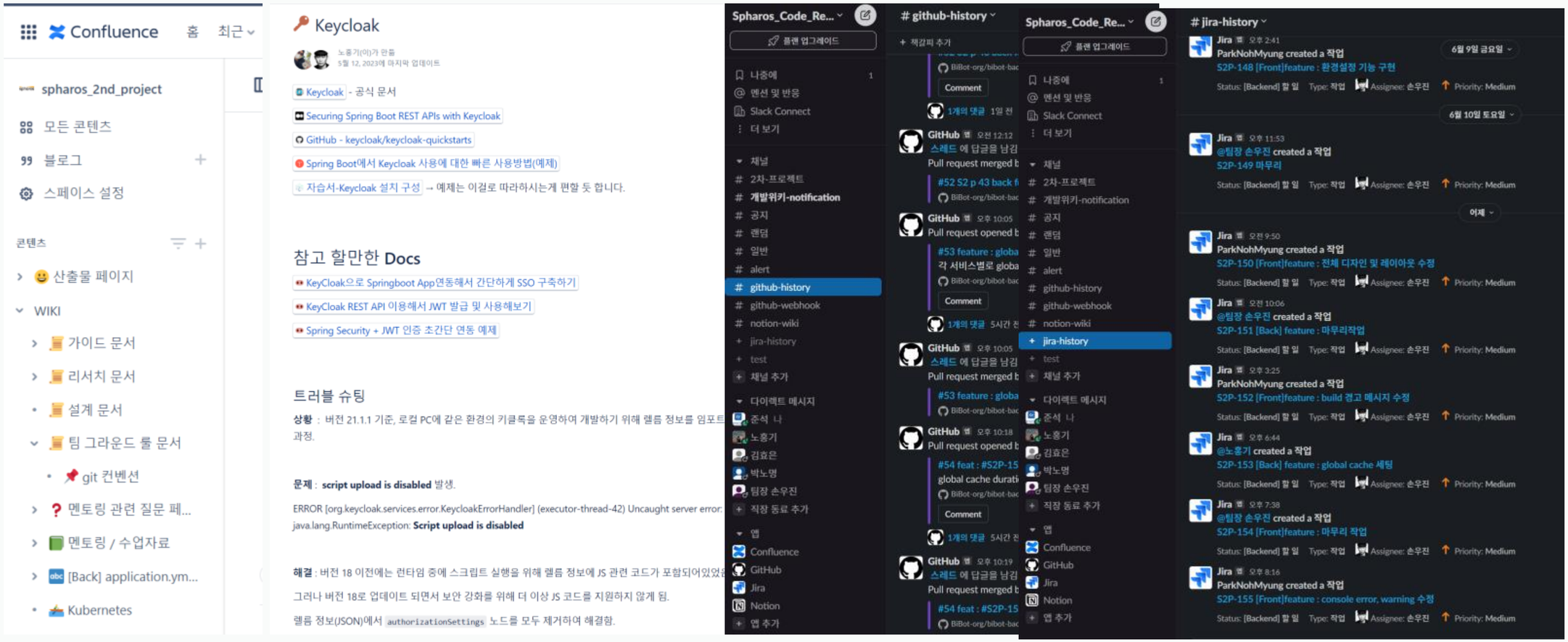


프로젝트 진행 - 협업툴



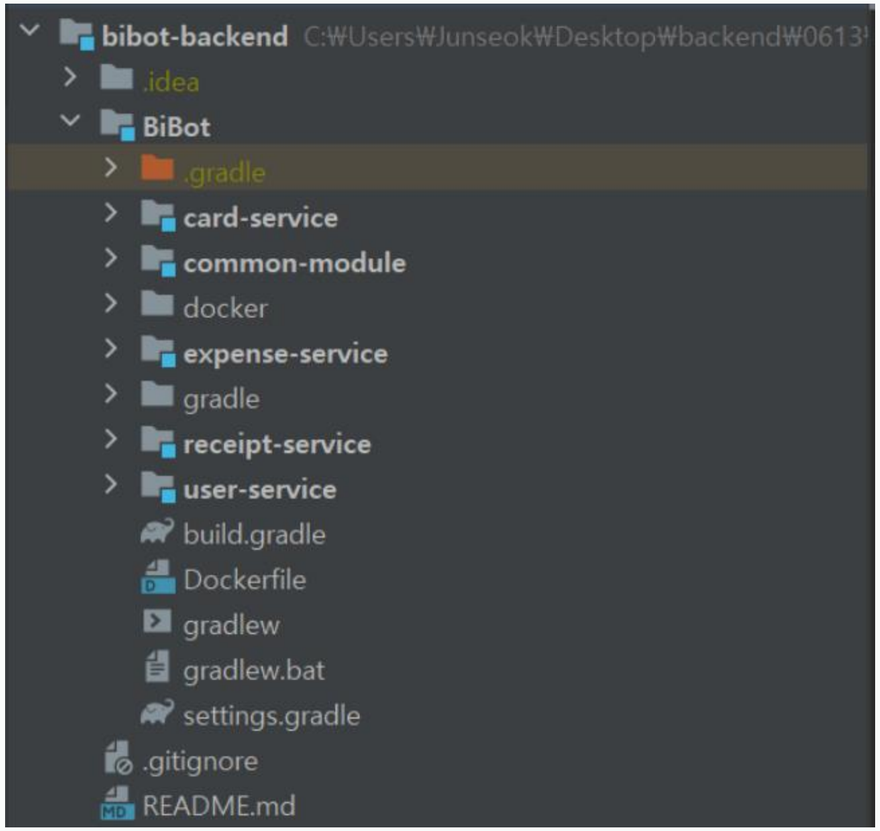
Jira를 활용하여 개발 일정 관리와 자동화를 활용한 이슈, 커밋 관리

프로젝트 진행 - 협업툴



Confluence를 활용하여 산출물과 Wiki 관리, Slack을 협업툴과 연동하여 프로젝트 관리

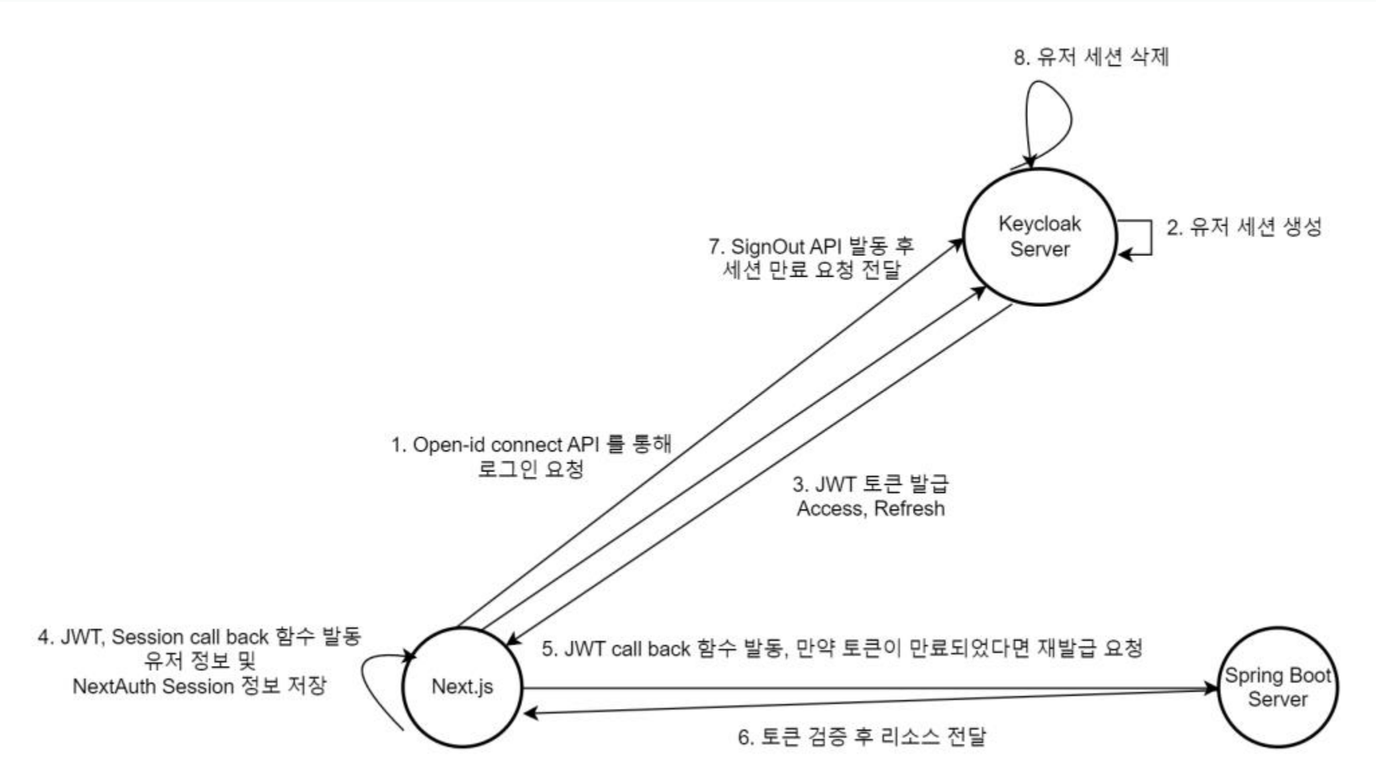
프로젝트 진행 - 멀티 모듈 구조




```
def commonProject : ArrayList<Project> =
        [project(':common-module')]
configure(commonProject) {

    bootJar {
        enabled = false
    }
    jar {
        enabled = true
    }
    bootBuildImage {
        enabled = false
    }
}


def domainProjects : ArrayList<Project> = [project(':card-service'), project(':expense-service')
                    , project(':user-service'), project(':receipt-service')]
configure(domainProjects) {
    dependencies {
        implementation project(':common-module')
//        implementation 'org.springframework.cloud:spring-cloud-starter-vault-config'
        implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
        implementation 'org.springframework.boot:spring-boot-starter-cache'
        implementation 'org.springframework.boot:spring-boot-starter-web'
        implementation 'org.springframework.boot:spring-boot-starter-validation'
        runtimeOnly 'org.postgresql:postgresql'
```

