

제 9 회 IT 경진 대회

작품 설명서

GitHub 활용 증진 서비스

GitFlow

2018 년 9 월

서론

IT 기업 중 많은 곳에서 Git 이라는 버전 관리 시스템을 사용하고 있습니다. 얼마 전 ‘마이크로소프트’ 사에 인수되어 현재 가장 주목받는 버전 관리 시스템이기도 합니다. 우리가 이름을 들으면 알 수 있는 큰 기업들과 마찬가지로 대부분의 신생 기업(스타트업)에서도 이미 Git 을 사용하고 있습니다.

위와 같이 Git 을 실제 업무에 활용하고 있고, 입사 지원 양식에 GitHub URL을 적는 칸이 있는 것을 본다면, 프로그래밍을 공부하는 IT 학부생에게 Git 은 필수 조건이 되어 가고 있습니다.

개발자에게 GitHub 은 자신의 프로젝트를 Back-Up 및 관리 할 수 있는 곳이고, 이것들이 쌓여 자연스럽게 포트폴리오가 만들어지게 됩니다.

하지만 현재 성공회대학교 IT 학부(학과 포함)는 극히 일부만 GitHub 을 활용하고 있습니다. 이러한 안타까운 현실을 극복하고 IT 학부생이 GitHub 을 많이 활용하게 된다면, 그 결과로 선의의 경쟁을 통해 자기 주도적인 발전을 할 수 있다고 생각했습니다.

키워드: Spring Boot, MySQL, MyBatis, Handlebars, AWS, Scheduler, nginx, https, api, scheduler

목차

I. 작품 개요

- 1-1. 주제 / 작품명
- 1-2. 팀원 소개 및 담당
- 1-3. 개발 동기 및 목적
- 1-4. 기대 효과
- 1-5. 지향점
- 1-6. 유사 서비스 및 관련 연구

II. 개발

- 2-1. 개발 일정
- 2-2. 시스템 구성도 및 개발 환경
- 2-3. 작품의 기능
- 2-4. 개발 경과 및 완성도
- 2-5. GitFlow Repository Analysis & Scheduling Logic

III. iOS 개발 & 라이브러리

IV. 참고 문헌

개요

1-1. 주제 / 작품명

GitHub 활용 증진 서비스

GitFlow (깃 플로우)

1-2. 팀원 소개 및 담당

201132034 소프트웨어공학과 조민국

- 총괄, DB 설계, Front-End, GitHub 연동 Back-End 개발, GitHub API 연구, 배포

201132045 소프트웨어공학과 박영환

- 로그인 관련 Back-End 개발, GitHub API 연구

201232012 소프트웨어공학과 노욱진

- iOS 애플리케이션 개발

1-3. 개발 동기 및 목적

- IT 학부의 GitHub 활용도가 낮은 현실을 극복하고자 개발하게 되었습니다.
- IT 학부의 적극적인 GitHub 활용 증진의 목적을 갖습니다.
- 라이브러리를 활용한 것을 제외하고, 본인이 얼마만큼의 코딩을 했는지 한 눈에 볼 수 있고, 소속 내의 위치를 보여주하고자 했습니다.

- 이 서비스가 적극적으로 활용된다면, 데이터가 축적될수록 Library Filtering 이 정확해지고, 더 나은 방향으로의 피드백도 반영해야 한다고 생각하기 때문에 GitFlow 가 학회로서의 모습을 가질 수 있도록 새로운 구성원을 모집할 것입니다.
- 원하는 프로젝트의 세부적인 코드 분석을 통해 장기적으로는 프로젝트 수업에서의 활용도까지 생각했습니다.

1-4. 기대 효과

- 1 학년부터 GitHub 활용을 시작할 수 있습니다.
- IT 학부 구성원의 GitHub 을 한눈에 볼 수 있고, 이를 통해 다양한 사람들의 개발 스택을 확인할 수 있습니다.
- 현재 IT 학부생들이 고학년이 되었을 때, 학과 간의 소통이 지금보다 활발해질 수 있습니다.
- 선의의 경쟁을 통해 자기 주도적인 발전을 기대할 수 있습니다.
- GitHub 을 활용하여 프로젝트를 관리함으로써, 정리하면서 한 번 더 프로젝트를 이해할 수 있고, 자연스럽게 자신의 포트폴리오(개발 이력)를 만들 수 있습니다.

1-5. 지향점

- 학과 체제가 학부 체제로 변화함에 따른 선후배 간의 소통이 줄어든 것과 학과 학생들 간의 소통이 없었던 것을 이 서비스를 통해 개선하고자 합니다.

- 본인의 발전을 위해 개발 동아리에 지원할 때, GitHub URL 을 제출하는 사람을 많이 볼 수 없었습니다. GitHub 관리를 했다면 자신의 프로젝트 경력을 증명할 수 있었을 텐데, 그렇지 못한 모습을 보면서 안타깝다는 생각이 들었고, 학생들에게 관리해야 한다는 동기를 유발하고 싶었습니다.
- 최종적으로 성공회대 IT 학부 구성원의 전체적인 발전을 지향합니다.

1-6. 유사 서비스 및 관련 연구

- GitHub 에서는 Follow 라는 기능을 통해 해당 사용자를 즐겨찾기와 같이 추가할 수 있지만, 그 사용자들을 소속 별로 모아서 볼 수는 없습니다.
- GitHub 에서 보여주는 코드의 수는 해당 Repository(저장소)에 올라가 있는 모든 파일(라이브러리 포함)의 줄 수입니다.
- GitHub API 를 활용한 서비스 중 사용자의 주 언어를 분석해주는 기능은 있지만, 의미 있는 코드의 수(라이브러리 제외), Commit 수의 통계를 산출하지는 않습니다.

