



## 추만석님을 응원해보세요!

# 이직/구직 중이에요 # 책임감 # 솔루션 지향 # 분석적 # 목표 지향적 # 문제 해결사

# 추만석

## 백엔드 개발자



뛰어난 문제 해결 능력을 가졌으며 문제를 같이 고민하며 협업하는 개발자 추만석입니다.  
크래프톤 정글에서 팀 프로젝트 중 발생한 CPU 과부하 문제를 해결하여 응답 시간을 96%(1200ms -> 50ms) 줄였고 동시 사용자 한계를 2명에서 100명까지 늘렸습니다.

운영체제 개발 프로젝트에서 페어 프로그래밍을 통해 지식을 공유하며 협업했습니다. 프로세스, 스케줄링, 동기화 등의 중요한 운영 체제 개념에 대한 깊은 이해를 갖추고 있습니다. CS 지식과 여러 프로젝트 경험을 바탕으로 사용자와 팀원을 위한 코드를 작성할 수 있습니다.

## 기술 스택

Java, Python, Spring, MySQL, JPA, Redis, Git, github-actions, Docker, NestJS

## 프로젝트

### 교육용 운영체제 PintOS 구현

크래프톤 정글  
2024.04. ~ 2024.04.

운영체제도 결국 하나의 프로그램이라는 것을 깨닫게 해준 프로젝트  
기존 스탠포드 운영체제 교육용 PintOS를 64bit로 발전시켜 운영체제 핵심 기능을 구현하는 프로젝트

프로젝트 링크: <https://github.com/maansuk> | 추천해요 🍷 | 제안하기 ✉ | 댓글 💬

기술스택: Ubuntu, C, Makefile, Docker

기여: 기술 문서를 번역하여 팀원과 공유했고 페어 프로그래밍을 제안하고 방식을 개선했습니다.

## 문제 정의 및 해결 방법

### # 어려운 문제를 동료와 함께 풀어나가는 페어 프로그래밍 경험 및 방식 개선

#### 문제 정의

- 하루 단위로 네비게이터와 드라이버를 정해서 기능 개발을 진행하여 피로도가 너무 높았습니다.
- 개발 중 같은 개념을 다르게 이해하고 있었습니다.
- 팀원이 기술 문서 이해에 어려움이 있었습니다.

#### 문제 해결

- 개발 단위를 기능 단위로 축소시키고 30분마다 역할을 바꿔서 개발했습니다.
- 개발에 들어가기 전 개념들을 먼저 정립하고 구현 로직을 공유하면서 서로 검증했습니다.
- 제가 알고 있는 개념을 활용하여 조금 더 이해하기 쉽게 기술 문서를 번역했습니다.

#### 결과


- 짧은 시간 동안 긴장감을 유지하면서 개발할 수 있어 개발 효율을 높일 수 있었습니다.
- 개념이 통일되면서 서로 생각을 더 잘 이해할 수 있고 로직을 공유하며 자신의 생각도 정리할 수 있었습니다.
- 팀원이 다른 번역된 기술 문서보다 훨씬 이해가 잘 됐다는 평을 남겨줬습니다.

### # 프로세스 생성(fork) 과정에서 자료구조 변경으로 메모리 효율 6배 개선

#### 문제 정의

- 프로세스 생성을 통한 Out-Of-Memory 테스트를 진행했었습니다.
- 기존의 프로세스가 가지는 파일 디스크립터 테이블은 동적 추가를 위해 링크드 리스트로 구현했습니다.
- 프로세스의 생성 속도가 느렸고 생성되는 프로세스 수가 20~30개가 한계였습니다.
- 프로세스를 최대 생성하고 제거하는 것을 10회 반복했는데, 속도가 너무 느려 2회까지만 반복되고 무한 대기 상태에 빠졌습니다.

#### 문제 해결

- 배열이 공간 지역성이 좋고 파일 디스크립터 번호를 인덱스로 하여 랜덤 액세스가 가능해 성능이 좋아질 것으로 판단했습니다.
- 마지막으로 사용된 파  |  | 댓글  !기시에 빠르게 번호를 부여받게 했습니다.

