포팅 매뉴얼

목차

    - 입장권
    - 구매 내역
    - 환불
    - 그 외 작업중인 추첨결과 및 티켓북 탭 화면

외부 서비스

- Amazon EC2
- Amazon S3
- Oracle Cloud Compute
- Cloudflare S3
- CoolSMS
- toss payments

빌드 및 배포 정보

카프카, 백엔드, 프론트엔드, 블록체인 서비스를 배포하는 데 필요한 정보를 정리합니다.

Linux 패키지 매니저: mise

mise 설치 명령어는 다음과 같습니다.

```
curl https://mise.run | sh
echo 'eval "$(~/.local/bin/mise activate bash)"' >> ~/.bashrc
source ~/.bashrc
mise --version
```

- Node.js 설치

  ```
  mise use -g node@20
  node --version
  ```

- OpenJDK 설치

  ```
  mise use -g java@17
  java --version
  ```

Windows 패키지 매니저: scoop

scoop 설치 명령어는 다음과 같습니다. (non-admin)

```
irm get.scoop.sh | iex
scoop --version
```

admin 권한에서는 다음 명령어를 통해 설치해야 합니다.

```
irm get.scoop.sh -outfile 'install.ps1'
.\install.ps1 -RunAsAdmin
scoop --version
```

- Node.js 설치

```
scoop install nodejs-lts
node --version
```

- OpenJDK 설치

```
scoop bucket add java
scoop install openjdk17
java --version
```

카프카

1. EC2 Instance 생성
    - AWS에서 Kafka를 구축하는 방법은 두 가지가 있습니다.
    - 첫 번째는 MSK(Managed Service Kafka)를 사용하는 것이고,
    - 두 번째는 EC2에서 인스턴스를 발급 받아서 설치 및 실행하는 방식입니다. 프로젝트에서는 후자를 선택했습니다.
    - OS는 프리 티어로 사용 가능한 Amazon Linux 2 AMI를 선택합니다.
    - 추후 주키퍼와 카프카 브로커는 JVM 위에서 돌아가는 애플리케이션으로서 힙 메모리를 지정해야 합니다.
    - 두 프로세스에 각각 400MB의 힙 메모리를 설정하려면 1G 이상의 램이 필요하므로 적당한 t2.micro type을 선택합니다.
    - SSH로 인스턴스에 접속하여 실습을 진행하려면 KeyPair를 발급 받아야 합니다. 이미 발급 받은 KeyPair가 없다면 '새 키 페어 생성'을 통해 발급 받아야 합니다. 나머지 설정은 Default 값을 사용할 예정이므로, 인스턴스 시작을 눌러 설정을 완료합니다.
2. 보안그룹 설정
    - EC2에 설치 된 브로커에 접속하기 위해서는 EC2 보안그룹의 Inbound 설정에 `9092`와 `2181` 포트를 열어야 합니다.
    - 실습을 위해 source IP를 `any`로 설정하겠습니다. 실제 상용 환경에서는 적합한 IP를 대상으로 설정하면 됩니다.
    - 보안그룹 인바운드 규칙에 `9092`와 `2181` 포트를 추가합니다.
    - 발급 받은 KeyPair에 위 명령어로 Read 권한만 가지고 있게 400으로 설정합니다.
    - EC2 > 인스턴스 > 연결 화면에 가서 ssh 명령어를 복사한 뒤, 터미널에 입력하여 EC2에 접속합니다.
3. 자바 설치

```
$ sudo yum install -y java-1.8.0-openjdk-devel.x86_64
...
Complete!
$ java -version
openjdk version "1.8.0_392"
OpenJDK Runtime Environment (build 1.8.0_392-b08)
OpenJDK 64-Bit Server VM (build 25.392-b08, mixed mode)
```

4. 주키퍼 & 카프카 브로커 실행

- 카프카 브로커를 실행하기 위해서 카프카 바이너리 패키지를 다운로드 합니다. wget 명령어와 바이너리 패키지의 URL을 넣으면 카프카 패키지를 EC2 인스턴스에 다운로드 할 수 있습니다. 다운로드가 완료 되면 tar 명령어와 xvf 옵션을 통해 압축 파일을 풉니다. cd 명령어로 kafka_2.12-2.5.0 디렉토리로 이동해보면 정상적으로 설치된 것을 확인할 수 있습니다.

```
$ wget https://archive.apache.org/dist/kafka/2.5.0/kafka_2.12-2.5.0.tgz
$ tar xvf kafka_2.12-2.5.0.tgz
$ ll
합계 60164
drwxr-xr-x 6 ec2-user ec2-user       89  4월  8  2020 kafka_2.12-2.5.0
-rw-rw-r-- 1 ec2-user ec2-user 61604633  7월  6  2020 kafka_2.12-2.5.0.tg
$ cd kafka_2.12-2.5.0
```

5. 카프카 브로커 힙 메모리 설정

```
$ export KAFKA_HEAP_OPTS="-Xmx400m -Xms400m"
$ echo $KAFKA_HEAP_OPTS
-Xmx400m -Xms400m
```

- 카프카 브로커를 실행하기 위해서는 힙 메모리 설정이 필요합니다. 카프카 브로커는 레코드의 내용은 페이지 캐시로 시스템 메모리를 사용하고, 나머지 객체들을 힙 메모리에 저장하여 사용한다는 특징이 있습니다. 이러한 특징으로 운영할 때 힙 메모리를 5GB 이상으로 설정하지 않는것이 일반적입니다. 카프카 패키지의 힙 메모리는 브로커는 1G, 주키퍼는 512MB로 기본 설정 되어 있습니다.
- 실습용으로 생성한 해당 EC2 인스턴스(t2.micro)는 1G 메모리를 가지고 있으므로 카프카 브로커와 주키퍼를 기본 설정값으로 실행하면 1.5G 메모리가 필요하기 때문에 Cannot allocate memory 에러가 출력되면서 실행 되지 않습니다.
- 이를 해결하기 위해 export 명령어로 힙 메모리 사이즈를 미리 환경변수로 지정한 뒤 실행해야 합니다.
- bashrc 설정

```
$ vi ~/.bashrc   (1)
-------------------------------------------------------------------------
---------
# .bashrc
# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi
# Uncomment the following line if you don't like systemctl's auto-paging
feature:
# export SYSTEMD_PAGER=
# User specific aliases and functions
export KAFKA_HEAP_OPTS="-Xmx400m -Xms400m"  (2)
$ source ~/.bashrc (3)
$ echo $KAFKA_HEAP_OPTS (4)
```

- 터미널에서 사용자가 입력한 KAFKA_HEAP_OPTS 환경변수는 터미널 세션이 종료되고 나면 초기화 되어 재사용이 불가능합니다. 따라서 ~/.bashrc 파일에 입력하여 bash 쉘이 실행될 때마다 설정할 수 있도록 수정합니다.

6. 카프카 브로커 실행 옵션 설정

- config 폴더에 있는 `server.properties` 파일에는 카프카 브로커가 클러스터 운영에 필요한 옵션들을 지정할 수 있습니다. 저희는 실습용 카프카 브로커를 실행할 것이므로 `advertised.listener`만 설정하면 됩니다. 해당 설정 파일에서 현재 접속하고 있는 인스턴스의 퍼블릭 IP와 카프카 기본 포트 `9092`를 `PLAINTEXT://`와 함께 붙여넣고 `advertised.listeners`를 주석에서 해제합니다.
- `advertised.listener` : 카프카 클라이언트 또는 커맨드 라인 툴을 브로커와 연결할 때 사용됩니다.

```
vi config/server.properties
```

- `#advertised.listeners=PLAINTEXT://your.host.name:9092` 부분에서 #을 삭제하여 주석을 해제하고,
- `advertised.listeners=PLAINTEXT://13.125.237.53:9092` 처럼 자신의 인스턴스 퍼블릭 IP로 변경합니다.
- 참고로, `log.retention.hours` 설정은 카프카 브로커가 저장한 파일이 삭제되기까지 걸리는 시간을 지정하는 옵션입니다. 가장 작은 단위를 기준으로 하므로 상용 환경에서는 `log.retention.hours` 보다는 `log.retention.ms` 값을 설정하는것을 추천합니다. 또한 해당 설정 값을 -1로 설정하면 파일은 영원히 삭제 되지 않습니다.