개발

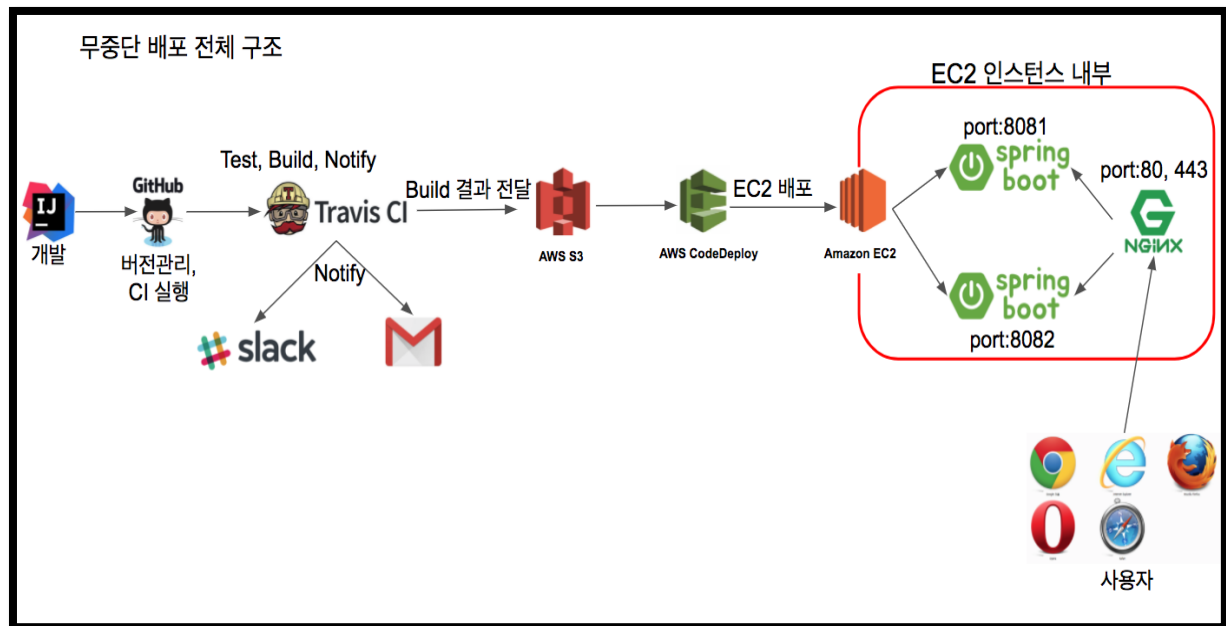
2-1. 개발 일정

- 여름 방학: 아이디어 구상 및 배포를 위한 AWS Credit 수급
- 2018. 08. 06 ~ 2018. 08. 24: 시스템 및 DB 설계
- 2018. 08. 25 ~ 2018. 09. 24: Front-End, Back-End 개발, iOS 개발

2-2. 개발 환경 및 시스템 구성도

- 개발 환경
 - Spring Boot, MyBatis 로 애플리케이션 서버 개발
 - MySQL 로 DB 구축
 - 템플릿 엔진 중 하나인 Handlebars 로 Front 개발
 - GitHub API (kohsuke) 활용
 - Swagger2 로 Rest API 자동 문서화
 - 구글을 통한 gitflow.org 도메인 구입
 - AWS Route 53, RDS, EC2 를 활용한 Deploy
 - (NGINX, Travis CI, AWS S3, Code Deploy 활용 무중단 배포 적용)

- 시스템 구성도

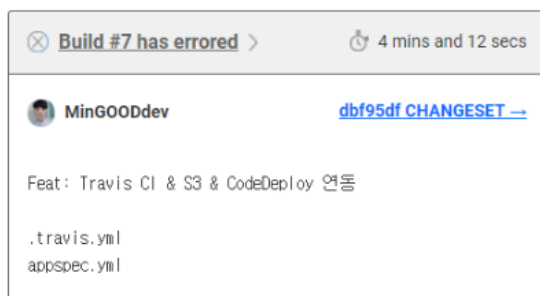


[무중단 배포시 전체 구조]

모든 서버는 AWS 를 통해 서비스됩니다. MySQL 은 AWS RDS 를 통해 배포되고, 애플리케이션 서버는 AWS EC2 를 통해 배포됩니다. 개발 단계에서는 프로젝트 구성원의 Continuous Integration (CI) 경험을 위해 실제 프로젝트에 위 무중단 배포 구조를 적용했습니다. 개발한 후, GitHub Repository master branch 에 Push 하게 되면, Travis CI 에서 해당 Repository 를 Clone 하고, Test, Build, Notify 를 진행하게 됩니다. 저희는 Gmail 로 Notification 을 받았습니다.