- 파일 디스크립터 접근을 하나의 함수로 추상화시켜줬었고 이 함수내에서 접근 방식을 변경했습니다.

## 결과

- 가상 환경에 메모리 추가 할당없이 자료구조 변경으로 프로세스 생성 수를 6배(30개 -> 180개)까지 증가시켰습니다.
- 자료구조의 차이를 성능으로써 확실히 느낄 수 있었고 적절한 선택이 중요하다는 것을 깨달았습니다.

## Learn On-Air (로아) | 온라인 강의 학습 도우미 | 크롬 익스텐션

크래프톤 정글

2024.04. ~ 2024.05.

*부하 테스트를 통한 서버 성능 측정부터 성능 개선까지 경험한 프로젝트*

온라인 학습시 집중에 어려움을 느끼는 수강생들을 위한 온라인 AI 학습 도우미 서비스

프로젝트 링크: <https://github.com/makeMyOwnWeapon> | 프론트엔드 2명, 백엔드 3명

기술스택: Nest.js | FastAPI | MySQL | Chrome Extension | AWS | OpenAI

회고글 링크: <https://10kseok.github.io/posts/wake-up-wake-up/>

기여: 프로젝트의 기술적인 부문에서 해결 방법을 주도적으로 제안하고 개선했습니다.

## 문제 정의 및 해결 방법

# CPU 과부하 문제를 해결하여 응답 시간을 96%(1200ms -> 50ms) 감소

### 문제 정의

- Vision AI를 사용하여 사용자의 수강 상태(졸음, 자러이탈)을 감지했습니다.
- 클라이언트에서도 동작하는 모델이었지만 크롬 익스텐션 정책 이슈로 백엔드로 이전하게 되었습니다.
- Node.js 환경에서 프로세스 생성을 통해 사용자 상태 추론 작업을 처리하니 응답 속도가 느려졌습니다.
- 사용자가 두 명으로 늘어나자 서버가 다운되면서 서비스 이용이 불가했습니다.
- 프로세스 생성/종료 비용이 크다고 판단했고 프로세스를 유지시키면서 응답 속도를 개선해야했습니다.
- 개선 후 CPU 사용률이 특정 구간을 넘지 못하여 CPU를 제대로 활용하지 못하고 있다고 판단했습니다.

### 문제 해결

댓글 💬

- 사용자 상태 추론만을 위한 별도의 비동기 서버(FastAPI)를 구축하여 프로세스를 유지시켰습니다.
- 사용자 상태 추론 모델을 매번 로드하는 작업을 첫 호출에만 불러오고 재사용하여 응답 시간을 줄였습니다.
- uvicorn worker 파라미터 조절을 통한 멀티 프로세싱으로 CPU 활용도를 높여 응답 속도를 추가적으로 개선했습니다.

## 결과

- 사용자 상태 분석을 위한 이미지 처리 API 호출 시 응답 속도를 96%(1200ms -> 50ms) 감소시켰습니다.
- 응답 속도 개선 이후, 부하 테스트를 통해 동시 사용자의 수가 100명까지 늘어난 것을 확인했습니다.

## # 줄임 기록, 학습 기록, 회원 등 전체 DB 스키마 및 RESTful API Spec 설계

### 문제 정의

- 팀원은 초기에 프로젝트에 사용할 기술을 학습해야했습니다.
- 비슷한 기술 스택에 대한 경험이 있는 저 혼자서 DB 스키마 설계를 책임져야 했습니다.
- 나중에 팀원이 이해하기 쉽고 데이터 중복을 줄이는 정규화를 지키는 구조로 설계해야했습니다.

### 문제 해결

- ERD 다이어그램과 DDL을 공유하여 팀원이 로컬에서도 실험해볼 수 있게 도왔습니다.
- 추가 학습 비용을 줄이기 위해 ORM을 도입했고 엔터티를 미리 정의해둬 즉시 개발이 가능하도록 했습니다.

## 결과

- 팀원이 별도의 DB 작업 없이도 손쉽게 영속성 관련 로직을 개발할 수 있었습니다.
- 설계시 API Spec을 문서화하여 클라이언트 개발자와의 소통 비용을 줄일 수 있었습니다.

