이력서:

ㅇㅇㅇ

인적사항	**성명** **휴대전화** **E-mail** **생년월일**
학력사항	**2015.03 ~ 2020.02** 00대학교 000학과 졸업
웹페이지	**Blog** https://chagokx2.tistory.com/ **Github** https://github.com/cyj199637
보유 기술	#Java #Spring Boot #IntelliJ #Maven #MySQL #MyBatis #Redis #Jenkins #Docker #Python #Django
경력 사항	**기간** 2019.03 - 2020.01 **장소** 000 **직무 및 직급** Research Engineer (정규직) - ㅇㅇㅇㅇ 서비스 개발 - ㅇㅇㅇㅇ 관리자 페이지 개발

대외 활동	**기간**	**활동명**	**기관**
		빅데이터 소셜마케팅 전문인력 양성과정	

기타 활동	
	활동명 파이썬 스터디 **기간** 2016.07 ~ 2016.12 **설명** - 파이썬 라이브러리를 활용한 데이터 분석 공부 - flask를 활용한 웹사이트 개발 공부

자기소개서:

ㅇ ㅇ ㅇ

#저는 이러한 사람입니다.

4년전, “Hello World”를 컴퓨터 화면에 출력하면서 프로그래밍 세계에 발을 들였습니다. 머릿속으로만 생각하던 것을 제 손으로 직접 구현하여 저만의 세계를 직접 구축할 수 있다는 점에 매력을 느껴 개발자의 길을 선택했습니다. 학교를 다니면서 프로그래밍 관련 수업 외에도 스터디 그룹, 외부 교육, IT 회사 근무 등 많은 경험과 협업을 통해 다양한 기술을 배웠습니다. 단순히 혼자 구현하고 만족하는 서비스가 아닌 성능과 유지보수성 또한 뛰어난 서비스를 만들기 위해 저만의 성실함과 꼼꼼함을 무기로 개발 공부를 하고 있습니다. 기술에 대한 더 깊이 있는 지식과 이해로 실력을 증명하는 개발자가 되겠습니다.

#저는 이렇게 개발하고 공부합니다.

저는 개발 능력과 전문성을 향상시키기 위해 다음과 같이 공부하고 개발합니다.

첫 번째, 이해가 될 때까지 반복하여 학습합니다. 저는 개발 서적을 읽거나 인터넷 강의를 보면서 공부할 때, 이해가 되지 않는 부분이 있다면 그 부분을 이해할 때까지 놓지 않습니다. 꼼꼼하게 보지 않는다면 후에 기술에 대한 깊이가 부족하여 제대로 활용하지 못하고 또한 정확히 짚고 넘어가야 더 오랫동안 기억에 남기 때문입니다. 공부하면서 바로 받아들이지 못하는 개념이 생긴다면 놓친 부분은 없는지 그 개념이 포함된 챕터나 인터넷 강의를 다시 살펴봅니다. 그래도 습득하지 못하는 경우, 해당 개념을 좀 더 전문적으로 다룬 개발 서적을 사서 따로 공부하거나 공식 도큐먼트를 참고하며 기존에 제가 알고 있던 지식과 비교하면서 더 자세하게 파악하려고 합니다. 이해가 끝났다면 지금까지 공부한 부분에 대해 글을 쓰면서 한 번 더 정리하거나 예제 코드를 작성해보면서 완벽하게 제 것으로 만들기 위해 노력합니다.

두 번째, 블로그 글보다는 기술 공식 도큐먼트를 참고합니다. 프로젝트를 진행하면서 새로운 기능을 추가하거나 혹은 현재 사용하고 있는 기능의 성능을 개선할 때 검색 사이트에서 블로그 글을 찾기보다는 관련된 기술의 공식 도큐먼트를 참고합니다. 보통 블로그 글에는 글쓴이가 글을 쓸 당시에 사용했던 버전의 내용으로 설명되어있기 때문에 제가 사용하고 있는 버전에 적합한 해결방법이 아닐 수도 있습니다. 하지만 공식 도큐먼트에는 해당 기술의 정확한 의도, 버전 별 기술 문서, 자세한 내부 동작 원리 등 좀 더 깊이 있는 내용을 알 수 있으므로 이를 기반으로 기술 공부를 했습니다. 특히, 새로운 기술을 도입할 때는 해당 기술을 올바르게 사용하기 위해 어느 정도 시간을 투자하여 공식 도큐먼트를 읽으며, 각 문서에서 제안하는 베스트

프렉티스를 활용하여 코드를 작성하려고 노력합니다.

체계적인 문서화를 위해 노력합니다.

