

|  |
| --- |
|  |



|  |
| --- |
| **Hi-Shop**  **-간편 무인결제 서비스-** |
| **포팅 메뉴얼** |
|  |
|  |
|  |
|  |
|  |

목차

[**I. 개요 2**](#_heading=h.1fob9te)

[**1. 프로젝트 개요 2**](#_heading=h.3znysh7)

[**2. 프로젝트 사용 도구 2**](#_heading=h.2et92p0)

[**3. 개발환경 2**](#_heading=h.tyjcwt)

[**4. 외부 서비스 3**](#_heading=h.3dy6vkm)

[**II. 빌드 3**](#_heading=h.4d34og8)

[**1. 환경변수 형태 3**](#_heading=h.2s8eyo1)

[**2. 빌드하기**](#_heading=h.17dp8vu) 4

[**3. 배포하기**](#_heading=h.3rdcrjn) 7

[**4. 하드웨어 세팅**](#_heading=h.26in1rg) 10

# 개요

## 프로젝트 개요

주말이나 휴일의 대형마트에는 많은 고객이 몰립니다. 쇼핑을 끝낸 수많은 고객이 결제를 위해 계산대로 모여 병목현상이 발생하고, 지루한 대기시간이 발생하게 됩니다.

물건 결제를 위해 바코드를 찍어야 합니다. 바코드를 찍기 위해 카트의 물품을 계산대로 꺼내고, 다시 카트에 담게 됩니다. 그리고 번거롭게도 포장을 위해 다시 물품을 꺼내야 합니다.

‘Hi-Shop’은 위의 두 가지를 개선하기 위해 탄생한 간편 무인결제 서비스입니다.  
저장된 회원의 결제 정보를 사용하여 결제 시간을 줄이고, 결제를 위해 물건을 꺼낼 필요도 없습니다.

## 프로젝트 사용 도구

이슈 관리 : JIRA

형상 관리 : Gitlab

커뮤니케이션 : Mattermost, Discord, Notion

디자인 : Balsamiq, Mui

UCC : Movavi, InVideo

CI/CD : Jenkins

하드웨어 : Raspberry Pi, Arduino, FSR-H3(RFID), FSR-A6(RFID 안테나), DE2110(바코드 리더기), ISO 15693 I-Code2 Card(RFID 태그), 9V 1A 전원 어댑터

## 개발환경

VS Code : 1.74.3

IntelliJ : 2022.3.1 (Ultimate Edition)

JVM : azul-11 (스프링은 11로 빌드)

SpringBoot: 2.7.7

Node.js : 18.13.0 (npm 9.3.0)

SERVER : AWS EC2 Ubuntu 20.04.5 LTS

RaspberryPI : Raspbian GNU/Linux 11 (bullseye)

Arduino Studio : 2.0.4-nightly-20230114

DB : Mysql , Sqlite3

## 외부 서비스

Porkbun : 도메인 구입

CloudFlare: (무료DNS서비스)

# 빌드

## 환경변수 형태

1) 라즈베리파이

websocket port = 3333

front server port = 3000

back server port = 8888

serial\_port = /dev/tty/ACM0

2) AWS EC2

raspberry\_port = 7777

/etc/ssh/sshd\_config 에서

AllowTcpForwarding yes

GatewayPorts yes

설정

## 빌드하기

1) RaspberryPI Back-end

pip install -r requirements

2) RaspberryPI Front-end

npm i

npm run build

3) Frontend DockerFile

FROM node:16.15.0 as build-stage

WORKDIR /var/jenkins\_home/workspace/deploy/front

COPY ./shopmobile/package.json .

COPY ./shopmobile/package-lock.json .

RUN npm install

COPY ./shopmobile .

RUN npm run build

FROM node:16.15.0 as kiosk

WORKDIR /var/jenkins\_home/workspace/deploy/kiosk

COPY ./kiosk/package.json .

COPY ./kiosk/package-lock.json .

RUN npm install

COPY ./kiosk .

RUN npm run build

FROM node:16.15.0 as admin

WORKDIR /var/jenkins\_home/workspace/deploy/admin

COPY ./adminpage/package.json .

COPY ./adminpage/package-lock.json .

RUN npm install

COPY ./adminpage .

RUN npm run build

FROM nginx:stable-alpine as production-stage

COPY --from=build-stage /var/jenkins\_home/workspace/deploy/front/build /usr/share/nginx/html/app

COPY --from=kiosk /var/jenkins\_home/workspace/deploy/kiosk/build /usr/share/nginx/html/kiosk

COPY --from=admin /var/jenkins\_home/workspace/deploy/admin/build /usr/share/nginx/html/admin

COPY --from=build-stage /var/jenkins\_home/workspace/deploy/front/deploy\_conf/nginx.conf /etc/nginx/conf.d/default.conf

EXPOSE 80

EXPOSE 443

CMD ["nginx", "-g","daemon off;"]

4) Backend DockerFile

FROM adoptopenjdk/openjdk11 AS builder

COPY gradlew .