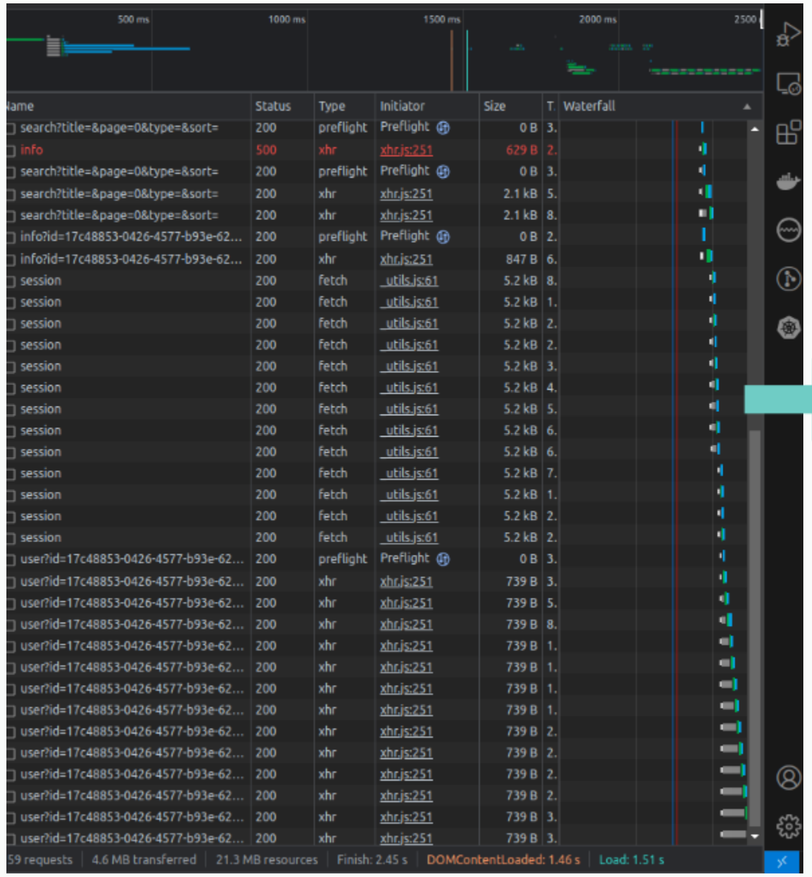
멀티 모듈을 활용한 Repo 간소화와 Dependency 버전 통일과 확장성 확보

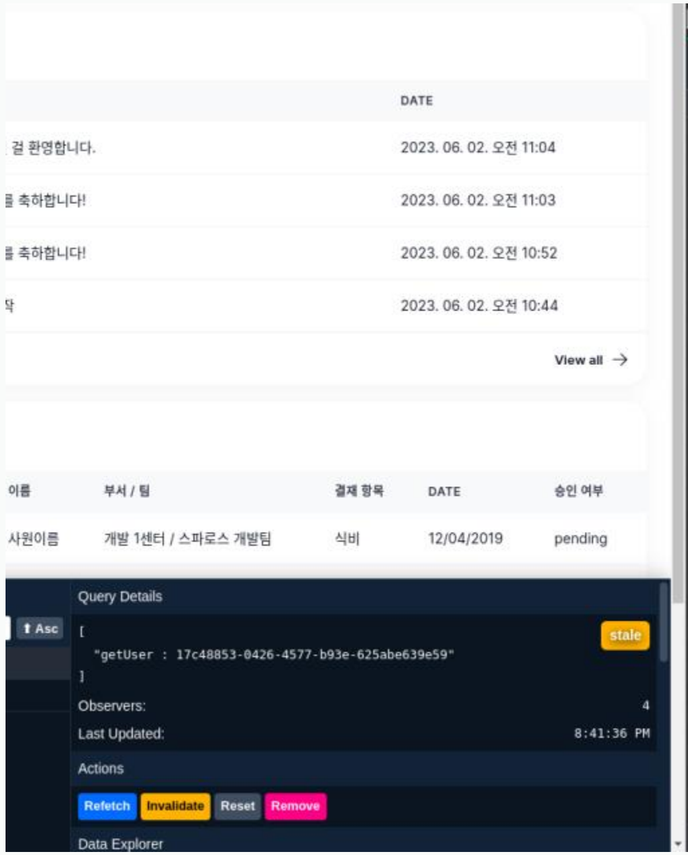
프로젝트 진행 - Keycloak, NextAuth

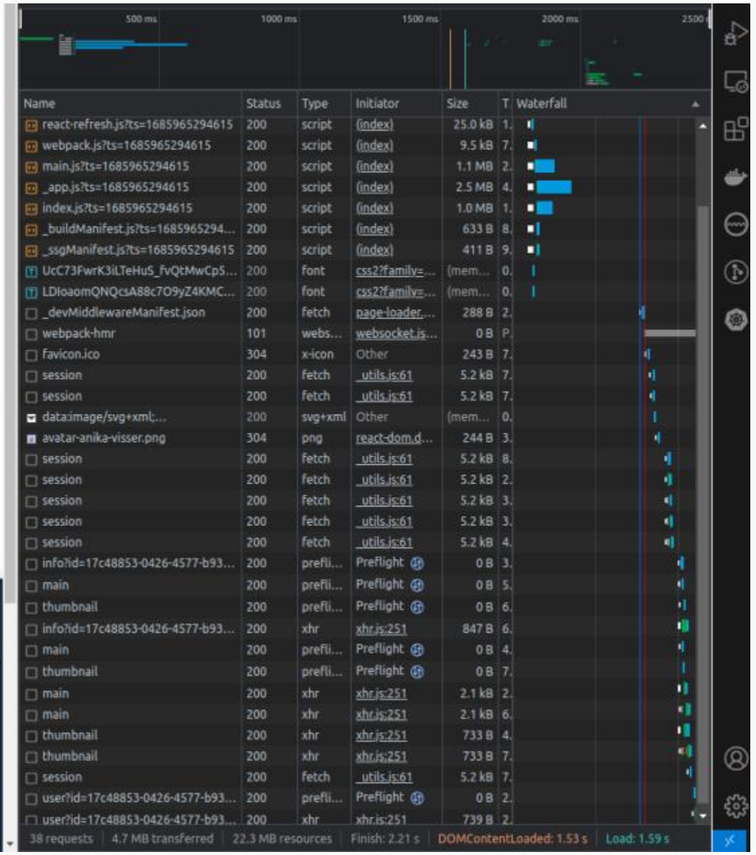


Keycloak을 통한 SSO 구현과 NextAuth 연동을 통해 확장성과 사용성 확보

프로젝트 진행 - React Query







API 호출이 많은 MSA에서 React Query를 이용한 캐싱처리를 통한 API 호출 감소와 랜더링 성능 향상

프로젝트 진행 - Global cache server

Case 1 : 100명의 유저가 초당 1회씩 1000번 요청

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput ↑	Received KB/sec	Sent KB/sec	Avg. Bytes
before Caching	100000	46	1	1075	74.04	0.00%	2049.9/sec	1214.54	264.24	606.7
after Caching	100000	23	0	608	48.58	0.00%	3842.2/sec	2276.44	495.28	606.7
TOTAL	200000	34	0	1075	63.67	0.00%	1811.9/sec	1073.55	233.57	606.7

user 100, period 1, Loop 1000

Case 2 : 1339명의 유저가 초당 1회씩 1회 요청

Label	# Samples	Average ↑	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
after caching	1339	12	2	103	13.05	0.00%	948.3/sec	562.13	122.24	607.0
before caching	1339	1180	258	2713	384.84	0.00%	471.1/sec	279.28	60.73	607.0
TOTAL	2678	596	2	2713	644.14	0.00%	95.7/sec	56.75	12.34	607.0

user 1339, period 1, Loop 1

Case 3 : 1339명의 유저가 초당 1회씩 2회 요청

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput ↓	Received KB/sec	Sent KB/sec	Avg. Bytes
after caching	2678	547	5	1102	225.66	0.00%	1150.3/sec	681.89	148.29	607.0
before caching	2678	829	5	2394	426.34	0.00%	811.0/sec	480.75	104.55	607.0
TOTAL	5356	688	5	2394	369.03	0.00%	125.4/sec	74.33	16.16	607.0

user 1339, period 1, Loop 2

Case 4 : 1339명의 유저가 초당 1회씩 3회 요청

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput ↓	Received KB/sec	Sent KB/sec	Avg. Bytes
after caching	4017	627	2	2989	436.73	0.00%	1324.9/sec	785.35	170.78	607.0
before caching	4017	1263	4	4050	587.49	0.00%	697.6/sec	413.54	89.93	607.0
TOTAL	8034	945	2	4050	607.45	0.00%	254.9/sec	151.12	32.86	607.0

user 1339, period 1, Loop 3

Case 5 : 1339명의 유저가 초당 1회씩 4회 요청

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput ↓	Received KB/sec	Sent KB/sec	Avg. Bytes
after caching	5356	530	2	3184	355.87	0.00%	1674.8/sec	992.78	215.89	607.0
before caching	5356	846	4	2818	472.95	0.00%	1064.6/sec	631.07	137.23	607.0
TOTAL	10712	688	2	3184	447.25	0.00%	247.9/sec	146.97	31.96	607.0

user 1339, period 1, Loop 4

Redis와 spring-boot-cache를 활용한 Global cache server를 구축
JMeter를 이용한 Metric Data 분석으로 리소스 관리