개인 프로젝트 같은 경우, 프로젝트의 README 문서에 프로젝트 개발 목적, 사용한 기술 스택, 프로젝트 주요 기능 등 프로젝트의 전반적인 부분을 작성하여 파악할 수 있도록 노력했습니다. README에서 언급한 부분에 대한 자세한 설명이나 그 외에 다루지 못했던 부분들은 github의 Wiki를 통해 문서화 작업을 하면서 어플리케이션 화면 설계, 어플리케이션 기능별 Use Case, 어플리케이션 ERD 등의 다양한 정보를 얻을 수 있도록 관리했습니다.

이와 더불어 읽는 사람들이 더 쉽게 이해할 수 있게 Application UI, ERD와 같은 전체적인 구조를 그림을 그려 시각적으로 표현했습니다. 그림을 그릴 때는 카카오 오븐이나 draw.io와 같은 툴을 사용하여 좀 더 깔끔하게 나타내고, 단시간 안에 마치며 생산성을 높일 수 있었습니다. 글이나 그림으로 설명하기 어렵다면 해당 기능에 대한 예시를 들어 모호한 부분을 최대한 제거하고자 했습니다. 시간적 여유가 있는 날에는 이제까지 작성한 문서를 다시 살펴보며 저만 아는 단어를 쓰지는 않았는지, 문맥에 맞게, 핵심만 담아서 썼는지 등을 확인하며 읽는 사람이 편하도록 문서 수정 작업에도 시간을 투자했습니다.

프로젝트를 진행하면서 기술적으로 고민한 사항들을 단순히 잊어버리지 않기 위한 용도로 적어 두는 것에 그치지 않고 기술 블로그에 다시 한 번 정리하여 글을 작성하면서 몰랐던 지식이라면 따로 더 공부하고, 이미 알고 있던 것이어도 한 번 더 검색하여 정확하게 짚고 넘어갔습니다. 이처럼 기록하고 문서화하는 것을 습관화하여 다양한 지식을 온전히 제 것으로 만들어 나갈 수 있었습니다.

#다른 사람들과 지식을 공유하는 것을 좋아합니다.

저는 다른 사람들과 다양한 지식을 공유하는 것을 좋아합니다. 완벽한 실력은 아니었지만, 회사에서 얻었던 지식과 경험을 공유하며 개발을 시작하려는 친구들에게 도움을 주고 싶었습니다. 그래서 그 친구들과 스터디 그룹을 만들어 파이썬 기초 문법, Django를 이용한 간단한 클론 코딩 등을 진행했었습니다.

시작은 도움을 주기 위한 것이었지만 저 또한 스터디를 진행하면서 많은 것을 얻어갈 수 있었습니다. 간단하게 알고 있던 부분도 더 자세하게 공부했고 그로 인해 Django Authentication 기능, Messages 프레임워크 등 회사에서는 사용하지 않던 Django의 기능들을 배우고 지식을 넓혀 나갈 수 있었습니다. 처음 웹 프로그래밍을 접하는 친구들이었기에 개념을 설명할 때는 그림과 함께 설명하여 쉽게 이해할 수 있도록 했고, 후에 혼동이 없도록 문장마다 용어의 의미를 명확하게 했습니다. 이러한 과정을 반복하면서 스터디를 준비하다 보니 커뮤니케이션 능력을 이전보다 한 단계 높일 수 있었습니다.

또한, 다른 친구들의 코드를 리뷰해주면서 다른 사람들이 읽기 쉽게 코드를 작성하는 것이 중요하다는 것을 깨닫게 되었습니다. 상수나 변수의 이름만으로도 그 역할이 최대한 드러나게 작성하거나 중첩된 조건문이나 반복문을

최대한 다른 메소드로 분리하여 작성하는 등 가독성을 고려하여 개발하기 시작했습니다. 이를 통해, 제가 작성한 코드를 스스로 검토하면서 코드의 품질을 높일 수 있는 계기가 되었습니다.

#자기 관리에 시간을 투자합니다.

저는 좀 더 즐겁게 개발할 수 있도록 자기 관리에도 시간을 투자합니다. 이전 회사에 재직할 당시 하나의 페이지를 개발하는 단계에 익숙해지면서 같은 코드만 기계적으로 짰기 때문에 개발 실력이 계속해서 정체한다고 느꼈습니다. 저는 여기에서 오는 회의감도 개발을 하는 데에 부정적인 요소 중 하나라고 생각했습니다. 그래서 단순 반복적인 개발에서 벗어나고자 개발 서적을 읽으며 더 좋은 코드를 짜기 위해서 노력했고, 개발할 때 요구사항과 관련된 여러 기술을 찾아보고 다양한 관점에서 비교하며 그중 가장 적절한 기술을 선택하여 사용하도록 노력했습니다.