7. 주키퍼 실행

- 카프카 바이너리가 포함된 폴더에는 브로커와 같이 실행할 주키퍼가 준비되어 있습니다. 분산 코디네이션 서비스를 제공하는 주키퍼는 카프카의 클러스터 설정 리더 정보, 컨트롤러 정보를 담고 있어 카프카를 실행하는 데에 필요한 필수 애플리케이션입니다. 상용에서는 안전하게 운영하기 위해서 3대 이상의 서버로 구성하여 사용하지만 실습에서는 동일한 서버에 카프카와 동시에 1대만 실행시켜 사용하겠습니다.
- 아래 명령어로 주키퍼를 실행합니다. 또한 주키퍼가 정상적으로 실행되었는지 확인하기 위해 jps 명령어를 입력합니다.
  - `jps`: JVM 프로세스 상태를 보는 도구로서, JVM 위에서 동작하는 주키퍼의 프로세스를 확인할 수 있습니다.
  - `-v option`: JVM에 전달된 인자 (힙 메모리 설정, log4j 설정 등)를 확인할 수 있습니다.
  - `-m option`: main 메서드에 전달된 인자를 확인할 수 있습니다.

```
bin/zookeeper-server-start.sh -daemon config/zookeeper.properties
jps -vm
```

8. 카프카 브로커 실행 및 로그 확인

- 이제 카프카 브로커를 실행할 마지막 단계입니다. `kafka-server-start.sh` 명령어를 통해 카프카 브로커를 실행한 뒤, jps 명령어를 통해 주키퍼와 브로커 프로세스의 동작 여부를 알 수 있습니다. 이후 `tail` 명령어를 통해 로그를 확인하여 정상 동작하는지 확인합니다.

```
$ bin/kafka-server-start.sh -daemon config/server.properties
$ jps -m
22226 QuorumPeerMain config/zookeeper.properties
9506 Kafka config/server.properties

$ tail -f logs/server.log
[2024-01-11 06:31:05,180] INFO [TransactionCoordinator id=0] Starting up.
(kafka.coordinator.transaction.TransactionCoordinator)
```

```
[2024-01-11 06:31:05,203] INFO [TransactionCoordinator id=0] Startup complete.
(kafka.coordinator.transaction.TransactionCoordinator)
[2024-01-11 06:31:05,204] INFO [Transaction Marker Channel Manager 0]: Starting
(kafka.coordinator.transaction.TransactionMarkerChannelManager)
[2024-01-11 06:31:05,347] INFO [ExpirationReaper-0-AlterAcls]: Starting
(kafka.server.DelayedOperationPurgatory$ExpiredOperationReaper)
[2024-01-11 06:31:05,457] INFO [/config/changes-event-process-thread]: Starting
(kafka.common.ZkNodeChangeNotificationListener$ChangeEventProcessThread)
[2024-01-11 06:31:05,510] INFO [SocketServer brokerId=0] Started data-plane
processors for 1 acceptors (kafka.network.SocketServer)
[2024-01-11 06:31:05,538] INFO Kafka version: 2.5.0
(org.apache.kafka.common.utils.AppInfoParser)
[2024-01-11 06:31:05,538] INFO Kafka commitId: 66563e712b0b9f84
(org.apache.kafka.common.utils.AppInfoParser)
[2024-01-11 06:31:05,538] INFO Kafka startTimeMs: 1704954665511
(org.apache.kafka.common.utils.AppInfoParser)
[2024-01-11 06:31:05,540] INFO [KafkaServer id=0] started
(kafka.server.KafkaServer)
```

콘팅은 다음 3가지 토픽을 사용합니다. 미리 토픽을 생성해놓아 warn 알림이 방지됩니다.

```
./bin/kafka-topics.sh --create --bootstrap-server ${카프카 인스턴스의 퍼블릭
IP}:9092 --replication-factor 1 --partitions 1 --topic success_order
./bin/kafka-topics.sh --create --bootstrap-server ${카프카 인스턴스의 퍼블릭
IP}:9092 --replication-factor 1 --partitions 1 --topic failure_order
./bin/kafka-topics.sh --create --bootstrap-server ${카프카 인스턴스의 퍼블릭
IP}:9092 --replication-factor 1 --partitions 1 --topic update_seat
```

백엔드

백엔드 서비스들은 `server` 디렉터리에 위치하며, 목록은 다음과 같습니다.

- batch-service
- blockchain-public-service
- blockchain-service
- catalog
- did
- discovery-service
- gateway
- notification
- payment
- queue
- seat
- ticket
- user

필요 환경

- OpenJDK 17.0.2
- Docker 26
- Node.js 20.11.1
- Pnpm 8.15.5
- ffmpeg 6.1.1
- yt-dlp 2024.03.10

catalog, did, discovery-service, gateway, notification, payment, queue, seat, ticket, user

다수의 백엔드 서비스는 아래와 유사한 프로젝트 구조를 가집니다. 서비스의 특성에 따라 `database` 디렉터리와 `docker-compose.yml`은 없을 수 있습니다.

```
 .
├──  gradle
└──  src
├──  build.gradle
├──  Dockerfile
├──  gradlew
├──  gradlew.bat
├──  Jenkinsfile
├──  settings.gradle
├── ...
```

`user` 프로젝트의 `Jenkinsfile`에 정의되어 있는 빌드 스크립트는 다음과 같습니다. gradle을 통해 생성한 대상 파일을 도커 이미지로 빌드하고 허브에 업로드합니다.

```
sh 'cd ./server/user && ./gradlew clean build'
dockerImage = docker.build("${repository}:User_${BUILD_NUMBER}", "-f
server/user/Dockerfile ./server/user")
sh "echo \${DOCKERHUB_CREDENTIALS_PSW} | docker login -u
\${DOCKERHUB_CREDENTIALS_USR} --password-stdin"
sh "docker push \${repository}:User_\${BUILD_NUMBER}"
sh "docker rmi \${repository}:User_\${BUILD_NUMBER}"
```

예를 들어, `user` 서비스를 빌드하기 위해 다음과 같은 셸 스크립트를 수행합니다.

```
./gradlew clean build
docker build -t 1w2k/c209:User_4 -f server/user/Dockerfile ./server/user
docker login -u 1w2k --password-stdin
docker push 1w2k/c209:User_4
docker rmi 1w2k/c209:User_4
```

빌드가 종료되면 지정된 각 서버에 ssh 연결을 수립하고, 도커 이미지를 풀링하고 실행하는 스크립트를 실행시킵니다.

```
stage("Deploy"){
  steps{
```

```
    sshPublisher(
      continueOnError: false, failOnError: true,
      publishers: [
        sshPublisherDesc(
          configName: "User_Service",
          verbose: true,
          transfers: [
            sshTransfer(execCommand: "sudo docker rm -f user"),
            sshTransfer(execCommand: "sudo docker pull " + repository
+":User_${BUILD_NUMBER}"),
            sshTransfer(execCommand: "sudo docker run -d -p 8881:8881 --name user
--log-driver=fluentd -e DISCOVERY_SERVER=\"${DISCOVERY_SERVER}\" -e
USER_DB_URL=\"${USER_DB_URL}\" -e USER_DB_USER_NAME=\"${USER_DB_USER_NAME}\" -e
USER_DB_PASSWORD=\"${USER_DB_PASSWORD}\" -e JWT_ACCESS=\"${JWT_ACCESS}\" -e
JWT_REFRESH=\"${JWT_REFRESH}\" -e COOL_SMS_API_KEY=\"${COOL_SMS_API_KEY}\" -e
COOL_SMS_SECRET_KEY=\"${COOL_SMS_SECRET_KEY}\" -e
PHONE_NUMBER=\"${PHONE_NUMBER}\" -e USER_HOST=\"${USER_HOST}\" "+ repository +
":User_${BUILD_NUMBER}")
          ]
        )
      ]
    )
  }
}
```