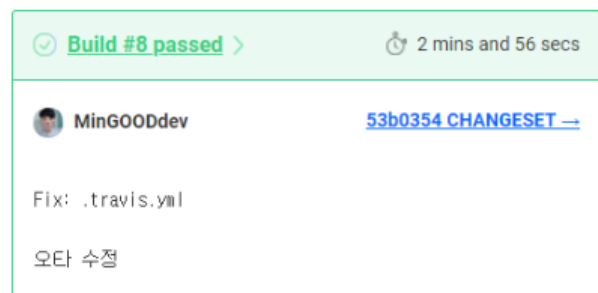
MinGOODdev / Git-Up

master



MinGOODdev / Git-Up

master

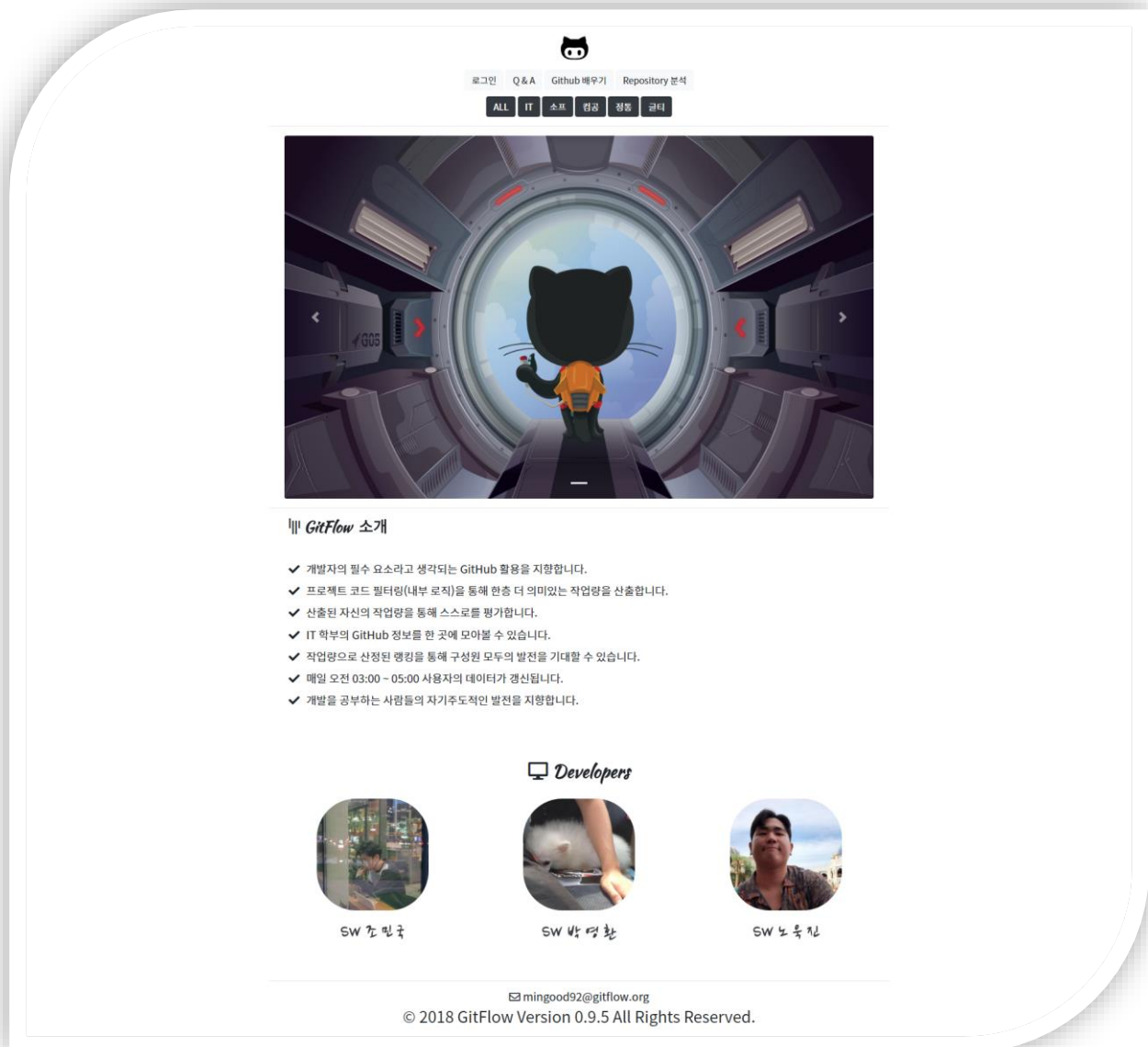


[Travis CI - Gmail Notification Image]

이후, 빌드 결과물은 AWS S3에 저장되고, Code Deploy가 AWS S3를 보고 새로운 버전이 있다면, EC2로의 배포를 진행합니다.

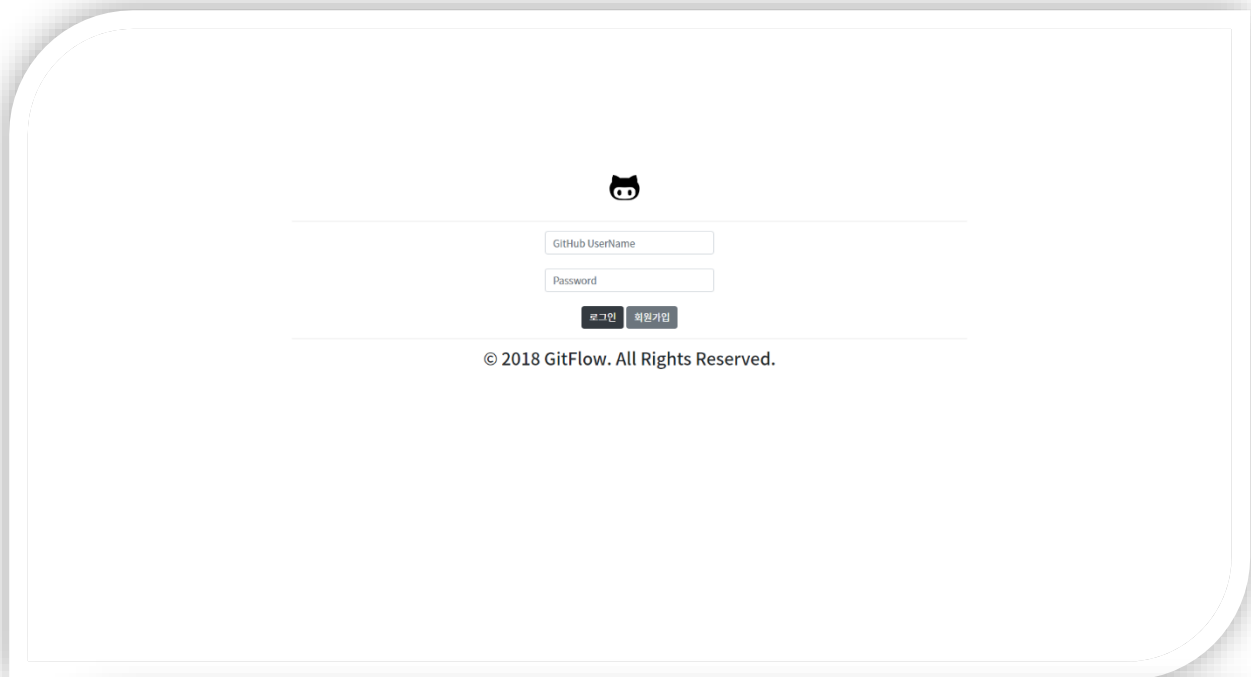
하지만 실제 서비스에서는 CI를 제외합니다. EC2에 새로운 버전의 애플리케이션을 배포하는 것은 Shell Script를 작성해 EC2에 올려놓는 것만으로도 충분히 간편하기 때문입니다. 실제 서비스를 사용하는 사용자가 늘어났을 때, 애플리케이션 서버를 분산하여 무중단 배포를 적용할 것입니다.