개발을 잘하는 것만큼 자기 관리도 중요하다는 것을 알게 되면서, 무작정 개발을 하기보다는 자기 관리에도 시간을 배분했습니다. 정해진 시간 동안에는 운동하거나 취미 활동에 집중하면서 잠시 개발에 대한 생각이 나지 않도록 휴식을 취했습니다. 이렇게 관리를 함으로써 실수를 하지 않게끔 머리와 몸을 재정비하고 다음 날 할 작업에 대해 기대감을 갖게 하여 더 즐겁고 효율적으로 개발을 할 수 있게 되었습니다.

포트폴리오

* 프로젝트는 최근에 진행한 순서대로 적었습니다.

Itda SNS 프로젝트

Project: https://github.com/f-lab-edu/sns-itda

Wiki: https://github.com/f-lab-edu/sns-itda/wiki

Blog: https://chagokx2.tistory.com/category/Java/%E2%96%B6-----it%3Bda

2020.06 ~

#JAVA #SPRING BOOT #IntelliJ #Maven #MySQL #MyBatis #Redis #Jenkins #Docker

Instagram, twitter와 같은 SNS 서비스 'Itda'를 개발하는 프로젝트입니다. 벡엔드 개발을 더 깊고 빠르게 배우기 위해서 처음부터 웹 서비스 전체를 개발하기 보다는 벡엔드에 초점을 맞춰 이에 대한 핵심역량을 먼저 키워야 한다고 생각했습니다. 그래서 애플리케이션의 UI만 카카오 오븐을 활용하여 구현하고 프론트엔드 부분은 생략하여 백엔드 개발에만 주력했습니다. 각 기능마다 구현하는 것에만 그치지 않고, 어떻게 하면 더 많은 트래픽을 견딜 수 있을지 고민하며 개발했습니다. 또한, 객체 지향 설계 원칙을 최대한 준수하며 코드의 재사용성을 높일 수 있도록 개발했고, 이로 인해 유지보수에 드는 비용을 줄일 수 있도록 노력했습니다.

효율적인 소스 코드 관리를 위한 브랜치 관리

이 프로젝트는 소스 코드의 버전 관리를 위해 Git을 활용했습니다. 특히, Git-flow 전략을 도입하여 브랜치를 더 체계적으로 관리했습니다. 해당 전략을 통해 기능별로 브랜치를 나누고 작업할 수 있었고, 이에 따라 해당 기능에만 초점을 맞추어 코드의 정확성과 품질을 높이는데 집중할 수 있었습니다. 또한, 코드 리뷰를 할 때 다른 사람들에게 대상 코드를 명확하게 나타낼 수 있어 코드 리딩에 소요되는 시간을 줄일 수 있었습니다. 이와 함께 아무 코드나 메인 브랜치에 반영되지 않도록 코드 관리 규칙을 정하고 이를 기반으로 개발 흐름을 관리했습니다. 먼저 'develop'에서 'feature'를 생성하여 이슈에 명시된 기능에 대한 개발을 진행하고 각 기능에 대한 개발을 완료하면 코드 리뷰를 요청했습니다. 혹여 실수라도 검증되지 않은 코드를 머지하는 것을 방지하기 위해 브랜치 보호 규칙을 추가하여 코드 리뷰를 통과한 코드만 베이스 브랜치에 머지할 수 있도록 했습니다.

README와 Wiki를 통한 프로젝트 문서화

프로젝트의 소개와 전반적인 내용은 README에, README의 내용을 더 자세하게 기록하거나 그 외에 프로젝트와 관련 있는 내용은 github의 Wiki에 분리하여 문서화를 진행했습니다. README에는 프로젝트 개발 목적, 사용한 기술 스택 및 개발 환경, 프로젝트 주요 기능 등을 작성했고 필요한 내용은 다 포함하되 요약하여 정리했습니다. github의 Wiki에서는 어플리케이션 화면 설계, 어플리케이션 기능과 같이 README에서 언급한 부분이나 코드만으로는 설명하기 어려운 부분을 좀 더 자세하게 설명하여 문서화하여 관리했습니다. 어플리케이션 화면 구성이나 ERD 혹은 개념을 설명하는 글에서는 다른 사람들이 더 직관적으로 프로젝트를 파악할 수 있도록 사진이나 그림을 활용하여 시각적으로 나타냈습니다.