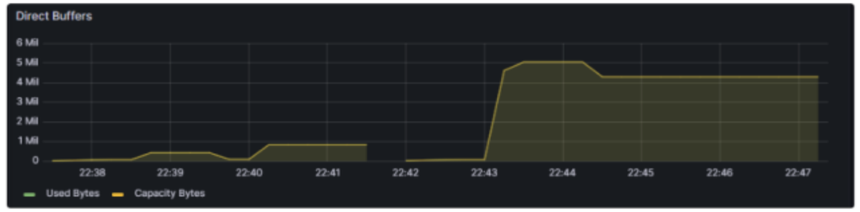
프로젝트 진행 - Global cache server

- 22:40 ~ 22:41 구간은 캐싱 처리를 하지 않았습니다.
- 22:43 ~ 22:44 구간은 캐싱 처리를 진행하였습니다.

CPU Usage

CPU 사용량과의 관계.

- 캐시는 자주 사용되는 데이터를 메모리에 저장하여 CPU가 데이터를 메모리에서 직접 읽습니다.
- CPU가 하드 디스크에서 정보를 가지고 오지 않아도 되기 때문에 CPU 사용량이 줄어듭니다.



Direct Buffers

Memory와의 관계.

- 캐싱을 사용하면 메모리 할당 및 프로모션 작업이 증가합니다.

Classes Loaded

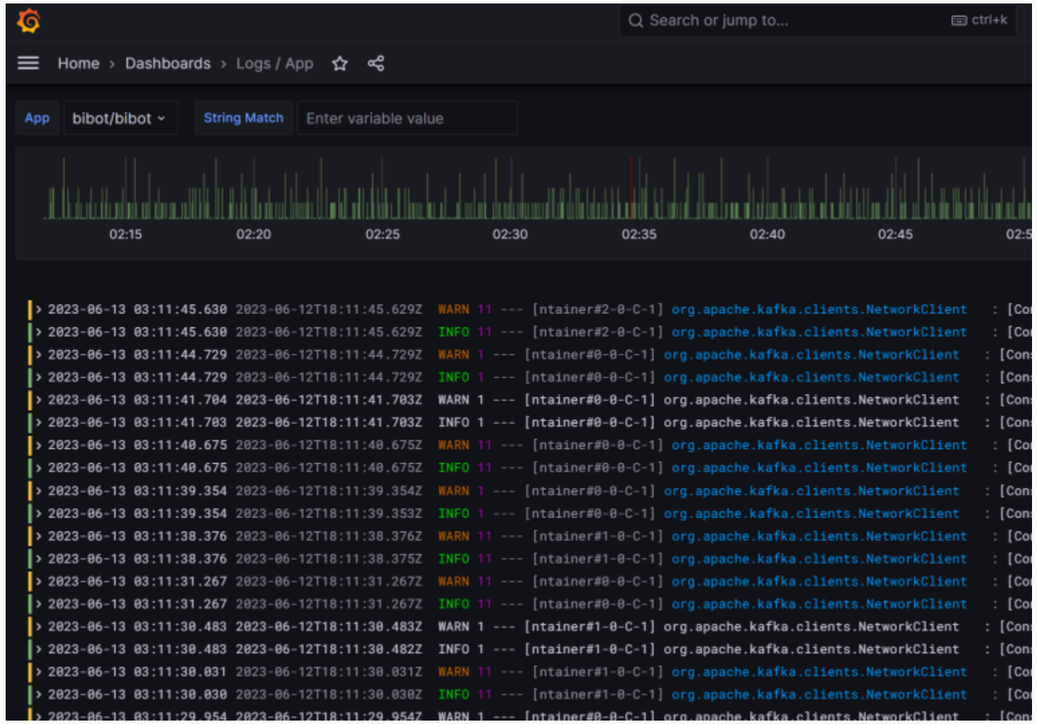
Classes Loaded와의 관계.

- 캐싱 처리를 적용할 때 클래스 로드의 증가가 발생할 수도 있습니다.
- 캐싱을 구현하기 위해 사용하는 라이브러리, 프레임 워크에서 자체적인 캐싱 기능을 사용한다면 관련된 클래스들이 추가로 로드될 수 있으며, 캐시 키/ 값의 직렬화, 역직렬화를 수행하는 관련 클래스들이 로드될 수 있습니다.

Hikari Connections

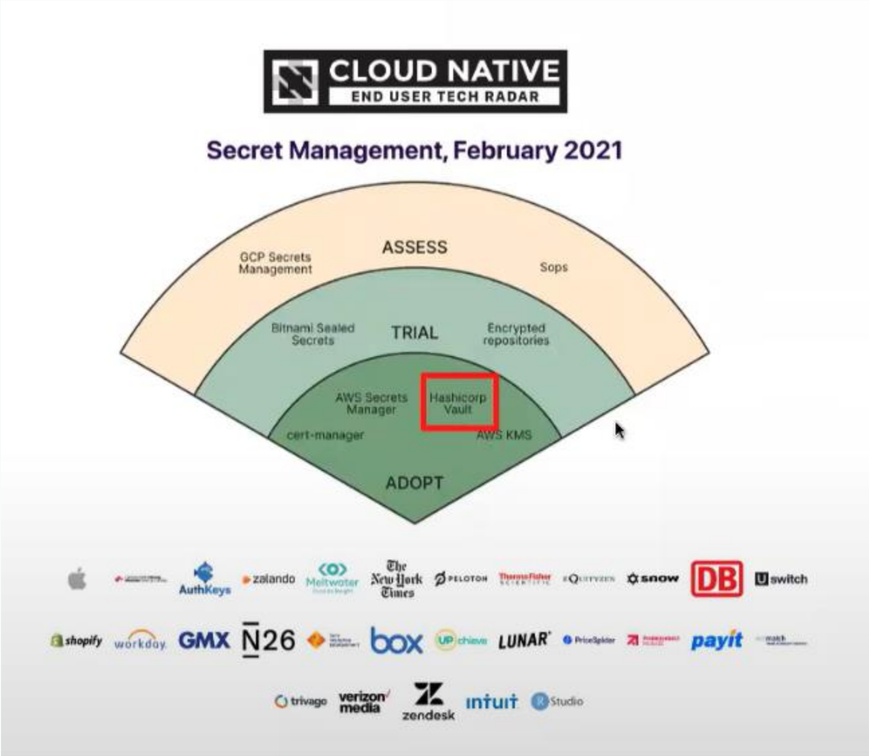
Grafana를 이용한 Metric Data 분석으로 테스트 시 리소스 추적