2-3. 작품의 기능



[2-3-1. 메인 페이지]

서비스의 첫 화면에는 상단 메뉴바와 하단 서비스 설명이 있습니다. 상단 메뉴바는 ALL 부터 소속별로 접근할 수 있는 버튼이 있습니다. 그 이외에 로그인, 문의사항, Repository 분석 기능으로 갈 수 있는 버튼이 있고, GitHub 을 배울 수 있는 사이트로 이동하는 버튼이 있습니다



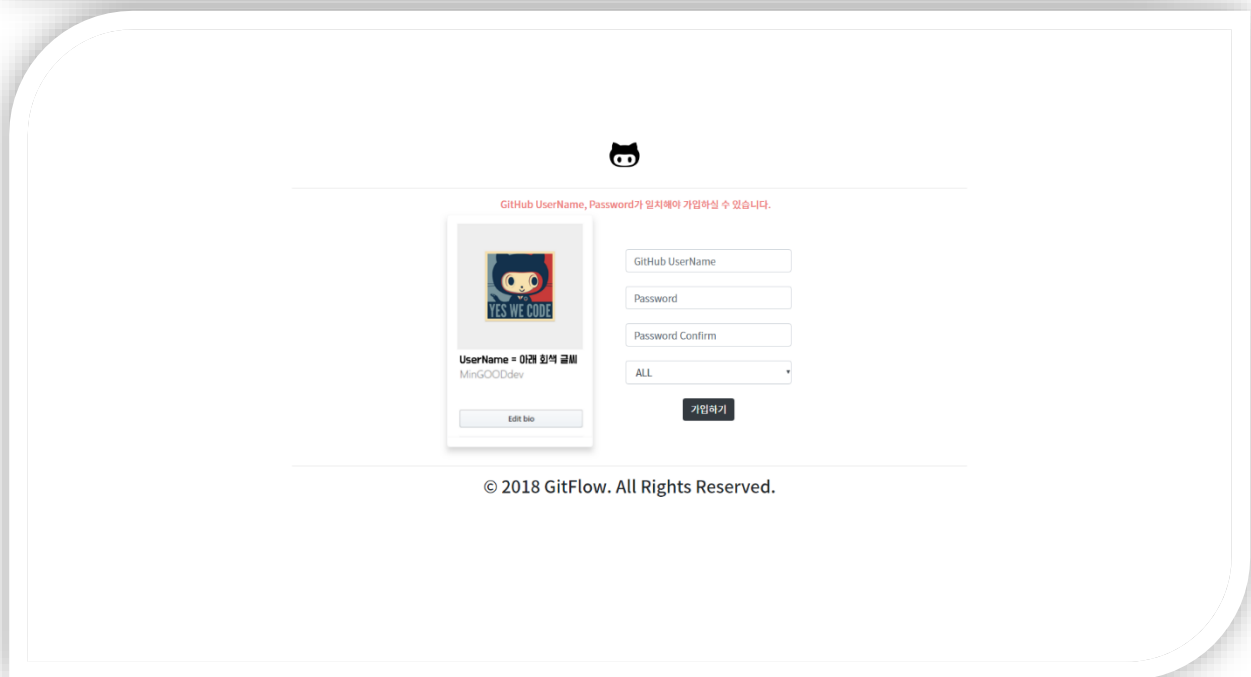
GitHub

GitHub UserName

Password

로그인 회원가입

© 2018 GitFlow. All Rights Reserved.



GitHub

GitHub UserName, Password가 일치해야 가입하실 수 있습니다.