분산 처리 환경에서의 대용량 트래픽을 고려한 로그인 기능 구현

처음에 개발한 로그인 기능은 단순히 동작하는 데에만 그쳐 있는 코드였기 때문에 만약 이대로 배포되어 다양한 상황들을 맞닥뜨린다면 많은 장애가 일어날 수 있다고 생각했습니다. 특히, 많은 사람들이 동시에 접속하여 거대한 트래픽이 발생한다면 이를 견디지 못하고 문제가 생길 가능성이 높다고 판단했습니다. 그래서 위의 문제 상황에 대처가 가능하도록 리팩토링 작업을 진행했습니다.

SNS 서비스는 많은 요청을 동시에 처리할 필요가 있지만 특성상 다른 서비스에 비해 데이터 정합성 이슈에 민감하지 않습니다. 그러나 장애가 발생하여 서비스가 중지되면 중지된 시간만큼 광고가 집행되지 못해 매출에 큰 타격을 입는 등 피해 범위와 정도가 크기 때문에 가용성이 높은 Scale Out 확장 방법이 적절하다고 생각했습니다. 그러나 각 서버의 WAS는 세션 데이터를 다른 서버와 공유하지 않기 때문에 Scale Out을 선택했을 때 세션 데이터 불일치 문제가 뒤따라 발생했습니다.

이 문제를 해결하기 위해 분산 처리 환경에서의 세션 관리 방법에 대해 찾아본 결과 대표적으로 Sticky Session, 세션 클러스터링, 세션 스토리지 분리 방식이 있었습니다. 그 중 세션 스토리지 분리 방식은 다른 방식과 달리 특정 서버에만 트래픽이 몰리거나 서버의 수를 늘리면서 발생하는 추가적인 부하가 없었습니다. 또한 서버를 쉽게 추가하고 제거할 수 있어 앞서 선택한 Scale Out과도 잘 어울린다고 생각했기 때문에 이를 선택하여 별도의 세션 스토리지에서 각 서버에서 세션 데이터를 공유할 수 있도록 했습니다.

세션 스토리지로는 대표적인 인메모리 데이터베이스인 Redis와 Memcached를 고려했습니다. 두 엔진 사이의 사용성이나 성능적인 차이는 거의 없었지만 Spring에서 Redis API를 지원했기 때문에 제가 진행한 프로젝트의 개발 환경에서 쉽게 사용할 수 있고 이에 따른 유지 보수성도 좋아질 것으로 생각했습니다. 따라서 Redis를 세션 스토리지로 선택하여 트래픽이 늘어나더라도 쉽게 확장할 수 있으면서도 세션을 효율적으로 처리할 수 있는 분산 처리 환경을 구성할 수 있었습니다.

AOP를 활용하여 반복되는 로그인 체크 로직 분리

itda SNS 서비스의 기능은 대부분 로그인을 해야 이용이 가능합니다. 그렇기 때문에 많은 부분에서 로그인 여부를 검사하는 코드들이 반복되고 있었습니다. 중복되는 코드를 제거하고, 한 곳에서 관리하기 위해 AOP를 적용했습니다.

먼저, 현재까지 개발된 기능들 중에서 로그인 검사를 위해 반복되고 있는 코드를 파악하고 이를 별도의 Aspect 클래스로 분리했습니다. 해당 로직은 항상 핵심 기능이 실행되기 전에 선행되야 하므로 @Before 어노테이션을 이용하여 포인트컷을 적용했습니다. 또한, 간편하게 포인트컷을 설정하고 로그인 검사 기능이 적용된다는 것을 명시적으로 나타내기 위해 커스텀 어노테이션을 만들어 분리한 Aspect 클래스와 연결했습니다. 위와 같은 리팩토링 작업으로 중복 코드를 제거하여 유지보수성을 높이면서도 어노테이션을 사용하여 기능 별로 로그인 체크 여부를 쉽게 파악할 수 있게 하였습니다.

로그인 검사를 통과하면 세션에 저장된 사용자 정보를 가져오게 되는데 이 코드 또한 메소드마다 반복되고 있었습니다. 이를 제거하기 위해 Controller Layer에서 @PathVariable이나 @RequestParam이 인자를 주입하는 것과 동일하게 동작하는 커스텀 어노테이션을 정의했습니다. Controller 메소드 파라미터에 사용자 객체가 있는 경우 HandlerMethodArgumentResolver를 구현한 ArgumentResolver로 세션에 저장된 사용자 객체를 리턴하여 파라미터에 바인딩했습니다. 이를 통해 현재 세션에 저장된 값을 가져온다는 의미는 명확하게 하면서 코드는 더 깔끔하게 유지했습니다.

