황현규

이메일: joseph95501@gmail.com
휴대폰: 010-9049-9550
포트폴리오: https://01joseph-hwang10.github.io/
Github: https://github.com/01Joseph-Hwang10/

2년차 주니어 개발자 황현규입니다. 웹 애플리케이션 개발과 SaaS 창업에 관심이 많습니다.

창업 아이템을 내 손으로 직접 구현하기 위해 개발을 시작했으며, 다수의 해커톤 및 창업 활동에서 활약하면서 나의 아이디어를 직접 구현하는 과정에서 개발에 대한 흥미와 재미를 느껴 개발에 몰두하고 있습니다.

최소의 자원으로 최대의 효과를 추구하는 린 방법론에 크게 공감하며, 개발을 할때도 어떻게 하면 더 적은 코드로 더 많은 일을 할 수 있을지 고민합니다. 기능 구현에 불필요한 단계가 있는지 주기적으로 살피며, 무엇을 하는지에 대한 뚜렷한 목적을 가지고 업무를 진행합니다.

지속가능한 개발을 지향하며, 유지보수가 용이하고 확장성이 뛰어난 프로그램을 작성하기 위한 사유 과정을 즐깁니다. 또한 명시적인 변수/함수 네이밍, 모듈화, 다양한 디자인 패턴을 통해 간결하고 직관적인 코드를 작성하기 위해 노력하며, 팀원이 처음 보는 코드라도 금방 이해하고 기여할 수 있게끔 노력합니다.

학력 사항

울산과학기술원 (2021.03 - 휴학 중)

- 컴퓨터공학과 학사과정 (주전공)
- 경영과학부 학사과정 (부전공)

경력 사항

트립빌더 (@tripbuilder)

- 2021.07 - 2021.12
- 개발팀 / Front-end Developer
- AI 기반 여행 계획 추천 모바일 앱 `TripBuilder` 개발 참여

랩이즈 (@labis)

- 2021.12 - 2022.11
- 개발팀 / Development Lead & Co-Founder
- 하드웨어 기반의 블록 코딩 교육 솔루션 `LearnQue` 기획 및 개발

쉐퍼드23 (@shepherd23)

- 2022.07 - 현재 재직중
- 개발팀 / Product Manager & Software Engineer

- E-Commerce 쇼핑몰을 위한 개인화 상품 추천 플러그인 PickHound 기획 및 개발

업무 경험

PickHound

=> `class-validator` 데코레이터 디버깅 및 소스 코드 분석

Backend Troubleshooting @shepherd23

- 모노리포 구성으로 클라이언트와 공유가 필요한 DTO를 공통 라이브러리로 분리해 사용중인 상황.
- `class-validator`를 통해 Decorate 된 공통 라이브러리의 DTO가 NestJS의 ValidationPipe 에 의해 타입이 검증되지 않는 문제 발견.
  - `class-validator`의 작동 방식을 알기 위해 소스 코드를 들여다보는 과정에서 내부적으로 `MetadataStorage` 객체를 이용해 Typescript -> Javascript 컴파일 과정에서 데코레이터 정보를 저장하고, 모노리포 내부의 라이브러리가 각각 별도의 `class-validator` 패키지를 설치하고 있어 발생하는 문제임을 확인.
- `class-validator`를 `peerDependencies`로 추가해 모든 패키지가 하나의 `MetadataStorage` 객체를 공유하도록 함.
- => 디버깅 과정을 통해 모노리포 상황이나 `npm` 패키지 퍼블리싱 상황에서 패키지 간의 호환성 이슈를 고려한 공유 라이브러리 구성이 중요함을 배움.

