iicdii / case-ing

Code Issues Pull requests Actions Security Insights

대법원 나의 사건 조회 자동화 API

15 stars 4 forks 1 watching 1 Branch 0 Tags Activity

Public repository

main 1 Branch 0 Tags Go to file t Go to file Add file Code

iicdii Add remark column 2 years ago

.github/workflows	Rename e2e spec path	2 years ago
.idea	Create serverless api, cypress tests	3 years ago
api	Add remark column	2 years ago
cypress	Add remark column	2 years ago
.gitignore	Refactor everything up-to-date	2 years ago
.nvmrc	Add .nvmrc to set node v16 as default	2 years ago
README.md	Add remark column	2 years ago
clear-sheet.js	Add clear sheet function	2 years ago
create-fixtures.js	Fix wrong spreadsheet id	2 years ago
cypress-partial.js	Rename e2e spec path	2 years ago
cypress.config.js	Refactor everything up-to-date	2 years ago
package.json	Add clear sheet function	2 years ago
yarn.lock	Remove dotenv	2 years ago

README

case-ing

대법원 나의 사건 조회 자동화 API & 크롤러

License MIT | Update Sheet Cron failing | Update Sheet Manually no status

case-ing는 case + ~ing의 합성어로 여러개의 사건 진행현황을 쉽게 조회하도록 도와주는 자동화 도구입니다.

사용된 기술

- Cypress - 크롤링
- Scikit-learn - Captcha 학습, 분석
- Serverless Framework - Captcha 분석 API, 구글 스프레드시트 조회/수정 API 생성

요구사항

- Node.js >= 14
- yarn v1.x
- Docker

폴더 구조

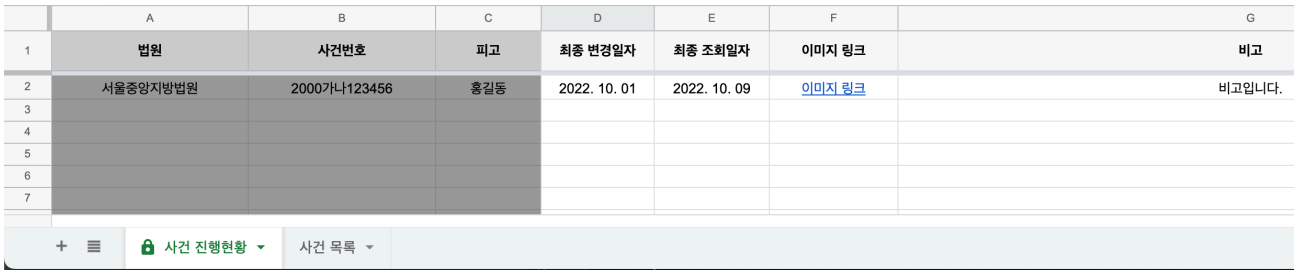
```
case-ing
└── api
|   └──certification
|   |   └ service-account.json  # 구글 스프레드시트 인증을 위한 파일
|   └──model
|   |   └ model.pickle          # 미리 학습된 scikit-learn 모델
|   ├ app.py                    # 캡챠 handler
|   ├ cases.py                  # 구글 스프레드시트 Read, Update handler
|   ├ docker-requirements.txt   # 캡챠 handler 도커 배포에 필요한 패키지 정의
|   ├ Dockerfile                # 캡챠 도커파일
|   ├ requirements.txt          # 구글 스프레드시트 handler 배포에 필요한 패키지 정의
|   ├ s3.py                     # s3 업로드 handler
|   └ serverless.yml            # 서버리스 설정파일
└── cypress
|   └──fixtures                 # 사건들이 create-fixtures.js 실행 후 여기에 생성됨
|   └──e2e
|   |   └ spec.cy.js            # 자동화 메인 스크립트
|   └──js
|   |   ├ caseTypes.js          # 대법원 홈페이지에서 추출한 사건 구분 데이터
|   |   └ courts.js             # 대법원 홈페이지에서 추출한 법원 데이터
|   └──plugins
|       └ index.js
├ create-fixtures.js            # 구글 스프레드시트를 조회해서 필요한 fixtures를 생성함
├ cypress-partial.js            # Cypress 병렬 실행용 스크립트
└ cypress.env.json              # 환경변수 저장소
```

사전 준비

1. 구글 스프레드시트 생성

사건 진행현황 시트 생성

자동화 입력이 끝난 후 결과가 기록될 시트입니다.



