Auto Store 포팅 매뉴얼

시스템 환경 및 소프트웨어 정보

백엔드 (Spring Boot)

- **프레임워크**: Spring Boot
- **버전**: 3.3.1
- **JVM 버전:** openjdk-22
- **빌드 도구**: Gradle 8.8
- **Docker 이미지**: Jenkins에서 빌드 후 Docker Hub에 푸시
- **IDE**: ****IntelliJ IDEA
- **IDE 버전**: ****2024.1.4

머신러닝

- **Framework**
  - **FastAPI**
  - **Python**: 3.12.5
- **Model**
  - **TensorFlow** or **PyTorch**
  - **LSTM model**
- **IDE:** Visual Studio
  - **IDE Version**: 1.90.2

영상 분석 AI

- **학습 데이터셋**
  - AI Hub: 실내(편의점, 매장) 사람 이상행동 데이터
  - https://aihub.or.kr/aihubdata/data/view.do?currMenu=&topMenu=&aihubDataSe=realm&dataSetSn=71550
  - 절도, 파손, 흡연, 방화, 실신 5가지 카테고리에 대하여 랜덤으로 80개씩 섞어 raw/batch_001... 순으로 영상 데이터 및 라벨링 데이터셋을 배치
  - 검증 데이터셋은 val/theft... 식으로 영상 데이터를 배치함
- **학습 환경**
  - **Ubuntu**
  - **Nvidia Tesla V-100**
  - **Nvidia Driver Version**: 535.183.01
  - **Cuda Toolkit**: 12.2
  - **Python**: 3.10.14
  - **conda**: 23.3.1
  - 그 외 패키지는 environments.txt로 conda 환경 설정

웹 프론트엔드 (Next.js)

- **프레임워크**: Next.js
- **버전**: ^14.2.13
- **Node.js 버전**: 20.18.0

- **빌드 도구**: npm 10.8.2
- eslint-config-next: ^14.2.8
- **Docker 이미지**: Jenkins에서 빌드 후 Docker Hub에 푸시
- **Electron**:
- **IDE**: ****Visual Studio Code
- **IDE 버전**: 1.90.2

Kiosk 프론트엔드

- **프레임워크**: Next.js
- **버전**: 14.2.13
- **Node.js 버전**: v20.15.1
- **빌드 도구**: electron-builder
- **Electron 버전**: ^25.0.0
- **IDE**: ****Visual Studio Code
- **IDE 버전**: Visual Studio Code 1.90.2

데이터베이스

- **Database:** MySQL: 9.0.1
- **Caching:** redis: 7.4.0

Infra

- **Server**
  - **AWS EC2 Ubuntu** 20.04
  - **Nginx**: 1.18.0
- **Storage**
  - **Amazon S3**
- **Containerization**
  - **Docker Engine - Community**: 27.3.1
- **Jenkins**
  - Username: a302
  - Password: ssafyA302!

배포 과정

- 변경된 프로젝트를 master branch에 push 합니다.
- code push시 GitLab이 Jenkins에 웹훅을 통해 알림을 보냅니다.
- Jenkins는 GitLab에서 소스 코드를 클론한 후 CI/CD 파이프라인을 실행합니다.
  - 백엔드(Spring Boot)와 프론트엔드(Next.js)를 빌드합니다.
  - 각각의 애플리케이션을 Docker 이미지로 빌드한 후, Docker Hub에 이미지를 푸시합니다.
- **Docker Hub**: Jenkins가 푸시한 이미지를 저장하고, AWS의 인프라로 이미지를 배포합니다.

Blue-Green Deployment

- **Nginx**는 사용자 요청을 프록시하여 Green 컨테이너 또는 Blue 컨테이너로 라우팅합니다.
- Blue-Green Deployment 전략을 사용하여 애플리케이션의 무중단 배포를 구현합니다.