> **팁**
>
> 아래 링크에서 위와 관련한 더 자세한 내용을 확인할 수 있습니다.
>
> https://01joseph-hwang10.github.io{/posts/class-validator-debugging

=> 상품 정보에 대한 Caching 구현

Backend Optimization @shepherd23

- 추천에 필요한 상품 정보를 Firestore에서 가져오는 과정에서 발생하는 불필요한 읽기 과정을 줄이기 위해 인메모리 캐시 도입.
- 상품 정보에 대한 캐싱 및 캐시 무효화 기능을 제공하는 NestJS 모듈 `ActionProviderModule` 구현.
- => 도큐먼트 읽기 횟수 `30,000,000/일 -> 1,500,000/일`로 감소 -> 매우 큰 비용 절감 효과

See Also

- PickHound: https://01joseph-hwang10.github.io/docs/career-description/pickhound
- `ActionProviderService`: https://01joseph-hwang10.github.io/docs/career-description/pickhound#상품-정보에-대한-caching-구현

=> Firestore 복합 인덱스 관리를 위한 명시적 메서드 네이밍 컨벤션 수립

Backend Improvement @shepherd23

- Firestore를 도입하고 여러 필드를 기준으로 쿼리를 수행하는 경우가 늘어나면서, 복합 인덱스에 포함된 프로퍼티, 정렬 기준 등을 혼동하는 경우가 빈번히 발생.
- 모든 프로퍼티 이름을 포함하는 네이밍 컨벤션을 통해 복합 인덱스 관리를 명시적으로 하도록 함.
- => 복합 인덱스가 필요한 쿼리를 수행하는 데 발생하는 휴먼 에러 발생률 감소, 복합 인덱스가 필요한 쿼리 구현에 필요한 시간 감소.

> #### 네이밍 컨벤션 예시: `listRewardRecordByMallIdIssuanceTypeTimerange`
>
> - list: 목록을 반환
> - rewardRecord: `reward-record` 컬렉션에서
> - by: 다음의 필드를 기준으로
> - MallId: `mallId` 필드
> - IssuanceType: `issuanceType` 필드
> - Timerange: `issuedDate` 필드를 기준으로 `class Timerange` 클래스에 정의된 `startDate`, `endDate` 프로퍼티를 사용해 필터링

=> 점진적인 코드베이스 이관을 통한 DB 마이그레이션

Backend Improvement @shepherd23

- 버져닝, JSDoc Block Tags, `backwardCompatible*` 등의 패턴을 활용한 점진적인 코드베이스 이관으로 일괄 DB 마이그레이션이 불가능한 스키마에 새로운 변경 사항을 반영

=> 상품 추천 속도 향상을 위한 로그 데이터 분석 및 코드 인스펙션

Backend Troubleshooting Optimization @shepherd23

- 특정 쇼핑몰에서 상품 추천 API의 응답 속도가 느린 것을 발견함.
- GCL 로그를 내려받아 `numpy`, `pandas`, `matplotlib`을 활용해 각종 지표를 살펴보았으나, 위 쇼핑몰을 제외하고는 응답 속도가 예상대로 나타남.
- 이에 상품 추천 API의 코드를 인스펙션하며, `Date.now()` 를 활용해 각 단계의 응답 속도를 측정함.
- => Firestore로의 count 쿼리가 응답 지연의 원인임을 발견하고, count 쿼리를 제거해 응답 지연 문제 해결.

=> Leaky Bucket이 적용된 엔드포인트를 위한 HTTP Client 구현

Feature @shepherd23

- 데이터 요청이 필요한 API의 Leaky Bucket Policy로 인해 1초에 2회 수준으로 API 호출이 제한됨.
- 각 상품의 상세 정보를 하나씩 전부 요청해야 하는 상황에서 단순 API 호출로는 HTTP 429 에러 발생.
- LeakyBucketClient 클래스를 구현해 호출의 빈도를 제한하고, 429 에러 발생 시 리퀘스트 재시도.
  - HTTP 에러 코드별로 `Exception` 클래스를 만들고 `instanceof` 연산자를 활용해 에러를 구분하고 대기 시간을 조절.
- => 상품 추천에 필요한 데이터인 상품의 상세 정보를 전부 요청하는 작업을 가능케 함.