예를 들어, `user` 서버에서는 다음과 같은 명령어가 실행됩니다.

```
sudo docker system prune -af
sudo docker rm -f user
sudo docker pull 1w2k/c209:User_4
sudo docker run -d -p 8881:8881 --name user --log-driver=fluentd -e
DISCOVERY_SERVER="http://3.36.51.150:8761/eureka" -e
USER_DB_URL="jdbc:mariadb://43.201.147.179:9996/user?
serverTimezone=UTC&useUnicode=true&characterEncoding=utf8" -e
USER_DB_USER_NAME="" -e USER_DB_PASSWORD="" -e JWT_ACCESS="" -e JWT_REFRESH="" -e
COOL_SMS_API_KEY="" -e COOL_SMS_SECRET_KEY="" -e PHONE_NUMBER="" -e
USER_HOST="43.201.147.179" 1w2k/c209:User_4
```

batch-service

빌드 명령어는 다음과 같습니다.

```
.\gradlew clean build
```

실행시 다음과 같은 환경변수를 요구합니다.

```
S3_ACCESS_KEY_ID=722cb3a800032f5f8c932ba31a6acf4f
S3_SECRET_ACCESS_KEY=8092d7f3bb3b4c7a24c4c68eff333e887e3d857071b37045b215562f8cd2
755e
S3_REGION=apac
S3_BUCKET=conting
S3_ENDPOINT=https://dcfbfd9f489ad8a785c3cdb81469e07d.r2.cloudflarestorage.com
S3_PUBLIC_BASE_URL=https://pub-42d3d2de01ff4e1baef74a4d07121130.r2.dev
WEB3_WRAPPER_BASE_URL=http://blockchain-service:3000
```

blockchain-public-service, blockchain-service

다음 명령어를 통해 종속성 모듈을 설치합니다.

```
corepack enable pnpm
pnpm i
```

`.env` 파일에 다음과 같이 환경변수를 설정합니다.

```
SERVER_PORT=3000
RPC_ENDPOINT=https://api.devnet.solana.com
WALLET_SECRET=
[135,165,80,133,50,218,215,55,201,42,252,241,128,85,70,128,120,228,96,149,132,233
,12,91,80,171,24,88,221,62,122,144,13,8,56,197,82,98,18,100,128,135,71,129,60,75,
168,250,197,12,248,200,27,139,105,84,204,10,28,2,12,190,20,159]⏎
```

서버를 실행합니다.

```
pnpm build
nohup sudo pnpm start &
```

데이터베이스

데이터베이스는 아래와 같이 `database` 디렉터리와 `docker-compose.yml` 파일이 있을 경우 실행될 필요가 있습니다.

```
 .
├── database
|   ├── config
|   |   └── mariadb.conf
|   └── Dockerfile
├── docker-compose.yml (또는 docker-compose-local.yml)
├── ...
```

`docker compose` 명령어를 통해 DB 서버를 정해진 포트, 유저네임, 패스워드 정보에 따라 실행합니다.

```
docker compose -f docker-compose-local.yml -p did-db up -d
```

프론트엔드

필요 환경

- Node.js 20.11.1
- Npm 10.2.4
- OpenJDK 17.0.2 (Kotlin)

환경 설치

윈도우 기준입니다.

- Android Studio
  - 인텔 프로세서는 UEFI/BIOS에서 `VT-x`를 활성화합니다.
  - Windows Features에서 `Windows Hypervisor Platform`(필수)와 `Hyper-V`(선택)를 활성화합니다.
  - https://developer.android.com/studio에서 설치 파일을 다운로드하고 `Android Virtual Device`과 함께 설치를 진행합니다.
  - 안드로이드 스튜디오를 실행하고 설치 유형 `Standard`을 고르고 `Android SDK`, `Android SDK Platform`을 포함해 추가 설치를 진행합니다.

에뮬레이터 실행

종속성 모듈을 설치합니다.

```
cd S10P22C209/app/frontend
npm i
```

안드로이드 스튜디오에서 에뮬레이터를 실행하고 다음 명령어를 실행합니다.

```
npm run android
```

metro 대화형 창에서 `a`를 입력합니다.

```
info React Native v0.73.6 is now available (your project is running on v0.73.5).
info Changelog: https://github.com/facebook/react-native/releases/tag/v0.73.6
info Diff: https://react-native-community.github.io/upgrade-helper/?from=0.73.6
info For more info, check out "https://reactnative.dev/docs/upgrading?
os=windows".
info Dev server ready

i - run on iOS
a - run on Android
d - open Dev Menu
r - reload app
```

APK 빌드

아래 명령어를 실행하여 bundle을 생성합니다.

```
npx react-native bundle --platform android --dev false --entry-file index.js --
bundle-output android/app/src/main/assets/index.android.bundle --assets-dest
android/app/src/main/res/
```

안드로이드 스튜디오로 `android` 디렉터리를 열고 초기 임포트 과정이 끝나면 `Build > Build Bundle(s) / APK(s) > Build APK(s)`을 실행합니다.

`android/app/build/outputs/apk/debug/app-debug.apk`이 생성됐는지 확인합니다.

블록체인

리눅스 OS를 가정합니다.

필요 환경

- Rust 1.77.0
- Node.js 20.11.1
- Yarn 1.22.22
- Solana CLI 1.18.8
- Anchor 0.29.0

환경 설치

- Rust

  ```
  sudo apt-get update
  sudo apt-get install -y \
      build-essential \
      pkg-config \
      libudev-dev llvm libclang-dev \
      protobuf-compiler libssl-dev

  curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh -s -- -y
  source ~/.bashrc
  cargo --version
  ```

- Node & Yarn

  ```
  corepack enable yarn
  yarn --version
  ```

- Solana CLI

  ```
  sh -c "$(curl -sSfL https://release.solana.com/v1.18.8/install)"
  export
  PATH="/home/$HOME/.local/share/solana/install/active_release/bin:$PATH"
  solana --version
  ```

- Anchor

```
cargo install --git https://github.com/coral-xyz/anchor avm --locked --force
avm install 0.29.0
avm use 0.29.0
anchor --version
```

사전 준비

개발을 위해 솔라나 네트워크를 mainnet-beta에서 devnet으로 변경하고 새 지갑을 생성합니다.

`solana airdrop 2` 명령어가 실패하면 `https://faucet.solana.com/`에 접속해 지갑 공개키(`solana address`)를 입력해 개발용 SOL을 요청합니다.

```
solana config set --url devnet
solana-keygen new
solana airdrop 2
solana balance
```

프로그램 빌드

아래 명령어를 통해 솔라나 프로그램을 빌드합니다.

```
cd S10P22C209/blockchain
anchor build
```

아래와 같은 에러가 발생할 시, 요구 버전(여기서는 `v1.18.8`)에 맞춰 Solana CLI 설치 과정을 다시 수행합니다.

```
error: package `solana-program v1.18.8` cannot be built because it requires rustc
1.75.0 or newer, while the currently active rustc version is 1.72.0-dev
Either upgrade to rustc 1.75.0 or newer, or use
cargo update -p solana-program@1.18.8 --precise ver
where `ver` is the latest version of `solana-program` supporting rustc 1.72.0-dev
```

빌드가 성공하면 `anchor keys list` 명령어를 실행하여 `did`, `event`, `market` 프로그램에 부여된 공개키를 확인합니다.

```
did: ArThQi7nofBbd34T4V7TuXivK9WRLYmUw9pkAQ6rsZxd
event: 3G46wqS8SHRvtqp7gLxoenn2G2apwke3MVNSnhq9sgrf
market: EJM4k5GaVQgucKUvKkJaKPmRdSEcavuMnC18nFExJVvB
```

공개키에 따라 `Anchor.toml`의 `[programs.localnet]`을 수정합니다.

```
# BEFORE:
[programs.localnet]
did = "DiDiDgTdcYhe7jemETo4u5B6GNtsv1BPDnRHBQJtVoEj"
event = "Even2kqboEgiEv8ozq4fMyiDi727VeRbTD7SQogF5vrn"
market = "MarqygkQw8N9f1byiDrWvtbKs6iDfeWiUmBpovQiJpi"
# AFTER:
[programs.localnet]
did = "ArThQi7nofBbd34T4V7TuXivK9WRLYmUw9pkAQ6rsZxd"
event = "3G46wqS8SHRvtqp7gLxoenn2G2apwke3MVNSnhq9sgrf"
market = "EJM4k5GaVQgucKUvKkJaKPmRdSEcavuMnC18nFExJVvB"
```