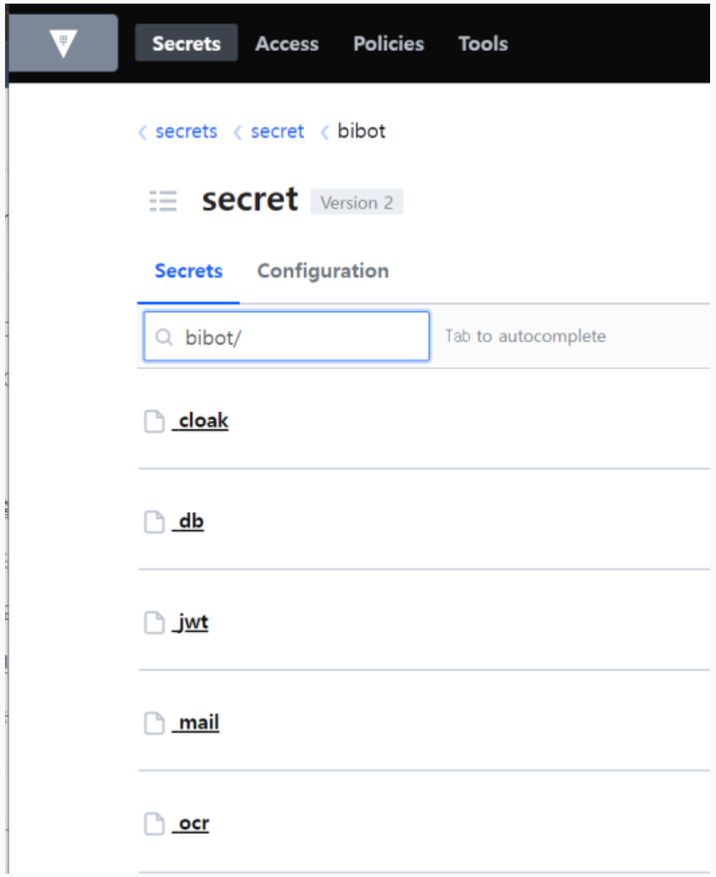
프로젝트 진행 - PLG Stack



Prometheus, Loki, Grafana(PLG)를 활용한 Kubernetes 리소스, 로그 관리 목적 모니터링 시스템 구축

프로젝트 진행 - Vault

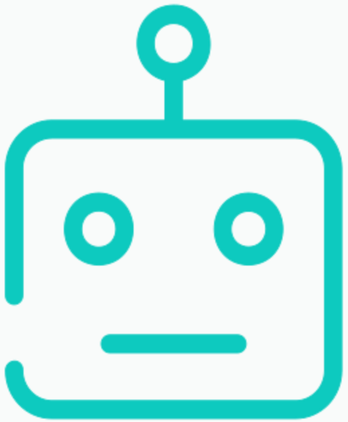




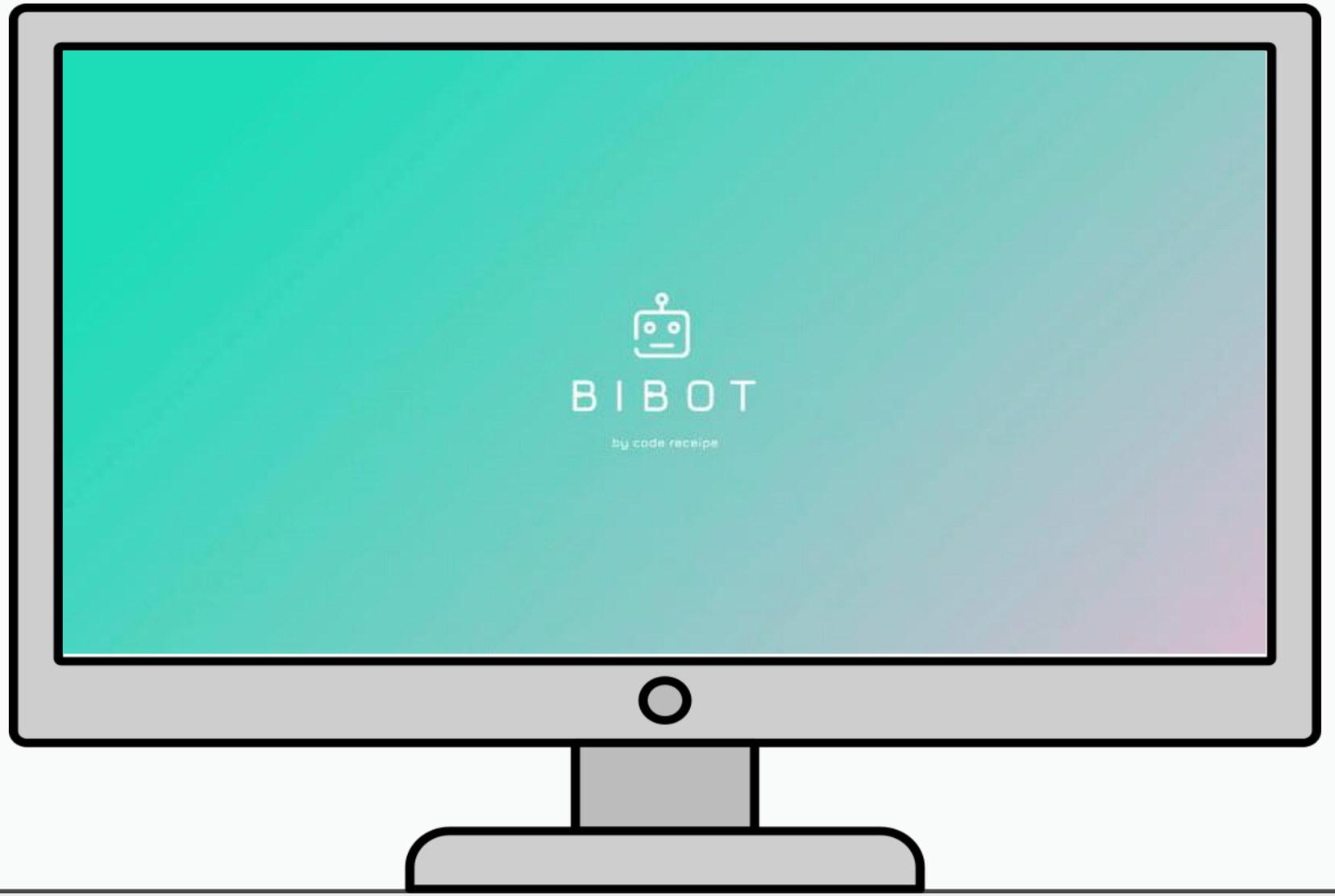

```yaml
datasource:
  driver-class-name: org.postgresql.Driver
  url: jdbc:postgresql://${host_name}/users
  username: ${db_name}
  password: ${db_pwd}

data:
  redis:
    host: ${redis_host}
    port: ${redis_port}
mail:
  host: smtp.gmail.com
  port: 587
  username: ${mail_id}
  password: ${mail_pwd}
  properties:
    mail.smtp.auth: true
    mail.smtp.starttls.enable: true

kafka:
  bootstrap-servers: ${kafka_url}
output:
  ansi:
    enabled: always
security:
  oauth2:
    resourceserver:
      jwt:
        jwk-set-uri: ${jwk_uri}
        issuer-uri: ${iss_uri}
```