단위테스트 작성

한 기능에 대해 테스트를 진행하려면 Controller Layer부터 DAO Layer까지 임의의 코드라도 어느 정도 틀이 갖춰져야 했고 Postman이라는 툴을 통해 일일이 그에 맞는 요청을 보내야했습니다. 간단하게 기능이 잘 동작하는지 확인하는 것이라도 요청마다 달라지는 데이터를 입력해야 하는 것이 불편했고, 테스트를 할 때마다 신경 써야하는 요소들이 많아지면서 테스트가 실패했을 때 그 원인을 파악하기 어려웠습니다.

이러한 불편함을 없애기 위해 '토비의 스프링' 이라는 책에서 읽은 단위 테스트를 프로젝트에 적용해 보았습니다. Mockito 프레임워크를 사용하여 테스트 진행 시 테스트 대상 객체가 의존하는 객체들은 Mocking 처리하여 의존 객체의 영향을 줄이고 테스트 대상 객체를 고립시켰습니다. 각 기능 별로 Mock 오브젝트의 행위를 지정하고 테스트 대상 객체가 Mock 오브젝트를 활용하여 제대로 동작했는지 행위 검증을 했습니다.

이러한 테스트 방식은 테스트하고자 하는 부분만 명확하게 테스트를 할 수 있었고, 자동화된 테스트로 이전보다 테스트를 효율적으로 진행하는데 도움이 되었습니다. 테스트를 통과한 코드에 대해서는 확신을 가지고 다른 기능을 개발할 수 있어 전체적인 개발 속도가 빨라질 수 있었습니다.

각 기능 별로 다양한 시나리오를 적용한 테스트를 작성하여 테스트 자체만으로도 해당 기능의 명세서 역할을 할 수 있도록 했습니다. 어떤 입력 값과 상황이 갖춰져야 올바르게 동작하는지 혹은 기대에 벗어나는 행동을 취했을 때 어떤 예외를 던지는지 등 케이스 별로 해당 기능을 자세하게 명세하고 있습니다. 이로 인해 다른 사람들이 코드를 보고 이해하는데 도움을 줄 수 있고, 코드의 활용성을 높일 수 있었습니다.

서버 비용 절감과 읽기 작업의 성능 향상을 위한 캐싱 적용

게시물 관련 기능을 개발하고 테스트를 하면서 서버에 동일한 요청을 여러 차례 보내게 되었습니다. 이런 과정에서 동일한 결과를 얻기 위해 매번 데이터베이스와 연결하여 똑같은 연산을 수행하는 것이 비효율적이라고 생각했습니다. 이런 비효율성을 개선하고 불필요한 서버 사용량을 줄이기 위해 캐싱을 적용하기로 결정했습니다.

캐싱 전략에는 크게 로컬 캐싱과 글로벌 캐싱이 있었습니다. 로컬 캐싱은 서버 내부 저장소에 캐시를 저장하기 때문에 속도는 빠르지만 서버 간의 데이터 공유가 안된다는 단점이 있었습니다. 이는 여러 서버 간 캐시 데이터에 대한 일관성 문제, 서버마다 중복된 캐시 데이터로 인한 리소스 낭비의 문제도 발생시킬 수 있었습니다. 따라서, 위의 문제를 해결할 수 있는 글로벌 캐싱 전략을 선택했습니다. 또한 현재 Redis를 세션 스토리지로 사용하고 있기 때문에 이를 캐시 저장소로도 활용하는 것이 효율성 측면에서도 좋다고 판단했습니다.

스프링이 제공하는 캐싱 추상화 기능을 사용하여 캐시 데이터를 관리했습니다. 케시 데이터는 최신 상태를 유지하면서도 캐시 미스가 최대한 일어나지 않도록 각 리소스의 성격에 따라 CacheName을 정의하고 CacheName마다 캐시의 만료시간을 다르게 설정했습니다.

모든 데이터를 캐싱하고 유지할 수 없기 때문에 메모리를 효율적으로 운영하고자 Maxmemory와 데이터 Eviction 정책을 설정하여 메모리 사용량에 제한을 두었습니다. 다양한 Eviction 정책 중 LRU 알고리즘은 최근 사용 여부 따라 데이터 보존 여부가 나뉘기 때문에 재요청될 가능성이 높은 데이터도 순위에 밀려 삭제가 될 수 있다고 생각했습니다. 요청이 많이 들어오는 데이터가 삭제된다면 그만큼 데이터베이스에 접근하는 횟수가 높아질 수 있다고 판단하여, LFU 알고리즘을 선택해 위와 같은 문제 상황이 없도록 했습니다.