`programs/did/src/lib.rs`, `programs/event/src/lib.rs`, `programs/market/src/lib.rs`에 정의된 `declare_id!`도 수정합니다.

```
// BEFORE
declare_id!("DiDiDgTdcYhe7jemETo4u5B6GNtsv1BPDnRHBQJtVoEj");
declare_id!("Even2kqboEgiEv8ozq4fMyiDi727VeRbTD7SQogF5vrn");
declare_id!("MarqygkQw8N9f1byiDrWvtbKs6iDfeWiUmBpovQiJpi");
// AFTER
declare_id!("ArThQi7nofBbd34T4V7TuXivK9WRLYmUw9pkAQ6rsZxd");
declare_id!("3G46wqS8SHRvtqp7gLxoenn2G2apwke3MVNSnhq9sgrf");
declare_id!("EJM4k5GaVQgucKUvKkJaKPmRdSEcavuMnC18nFExJVvB");
```

`~/.config/solana/id.json`에 저장된 64개 배열(개인키 + 공개키)에서 하위 32개 배열(공개키)을 추출하여 각 `libs.rs`에 정의된 `SERVER_PUBKEY`를 변경합니다.

```
const SERVER_PUBKEY: Pubkey = Pubkey::new_from_array([
    136, 5, 219, 160, 32, 90, 40, 48, 191, 238, 134, 32, 24, 42, 140, 100, 90,
234, 161, 150, 187,
    81, 89, 3, 188, 143, 164, 145, 14, 44, 167, 74,
]);
```

설정과 코드 수정이 완료되면 `anchor build`를 다시 실행합니다.

인스트럭션 테스트

`Anchor.toml`에는 `devnet`에서 복사할 대상이 정의되어 있습니다. 기존 항목을 참고해 계정과 NFT를 생성하고 테스트셋을 설정합니다.

`Anchor.toml` 설정이 끝나면 `.env` 파일을 생성하고 아래와 같이 환경변수를 설정합니다.

```
AGENCY_PUBKEY=AgenyMmqxHrgfkpd5bYMPrRfCy19TeJqc36eDhZPnnks
SINGER_PUBKEY=SingYu179NHKSG2nZdzewMedFB5XBukWRQHtukxJQoh
SELLER_PUBKEY=Se1btqDGaoerSt3WKeZvnuk3x2RzKCyi58bepQQUKNg
BUYER_PUBKEY=Buyrnk6tmcAgo6pPSvGeJDfCpsKW79UiuUph7uyfmpWd
PARTICIPANT_1_PUBKEY=Pa1Kw6Vp7rQ7Eqr9roAhBryTqRVbBTypaQBTnTvu6J3
PARTICIPANT_2_PUBKEY=Pa2EgvfnhAVGQ2DWz9LAHwu8zuhxfi8hdwLx9M9Je4U
PARTICIPANT_3_PUBKEY=Pa3dfT2TUDHxnQtTDjNRPHukKmCfgJ7ioS4AnKAi5Hf
AGENCY_SECRET=
SINGER_SECRET=
SELLER_SECRET=
BUYER_SECRET=
PARTICIPANT_1_SECRET=
```

```
PARTICIPANT_2_SECRET=
PARTICIPANT_3_SECRET=
COLLECTION_MINT=Co5WuDEf6FddVjDypFxJW93tRZL3VCzar8EuzLXQBk54
SELLERS_MINT=Ase5KMdWssiBKyE3iSQU4cKys4XqBeQyEmmDEQPfY9C4
PARTICIPANT_1_MINT=AsGTqXeLLP37MEB5W49X6fh2G5LiHccSXZ6TzYfB86C1
PARTICIPANT_2_MINT=Asexa6m2XnoTXq3Bhe83XA6jeRk5Gdrxhh7VEC8jcnNJ
PARTICIPANT_3_MINT=AshKvETDdyTYJqEmKk7Eeqqd5ggnRLgvqTYN3ywroemQ
```

`anchor test --detach` 명령어를 실행하여 테스트 코드를 실행합니다.

```
$ anchor test --detach
  did
TxHash ::
544bxrb7h4nr1mWAzMcyXsjeBSLB7Jwcr8NQSAmhvYJbpHXDwgeAYtDDnWsHxyUJs2Lr8ytRWaonEwdJi
3vJM2od
    ✓ Issue Cert 1+2 (360ms)
TxHash ::
2qnfXpvTZBKf8pb99CLWhkU4di3bVQeb9gnGp5WtGNEMTsA8YrkhF8vi4pBeKEez6t7o51pjUDhW6kCQZ
azFRnPq
    ✓ Issue Cert 2+3 (403ms)
TxHash ::
4ArsCKXQCdzUqXXo1QjbQrpJpSZdvXjdJeeijSxNsRacrffg6hP6JSSX9zrXHo4zK8KJhfMAxAkwEDEvV
A9VVBw8
    ✓ Revoke Cert Lower (408ms)
TxHash ::
NfbJiSjc4PaaTFnBbcuAEMCx5TSbYazJNQKfJLe4V67ab4Mr9798P7ZLFRW1MEf9b6puH7mXG2a5gDr57
73gHnR
    ✓ Revoke Cert Upper (407ms)

  event
TxHash ::
237T3GpRDX33Wx26U7YuEgvv6UfZkCS2ySEcsdVsT2ZyDB2p2zuUfd6XfSm2A1RvtY6dSWN3nyZpM9aZU
12iYgzZ
    ✓ Create Event (620ms)
TxHash ::
5aLjh2vBzN7QGNRd2Yqggf479ZAEp1ZvWYWzRcKZuHp13N8GaMhdxudSV8KCZ1TLbPTUkVnGs2BYvwoHy
tjQWQg1
    ✓ Entry Event (401ms)
TxHash ::
3RTh9euG1jbv5TdnJz6bRLY55YyoQ7iKgMMhupE1J4EuQAuSmRi3BL7cPUfPbLQKygdDXtughzfsGpv4w
iAaPZAQ
    ✓ Pick Winner (3105ms)

  market
TxHash ::
4ZsZ6gYzAouqu5xoqHHdcJ569outdqVuU9heGXhgVPEGaMYF2zX4X4Pxcqapc9R2zVjbnCoDr5XcvZWMV
is741We
    ✓ Sells a ticket of seller (226ms)
TxHash ::
4qEAGQXXVqy18Xw7Vvx2LNpnXKcy8HDRqzXd35D7Mqj1QDk2V96zP7NvscjvXZkXKMxTN563B5Ajf16Sp
AFBqaD9
    ✓ Buys a ticket of seller (410ms)
TxHash ::
PiNS2NgDEsbF9w4t5B6FHAxB2J4moBHjC42WqWWMmvoAjBm33PP1L3qGofCbQve5GuGg8r3yRQaXnynBH
wNLjTu
```

```
    ✓ Sells a ticket of buyer (406ms)
TxHash ::
5ib12CaQziuGLvz9uAbgzBRTPKQhctNMB3q3mSy31CYYMANrCvphkuwpYtYyFgGWMm5DNTvpZGCswKvLtnNCCo2D
    ✓ Cancels a trade of buyer (410ms)


  11 passing (10s)

Done in 11.71s.
Local validator still running. Press Ctrl + C quit.
```

인스트럭션 수행 결과는 `https://explorer.solana.com/tx/{txHash}?cluster=custom`에서 확인할 수 있습니다.