See Also

- `LeakyBucketClient`: https://01joseph-hwang10.github.io/docs/career-description/pickhound#leaky-bucket이-적용된-엔드포인트를-위한-http-client-구현

=> Bandit Engine 서비스와의 연동을 위한 HTTP Client 구현

Feature @shepherd23

- RecommendationAPI를 Bandit Engine과 연동하기 위해 Flask 서비스와 통신하는 HTTP Client 구현.
- `snake_case`를 사용하는 Flask 서비스와 `camelCase`를 사용하는 NestJS 서비스 간의 데이터 교환을 위해 `objectFromSnakeToCamel`, `objectFromCamelToSnake` 함수를 구현.

See Also

- `BanditClient`, `objectFromCamelToSnake`, `objectFromSnakeToCamel`: https://01joseph-hwang10.github.io/docs/career-description/pickhound#bandit-engine-서비스와의-연동을-위한-http-client-구현

=> 긴 프로세스를 담당하는 Worker 인프라 구축

Devops Feature @shepherd23

- 기존에는 모든 서비스를 Google Cloud Run을 이용해서 배포했음.
  - 데이터 요청이 필요한 API의 Leaky Bucket Policy로 인해, 상품 정보를 자사 Firestore DB로 가져오는 프로세스가 길어지면서 Cloud Run의 프로세스 제한 시간인 1시간을 초과하는 문제 발생.
- GCE 인스턴스 그룹과 Cloud Load Balancer를 활용해 긴 프로세스를 담당하는 Worker 인프라를 구축하여 이슈 해결.
  - Cloud Build Substitution Variable 과 Docker Build Args를 활용해, Cloud Run과 Worker 인프라 배포 코드 공유.
- => Cloud Build와 Docker를 이용한 CI/CD 자동화로 인력 소비 최소화 및 GCE Instance Group과 GCLB를 활용한 시스템의 유연성 확보.
- => 인프라 구축 과정에서 외부 API의 제약사항을 고려한 Worker 인프라 구축과정에서 실제 프로덕션 환경에서의 문제 해결 능력 향상

See Also

- `cloudbuild.yaml`, `deploy.sh`, `Dockerfile`: https://01joseph-hwang10.github.io/docs/career-description/pickhound#긴-프로세스를-담당하는-worker-인프라-구축

=> GCE instance group autoscaler 의 instance termination으로 인한 프로세스 중단 문제 해결

Backend Devops Troubleshooting @shepherd23

- GCE instance group autoscaler가 scale down을 하면서 instance를 종료하는 과정에서, 종료되는 instance에 할당된 프로세스를 중단하는 문제가 발생함.
- => NestJS의 `OnApplicationShutdown` 인터페이스를 구현하여 프로세스 종료 시점에 진행중인 작업을 재시작하는 로직을 구현해 이슈 해결.
  - Fire and forget 방식으로 request를 보내 Shutdown process에 영향을 주지 않도록 함.

See Also

- `MallSnapshotService.onApplicationShutdown`: https://01joseph-hwang10.github.io/docs/resume/backend-agnostic#gce-instance-group-autoscaler-의-instance-termination으로-인한-프로세스-중단-문제-해결

=> 모델 파라미터 압축을 통한 성능 향상

Backend Optimization @shepherd23

- Bandit Model의 파라미터가 너무 커 HTTP 413 에러를 반환하는 문제를 발견함.
- 이에 `np.float32`, gzip 을 도입하고 파라미터를 압축, 압축해제하는 로직을 추가함.
- => 모델 파라미터 크기 10배 축소
- => 상품 추천 응답 시간 2배 증가