	A	B	C	D	E	F	G
1	법원	사건번호	피고	최종 변경일자	최종 조회일자	이미지 링크	비고
2	서울중앙지방법원	2000가나123456	홍길동	2022. 10. 01	2022. 10. 09	이미지 링크	비고입니다.
3							
4							
5							
6							
7							

사건 진행현황 사건 목록

사건 목록 시트 생성

조회하고 싶은 사건을 입력할 시트입니다.

스프레드시트 id 입력

`api/cases.py` 의 `SPREADSHEET_ID` 에 생성한 스프레드시트의 id를 입력합니다. 스프레드시트의 id는 스프레드시트가 열려있는 탭의 URL의 중간에 있는 유니크 아이디를 가져옵니다.

`https://docs.google.com/spreadsheets/d/여기에 스프레드시트 id가 있습니다/edit#gid=0`

2. 서비스 계정 생성 및 스프레드시트 편집자 권한 부여

구글 개발자 콘솔에서 프로젝트를 만들고 Google Sheets API를 활성화합니다.

그 후 서비스 계정의 API 키를 생성 및 다운로드 받은 후 `api/certification/service-account.json` 에 서비스 키를 저장합니다.

위 작업이 끝나면 스프레드시트 우측 상단의 공유 버튼을 클릭합니다.

공유 모달에서 서비스 계정의 이메일을 입력하고 편집자 권한을 부여합니다.

3. Captcha 우회용 Scikit-learn 학습

캡챠 예측에 필요한 모델을 학습시킨 후 `api/model/model.pickle` 에 모델을 저장합니다.

이 레포에서는 학습 모델을 따로 제공하지 않습니다.

4. S3 버킷 생성

사건 진행 내용 스크린샷을 저장할 S3 버킷을 생성합니다.

5. IAM 역할 생성

정책 예)

- `AWSLambdaBasicExecutionRole`
- `AmazonS3FullAccess`
- `AWSLambda_FullAccess`

IAM 역할을 생성한 후 `api/serverless.yml` 에 ARN을 입력합니다. (키: `provider.iam.role`)

서버리스 백엔드 준비하기

서버리스 백엔드는 캡챠 예측, 구글 스프레드 시트 조회 & 업데이트를 위해 필요하며, 서버리스 프레임워크를 통해 배포됩니다.

시작하기

서버리스는 `api/serverless.yml` 에 기재된 `frameworkVersion` 의 메이저 버전과 일치해야 합니다.

~~현재는 설정이 2.xx 버전 기준이므로~~ ~~`@latest 2`~~ ~~태그를 붙여서 글로벌로 패키지 설치를 진행합니다.~~

현재 최신 버전인 3 버전으로 글로벌 설치를 진행합니다.

```
npm install -g serverless@latest
```

서버리스 배포를 위해 aws-cli를 통한 프로필 설정을 진행해주세요. 이미 설정이 되있다면 넘어가셔도 됩니다.

```
$ aws configure
AWS Access Key ID: foo
AWS Secret Access Key: bar
Default region name [us-west-2]: ap-northeast-2
Default output format [None]:
```

테스트 (local)

로컬 환경에서 테스트 하는 방법을 설명합니다. 노드, 파이썬 패키지 설치와 도커 빌드 과정이 필요합니다.

노드 패키지 설치

```
cd api
yarn install
```

파이썬 패키지 설치

파이썬 패키지의 전역 설치를 피하기 위해 가상환경을 구성합니다.