프로그램 배포

`Anchor.toml` 파일의 `provider.cluster`를 `Localnet`에서 `devnet`으로 변경합니다.

```
# BEFORE
[provider]
cluster = "Localnet"
# AFTER
[provider]
cluster = "devnet"
```

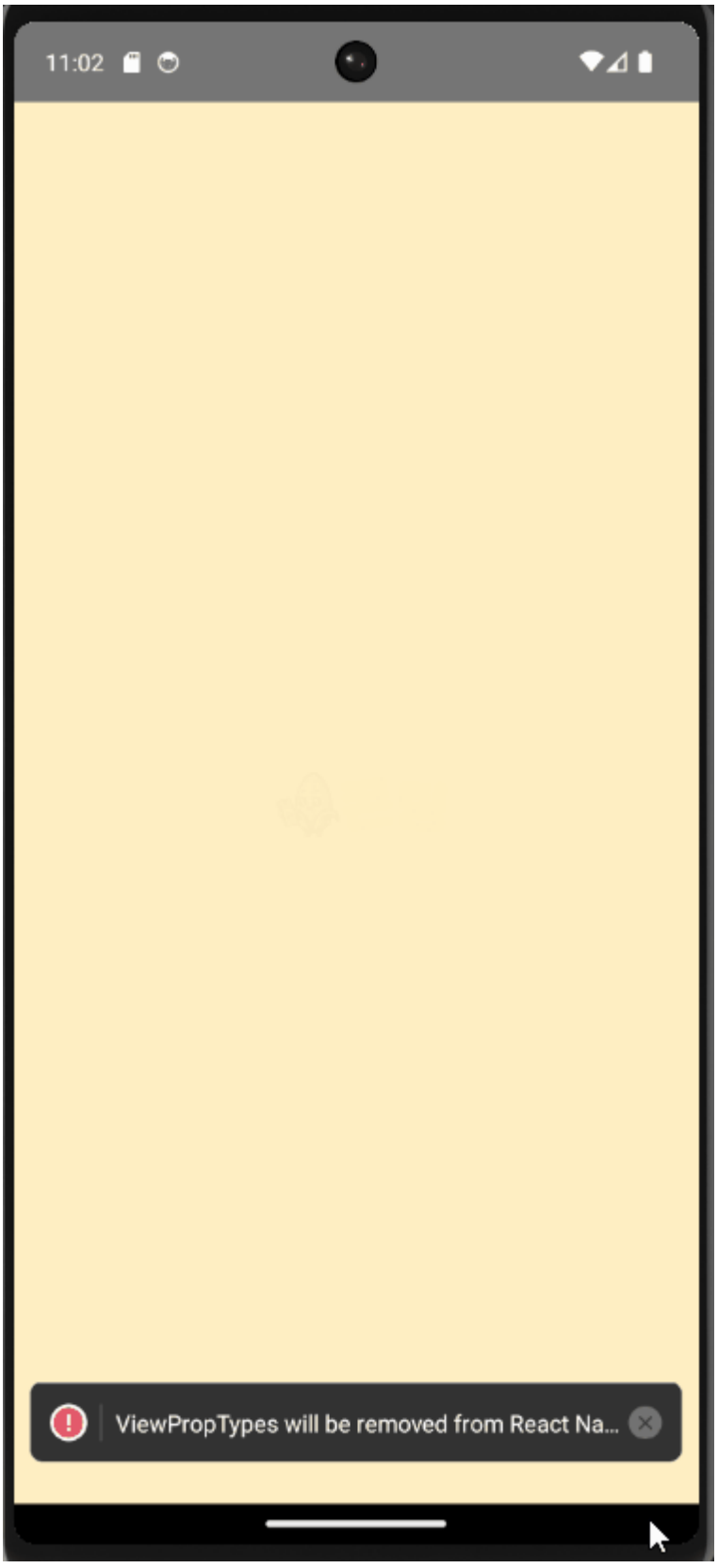
`anchor deploy`를 실행합니다.

솔라나 네트워크에 배포된 프로그램은 `https://explorer.solana.com/address/{programId}?cluster=custom`에서 확인할 수 있습니다.

시연 시나리오

스플래시 뷰

- 해당 화면 FE 코드 보기



스플레쉬 화면에서 서버에 모든 콘서트 정보를 요청합니다.
해당 요청이 완료가 된경우 앱 내 저장소에 저장 후 로그인 페이지로 이동합니다.
요청이 실패한 경우는 앱을 재 구동해달라는 메세지를 보냅니다.

로그인

- 해당 화면 FE 코드 보기
- 해당 화면 BE 코드 보기

```
[작업 내용]
이메일 형식을 검증합니다. 올바른 이메일 형식이 아니라면 올바른 이메일 형식을
알려주는 메세지를 보여줍니다.
비밀번호 패턴 검증을 합니다. 올바른 비밀번호 패턴이 아니라면 로그인 버튼이
활성화 되지 않습니다.
서버에 로그인 요청을 보낸 후 성공한다면 JWT 토큰을 전역으로 상태관리하여
인스턴스 함수에서 사용되게 하여 프론트서버에서 보내게 되는 API는 토큰을
넣거나, 재발행 같은 예외처리를 자동화하여 프론트 팀에서 작업하게 해야하는
불편함을 해소했습니다.
```

메인화면

- 해당 화면 FE 코드 보기

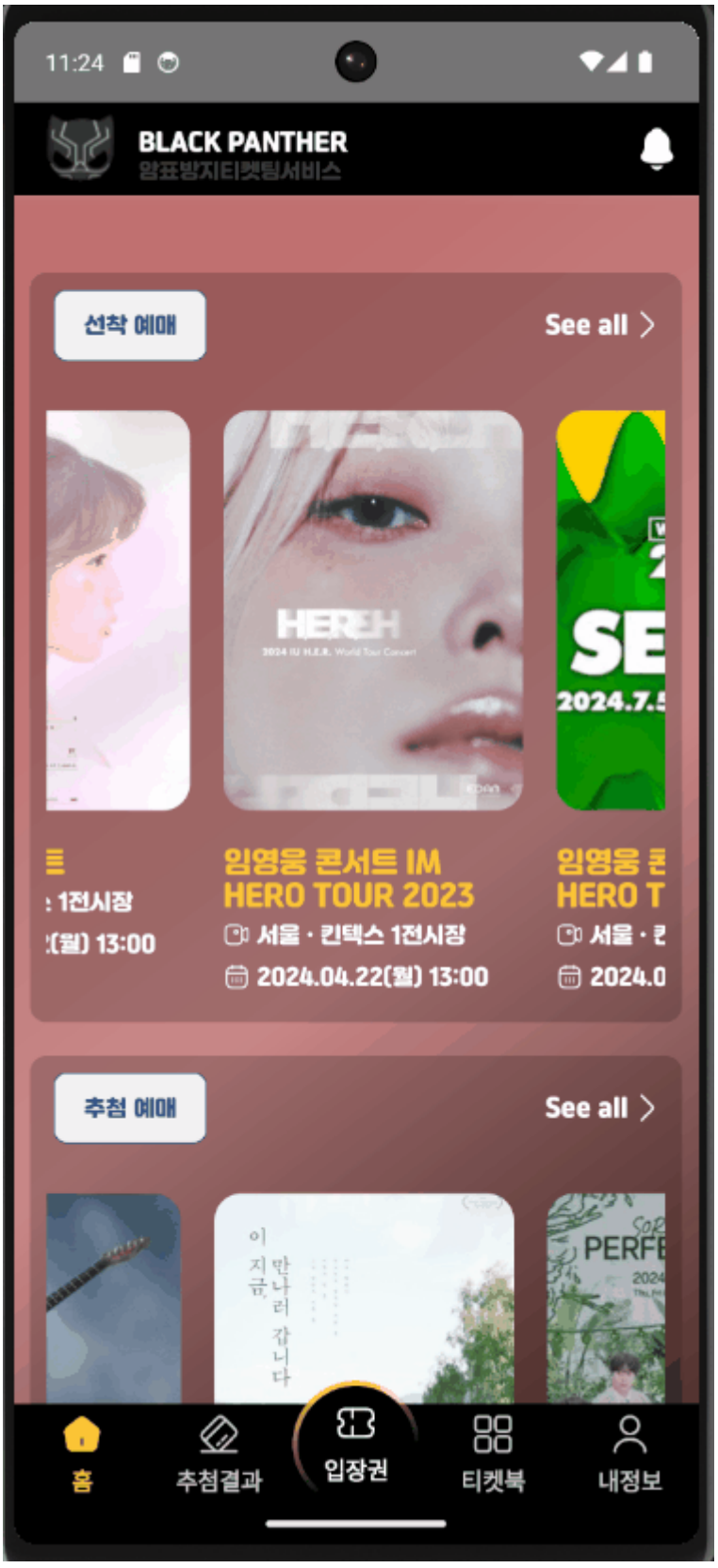
- [해당 화면 BE 코드 보기]


```
[작업 내용]
메인 포스터 전체 이미지 색에 따라 배경이 동적으로 변화합니다.
```

- 선착순 공연

[작업 내용]
예매 방식이 선착순 예매인 공연을 횡스크롤로 확인할 수 있습니다.