## # 백엔드 서버 배포 헬 스크립트 작성으로 배포 작업 시간 90%(120s -> 12s) 단축

### 문제 정의

- ssh 연결과 scp를 이용한 파일 전송으로 수동으로 배포를 진행했습니다.
- 개발이 후반으로 갈수록 백엔드 서버를 배포할 일이 잦아졌습니다.
- 배포된 서버에서 동작을 확인할 일이 많아졌는데, 수동으로 하니 번거로웠습니다.

### 문제 해결

댓글 💬

- 쉘 스크립트에 ssh로 접속 및 서버 애플리케이션을 배포하고 실행시키는 로직을 작성했습니다.
- 서버가 정상적으로 띄워졌는지 확인하기 위해 health check 작업을 추가했습니다.

## 결과

- 쉘 스크립트 실행 한번으로 배포 관련 작업을 모두 자동화시킬 수 있었습니다.
- 빠른 배포를 가능하게 했으며 배포를 책임짐으로써 팀원의 부담을 덜었습니다.

## # OpenAI의 Chat Completion API를 활용하여 강의 내용 기반 문제집 생성 기능 구현

### 문제 정의

- 강의 내용으로 문제 생성을 지시하였으나 문제 수준이 국어 문제 수준으로 매우 낮았습니다.
- Claude와 같은 다른 API를 사용해봐도 비슷하게 낮은 수준의 문제가 생성되어 프롬프트에 문제가 있다고 판단했습니다.
- 정답이 여러 개가 나오고 json 형식도 예시로 제시했으나 일관적이지 않은 결과가 나왔습니다.

### 문제 해결

- 프롬프트에 컨텍스트를 제시했으며 문제 생성 과정을 순차적으로 지시했습니다.
- 프롬프트에서 입력 예시와 출력 예시에 실제 데이터를 입력하여 올바른 결과를 제시해줬습니다.
- API 옵션 중 json formatting 기능을 활성화하여 응답을 json 형식으로 하도록 보장했습니다.

## 결과

- 프롬프트 개선을 통해 생성되는 문제의 질을 높이고 원하는 형식의 결과를 응답 받을 수 있었습니다.
- LLM 서비스 이용해보며 프롬프트 엔지니어링의 중요성을 깨닫게 되었습니다.

---

## Cook-Shoong(쿡슝) | 음식 배달 플랫폼 프로젝트

NHN 아카데미

2023.07. ~ 2023.09.

최대한 실무와 비슷한 환경으로 개발해보려 노력했던 프로젝트

MSA와 유사한 구조로 다중 서버 환경으로 구성된 프로젝트

프로젝트 링크: [https://github.com/nhnacademy-be3-](https://github.com/nhnacademy-be3-CookShoong#%ED%9A%A1)

[CookShoong#%ED%9A%A1](https://github.com/nhnacademy-be3-CookShoong#%ED%9A%A1)

댓글 💬

기술스택: Spring Boot | Java | MySQL | JPA | Redis | Docker | JWT | Spring Cloud

회고글 링크: <https://10kseok.github.io/posts/digital-ticket-jwt/>

기여: 여러 도메인에서 공통적으로 사용하는 로직을 개발하여 코드 중복을 줄였습니다.

## 문제 정의 및 해결 방법

### # 팀원들이 기피하는 도메인을 책임지고 개발

#### 문제 정의

- 프로젝트 개발전 자신의 선호도로 도메인을 맡아 개발해야했습니다.
- 모두가 이전 프로젝트에서 Spring Security 프레임워크의 복잡함을 경험한 뒤 인증을 기피했습니다.
- 저또한 Spring Security와 인증에 대한 이해가 부족한 상태였습니다.

#### 문제 해결

- Spring Security in Action 서적을 구매해서 프레임워크에 대한 학습을 진행했습니다.
- 인증 로직에 관여하는 모든 클래스들에 브레이크 포인트를 찍어 실행 흐름을 파악했습니다.
- 공식 문서와 서적, 블로그를 참조하여 프로젝트 상황에 맞게 변형했습니다.