`venv` , `virtualenv` 등의 선택지가 있는데 여기선 `virtualenv` 를 사용하겠습니다. (`virtualenv` 설치 가이드)

```
cd api
virtualenv venv
source venv/bin/activate
```

- `activate` 는 가상 환경에 설치한 파이썬과 패키지를 시스템에 설치된 것보다 먼저 사용하도록 환경 변수를 구성하는 역할을 하므로 반드시 실행해야 합니다.

```
pip3 install -r requirements.txt
deactivate
```

캡챠 예측 API의 경우 `serverless-offline` 으로 로컬에서 테스트가 불가능하므로 별도 빌드 작업이 필요합니다.

```
docker buildx build --platform linux/amd64 -t predict-captcha .
```

- `--platform linux/amd64` 플래그를 붙이면 맥북 M1 환경에서도 빌드가 가능해집니다.

테스트 실행

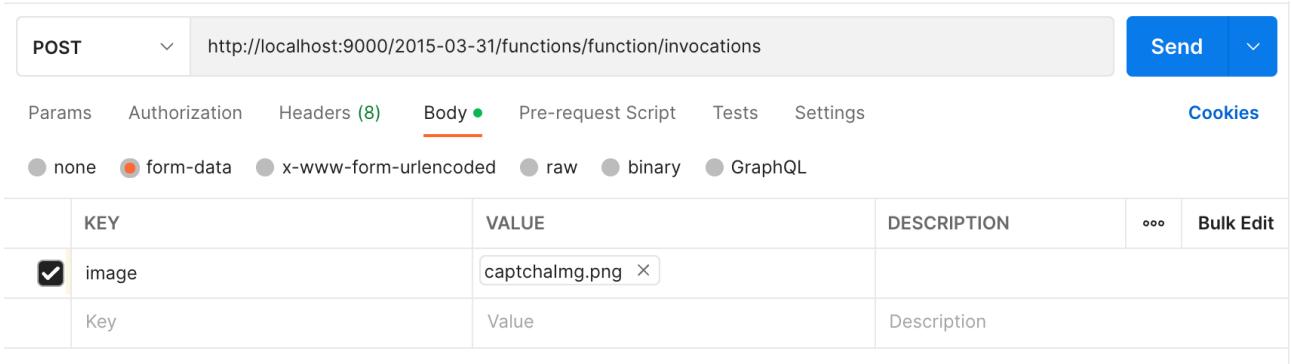
```
cd api
sls offline
```

http://localhost:3000/cases 에 `GET` 요청을 보내서 스프레드시트의 데이터를 읽어올 수 있는지 테스트 해봅니다.

```
cd api
docker run -p 9000:8080 predict-captcha:latest
```

http://localhost:9000/2015-03-31/functions/function/invocations 에 `POST` 요청을 보내서 응답이 오는지 테스트 해봅니다.

아래 이미지는 POST 요청 예시입니다.



캡챠 예측 API의 경우 로컬에서 에러가 발생할 수 있습니다. 이런 경우 아래 서술할 beta 환경에 배포 후 테스트를 진행할 수도 있습니다.

테스트 (beta)