Nginx Config

```
# 백엔드 부분
upstream backend {
      server localhost:8081;

      # Example health check configuration (requires Nginx Plus or a third-party module)
      # Uncomment if you have the required module
      # health_check interval=3s fails=3 passes=2;
  }
upstream ml {
      server localhost:8001;

  }
  # 프론트엔드 부분
upstream frontend {
      server localhost:3002;
    }
  server {
      listen 80;
      server_name j11a302.p.ssafy.io;
      return 301 https://$host$request_uri;
  }

  server {
      listen 443 ssl;
      server_name j11a302.p.ssafy.io;

      ssl_certificate /etc/letsencrypt/live/j11a302.p.ssafy.io/fullchain.pem;
      ssl_certificate_key /etc/letsencrypt/live/j11a302.p.ssafy.io/privkey.pem;

      ssl_protocols TLSv1.2 TLSv1.3;
      ssl_ciphers HIGH:!aNULL:!MD5;

      # API 요청을 백엔드 서버로 라우팅
        location /api/ {
                        #Cors 설정
                        # add_header 'Access-Control-Allow-Origin' '<https://i11a508.p.ssafy.io>';
            proxy_pass <http://backend>;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
            proxy_connect_timeout 300s;
            proxy_send_timeout 300s;
            proxy_read_timeout 300s;
            send_timeout 300s;
            proxy_next_upstream error timeout http_500 http_502 http_503 http_504;
        }

        location /ml/ {
            proxy_pass <http://ml>;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X_Forwarded_For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
        }
        # 나머지 모든 요청을 프론트엔드 서버로 라우팅
        location / {
            proxy_pass <http://frontend>;
            proxy_set_header Host $host;
```

```
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
                }

  }
```

ML

```
pipeline {
    agent any

    environment {
        DOCKER_TAG = "latest"
        DOCKER_HUB_REPO = 'junhyeok302/ml'
        GIT_REPO = '<https://lab.ssafy.com/s11-ai-image-sub1/S11P21A302.git>'
        GIT_BRANCH = 'dev/ML/master'
        GIT_CREDENTIALS_ID = 'Gitlab'
        DOCKER_CREDENTIALS_ID = 'DockerHub'
    }

    stages {
        stage('Clone') {
            steps {
                script {
                git branch: "${GIT_BRANCH}", url: "${GIT_REPO}", credentialsId: "${GIT_CREDENTIALS_ID}"
                }
            }
        }

        stage('Build Docker Image') {
            steps {
                script {
                    dir('ml') {
                        def imageTag = "${DOCKER_HUB_REPO}:${DOCKER_TAG}"
                        sh "docker build -t ${imageTag} -f Dockerfile ."
                        env.DOCKER_IMAGE = imageTag
                    }
                }
            }
        }

        stage('Push Docker Image') {
            steps {
                script {
                    def imageTag = "${DOCKER_IMAGE}"
                    withCredentials([usernamePassword(credentialsId: DOCKER_CREDENTIALS_ID,
passwordVariable: 'DOCKER_PASSWORD', usernameVariable: 'DOCKER_USERNAME')]) {
                        sh """
                        echo "$DOCKER_PASSWORD" | docker login -u $DOCKER_USERNAME --password-stdin
                        docker push ${imageTag}
                        docker tag ${imageTag} ${DOCKER_HUB_REPO}:latest
                        docker push ${DOCKER_HUB_REPO}:latest
                        """
                    }
                }
            }
        }
```

```
        stage('Deploy') {
            steps {
                script {
                    def isBlue = sh(script: "docker ps | grep blue-ml", returnStatus: true) == 0

                    def newPort = isBlue ? 8000 : 8001
                    def oldPort = isBlue ? 8001 : 8000
                    def newContainer = isBlue ? 'green-ml' : 'blue-ml'
                    def oldContainer = isBlue ? 'blue-ml' : 'green-ml'

                    sh """
                        docker pull ${DOCKER_HUB_REPO}:latest
                        docker stop ${newContainer} || true
                        docker rm ${newContainer} || true
                        docker run -d --name ${newContainer} -p ${newPort}:8001 ${DOCKER_IMAGE}
                    """
                    lock('nginx-config') {  // NGINX 설정 파일에만 락을 걸어 동시 접근 방지
                        sh """
                        sudo sed -i 's#server localhost:800[01]#server localhost:${newPort}#'
/etc/nginx/sites-available/default
                        sudo nginx -t && sudo systemctl reload nginx
                        """
                    }

                    sh """
                        docker stop ${oldContainer} || true
                        docker rm ${oldContainer} || true
                    """
                }
            }
        }
    }

    post {
        success {
            script {
                def Author_ID = sh(script: "git show -s --pretty=%an", returnStdout: true).trim()
                def Author_Name = sh(script: "git show -s --pretty=%ae", returnStdout: true).trim()
                // Add your notification logic here if needed
            }
        }
        failure {
            script {
                def Author_ID = sh(script: "git show -s --pretty=%an", returnStdout: true).trim()
                def Author_Name = sh(script: "git show -s --pretty=%ae", returnStdout: true).trim()
                // Add your notification logic here if needed
            }
        }
    }
}
```