=> 점진적인 코드베이스 이관을 통한 서비스 장애 방지

Backend Improvement @shepherd23

- Bandit 모델의 버전이 많아짐에 따라 각 모델 버전을 engines 폴더 아래에 디저트 이름을 붙여 모듈화함.
- 이와 함께 엔드포인트 구조를 체계적으로 변경하는 과정에서 기존 엔드포인트를 제공하는 `blueprint` 가 사라지면 서비스에 장애가 발생함을 인지함.
- 기존의 코드베이스와의 호환성을 유지하기 위해 기존 코드베이스에 존재하는 엔드포인트를 새로운 엔드포인트로 Redirect 하도록 구현함.
- => 서비스 간의 업데이트 속도 차이로 인해 발생하는 서비스의 일시적인 장애 없이 코드베이스 이관할 수 있었음.

=> 데코레이터 패턴을 활용한 AI 팀의 업무 효율화

Backend Improvement @shepherd23

- 데코레이터 패턴을 활용해 각 엔드포인트의 핸들러 함수를 데코레이터로 감싸 에러 핸들링, 인증 등의 공통된 로직을 분리함.
- => 웹과 관련한 코드를 데코레이터를 통해 추상화하여 웹에 익숙하지 않은 AI 팀의 업무 효율화.
- => 신입 AI 팀원도 별도의 웹 관련 교육 없이 바로 업무에 참여 가능한 수준.

See Also

- Before & After: https://01joseph-hwang10.github.io/docs/resume/backend-agnostic#데코레이터-패턴을-활용한-ai-팀의-업무-효율화

=> 모노리포 구성, 패키지 버져닝 및 프로젝트 간 공통 모듈 공유

Automation Improvement @shepherd23

- Lerna, Python Module을 활용해 Monorepo 구성 (Recommendation API, Bandit Engine, Banner Manager)
- Lerna, `standard-version`을 활용한 패키지 버전 관리 (모든 프로젝트)
- 공유가 필요한 코드는 따로 패키지로 분리해 다른 리포지토리에서도 사용 가능하도록 구성 (Recommendation API, Banner Manager)
  - 공유 패키지 빌드 프로세스 구성 및 Github Actions, Google Artifact Registry를 활용해 자동 배포
  - => 비효율적인 복사 붙여넣기 방식의 코드 공유를 최소화.

=> 불필요한 코드 빌드 최소화

Optimization @shepherd23

- 대부분의 CI/CD 빌드 트리거가 Github Actions의 `push` 이벤트에 의해 발생함.
- 모노리포 상황에서 하나의 패키지에만 변경사항이 있는 경우에도 모든 패키지의 빌드가 발생하는 문제가 있었음.
- Lerna를 이용한 버저닝 관리와 함께 패키지 레지스트리에 배포되어 있는 패키지의 버전과 현재 레포지토리의 패키지 버전을 비교해 변경사항이 있는 경우에만 빌드가 발생하도록 구성함.

- => 불필요한 빌드 최소화 및 CI/CD 자원 소비 최소화

=> Jest, Pytest를 활용한 TDD로 개발 흐름 효율화

Backend Devops @shepherd23

- Jest, Pytest를 활용해 주요 엔드포인트 및 서비스 로직에 대한 Unit/E2E 테스트 작성.
- Insomnia 등의 툴을 이용한 수동 테스트 시보다 훨씬 효율적인 개발 흐름을 구성할 수 있었음
  - => 기존 수동으로 테스트를 진행했을 때와 비교해 작업 시간 약 90% 감소. (기존 약 20분 → TDD 도입 후 최대 2분)
  - => 다른 팀과 버그 수정 관련한 불필요한 대화가 오고 가는 빈도 약 80% 감소. (기존 평균 4 ~ 5회 → TDD 도입 후 0 ~ 1회)

=> 사내 내부 도큐멘테이션 배포를 위한 Private File Server 구축

Backend Devops @shepherd23

- 정적 웹페이지를 회사 내부 인원만 열람할 수 있도록 하는 Private File Server 구축.
- static-webpage-with-auth 로 도커라이즈 하여 여러 도큐멘테이션 페이지에 쉽게 적용할 수 있도록 만듬.