Kubernetes에서 Secret 관리를 위한 Vault 사용으로 보안 강화와 관리 간소화

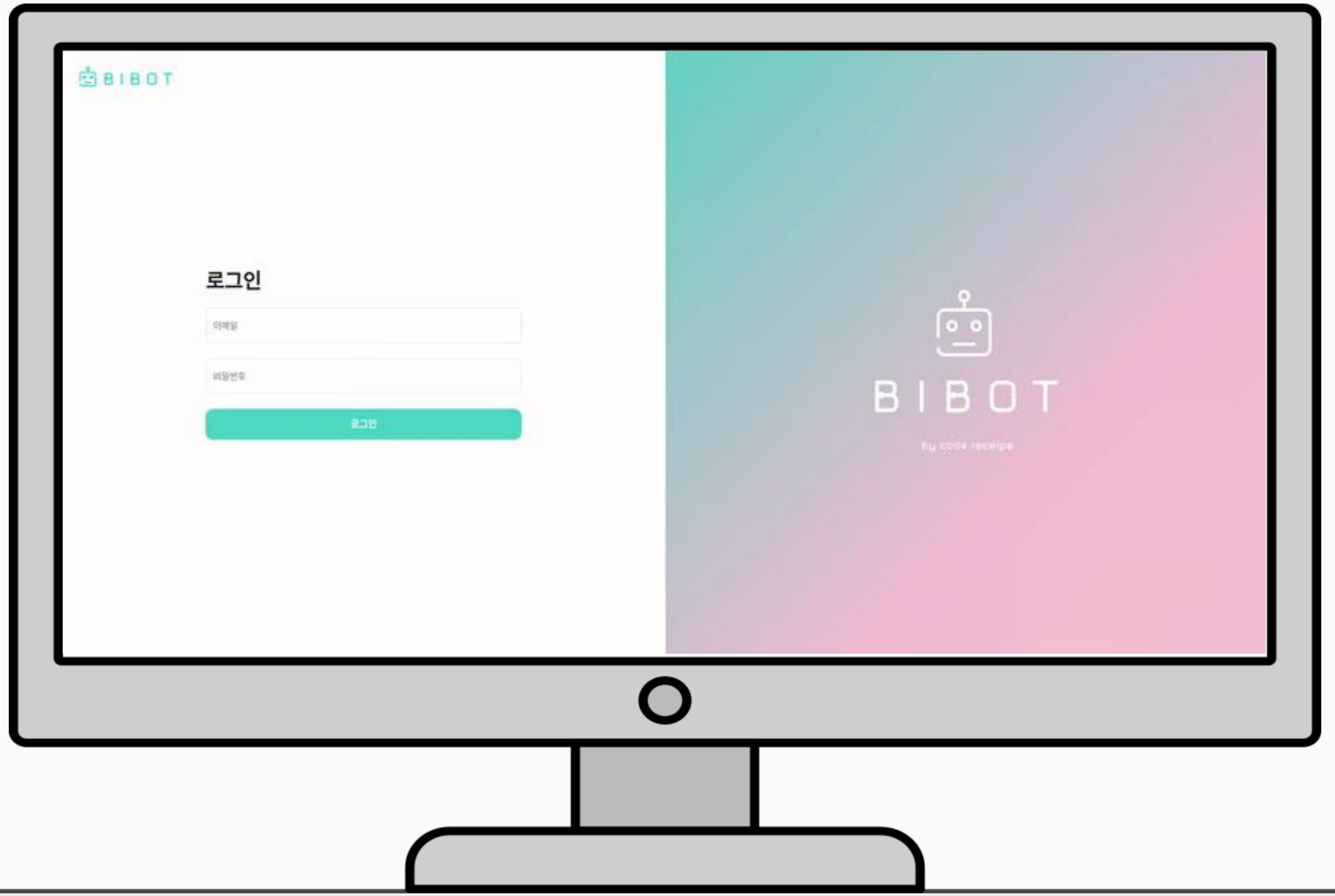
4. 시연



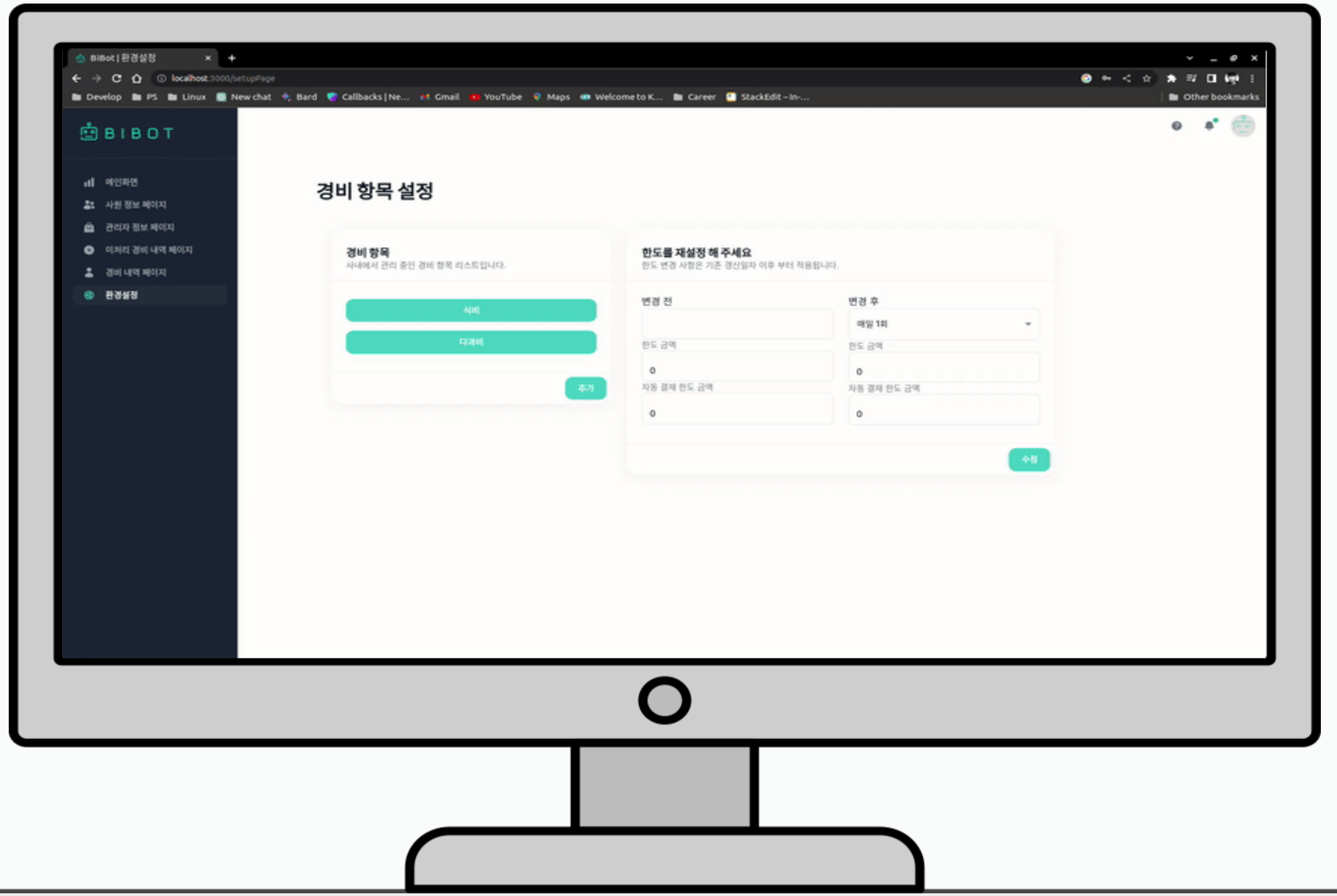
시연 - 관리자 메인 화면



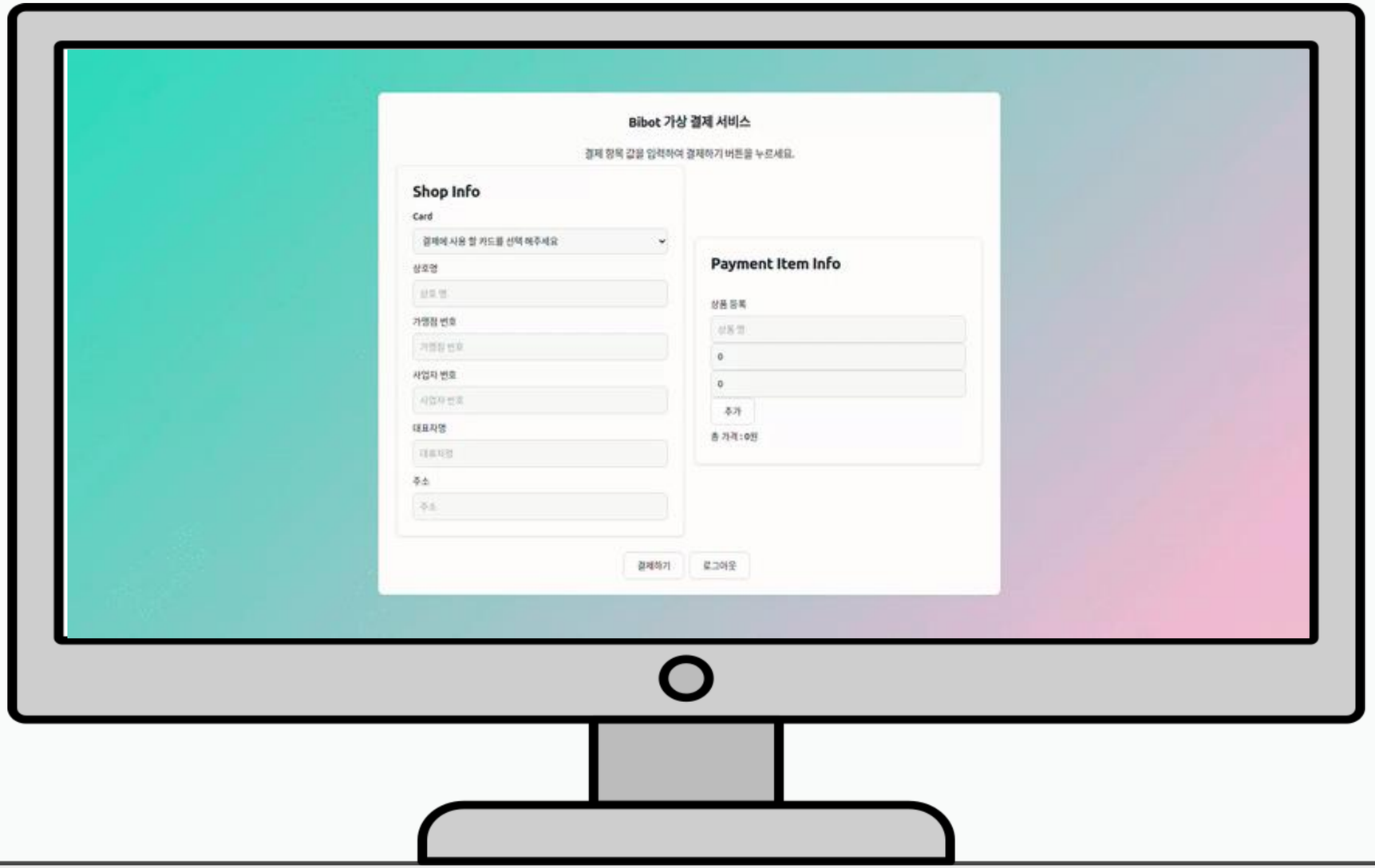
시연 - 관리자 로그인 화면



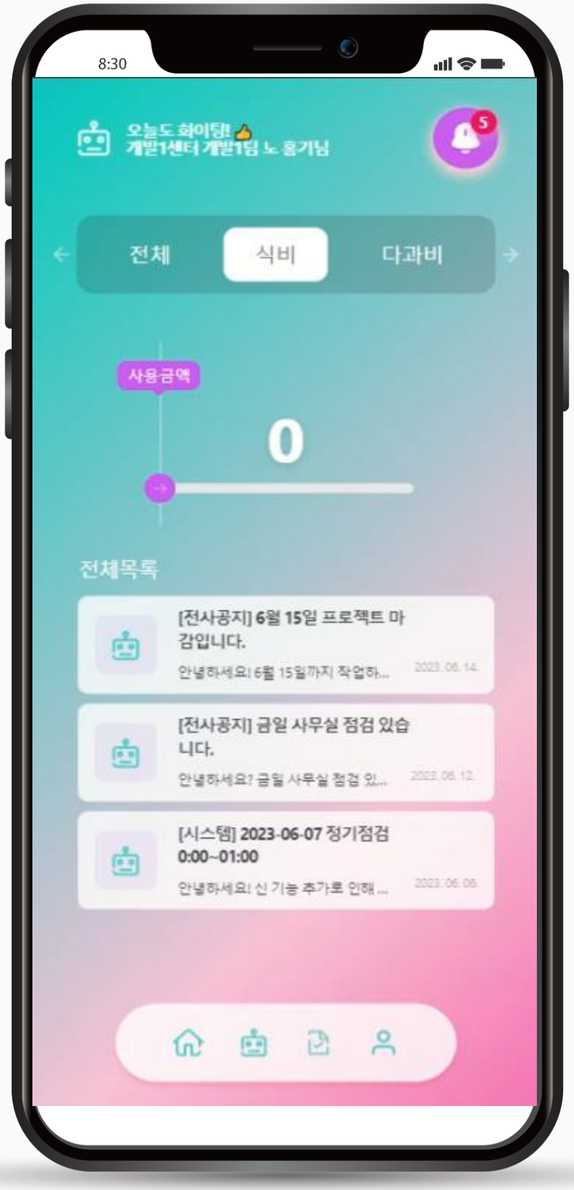
시연 - 관리자 경비 환경 설정



시연 - 영수증 이미지 생성

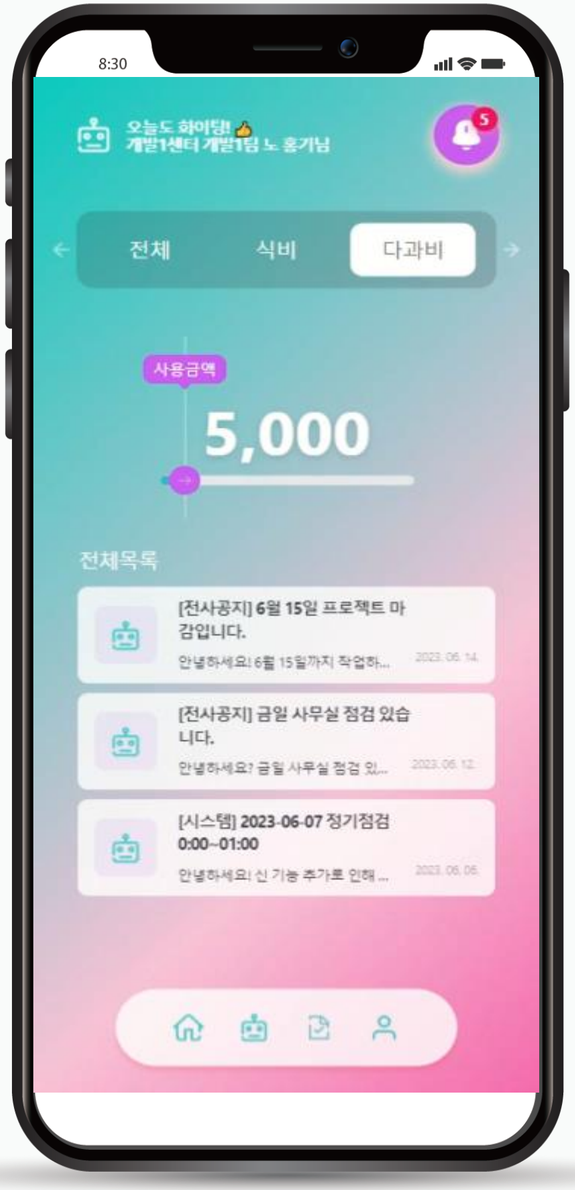

Bibot 가상 결제 서비스
결제 항목 값을 입력하여 결제하기 버튼을 누르세요.