- 추첨 공연



[작업 내용]
예매 방식이 추첨 예매인 공연을 횡스크롤로 확인할 수 있습니다.

- 광고 배너 및 응모 이벤트 리스트

[작업 내용]
NFT 응모권을 활용한 응모 굿즈를 확인할 수 있습니다.

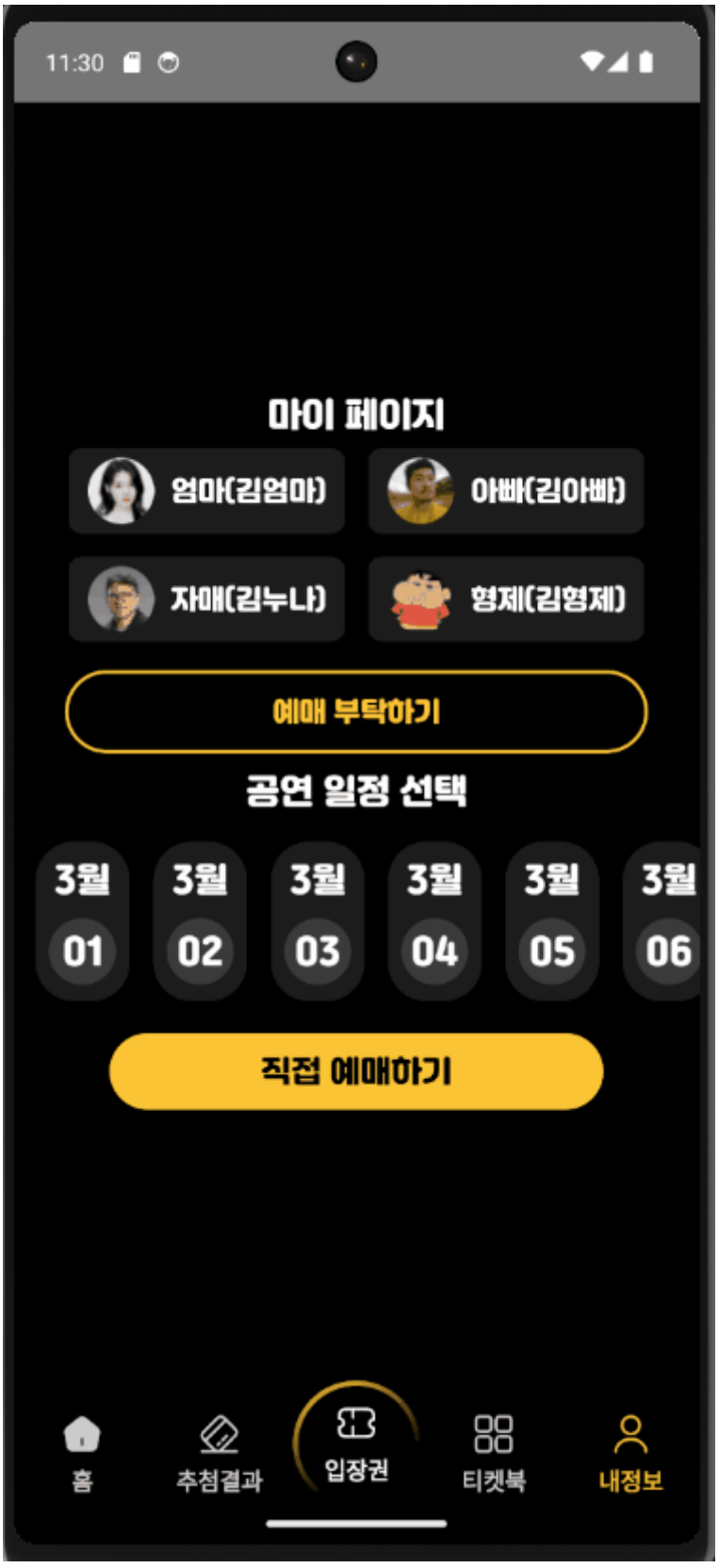
공연 상세 페이지에서 토글이 될 예매부탁/직접예매 버튼

- 해당 화면 FE 코드 보기

[작업내용]
가족에게 티켓 구매 권한을 넘기는 UI 및 직접 구매를 하는 버튼 UI를 구현했습니다.

예매할 경우 대기열 입장

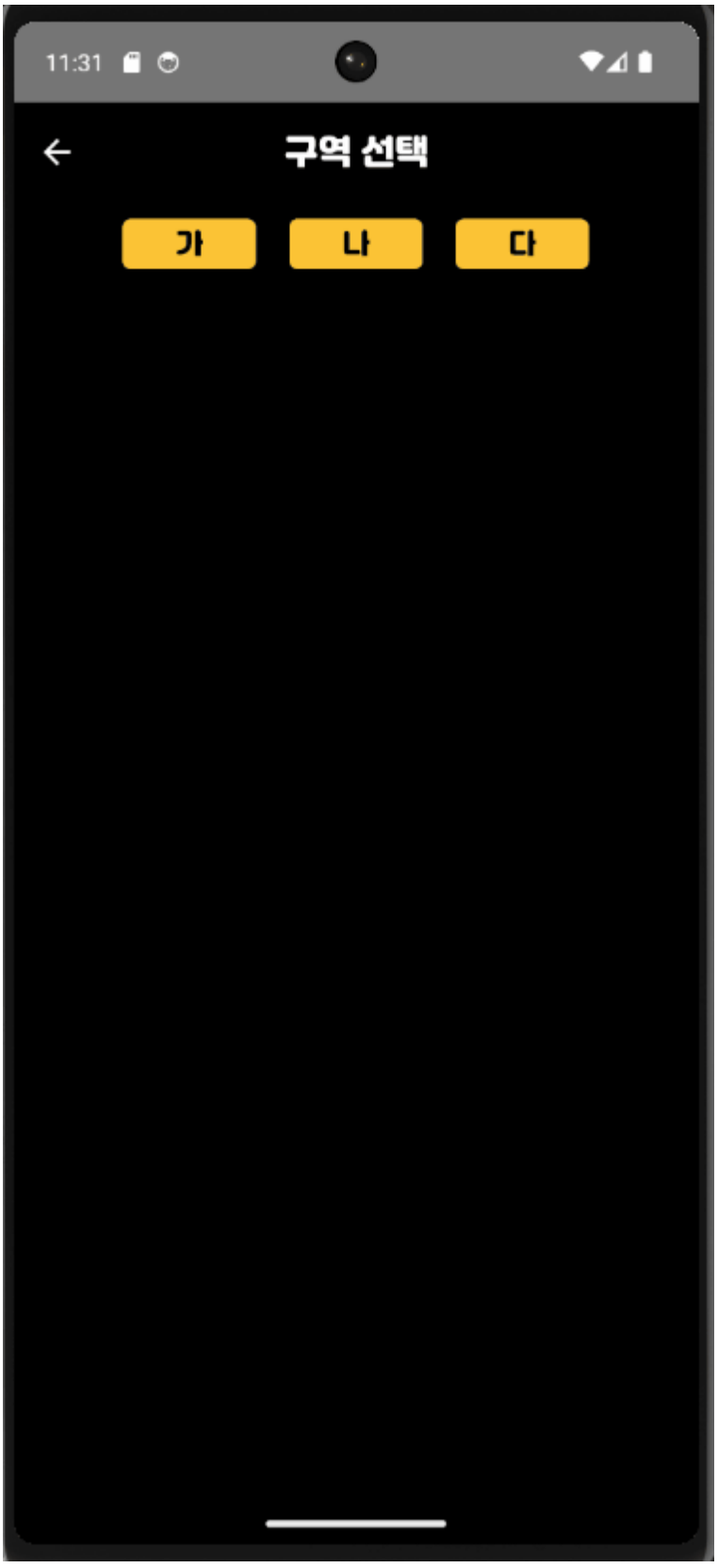
- 해당 화면 FE 코드 보기
- 해당 화면 BE 코드 보기




```
[작업내용]
대용량 트래픽을 처리하기 위한 대기열을 구성했습니다.
```