See Also

- static-webpage-with-auth: https://hub.docker.com/r/josephhwang02/static-webpage-with-auth

=> Google Sheets를 활용한 간이 어드민 대시보드 구축

Feature @shepherd23

- NestJS는 Django처럼 Admin Dashboard를 제공하지 않고, Thrid Party 라이브러리 (e.g. AdminJS) 도 Firestore 를 DB로 사용하는 경우를 고려한 라이브러리는 없었음.
- 이에 Google Sheets, Sheets API, Cloud Functions 를 활용해 사용자 데이터 시각화 및 쇼핑몰 관리 기능을 포함하는 간이 대시보드 구축함.
- => UI 구현 필요 없이 간단한 로직 구현만 필요했기에 빠른 대시보드 구현이 가능했음.
- => 엑셀에 기반한 대시보드 구현으로 비개발 인원도 쉽게 지표를 확인하고 쇼핑몰을 관리할 수 있었음.

=> Cloud Logging & Slack을 활용한 에러 모니터링 시스템 구축

Devops Feature @shepherd23

- 에러 확인을 위해 주기적으로 GCP 대시보드에 접속하는 일이 매우 번거롭다는 문제가 있었음.
- 이에 GCL 로그 라우터와 Cloud Functions를 활용해 에러가 발생할 때마다 Slack으로 알림을 전송하는 시스템 구축함.
- => 에러 확인을 위해 주기적으로 GCP 대시보드에 접속하는 비효율 제거.
- 4XX 에러나 5XX 코드를 가지는 로그도 라우터에 포함되어 알림이 전송되는 문제가 있었음.
- GCL의 Logging Query Language 를 이용해 알림을 받을 필요가 없는 로그를 필터링하는 기능을 추가함.
- => 불필요한 알림 방지.

=> 코드주석과 Notion을 연계한 사내 도큐멘테이션 작성 효율화

Devops Improvement @shepherd23

- 기존에는 비개발 인원이 개발 상황을 파악하기 위해 개발자에게 직접 문의하거나, 개발자가 이미 코드 주석으로 작성된 내용을 Notion에 다시 작성하는 등의 비효율이 존재했음.
- 이에 Swagger, Typedoc, Sphinx 등의 툴을 활용해 도큐멘테이션을 생성하고, 노션 페이지와 도큐멘테이션 페이지의 웹 링크를 연계하는 도큐멘테이션 작성 컨벤션 수립.
- => 중복 작업 최소화로 개발 효율성 향상
- static-webpage-with-auth 를 개발해 `Google Cloud Run`에 배포하여 사내 인원만 접근할 수 있도록 구성.
- => 비교적 적은 개발 자원 소비로 Netlify의 유료 솔루션 등을 사용하지 않고도 사내 인원만 접근할 수 있는 도큐멘테이션 페이지 구축.

See Also

- static-webpage-with-auth: https://hub.docker.com/r/josephhwang02/static-webpage-with-auth
- 사내 내부 도큐멘테이션 배포를 위한 Private File Server 구축: https://01joseph-hwang10.github.io/docs/career-description/pickhound#사내-내부-도큐멘테이션-배포를-위한-private-file-server-구축

스킬셋

Typescript HTML5 CSS3 ES6 CommonJS Webpack Babel Rollup Vite Lerna
Python Numpy Pandas Scikit Learn Django Flask PyTest Selenium Sphinx React
NextJS Redux RTK Query TailwindCSS MUI Emotion Cypress Storybook StencilJS
D3 Figma NestJS Express Jest Swagger TypeORM Typedoc SQL Firestore
Cloud Computing Firebase Cloud Run Cloud Build Cloud Storage Cloud Logging
Cloud Compute Engine Docker Linux Git Github Slack Notion Korean (Native)
English (TOEFL iBT 98 @ 2023.02)

*최종 수정: **2023년 9월 16일**에*