#### 결과

- JWT 인증과 블랙 리스트 도입, 레퍼런스가 없던 OAuth2 페이코 인증까지 성공적으로 구현했습니다.
- 예시가 없었던 인증 객체를 커스텀하여 불필요한 정보를 없이 인증에 필요한 정보만 담을 수 있었습니다.
- <https://shorturl.at/sILKO>

### # 에러 로깅 로직에 AOP를 사용하여 코드 중복 제거

#### 문제 정의

- 서버가 예상치 못하게 다운 됐을 때 원인을 꺾알 수 없었습니다.
- HTTP 트랜잭션간에 오류 발생시 예외 핸들러를 통해 모든 예외 처리 로직에 일일이 로깅했습니다.
- 똑같은 로깅 코드가 반복됐고 로깅 내용을 변경했을 때 다시 일일이 바꿔야하는 불편함이 생겼습니다.

#### 문제 해결

- 예외 처리 로직을 하나
  - 잦은 로깅을 막기 위해
- 댓글 
- 했습니다.

## 결과

- 반복되는 로깅 코드를 제거하고 서버 다운의 원인을 로그를 통해 알 수 있게 했습니다.
- 로깅의 일괄적인 처리가 가능해져 팀원이 따로 로깅 로직을 작성할 필요가 없게 했습니다.
- <https://shorturl.at/vu4Rb>

## # API 호출 시 인증 토큰 삽입에 **Interceptor**를 사용해 인증 로직 중복 제거

### 문제 정의

- Server Side Rendering을 하는 프론트 서버에서 백엔드 API를 호출하려면 인증 토큰이 필요했습니다.
- 매 API 호출마다 인증 토큰을 가져와서 RestTemplate 헤더에 추가해줘야 했습니다.
- API 호출을 위해선 토큰이 어디에 저장돼있고 어떻게 가져오는 건지 팀원 모두가 알아야했습니다.

### 문제 해결

- 공통으로 사용할 RestTemplate을 Bean으로 등록했습니다.
- API 호출시에 HTTP 헤더에 인증 토큰을 추가해줄 Interceptor를 구현했습니다.
- RestTemplate에 Interceptor를 추가하여 API 호출 전처리를 진행했습니다.

## 결과

- 팀원들이 추가 학습 없이도 기존에 쓰던 RestTemplate을 이용하면 인증 로직이 적용됐습니다.
- API 호출시 반복적으로 작성해야했던 인증 로직을 제거할 수 있었습니다.
- <https://shorturl.at/aVrZt>, <https://shorturl.at/mOGp4>

## # DTO 검증에 쓰이는 정규 표현식 작성

### 문제 정의

- DTO에 담긴 문자열을 특정 패턴에 맞게 검증해야했습니다.
- 제가 만든 DTO에서만 정규표현식을 사용했는데 팀원들도 정규표현식 사용이 필요하다는 것을 알게 됐습니다.
- 팀원들은 정규표현식을 낯설어 했으며 어려워했습니다.

### 문제 해결

- 팀원들에게 필요한 정규표현식을 물어보고 자주 쓰이는 패턴들을 정리했습니다.
- RegularExpressions라는 클래스에 자주 쓰이는 패턴들을 모아놓고 사용하도록 했습니다.

## 결과

댓글 💬

- 모아놓은 정규표현식이 DTO 검증에 약 27번 이상 사용됐습니다.
- 팀원이 변수명을 보고 유추해서 직접 정규표현식을 추가할 수 있었습니다.
- <https://shorturl.at/57kiU>

## # Assertj와 Mockito를 사용하여 단위, 통합 테스트 작성으로 회원 도메인 코드 커버리지 100% 달성

### 문제 정의

- 서비스 계층만을 검증하기 위한 슬라이스 테스트에서 인증 서비스 객체의 토큰 정상 발급을 테스트하려했습니다.
- 인증 서비스 객체는 토큰 발급 객체를 주입받고 있어 인증 서비스 객체를 mocking 하면 토큰 발급 객체도 mocking 됐습니다.
- 토큰 발급은 정상적으로 이뤄져야함으로 토큰 발급 객체는 실제 동작을 해야했습니다.
- 토큰 발급 객체는 토큰 설정값 객체를 주입받고 있어 테스트용 토큰 설정값 객체가 필요했습니다.