Redis 세션 저장소와 캐시 저장소 분리

프로젝트 규모가 커지고 캐싱되는 데이터가 많아질수록 서버 한 대가 많은 요청을 처리해야하고 이에 따라 응답 속도가 저하될 것으로 예상이 되어 부하 분산의 필요성을 느꼈습니다. 특히, Redis는 싱글 스레드로 운영되기 때문에 CPU를 하나 밖에 쓰지 못합니다. 만약 목적에 따라 Redis 서버를 분리한다면 그만큼 CPU를 더 많이 사용할 수 있으므로 처리 능력을 높일 수 있었습니다. 또한 Redis에 저장된 세션 데이터와 캐시 데이터 간에는 연관성이 없기 때문에 데이터를 분산시켜도 문제가 없을 것이라고 판단했습니다.

먼저 새로운 Redis 서버를 생성하여 기존에 사용하던 Redis 서버로부터 캐시 서버를 분리했습니다. 캐시 서버로 연결하는 RedisConnectionFactory 빈을 생성했습니다. 기존에 만들어진 같은 타입의 빈과 구분되도록 이름을 붙이고, @Autowired 대신 @Qualifier를 선택하여 빈을 적절하게 주입했습니다.

MySQL Replication 구성 및 쿼리 요청에 따른 분기

프로젝트를 진행하면서 서비스가 커짐에 따라 다양한 쿼리 요청이 발생했고 MySQL 서버 한 대가 모두 처리하기에는 한계에 다다를 것이라고 예상했습니다. 해당 프로젝트에서는 쓰기 작업보다 읽기 작업이 훨씬 많이 발생할 것이라고 생각했기 때문에 이를 대비한 부하 분산이 필요하다고 생각했습니다.

처음에는 단순히 서버별로 다른 설정 값을 가진 DataSource 빈을 등록하여 쓰기 작업은 Master Datasource에서, 읽기 작업은 Slave Datasource에서 작업하도록 환경을 구축하려고 했습니다. 그러나 이 방법은 서버 확장에 따라 관리해야 하는 파일들이 증가하므로 불편함을 일으킬 것이라고 생각했습니다. 그래서 Spring에서 제공하는 AbstractRoutingDatasource 클래스로 동적으로 DataSource를 결정하도록 구현했습니다.

각 메소드에서 쿼리 요청에 따라 지정한 ThreadLocal 변수에 값을 다르게 저장하고, 저장된 값을 바탕으로 Master 서버와 Slave 서버를 분기했습니다. 그러나 하나의 메소드 안에서 쓰기와 읽기가 동시에 발생한다면 Master 서버에 방금 작성된 데이터가 Slave 서버에 전달되지 않아 이를 읽지 못한다는 문제가 발생했습니다. 따라서, 쿼리 단위가 아닌 트랜잭션 단위로 ThreadLocal 값을 변경하기로 했습니다.

먼저 전체 코드에 트랜잭션 기능을 적용하고 단순 읽기 작업만 있는 메소드에 readOnly 옵션을 활성화하여 해당 메소드는 Slave 서버에서 요청을 처리하도록 했습니다. 일반적인 트랜잭션 로직에서는 Connection 객체를 먼저 확보하고 트랜잭션 동기화를 진행합니다. 하지만 readOnly 옵션에 따라 서버를 분기하려면 트랜잭션 동기화를 먼저 마치고 해당 트랜잭션이 readOnly 옵션이 활성화가 되었는지에 따라 다른 Connection 객체를 가져야 하기 때문에 작업 순서를 바꿔야했습니다. 이를 위해 트랜잭션 시작시에 Connection Proxy 객체를 리턴하고 실제로 쿼리가 발생할 때 데이터소스에서 올바른 Connection 객체를 얻도록 했습니다.

비동기로 푸시 메시지 전송 기능 개선

서비스 사용자 간의 팔로우 기능을 추가하게 되면서 사용자의 활동을 알려주는 푸시 서비스도 추가했습니다. 현재까지는 푸시 서비스가 적용된 곳이 팔로우 기능뿐이지만 프로젝트를 진행하면서 다양한 기능에 푸시 메시지를 적용하거나 서비스 사용자의 수도 많은 상황을 가정한다면 성능 저하가 예상되었기 때문에 푸시 서비스의 로직을 개선해야 한다고 생각했습니다.

푸시 서비스에 대해 검색해보았을 때 위의 고민의 해결책으로 사용할 수 있는 FCM 서비스를 알게 되었지만 바로 도입하기 전에 어느 부분에서 문제가 되는지 알아보기 위해 REST API로 로우하게 구현해보려고 했습니다.