Shop Info
Card
결제에 사용 할 카드를 선택 해주세요
상호명
가맹점 번호
사업자 번호
대표자명
주소
Payment Item Info
상품 등록
0
0
추가
총 가격 : 0원
결제하기
로그아웃

시연 - 메인 화면 및 공지 기능

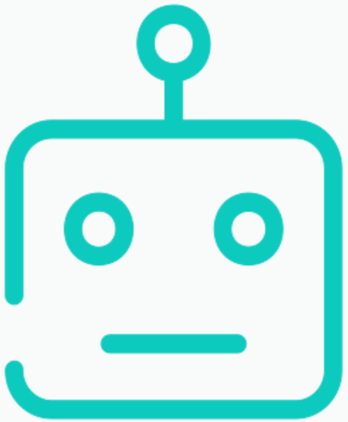


시연 - 경비처리 성공 케이스

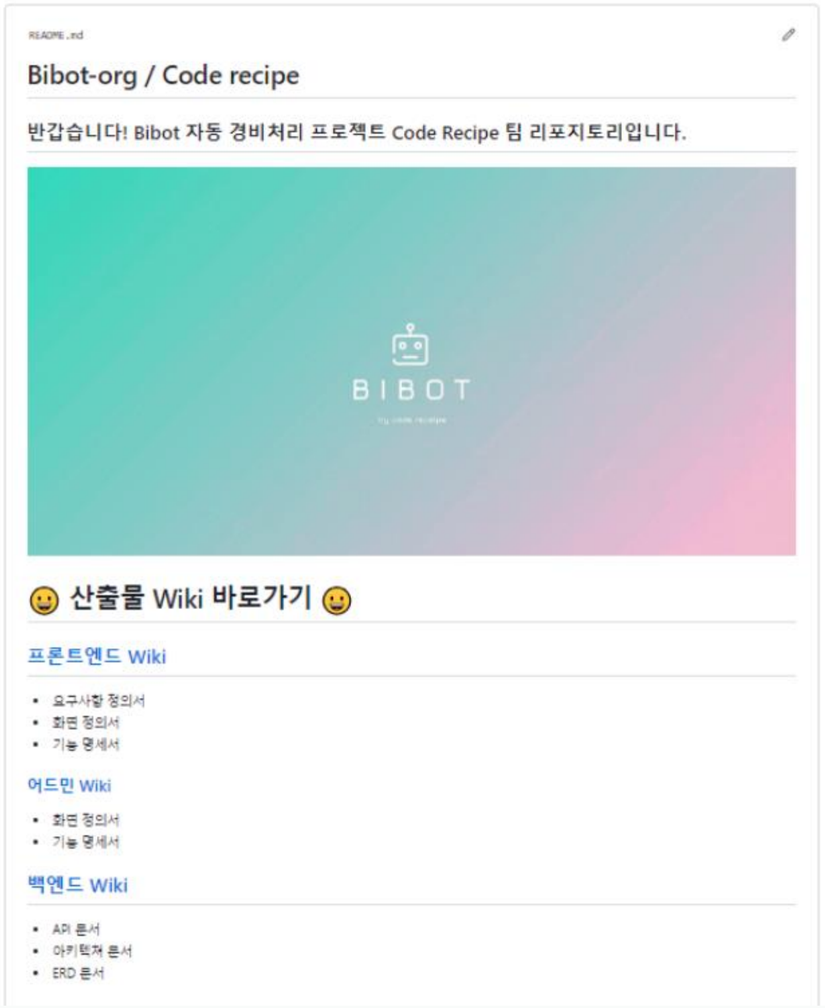
시연 - 경비 처리 실패, 재전송



5. 후기



후기



README.md

Bibot-org / Code recipe

반갑습니다! Bibot 자동 경비처리 프로젝트 Code Recipe 팀 리포지토리입니다.