COPY gradle gradle

COPY build.gradle .

COPY settings.gradle .

COPY src src

RUN chmod +x ./gradlew

RUN ./gradlew bootJAR

FROM adoptopenjdk/openjdk11

COPY --from=builder build/libs/\*.jar app.jar

EXPOSE 8080

ENTRYPOINT ["java", "-jar", "/app.jar"]

5) Docker Compose

version: '3'

services:

jenkins:

image: jenkins/jenkins:lts

container\_name: jenkins

volumes:

- /var/run/docker.sock:/var/run/docker.sock

- /jenkins:/var/jenkins\_home

- ./etc/cloudflare/:/etc/cloudflare

- /usr/bin/docker:/usr/bin/docker

ports:

- "9090:8080"

privileged: true

user: root

6) Jenkins Build Steps -> Execute Shell

docker build -t backimg ./Backend

if (docker ps | grep "backimg"); then docker stop backimg; fi

docker run -it -d --rm -p 8080:8080 --name backimg backimg

echo "Run Backend Project"

#Jenkins 내의 ssl key를 docker image로 옮긴다.

docker build -t frontimg-build ./Frontend

if(docker ps -a | grep "frontimg-tmp"); then docker rm frontimg-tmp; fi

# Create a container from the image called foo-tmp

docker create --name frontimg-tmp frontimg-build

# Run the copy command on the container

docker cp /etc/cloudflare frontimg-tmp:/etc/cloudflare

# Commit the container as a new image

docker commit frontimg-tmp frontimg

# The new image will have the files

# docker run foo ls /dest

if (docker ps | grep "frontimg"); then docker stop frontimg; fi

docker run -it -d --rm -p 80:80 -p 443:443 --name frontimg frontimg

echo "Run Frontend Project"

## 배포하기

1)

Nginx 설정

limit\_req\_zone $binary\_remote\_addr zone=ddos\_req:10m rate=1r/s;

upstream backend{

ip\_hash;

server 172.26.5.42:8080;

}

server {

listen 80;

listen [::]:80;

server\_name himart.shop;

return 301 HTTPS://$server\_name$request\_uri;

}

server {

listen 443 ssl;

listen [::]:443 ssl;

server\_name XXX;

ssl\_certificate /etc/cloudflare/hishop.pem;

ssl\_certificate\_key /etc/cloudflare/hishop.key;

return 301 https://himart.shop$request\_uri;

}

server {

client\_body\_timeout 5s;

client\_header\_timeout 5s;

listen 443 ssl;

listen [::]:443 ssl;

server\_name himart.shop;

ssl\_certificate /etc/cloudflare/hishop.pem;

ssl\_certificate\_key /etc/cloudflare/hishop.key;

access\_log /var/log/nginx/access.log;

error\_log /var/log/nginx/error.log;

if ($http\_range ~ "d{9,}") {

return 444;

}

location /app {

limit\_req zone=ddos\_req burst=5 nodelay;

root /usr/share/nginx/html;

index index.html index.htm;

try\_files $uri $uri/ /app/index.html;

}

location /api {

limit\_req zone=ddos\_req burst=5 nodelay;

proxy\_pass http://backend;

proxy\_redirect off;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

}

location /kiosk {

limit\_req zone=ddos\_req burst=5 nodelay;

root /usr/share/nginx/html;

index index.html index.htm;

try\_files $uri $uri/ /kiosk/index.html;

}

location /admin {

limit\_req zone=ddos\_req burst=5 nodelay;

root /usr/share/nginx/html;

index index.html index.htm;

try\_files $uri $uri/ /admin/index.html;

}

#error\_page 404 /404.html;

# redirect server error pages to the static page /50x.html

#

error\_page 500 502 503 504 /50x.html;

location = /50x.html {

root /usr/share/nginx/html;

}

}

2) 라즈베리파이 서버 실행

* kiosk backend server

uvicorn main:app --host=0.0.0.0 --port=8888

* kiosk front server

npm start

* kiosk websocket server

node index.js

* kiosk ssh reverse ssh tunnelling 연결하기

ssh -i {SSH\_PEM\_KEY} - R {PORT}:localhost:{PORT} {AWS-EC2 계정@주소}

지금 상황의 경우

: 7777:localhost:8888

: ubuntu@XXX

## 하드웨어 세팅

1. 아두이노 - FSR-H3(RFID 리더기)  
   FSR-H3 전원부 GROUND선 Arduino Digital GND 핀 연결

FSR-H3 TX선 Arduino Digital 4번 핀 연결

FSR-H3 RX선 Arduino Digital 7번 핀 연결

FSR-H3 와 안테나(FSR-A6) 전용 선으로 연결

1. 라즈베리파이

CSI-camera 포트에 라즈베리파이 카메라 연결

미니 HDMI 포트 - 디스플레이 연결

usb - 터치 연결

해상도 설정 - 1280 x 720