처음에는 한 번에 많은 요청을 병렬적으로 처리하기 위해 스레드의 수를 늘리는 방법을 생각했습니다. 그러나 요청을 빠르게 처리하기 위해 요청 수만큼 스레드가 만들어진다면 메모리 부족이나 성능 저하 등 다른 문제들이 발생할 가능성이 있었습니다. 그래서 위의 문제점을 보완할 수 있도록 스레드 풀을 생성하고 최대 스레드 수를 제한하는 방법을 생각했습니다.

스레드 풀로 많은 요청을 동시에 처리하고 적정한 수의 스레드를 유지해도 tomcat에서 생성되는 스레드는 요청들을 동기적으로 처리하기 때문에 스레드의 처리 능력 이상으로 요청이 들어올 경우 결국 병목이 발생할 수 있다는 것을 알게 되었습니다. 따라서, Spring에서 제공하는 @Async 어노테이션을 사용하여 새로운 스레드풀로 tomcat 스레드는 블록시키지 않도록 구현했습니다.

이와 같은 비동기 방식은 FCM에서도 지원하고 있었기에 좀 더 안정적이면서도 더 나은 성능을 위해 @Async 어노테이션 대신 FCM의 sendAsync 메소드로 대체했습니다.

Jenkins를 사용하여 CI/CD 과정을 자동화

어플리케이션을 패키징하고, 패키징한 파일을 배포 서버에 옮겨 실행시키고, 성공적으로 실행시키면 다시 처음부터 테스트하는 등 이전에 경험한 배포 과정은 불편한 점이 많았습니다. 이러한 문제를 해결하고 일련의 과정을 자동으로 진행해주는 툴을 찾게 되었습니다. 여러 가지 CI/CD 관련 툴이 있었지만 Jenkins는 무료로

사용할 수 있고 다른 툴에 비해서 활용 사례와 레퍼런스가 다양했기 때문에 Jenkins를 선택했습니다.

먼저 언제든지 자동으로 코드를 통합하고 배포할 수 있도록 클라우드 서비스를 이용하여 Jenkins 서버를 따로 구축했습니다. Github에 새로운 Pull Request가 발생하면 자동으로 빌드하기 위해 Jenkins에서 저의 프로젝트에 대한 Multibranch Pipeline을 생성했습니다.

Github에서 코드 변경 사항을 감지할 때마다 Jenkins로 Webhook을 날리면, Jenkins에서는 미리 작성한 Jenkinsfile에 따라 Maven의 package, clean, test 과정을 진행하도록 했습니다. 코드를 통합하면서 PR 페이지 하단에 단위 테스트나 통합 테스트 결과를 볼 수 있게 했고, 테스트가 실패하면 이를 바로 인식할 수 있도록 메일로도 알림을 받을 수 있도록 했습니다.

추가로 로컬 서버는 윈도우 OS였지만 운영 서버는 ubuntu OS였기에 어플리케이션을 OS 환경에 종속적이지 않게 만들 필요가 있었습니다. 그래서 Docker를 사용하여 어플리케이션 이미지를 빌드하고 이를 Docker hub 저장소에 올리도록 Jenkinsfile을 작성했습니다. 이후 배포가 성공적으로 완료되면 운영 서버에서 미리 작성한 스크립트 파일에 따라 Docker hub 저장소에 저장한 이미지를 다운받고 실행하도록 하여 CD 자동화 작업을 마칠 수 있었습니다.

ㅇㅇㅇㅇ 서비스 개발

2019.03 ~ 2019.06

#Python *#Django* *#MySQL*

ㅇㅇㅇㅇ 서비스는 초등학생부터 중학생을 대상으로 한 코딩 교육 플랫폼입니다. 선생님들을 위한 학습자 관리, 학생 별 수업 진도 관리, 문제를 통한 학습 체크 등 서비스 내의 다양한 기능이 포함된 프로토타입을 개발했습니다.

S3를 활용한 동영상 관리

이 서비스의 주요 컨텐츠는 동영상이었습니다. 그렇기에 동영상 컨텐츠 관리가 가장 중요한 이슈였습니다. 그래서 EC2 서버에 저장하는 대신 S3를 사용하여 파일의 관리를 용이하게 하였고, 파일 관리에 따른 비용을 줄일 수 있었습니다. 또한 동영상 및 다양한 웹 컨텐츠들을 동적인 컨텐츠들과 분리하여 서버를 거치지 않고 서비스함으로써 서버의 부하를 줄일 수 있었습니다. S3에 올려놓은 컨텐츠들마다 접근 권한을 다르게 설정했고 보안이 중요한 동영상 컨텐츠의 경우 해당 파일에 접근하는 http 요청의 referer header를 검사하여 저희 사이트를 거치지 않은 사용자는 접근할 수 없도록 하였습니다.