BIBOT

😀 산출물 Wiki 바로가기 😀

프론트엔드 Wiki

- 요구사항 정의서
- 화면 정의서
- 기능 명세서

어드민 Wiki

- 화면 정의서
- 기능 명세서

백엔드 Wiki

- API 문서
- 아키텍쳐 문서
- ERD 문서

코드레시피 깃허브 주소
-> https://github.com/BiBot-org

후기

01

협업툴을 이용한 개발

협업툴과 git 정책에
따라 개발을 진행하여
프로젝트 진행시
협업 및 관리가
잘 이루어짐

02

MSA 환경 개발

MSA 서비스를
구현하면서
Next.js, TypeScript
이벤트기반 아키텍처,
Kubernetes 등
여러 기술 스택을 사용

03

기업 연계 프로젝트

기업 연계 프로젝트를
수행하면서 B2B
서비스에 대한 이해와
실제 기업에서
사용하는 경비처리
로직을 개선함

후기 - 기획 산출물

요구사항 정의서

사용자

식별코드	요구명	설명	비고
RQ-USR-01	초기 셋업	• 사용자는 마켓 (앱스토어 및 플레이스토어) 에서 애플리케이션을 설치할 수 있어야 한다. • 사용자는 회사 별 워크스페이스 URL을 입력해서 자신의 회사 워크스페이스에 접속할 수 있어야 한다. ◦ 해당 설정이 적용되지 않는다면 사용자는 서비스를 사용할 수 없다.	
RQ-USR-02	로그인 / 로그아웃	• 사용자는 발급 된 사원 전용 계정을 통해 해당 서비스에 로그인 할 수 있어야 한다. • 사용자는 회원가입을 직접 진행할 수 없다. ◦ 사내 시스템과 별개로 서비스는 Third Party로써 제공된다.	
RQ-USR-03	비밀번호 변경	• 사용자는 자신의 비밀번호를 변경할 수 있어야 한다.	
RQ-USR-04	유저 정보 확인 (소속 부서, 직급)	• 사용자는 자신의 정보를 해당 앱에서도 확인할 수 있어야 한다. • 확인할 수 있는 정보는 아래와 같다. ◦ 소속 부서 ◦ 경비 처리 한도 ▪ 지금까지 사용한 경비 ▪ 앞으로 사용할 수 있는 경비	
RQ-USR-05	영수증 경비 처리 요청	• 사용자는 업무 시 사용한 경비 처리를 해당 시스템을 통해 할 수 있다. • 사용자는 영수증의 사진을 찍음으로써, 영수증 데이터 취합 서비스에 데이터 분석 요청을 보낼 수 있다. ◦ 하나의 결제 건에서 1개의 영수증이 등록된다. ◦ 사용자는 영수증 분석 서비스의 분석결과를 직접 확인 할 수 있다. ▪ 45000원이 총 합인데, 47000원이 입력 되었 던 경우 • 사용자는 경비 처리 항목을 선택 할 수 있다. 처리할 수 있는 경비 항목은 아래와 같다. 경비 처리 항목은 사내 정책에 따라 달라질 수 있다. ◦ 식비 ◦ 유류비 • 사용자는 사내 정책에 따라 자동 결제 유무를 확인할 수 있다. ◦ 해당 요구사항은 사내 정책에 따라 달라 질 수 있다. ▪ 한 번의 결제에서 자동 결제 처리되는 식비의 범위는? • 사용자는 카드 결제내역에서 결비 처리요청을 진행할 수 있다.	
RQ-USR-06	영수증 경비 처리 내역 확인	• 사용자는 처리하였던 경비들에 대한 내역을 조회할 수 있다. • 사용자는 기간 별로 경비 처리 내역을 조회할 수 있다. • 사용자는 처리 항목 별로 경비 처리 내역을 조회할 수 있다. • 사용자는 요청의 승인 여부에 따라 경비 처리 내역을 조회 할 수 있다.	

화면 정의서

애플리케이션(사용자)

	화면 명	비고
	메인 로고 화면	• 앱 실행 및 로고화면
	Url 초기 셋업 화면	• 최초 앱 접속 시 회사 URL 입력 화면
	로그인 화면	• 로그인(이메일, 비밀번호) • 비밀번호 찾기
	비밀번호 찾기	• 등록된 이메일로 임시 비밀번호 발급
	메인 화면	• 유저 정보 ◦ 유저 이름 / 팀 => 환영합니다 ◦ 경비 항목 별 경비 처리 현황 ▪ ex) 식비 / 경비 처리 금액 / 앞으로 청구할 수 있는 금액 • 알림 아이콘 (알림 개수) • 공지사항 (ex) 가장 최근 3개~5개) • 하단 메뉴바 ◦ 홈 ◦ 카드 사용내역 ◦ 결제 내역 ◦ 마이페이지
	알림 내역	• 알림 내역 (읽음 여부 표시)
	공지 사항 목록 화면	• 공지사항 페이징 조회 • 공지사항 조건 별 정렬 (미정) ◦ 일반 공지 ◦ 시스템 공지
	공지 사항 상세 화면	• 해당 공지사항 상세 내역 • 이전글 / 다음글
	카드 목록 및 카드 결제 내역 조회 화면	• 카드 등록 / 삭제 • 사용자에게 등록 된 카드 목록 • 카드 별 결제 내역 조회 (결제 별 결제 요청 여부)
	카드 사용내역 결제 요청 화면	• 영수증 촬영 버튼 및 영수증 이미지 등록 • 경비 항목 선택 버튼 • 결제 요청
	결제 내역 목록 화면	• 전체 사용 경비와 남은 경비 • 결제 내역 목록 • 검색조건(최신순, 기간순)

API 명세서

Receipt Service

Receipt

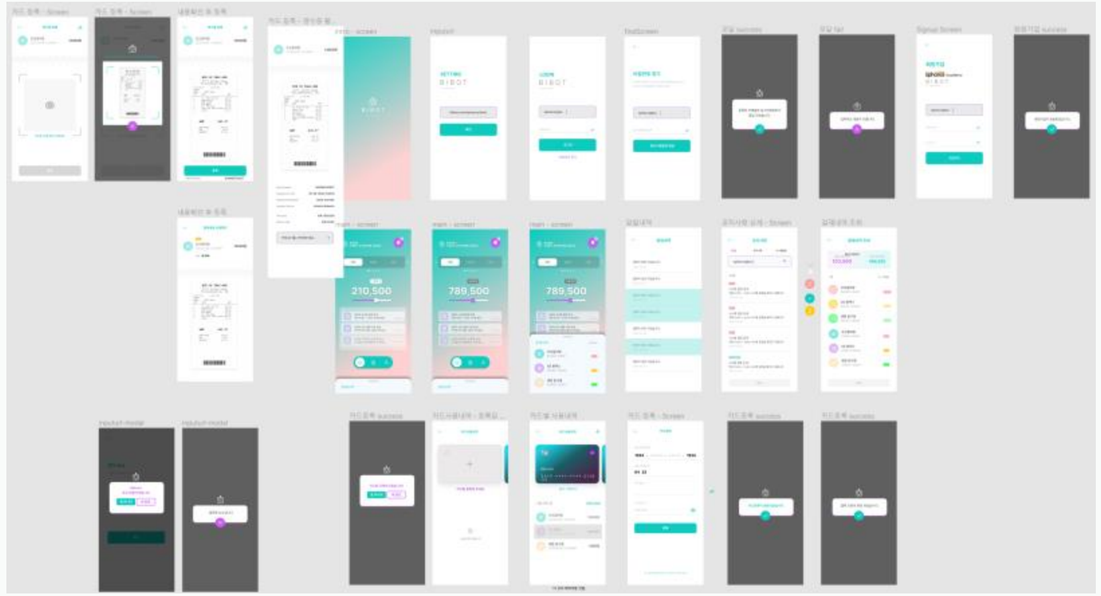
API-RECEIPT-001	영수증 조회	/api/v1/receipt	GET	id : number	`{ status : HttpStatus, data : { id : String, cardId : number, paymentDestination : String, amount : number, approvalId : String, isRequested : boolean } message : "SUCCESS" }`
API-RECEIPT-002	영수증 사진 업로드	/api/v1/receipt/image	POST	`{ file : Multipartfile, cardId : number, categoryId : number, paymentId : String, userId : String }`	`{ status : HttpStatus, data : [sotrageURL : String], message : "SUCCESS" }`
API-RECEIPT-003		/api/v1/receipt/test	POST		

Department

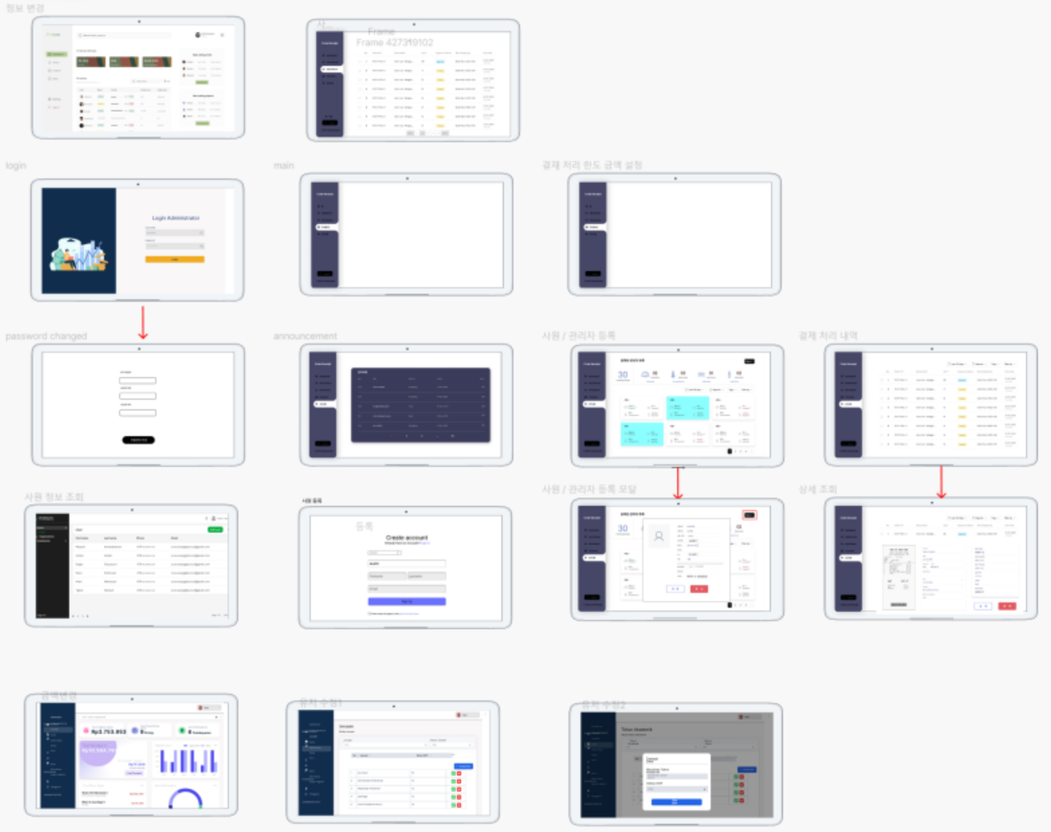
식별 코드	이름	URI	Method	Parameter	Return
API-USER-ADM-009	부서 추가	/api/admin/v1/department	POST	`{ name : string; }`	`{ status : HttpStatus, data : {departmentId : number}, message : "SUCCESS" }`
API-USER-ADM-010	모든 부서 정보 조회	/api/admin/v1/department/all	GET		`{ status : HttpStatus, data : { id : number, name : string; }[], message : "SUCCESS" }`
API-USER-ADM-011	모든 부서 상세 정보 조회	/api/admin/v1/department/info/all	GET		`{ status : HttpStatus, data : { department : { id : number, name : string }, team : { id : number, name : string }, }[], message : "SUCCESS" }`