Electron 배포 과정

프로젝트 환경 설정

1. **Node.js 및 Electron 설치**

```
npm install
```

2. **Next.js와 Electron 환경 동시 실행**

```bash
npm run dev
```

3. **프로덕션 빌드 및 패키징**
   - Electron과 Next.js를 각각 빌드 및 패키징하여 실행 파일(.exe) 생성

```bash
npm run build
npm run electron
npm run package
```

Electron 빌드 구성

`package.json` 에서 Electron 및 빌드 관련 스크립트는 다음과 같이 구성됩니다:

```json
{
  "name": "kiosk",
  "version": "1.0.0",
  "main": "electron/main.js",
  "scripts": {
    "dev": "concurrently \\"next dev\\" \\"npm run electron\\"",
    "build": "next build && electron-builder build",
    "start": "next start",
    "electron": "electron .",
    "package": "electron-builder"
  },
  "build": {
    "appId": "com.kiosk.app",
    "productName": "AutoStoreKiosk",
    "files": [
      "build/**/*",
      "electron/**/*"
    ],
    "directories": {
      "output": "dist"
    },
    "win": {
      "target": [
        {
          "target": "nsis",
          "arch": [
            "x64",
            "ia32"
          ]
        }
      ],
      "icon": "public/icon.ico"
```

```
    }
  }
}
```

Electron 배포

1. **Electron Builder를 사용한 빌드**

```
npm run package
```

2. **생성된 실행 파일 (`.exe`) 배포**
   - `dist` 폴더에 생성된 `.exe` 파일을 필요에 따라 PC에 배포 및 실행

PC/SC 카드 리더기 통합

PC/SC 라이브러리 설치

Electron에서 **PC/SC**를 이용해 NFC 리더기와 통신하기 위해 `pcsclite` 및 `nfc-pcsc` 패키지를 사용합니다.

1. **pcsclite** 및 **nfc-pcsc** 설치:

```
npm install pcsclite nfc-pcsc
```

PC/SC 카드 리더기 코드

`electron/main.js` 파일에 NFC 리더기와의 통신을 설정하는 코드를 작성합니다:

```
javascript
코드 복사
const pcsclite = require("pcsclite");

function connectNFCReader() {
  const pcsc = pcsclite();

  pcsc.on("reader", (reader) => {
    console.log(`Reader detected: ${reader.name}`);

    reader.on("status", (status) => {
      const changes = reader.state ^ status.state;

      if (
        changes & reader.SCARD_STATE_PRESENT &&
        status.state & reader.SCARD_STATE_PRESENT
      ) {
        reader.connect(
          { share_mode: reader.SCARD_SHARE_SHARED },
          (err, protocol) => {
            if (err) {
              console.error("Error connecting to card:", err);
```

```
              return;
            }

            console.log("Card connected, protocol:", protocol);
            readSpecificBlock(reader, protocol, 12);
          }
        );
      } else if (
        changes & reader.SCARD_STATE_EMPTY &&
        status.state & reader.SCARD_STATE_EMPTY
      ) {
        console.log("Card removed");
        reader.disconnect(reader.SCARD_LEAVE_CARD, (err) => {
          if (err) {
            console.error("Error disconnecting:", err);
          } else {
            console.log("Card disconnected");
          }
        });
      }
    });

    reader.on("end", () => {
      console.log(`Reader removed: ${reader.name}`);
      reader.close();
    });

    reader.on("error", (err) => {
      console.error("Reader error:", err);
    });
  });

  pcsc.on("error", (err) => {
    console.error("PCSC error:", err);
  });
}

function readSpecificBlock(reader, protocol, blockNumber) {
  const command = Buffer.from([0xff, 0xb0, 0x00, blockNumber, 0x10]); // 블록 읽기 명령어

  reader.transmit(command, 40, protocol, (err, data) => {
    if (err) {
      console.error(`Error reading block ${blockNumber}:`, err);
      return;
    }

    let fullData = data.toString("hex"); // hex 데이터를 문자열로 변환
    console.log(`Block ${blockNumber} data:`, fullData);

    // 데이터를 정제하여 프론트엔드로 전송
    const barcode = Buffer.from(fullData, "hex").toString("ascii").trim();
    const numericBarcode = barcode.replace(/\D/g, ""); // 숫자만 추출
    mainWindow.webContents.send("nfc-data", numericBarcode); // 정제된 바코드를 프론트엔드로 전송
  });
}

app.whenReady().then(() => {
  createWindow();
  connectNFCReader(); // NFC 리더기 연결
});
```