### 문제 해결

- 토큰 발급 객체를 mock 객체가 아닌 spy 객체로 실제 동작을 하는 객체로 생성했습니다.
- 토큰 설정값 객체는 mock 객체로 만들어 주입시킨 뒤 테스트 전처리 과정에서 임의의 설정값을 구성했습니다.

### 결과

- 테스트하기 힘든 객체를 @Spy 기능을 활용하여 가능하게 만들었습니다.
- 별도의 테스트용 설정 파일을 만들지 않고도 테스트 환경에 맞는 설정값 객체를 생성했습니다.
- <https://shorturl.at/JGvfw>

## Stemo (스테모) | 주식 매매일지 iOS 앱

개인

2021.01. ~ 2021.02.

소프트웨어로 문제 해결을 처음 진행한 프로젝트며 기획부터 앱스토어 배포까지 전과정을 경험한 프로젝트

주식 고수들이 매매일지를 작성하는 것을 보고 일반인들도 쉽게 매매일지를 작성하여 투자를 돕는 앱을 개발하고자 했습니다.

기술스택: Swift, UIKit, FSCalendar, Charts, RealmSwift

- 누적 다운로드 수 2000+ | [앱 소개](#) | [데모](#) | [댓글](#) | [서비스 지원](#)



- FSCalendar를 사용하여 일자별 매매일지를 보여주고 Charts를 통해 수익률 추이 제공
- RealmSwift를 이용하여 사용자 기기에서만 데이터 접근 및 저장
- 모바일 애플리케이션의 동작 방식을 이해하고 배포 과정에서 플랫폼의 한계를 느낌
- 사용자가 의도와 다르게 기능을 사용할 수도 있다는 것을 배움
- 사용자가 기다려왔고 필요한 앱이었다는 리뷰를 통해 보람을 느낌

---

## 포트폴리오

### URL

[기술 블로그](#)



[깃헙 주소](#)



---

## 교육

### 크래프톤 정글

사설 교육 | CS 중점 교육 (1294시간)  
2024.01. ~ 2024.05. | 졸업

---

### NHN Academy

사설 교육 | 자바 백엔드 과정 (1,200시간)  
2023.01. ~ 2023.08. | 졸업

---

### 창원대학교

대학교(학사) | 신소재공학/스마트제조ICT(OS,DB,네트워크,알고리즘,자료구조) | 학점 4.2/4.5  
2017.03. ~ 2023.02. | 졸업

---

## 자기소개

댓글 💬

## 개발자가 되려는 이유

저는 제 생각을 프로그래밍으로 실체화시키는 것과 끝까지 파고들어 문제가 해결됐을 때 오는 성취감을 즐깁니다.

본 전공에서는 제 생각 하나를 테스트하고 검증하려면 이틀에서 삼일 정도의 시간이 필요했습니다. 하지만 파이썬을 처음 접했을 때 사소한 생각이라도 수 초에서 몇 시간내로 검증할 수 있었습니다. 제가 잘못 생각한 점은 그 자리에서 바로 확인 할 수 있었고, 어떠한 문제가 생겼을 때, 에러 로그를 유심히 읽고 문제를 해결하는 것에서 흥미를 느꼈습니다. 더 깊이 있는 지식을 학습하고 싶었고, 관련 학과를 복수전공 하면서, 마법같이 느껴졌던 동작들의 원리를 이해할 수 있습니다.

저는 어제보다 더 나은 사람이 되고 싶어 노력합니다. 그러나, 어느 수준에 다다르면 오만해지기도 합니다. 개발 업종은 기술들의 변화가 굉장히 빠르고 성장 없이는 살아남기 힘든 업종으로 알고 있습니다. 이러한 업종에서 일한다면 나태해지지 않고, 끊임없이 부족함을 느끼며, 계속해서 성장하는 사람이 될 것이라 생각했습니다.