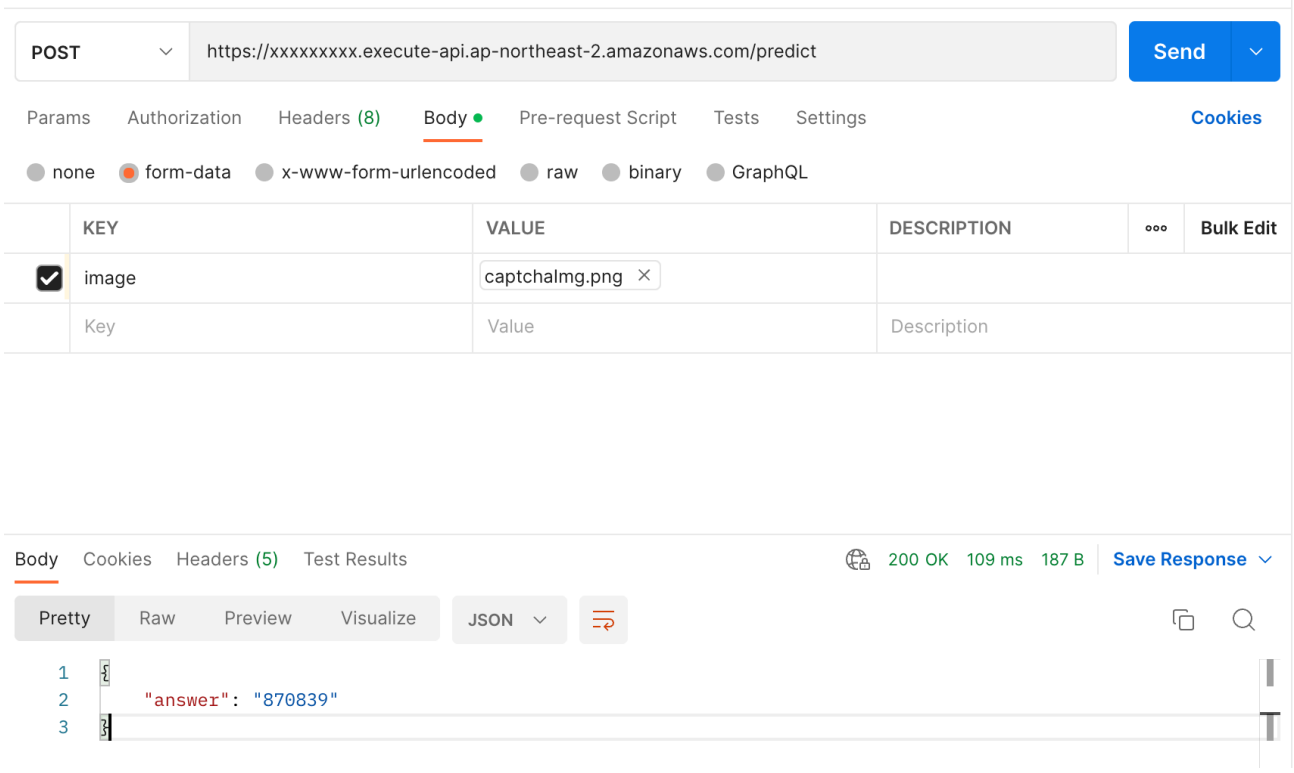
베타에서 테스트 하는 방법을 설명합니다. 노드 패키지 설치 후 서버리스에 `beta` 환경으로 배포합니다. 배포를 진행하기 전에 도커 실행이 필요합니다.

```
cd api
yarn install
serverless deploy --stage beta
```

서버리스 배포가 완료되면 배포된 URL이 콘솔에 뜰텐데 해당 URL로 요청을 보내봅니다. 아래는 예시입니다.

- (GET) https://xxxxxxxxx.execute-api.ap-northeast-2.amazonaws.com/cases
- (POST) https://xxxxxxxxx.execute-api.ap-northeast-2.amazonaws.com/predict

아래는 캡챠 API에 대한 요청 및 응답 예시입니다.



배포

캡챠 예측의 경우 다른 API와 다르게 약 1GB가 넘는 의존성이 필요하므로 도커 이미지로 빌드됩니다.

도커 이미지 빌드를 위해 도커를 실행합니다. 도커가 없다면 공식 홈페이지에서 설치해주세요.

```
cd api
yarn install
sls deploy
```

Cypress 자동화 준비하기

1. 루트 경로에 cypress.env.json 생성

```
{
  "CYPRESS_LAMBDA_API_URL": "<Lambda 서버리스 API URL>",
  "CYPRESS_S3_BUCKET_URL": "<S3 Bucket URL>"
}
```

- URL 마지막에 `/` 는 적지 않습니다.
- CYPRESS_LAMBDA_API_URL은 앞에서 배포된 서버리스 백엔드의 URL Endpoint를 기입합니다.