UserName = 아래 형석 굿비
MinG00Ddev

GitHub UserName

Password

Password Confirm

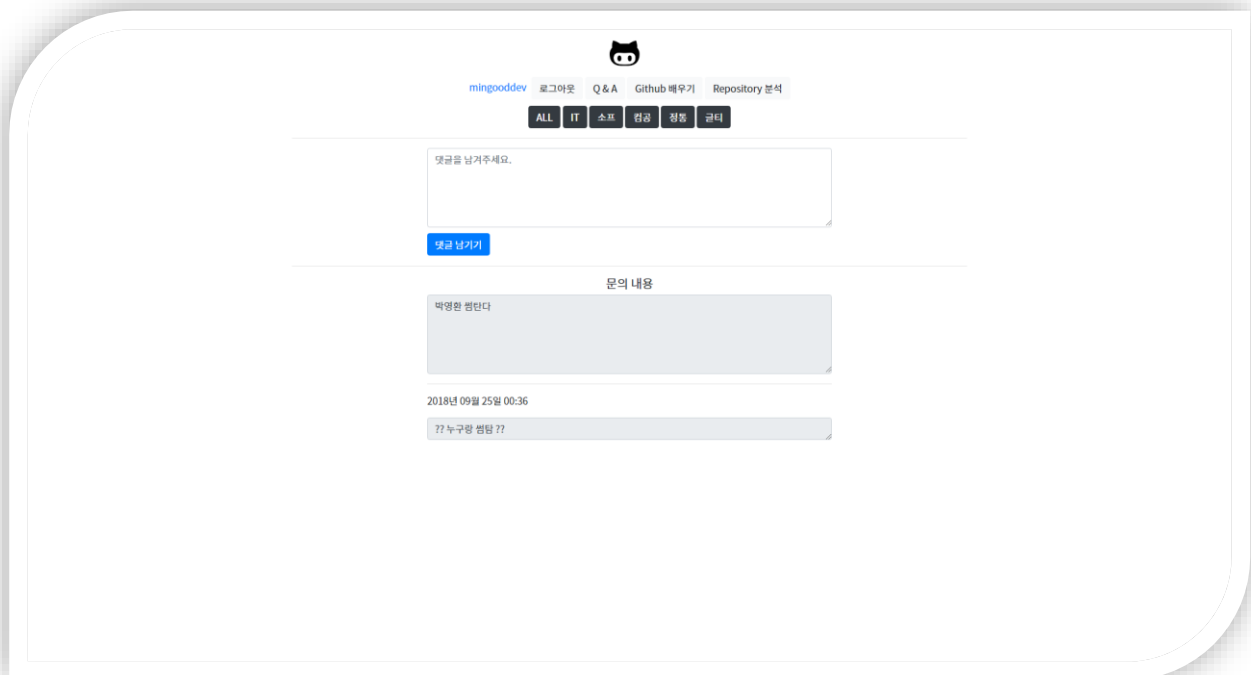
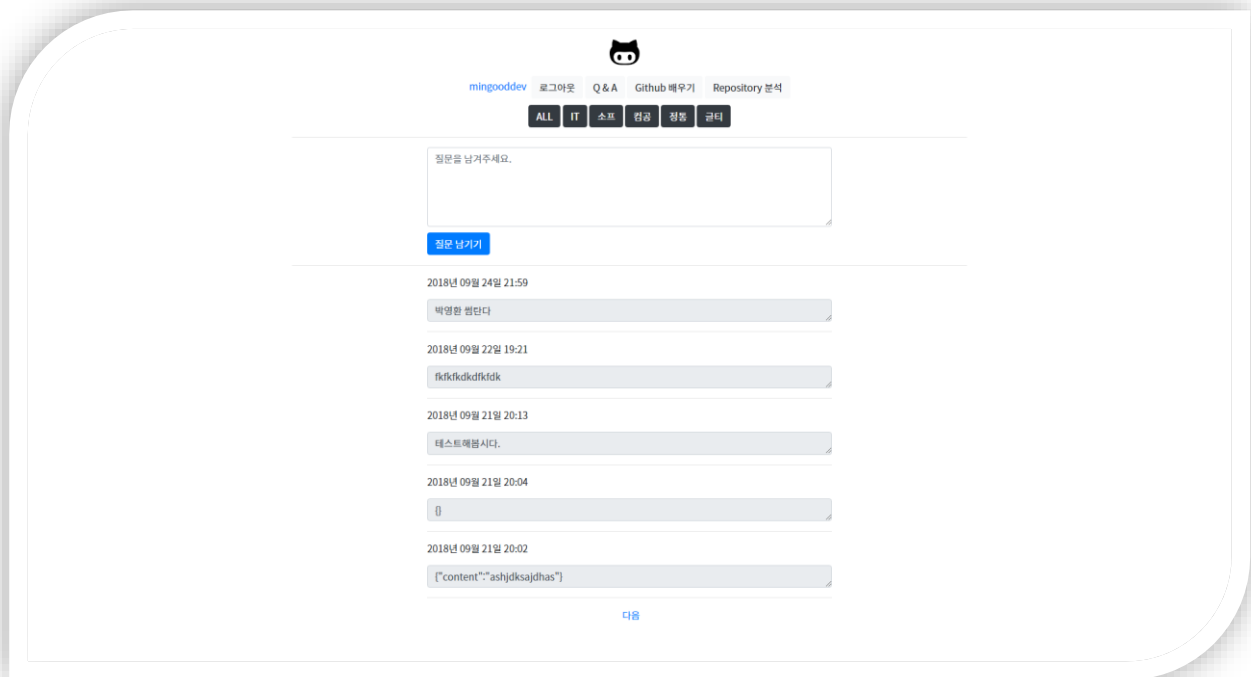
ALL

가입하기

© 2018 GitFlow. All Rights Reserved.


[2-3-2. 로그인과 회원가입]

회원가입은 계정 도용을 방지하기 위해 GitHub 의 UserName 과 Password 까지 일치해야 가능하므로 회원가입 View 에서 안내 메시지를 띄우고 있습니다.



[2-3-3. 문의 사항과 문의 사항 댓글]

궁금한 점을 거리낌 없이 올릴 수 있도록 익명의 문의 사항 게시판이 있고, 그 문의 사항에 대한 댓글 또한 익명으로 달 수 있습니다


[mingooddev](#)
[로그아웃](#)
[Q & A](#)
[Github 배우기](#)
[Repository 분석](#)