선생님들을 위한 간편한 LMS 기능

이 서비스에서는 선생님들께서 자신이 개설한 수업을 들을 학습자를 직접 등록해야 합니다. 많은 학생들을 한 명씩 직접 등록하기 어렵기 때문에 파일을 업로드하면 간편하게 학습자를 등록하고 관리할 수 있도록 기

능을 구현했습니다. 또한 선생님들께서 개설한 수업 별로 '전체 학생 진행 상황' 페이지를 두어 수업 달성률, 수업 이해도, 한 주간 수강 횟수 등 다양한 수치로 학생들의 학습 이력을 실시간으로 파악할 수 있는 진도 관리 기능을 구현했습니다. '문제 설정' 기능을 통해 선생님들은 각 영상 별 평가 문제 중 원하는 문제를 선택하여 시험을 출제하고, 학생들의 성취 정도를 객관적으로 평가할 수 있도록 하였습니다. 시험 결과는 '채점 결과 보기' 기능을 통해 학생 별로 상세하게 채점한 결과를 제공했습니다. 오프라인에서도 해당 서비스의 컨텐츠를 활용하여 수업을 진행할 수 있도록 '강의 찾기' 페이지에서 학습지도교안, 활동지를 제공하였습니다.

학생들을 위한 학습 및 평가 기능

이 서비스에서는 학생들은 선생님이 개설한 수업을 선택하여 수업 내의 코딩 교육 영상을 통해 학습합니다. 학생들이 영상을 시청한 후, '평가하기' 페이지에서 자체적으로 해당 영상에 대한 이해도를 체크하고, '결과 확인' 페이지에서 자신의 수준이 어느 정도인지 파악할 수 있는 자체 평가 기능을 제공했습니다. 또한 '문제 풀기' 기능을 통해 선생님이 출제한 문제를 풀어 한 번 더 자신의 이해도를 체크할 수 있게 하였습니다. 시험을 제출할 경우, 바로 시험 결과와 풀이를 제공하여 부족한 개념에 대한 반복 학습이 가능하도록 하였습니다. 학생들이 수업에 더 적극적으로 참여할 수 있도록 하나의 강좌에 있는 모든 영상을 수강할 경우, 뱃지를 부여하는 기능을 구현했습니다.

빅데이터 소셜 마케팅 분석 보고서 공모전

2018.12 ~ 2019.02

#python *#R*

데이터 분석을 통한 인테리어 플랫폼 서비스 '오늘의집'의 인지도 향상 마케팅 전략

오늘의집 서비스의 핵심 고객층은 신혼부부이며, 서비스 자체에서도 신혼부부를 위한 별도의 탭이 마련되어 있고 관련 이벤트들이 주를 이루고 있었습니다. 그러나 통계청의 인구동향조사에 따르면 매년 혼인 건수는 꾸준히 감소하는 추세이며, '오늘의집' 이라는 키워드로 검색된 SNS 데이터에서 키워드 분석을 한 결과 '신혼'과 관련된 키워드의 비율은 낮게 나옴으로써 신혼부부의 버즈량은 정체된 것으로 파악되었습니다. 이를 통해 주요 타겟층 확장의 필요성을 감지했고 사용자의 니즈를 파악하기 위해 서비스 내의 '질문과 답변'에 작성된 약 5000건의 사용자들의 글을 파이썬의 BeautifulSoup, urllib, selenium 라이브러리를 활용하여 크롤링했습니다. 크롤링한 데이터에서 키워드를 추출한 결과, '자취', '원룸' 등 1인 가구와 관련된 키워드의 비율이 높게 나왔습니다. 앞선 데이터를 기반으로 현재의 마케팅은 1인가구의 공감을 얻지 못하고 있다고 판단되었고, 저희 팀은 이들을 위한 새로운 마케팅 전략을 제안했습니다. 저희는 먼저 청년들이 자주 사용하는 SNS인 인스타그램에서 청년 1인가구를 대표하는 '자취'라는 키워드로 게시글을 검색한 후, 해당 게시글에 같이 달린 해시태그를 추출, 추출건수가 100건 이상인 키워드들과 '자취' 키워드 간의 연관분석을 실시하여 그들의 라이프스타일을 파악했습니다. 그 결과 '혼밥', '요리', '고양이'라는 키워드가 가장 연관성이 높다고 나왔고, 이를 활용한 1인 가구가 가장 많이 분포한 관악구를 지나다니는 버스 및 지하철의 배너 광고, 자취 생활 웹툰을 활

용한 PPL 광고, 고양이와 혼밥에 관한 내용을 담은 영상 광고 등의 마케팅 방안을 제시했습니다. 그 결과 해당 공모전에서 성과를 인정받아 우수상을 수상할 수 있었습니다.