후기 - 화면설계서

User

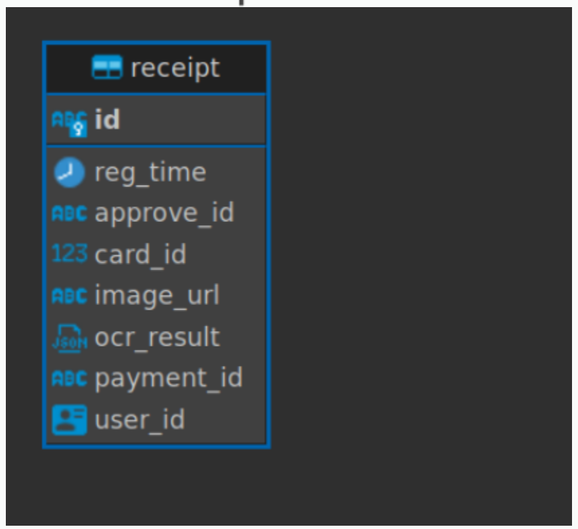


Admin

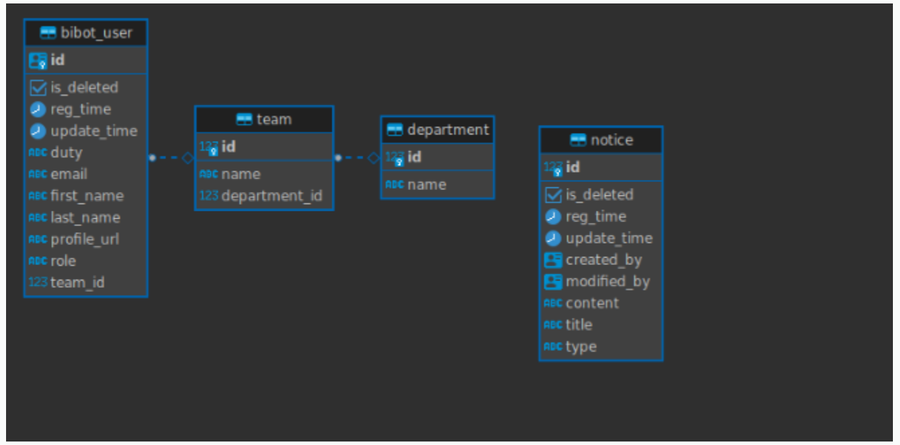


후기 - ERD Diagram

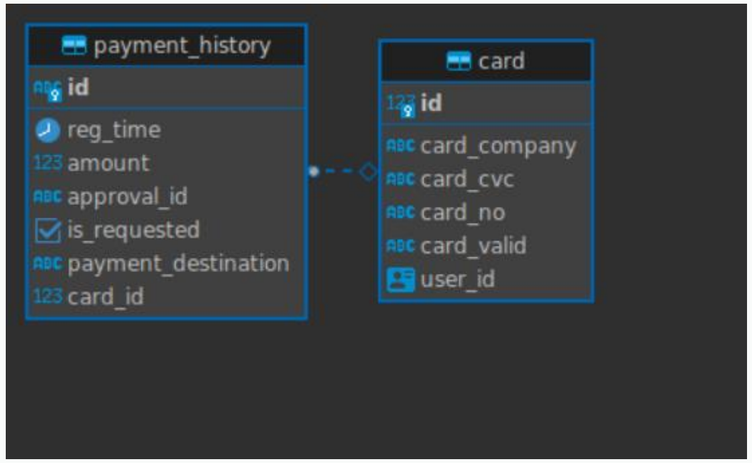
Receipt Service



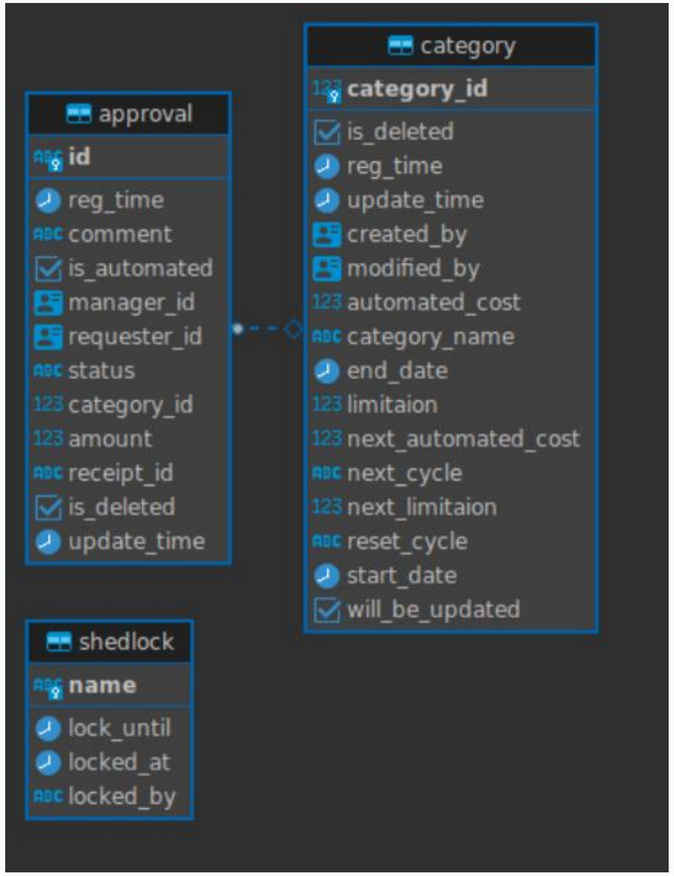
User Service



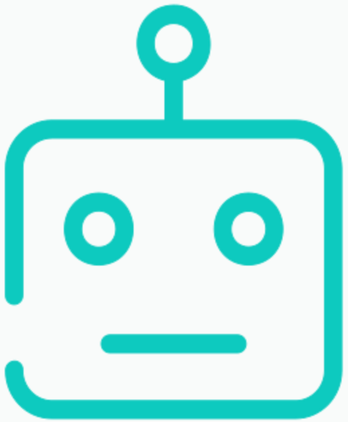
Card Service



Expense Service



6. 팀 소개



코드를 더 맛있게! 우리는 코드레시피!

손우진(FE&BE)

- 애플리케이션 아키텍트
- Kafka 비동기 개발
- SSO 구축(Keycloak)
- 앱인증 로직 개발 (NextAuth)
- 관리자 애플리케이션 개발

박노명(FE)

- 유저 애플리케이션 화면 및 기능 구성
- 유저 애플리케이션 퍼블리싱
- 서버 api(유저, 공지사항, 카드내역, 결재내역) 연동
- 이미지 업로드 기능

김효은(FE)

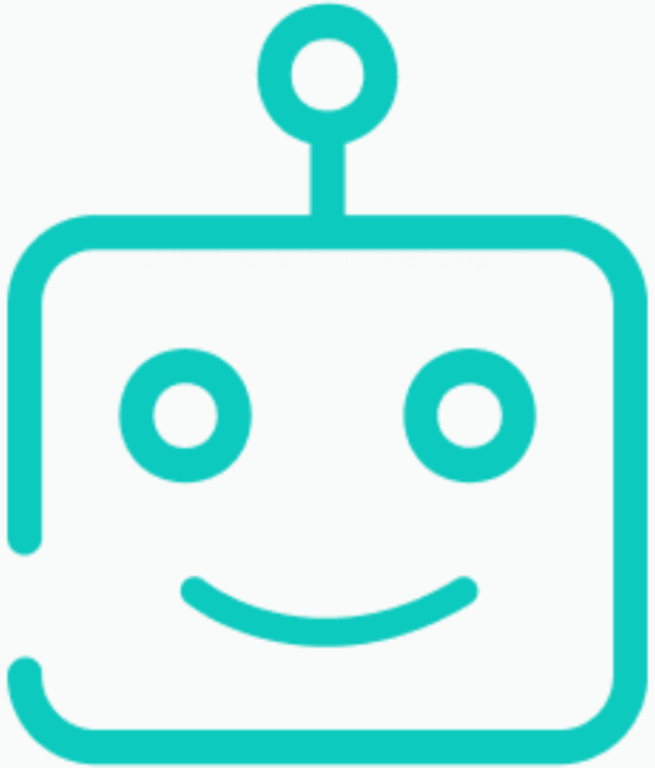
-유저 애플리케이션 화면 및 기능 구성
- 유저 애플리케이션 퍼블리싱
-NextUI(components) 적용
- PWA

노홍기(BE)

- OCR API 연동 및 로직 개발
- 스케쥴러 기반 경비처리 주기 업데이트 로직 개발
- Redis 기반 API 캐싱 환경 구축
- 테스트 엔지니어

박준석(DevOps)

- Google Cloud Platform k8s 서버 운영
- GitOps CI/CD 구축
- Vault 기반 시크릿 서버 구축
- Prometheus & Grafana & Loki 기반 로그 수집 및 모니터링 구축
- DB 서버 운영 및 연결



Thanks!