ALL

IT

소프














컴공

정통

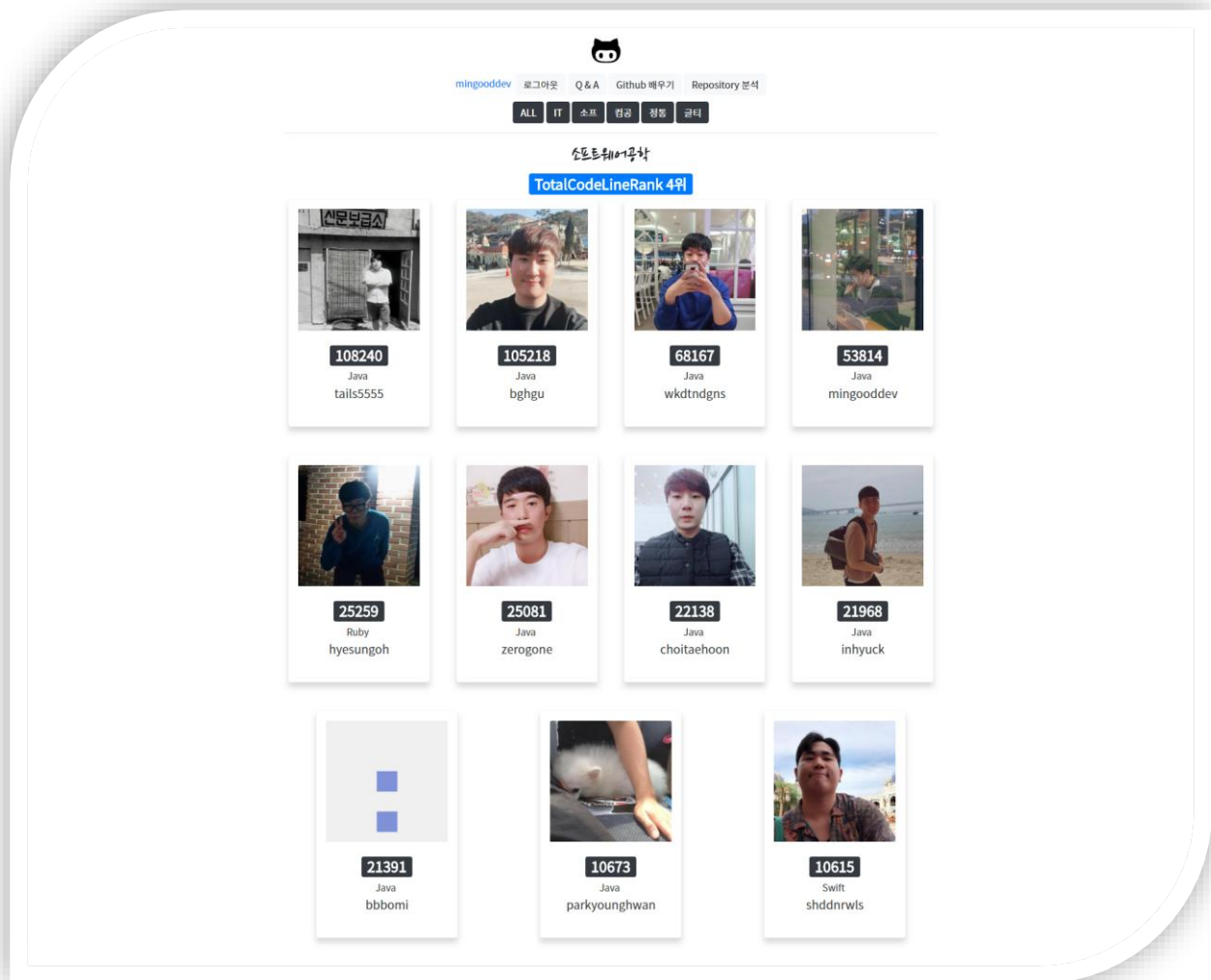
글타

ALL

TotalCodeLineRank 4위

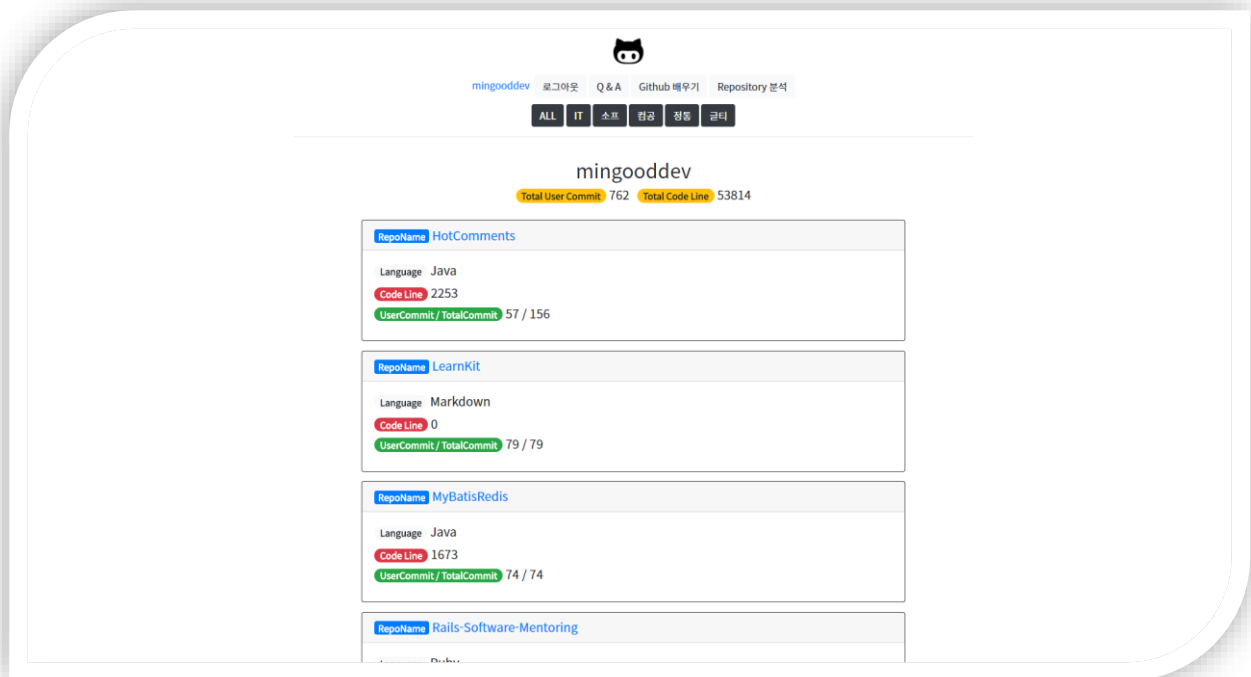
 <div> 108240 Java 소프트웨어공학 tails5555 </div>	 <div> 105218 Java 소프트웨어공학 bghgu </div>	 <div> 68167 Java 소프트웨어공학 wkdtndgns </div>	 <div> 53814 Java 소프트웨어공학 mingooddev </div>
 <div> 25259 Ruby 소프트웨어공학 hyesungoh </div>	 <div> 25081 Java 소프트웨어공학 zerogone </div>	 <div> 22138 Java 소프트웨어공학 choitaehoon </div>	 <div> 21968 Java 소프트웨어공학 inhyuck </div>
 <div> 21391 Java 소프트웨어공학 bbbomi </div>	 <div> 16207 Ruby IT학부 hanhyunsoo </div>	 <div> 12069 Ruby IT학부 kyeul </div>	 <div> 10673 Java 소프트웨어공학 parkyoungwan </div>
 <div> 10615 Swift 소프트웨어공학 shddnrwls </div>			