2. Actions Secrets 입력

깃허브 메뉴의 Settings -> Secrets -> Actions 에 들어가서 `CYPRESS_ENV_CI` 키를 아래와 같이 생성합니다.

```
{
  "CYPRESS_LAMBDA_API_URL": "<Lambda 서버리스 API URL>",
  "CYPRESS_S3_BUCKET_URL": "<S3 Bucket URL>"
}
```

여기서 입력한 json 데이터는 workflow 실행 중 아래 명령어에 의해 자동으로 생성됩니다.

```
echo '${{ secrets.CYPRESS_ENV_CI }}' > cypress.env.json
```

Cypress 자동화 테스트

```
node create-fixtures.js
yarn test
```

GitHub Actions workflow

- `.github/workflows/update-sheet-cron.yml`
  - 일정 주기에 따라 실행되는 워크플로우입니다.
- `.github/workflows/update-sheet-manually.yml`
  - 수동으로 실행하는 워크플로우입니다.

더 많은 사건을 빠르게 처리하기 위해 GitHub Actions의 Job Matrix에 여러개의 기기를 두어 병렬 실행을 하도록 구성되어있습니다. Cypress에서 제공하는 github-action에서도 병렬 실행을 제공하고 있지만, 유료 플랜을 사용하지 않는 경우 제한이 걸려있기 때문에 별도의 Node.js 스크립트를 사용해서 병렬 실행을 하도록 구성하였습니다. `create-fixtures.js` 와 `cypress-partial.js` 에서 관련 코드를 확인할 수 있습니다.

실제 워크플로우 실행은 아래와 같은 단계로 구성됩니다.

1. 픽스쳐 생성 (create-fixtures)

```
node create-fixtures.js
```

- Serverless를 통해 배포된 Lambda API에서 스프레드시트 데이터를 받아옴
- `cypress/fixtures` 폴더에 5개 단위로 `cases_chunk_<number>.json` 를 생성함. 5개씩 나누는 이유는 Cypress 병렬 실행을 위함임.
- `cypress/e2e` 폴더에 5개 단위로 `spec_chunk_<number>.cy.js` 를 생성함.

이후 최신 정보를 유지하기위해 구글 스프레드시트의 사건 진행현황 시트 데이터를 초기화합니다.

2. 자동화 메인 스크립트 실행 (cypress-run)

```
yarn start --spec $(node cypress-partial.js n ${{ matrix.containers }})
```

- `n` 개의 러너 중 해당 러너가 실행해야 할 번호의 spec을 실행

3. 아티펙트 삭제 (delete-artifacts)

워크플로우 실행중에 artifact에 보관한 파일들을 삭제합니다. 해당 작업을 진행하지 않으면 실행한 데이터가 남아서 개인정보가 유출 위험이 존재하며 깃허브에 보관 비용을 낼 수도 있습니다.

spec.cy.js 프로세스 설명

1. 나의 사건 조회 페이지 접속

2. 페이지 안의 모든 이미지들이 불러와졌는지 검사

가끔 캡챠 이미지가 깨질때가 있는데 미리 검사해서 실패시 빠른 재시도를 할 수 있음

3. 사건 번호 / 법원 데이터 강제 입력

EUC-KR 인코딩으로 인해 깨진 데이터들을 아래의 파일에서 직접 넣어줌

- `cypress/js/caseTypes.js`
- `cypress/js/courts.js`

4. 자동입력방지 문자 입력

Lambda API에 캡챠 이미지를 전송하여 예측된 숫자를 입력함 단, 정확도가 100%가 아니므로 실패시 최대 20번까지 재시도함

5. 사건 진행내역 스크린샷 캡쳐

Languages

JavaScript 96.4% Python 2.9% Dockerfile 0.7%