구역 선택

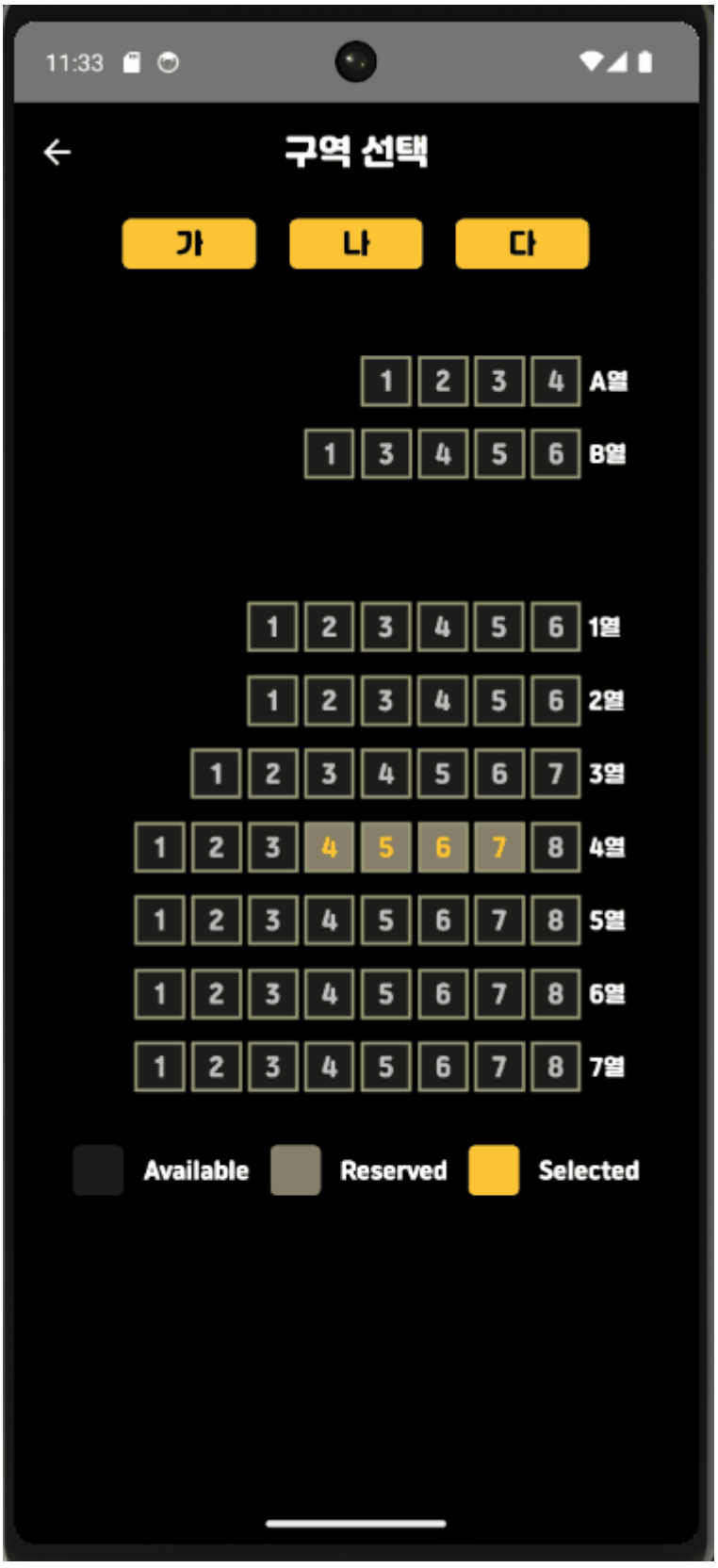
- 해당 화면 FE 코드 보기




```
[작업내용]
전체 구역을 다 보여준다면, 자리 선택하는 UI가 너무 작아지기 때문에
구역을 탭으로 나누어 선택할 수 있는 UI를 구현했습니다.
```