[2-3-4-1. 전체 랭킹]



[2-3-4-2. 본인 소속 랭킹]

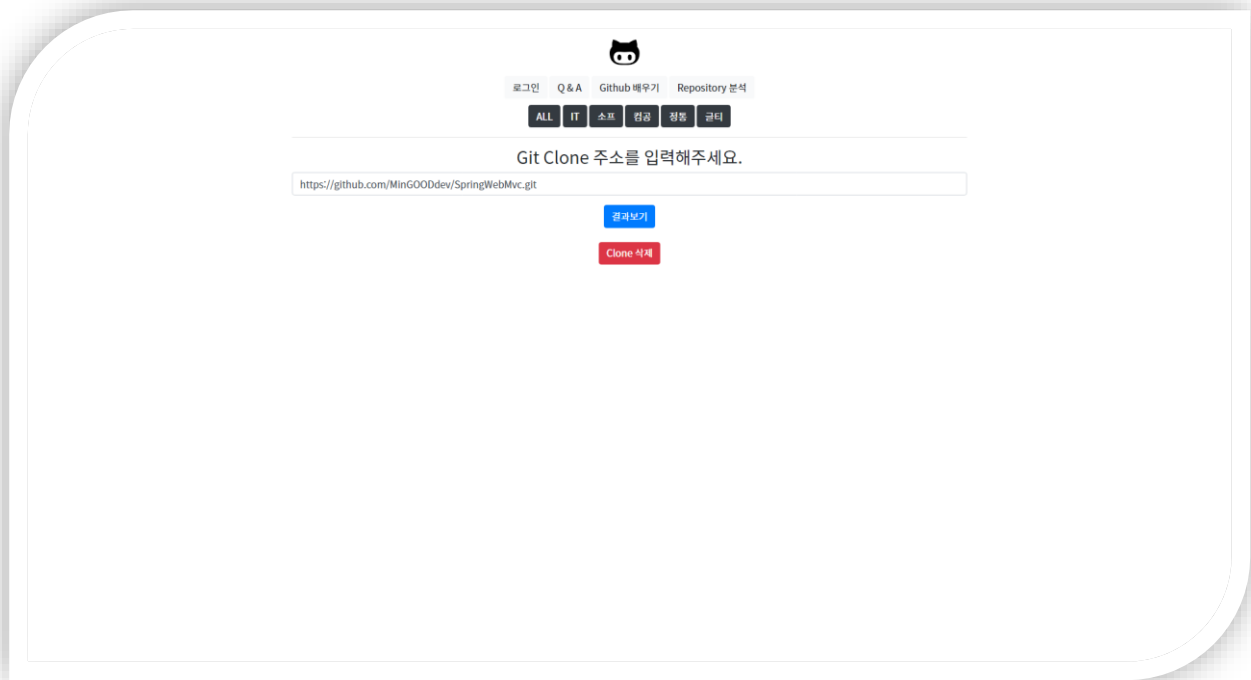
ALL 과 소속 별로 사용자 목록을 조회할 수 있습니다. 이때 라이브러리와 같은 코드를 제외하고, 본인이 구현한 라인 수의 총합으로 산출된 순위 순으로 보이게 됩니다.



[2-3-5. 사용자의 세부 정보]

위 전체 순위 또는 소속 순위를 볼 수 있는 화면에서 A 사용자를 클릭할 경우 볼 수 있는 View 입니다. 이 View에서는 A 사용자의 전체 Commit 수와 의미 있는 전체 라인 수를 상단에서 확인할 수 있습니다. 그 아래로는 A 사용자의 Repository 목록을 조회할 수 있는데, Repository 마다 가장 많이 사용된 언어, 의미 있는 라인 수를 확인할 수 있습니다. 또, 그 Repository의 전체 Commit 수와 A 사용자의 Commit 수를 볼 수 있습니다.

GitFlow 에 등록된 모든 사용자의 정보들은 매일 새벽 설정된 스케줄러를 통해 자동으로 갱신됩니다.



[2-3-6. Repository 분석]

GitFlow 의 기본적인 기능은 전체 사용자의 GitHub 정보를 한눈에 모아서 보여주고, 순위를 산출하여 선의의 경쟁을 유도하여 자기 주도적인 발전을 지향하는 것이었습니다.