Electron에서 NFC 데이터 전송

`preload.js`를 사용하여 Electron의 백엔드에서 읽은 NFC 데이터를 프론트엔드로 전달합니다:

```
javascript
코드 복사
const { ipcRenderer, contextBridge } = require("electron");

contextBridge.exposeInMainWorld("electronAPI", {
  onRFIDDetected: (callback) => {
    const handler = (event, data) => {
      console.log("NFC Data received in preload:", data);
      callback(data); // 데이터를 프론트엔드로 전달
    };

    ipcRenderer.on("nfc-data", handler);

    return () => {
      ipcRenderer.removeListener("nfc-data", handler);
    };
  },
});
```

NFC 데이터 처리

프론트엔드에서 `window.electronAPI.onRFIDDetected`를 통해 NFC 데이터를 수신하고 처리할 수 있습니다:

```
javascript
코드 복사
window.electronAPI.onRFIDDetected((data) => {
  console.log("Received NFC data:", data);
  // 받은 데이터를 처리하여 화면에 표시하거나, 다른 작업을 수행
});
```

AI Train / Serve

학습 실행

1. **Conda 설치**

```
# Miniconda 설치 스크립트 다운로드
wget <https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh>

# 설치 스크립트 실행
bash Miniconda3-latest-Linux-x86_64.sh

# 설치가 완료된 후 터미널 재실행 또는 아래 명령어 실행
source ~/.bashrc
```

2. **Conda 환경 설정**

- `conda create --name <new_env_name> --file environment.txt`
  - conda 환경을 설정하고, 필요한 패키지를 설치
  - `<new_env_name>`은 설정하고 싶은 환경의 이름으로 대체

1. 학습 및 검증 실행

- `result/scripts/train_gpu.py` 를 실행
- 실행 명령어는 `train_gpu.sh`

```
export CUDA_VISIBLE_DEVICES=3

nohup python result/scripts/train_gpu.py > output.log 2>&1 &
```

모델 서빙

1. `archive_ensemble_model` 로 result/models 의 두 모델에 대한 앙상블 .mar 파일을 생성

```
torch-model-archiver \\
--model-name ensemble_model \\
--version 1.0 \\
--serialized-file result/models/x3d_best_model.pth \\
--handler TorchServe/settings/ensemble_handler.py \\
--extra-files "result/models/slowfast_best_model.pth,TorchServe/settings/index_to_name.json" \\
--export-path TorchServe/model_store
```

2. `initialize_torchserve` 로 torchserve 를 실행하여 앙상블 모델 서빙

```
export CUDA_VISIBLE_DEVICES=3

nohup torchserve \\
    --start \\
    --ncs \\
    --model-store ../TorchServe/model_store \\
    --models ensemble=../TorchServe/model_store/ensemble.mar \\
    --ts-config ../TorchServe/config.properties \\
    --disable-token-auth \\
    > ../results/torchserve.log 2>&1 &
```