좌석 선택

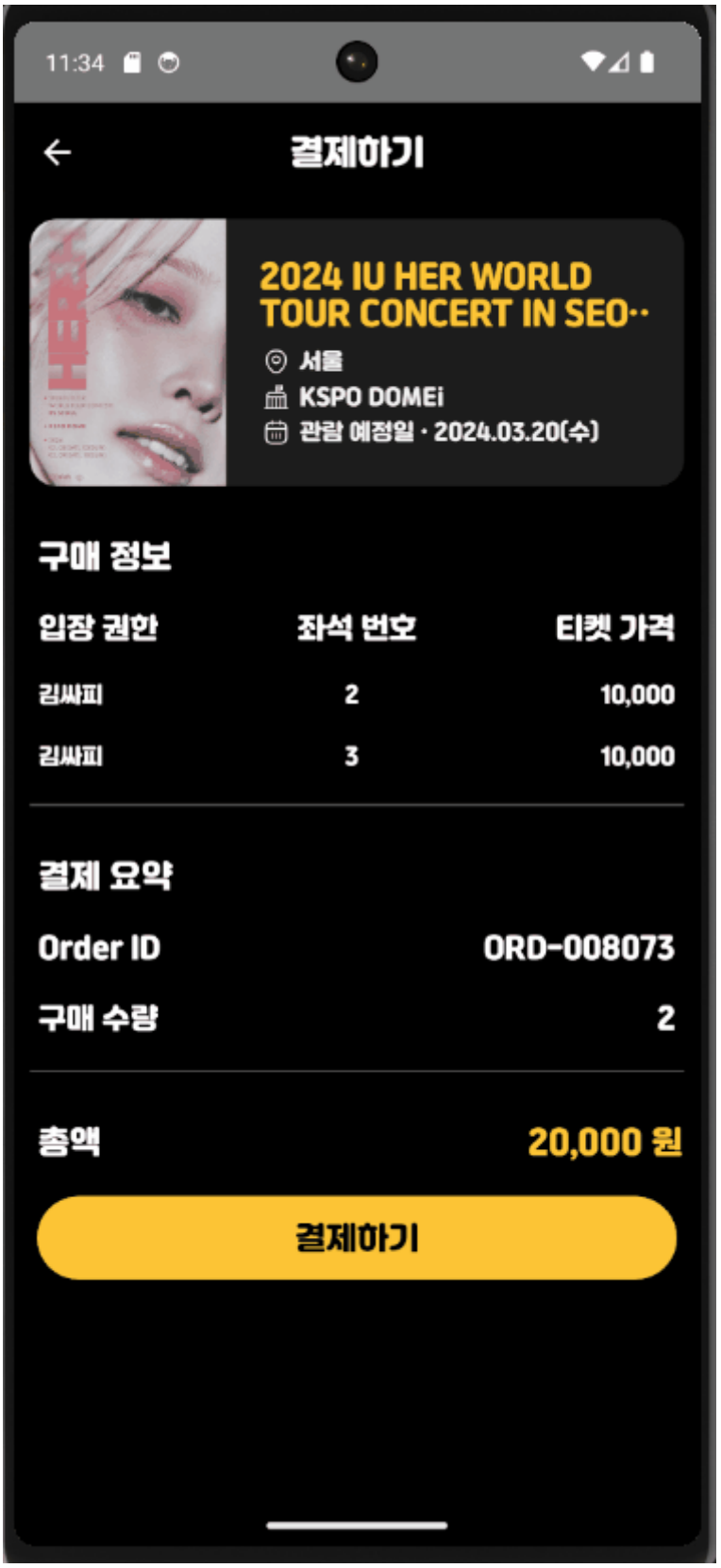
- 해당 화면 FE 코드 보기



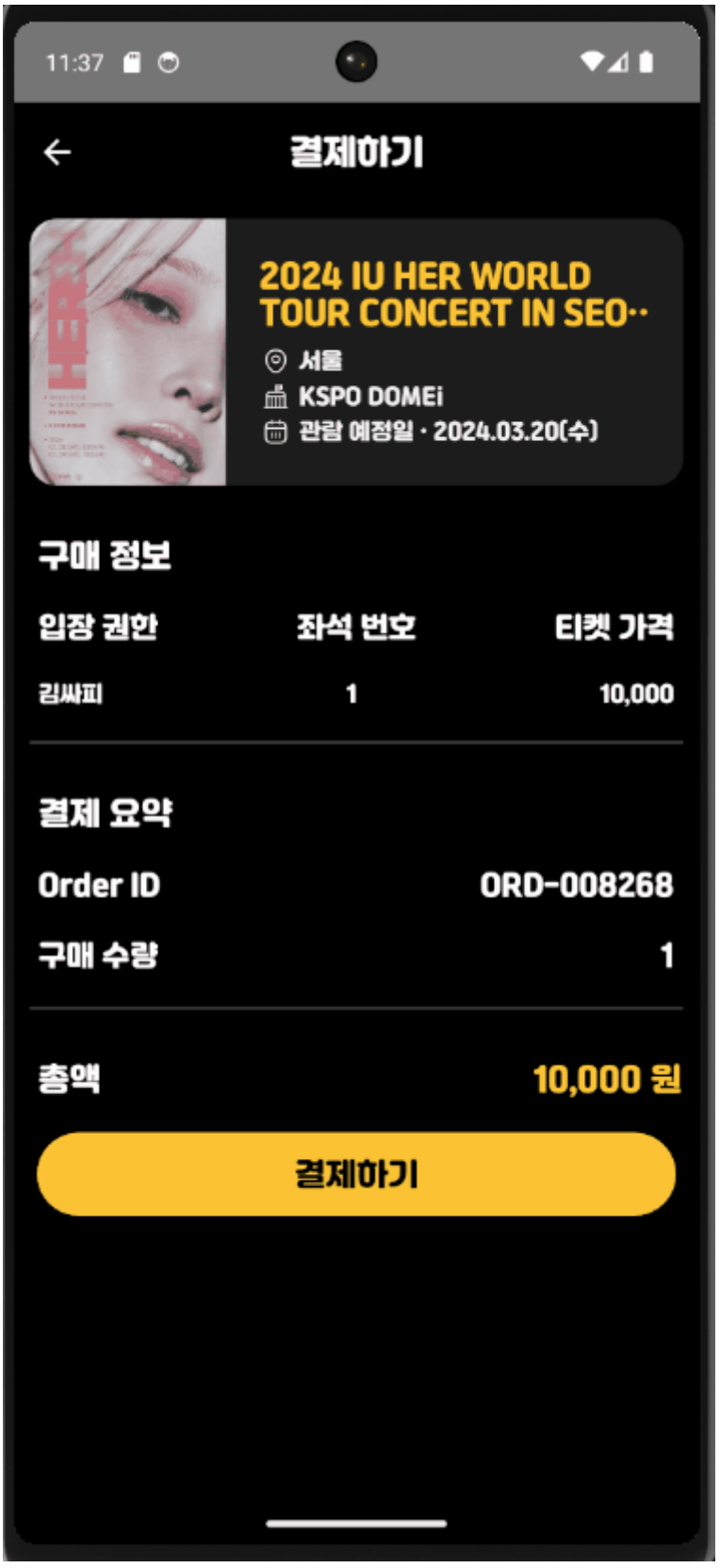
좌석 이미지의 좌석 좌표를 찾은 뒤 해당 좌석을 터치로 접근할 수 있게 하는 UI를 구성했습니다.

결제 정보 및 결제 결과

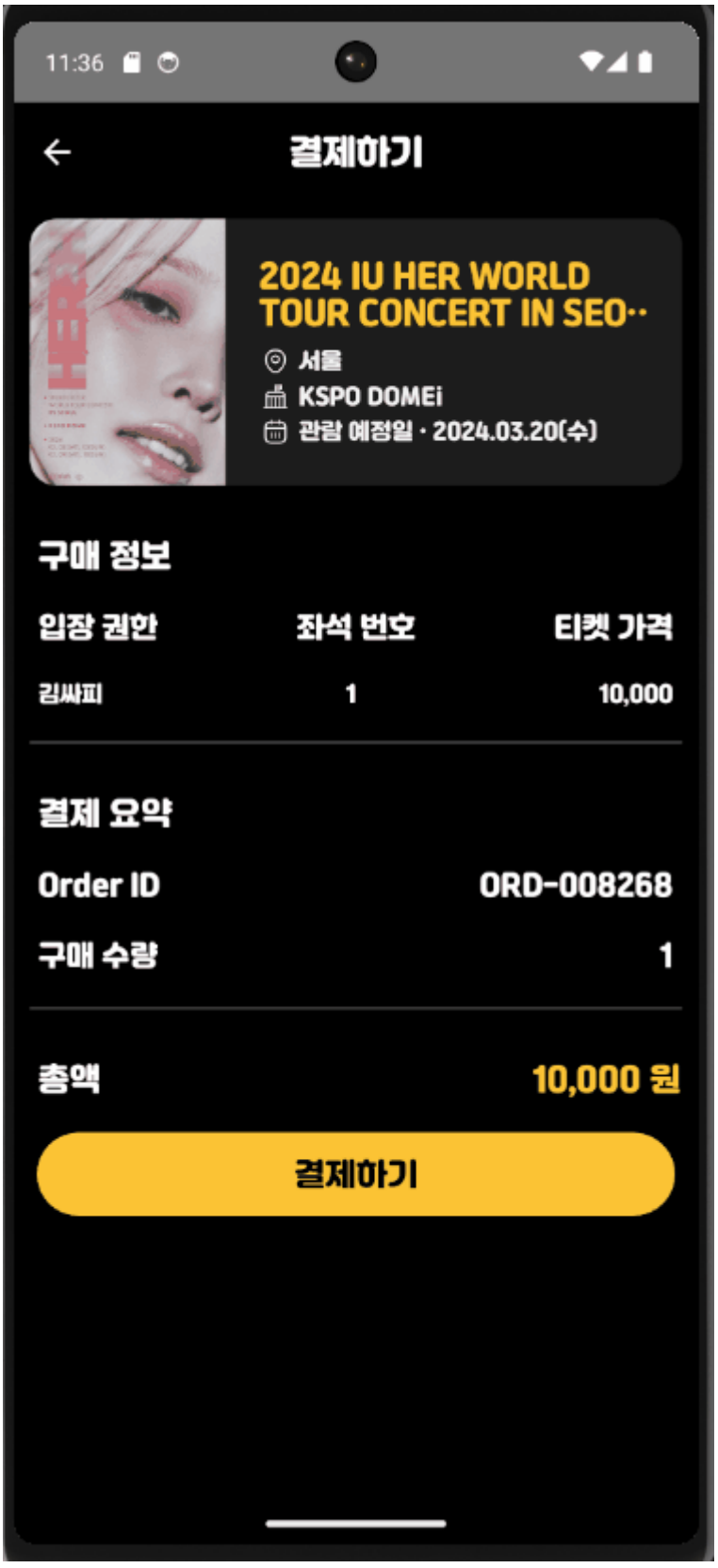
- 해당 화면 FE 코드 보기
- 결제 성공 시



- 응모 성공 시


11:37
결제하기
2024 IU HER WORLD TOUR CONCERT IN SEO··
서울
KSPO DOMEi
관람 예정일 · 2024.03.20(수)
구매 정보
입장 권한
좌석 번호
티켓 가격
김싸피
1
10,000
결제 요약
Order ID
ORD-008268
구매 수량
1
총액
10,000 원
결제하기

- 결제 실패 시




```
[작업내용]
결제,응모 결과에 따른 UI를 구현했습니다.
```

입장권

- 해당 화면 FE 코드 보기

구매 내역

- 해당 화면 FE 코드 보기


```
[작업내용]
콘서트 NFT 티켓 입장권을 UI를 구성했습니다.
```

환불

- 해당 화면 FE 코드 보기
- 성공 시


11:41
구매 내역
2024 IU HER WORLD TOUR CONCERT IN SEO··
서울
KSPO DOME
관람 예정일 · 2024.03.20(수)
구매 정보
입장 권한
좌석 번호
티켓 가격
김싸피
7
99,000
김싸피
8
99,000
김싸피
9
99,000
취소 수수료
예매일 기준
예매 후 7일 이내
없음
예매 후 10일 이내
장당 4,000원 (10% 한도)
관람일 기준
관람일 9일전
10%
관람일 6일전
20%
관람일 2일전
30%

- 실패 시

[작업내용]
환불 결과에 따른 UI를 구현했습니다.

그 외 작업중인 추첨결과 및 티켓북 탭 화면

- 해당 화면 FE 코드 보기
- 해당 화면 FE 코드 보기