제 생각을 통해 다른 사람에게 도움을 주는 일을 하고 싶어합니다. 개인 프로젝트로 iOS 앱을 개발하며 많은 사람들에게 큰 도움을 주려는 서비스를 만드려면, 여러 사람의 힘이 필요하다는 것을 느꼈습니다. 팀 프로젝트, 창업 등을 거치며, 다양한 역할 속에서 제가 가장 잘할 수 있고, 제 생각을 드러낼 수 있는 분야는 개발 분야라는 것을 깨달았습니다.

회사에서 개발자로서 역할을 제대로 수행하기 위해 관련 지식을 미리 습득하고 지원하고 싶었습니다. **실무 지식 습득 목적으로** 간 NHN 아카데미에서 웹 기반 기술과 다양한 베이스를 가진 **동료들과 소통, 코드 리뷰, 팀 프로젝트**를 경험했습니다. 같이 일하고 싶은 사람이 되고 싶어 노력했고, 그 결과 **"팀 분위기를 긍정적으로 만들고, 본인이 공부한 기술을 팀원들에게 적극적으로 공유함. 기술을 깊이 있게 고민하고 구현하였음. 코드 리뷰를 성실하게 진행함"** 과 같은 평가를 받을 수 있었습니다.

## 개발 경험

### [말은 일은 스스로 검증합니다]

부하 테스트를 통해 서버 한계를 측정하고 성능을 개선한 경험이 있습니다. 클라이언트에서 백엔드로 이미지 처리를 이관하게 되면서 두 명의 사용자만으로 서버가 한계에 도달했습니다. 이미지 처리 요청마다 프로세스 생성/종료가 일어나 응답시간이 매우 길었습니다. 별도의 이미지 처리 비동기 서버를 개발했고 응답시간을 96%(1200ms > 50ms) 줄였습니다. locust를 통해 초당 하나의 이미지를 전송하는 부하 테스트를 진행했고 동시 요청 50명까지는 CPU 사용량이 30%대로 확인했습니다. 사용자를 늘려도 CPU 사용량은 그대로 있으며 응답시간이 1초를 넘기는 문제가 발생했습니다. 멀티 프로세싱을 이용하여 CPU 사용량을 늘렸고 100명의 사용자까지 안정적으로 버티는 것을 확인했습니다.

[협력을 통해 문제를 해결

댓글 0

페어 프로그래밍을 통해 어려운 문제를 동료와 함께 풀어나간 경험이 있습니다. 구현 전 개념에 대한 지식을 공유하고 변수명부터 함수 파라미터, 위치까지 상세하게 설명했습니다. 개념도 서로 다르게 아는 게 있었으며, 생각대로 의도가 잘 전달되지 않았습니다. 생각을 도식화하여 전달하고 추상화된 형태를 먼저 말하고 의문이 생기면 구체화하는 식으로 진행했습니다. 구현 로직을 말로 전달하는 과정에서 제가 모르는 개념을 깨달을 수 있었으며 혼자 개발 시 발생하는 사소한 실수를 줄일 수 있었습니다.

#### [팀원이 똑같은 불편함을 겪지 않았으면 합니다]

기술 문서를 번역하여 팀원과 공유한 경험이 있습니다. 운영체제 구현 기술 문서가 영문서였고, 웹상에 번역된 자료들은 배경지식이 동반되지 않은 텍스트 그 자체의 번역이어서 제대로 이해하기 힘들었습니다. 26,220자를 번역했고 기존 번역문에서 약 5000자 분량을 첨삭했으며 직접 설명을 통해 팀원의 이해를 도왔습니다. 그 결과, 팀원과 같은 개발 맥락을 가지고 협업할 수 있었고 의사소통이 원활해졌습니다.

리눅스 환경을 쉽고 빠르게 구축하는 방법을 공유한 경험이 있습니다. 프로젝트들이 리눅스 환경을 요구했고 여러 가상 환경 구축 방법 중 스크립트만으로도 환경을 구축할 수 있는 도커를 선택했습니다. 개발 환경 구축에 어려움을 겪는 동료를 위해 Dockerfile을 작성한 뒤 공유하여 손쉽게 리눅스 환경 구축을 할 수 있도록 도왔습니다.

Powered by Rallit.