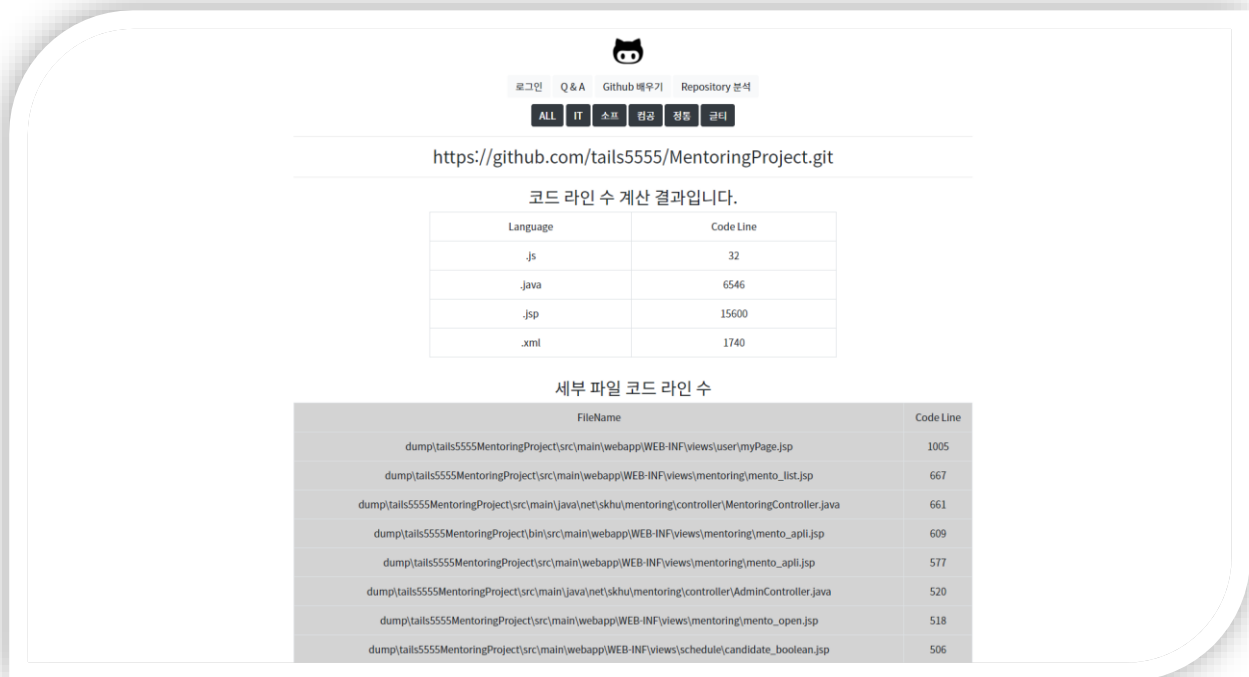
지금부터 말씀드릴 Repository 분석 기능은 GitFlow 가 산출하는 데이터가 어떻게 나오는지, 정말 정확한 것인지에 대해 확인할 수 있습니다. 사용자가 원하는 Repository 를 분석한 결과를 실제로 확인함으로써 GitFlow 의 신뢰도를 높일 수 있습니다.



[2-3-7. Repository 분석 결과]

Repository 를 분석한 결과 View 입니다. 해당 Repository 의 의미 있는 코드 라인 수를 계산합니다. 그 코드 라인 수는 세분되어 확장자 별로 그 수치를 산출합니다. 그리고 GitFlow 가 어떤 파일들을 읽었는지 사용자도 확인할 수 있도록 아래 GitFlow 가 읽은 기록을 보여주게 됩니다. 라이브러리와 같은 파일이 등록되어 있지 않아 Filtering 이 되지 않았을 경우를 생각하여 코드 라인 수의 역순으로 로그가 보이게 됩니다. 이를 통해 제외해야 하는 코드들의 데이터를 축적할 수 있습니다.

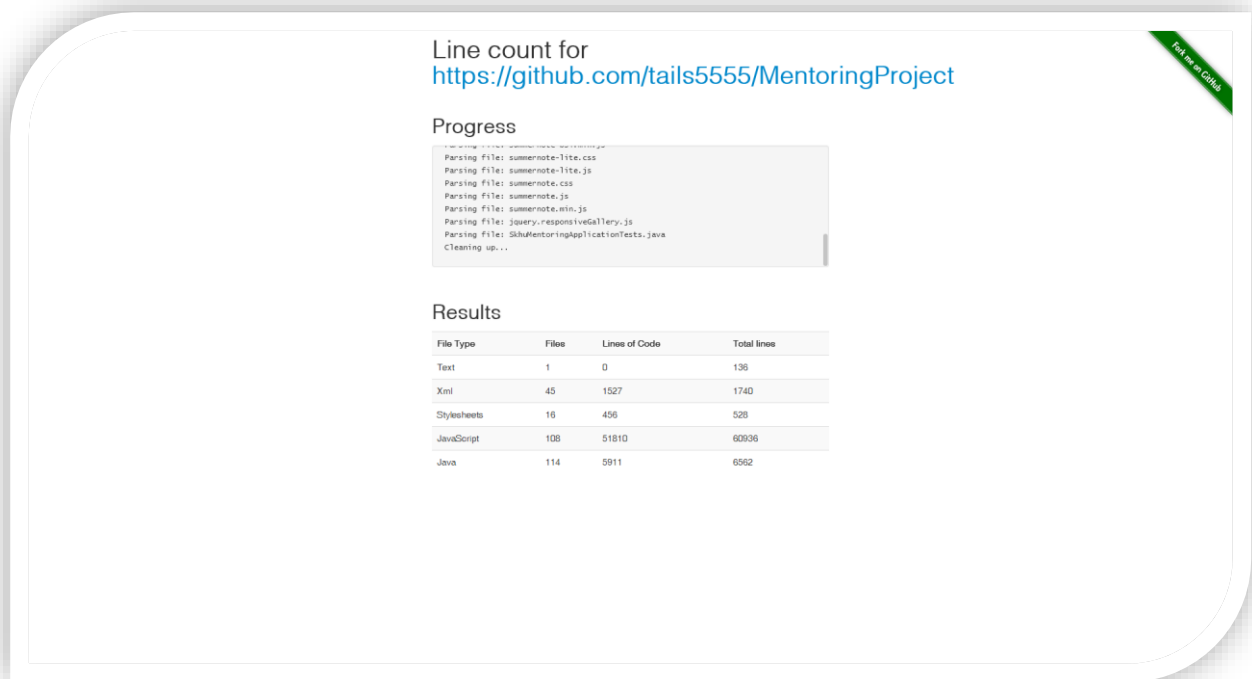
다음으로는 GitFlow 의 라이브러리를 제외한 분석 결과와 다른 GitHub API 활용 플랫폼에서 라이브러리를 포함한 결과를 비교해보도록 하겠습니다.



[2-3-8. 라이브러리를 제외한 Repository 분석 결과]

GitFlow 의 분석 결과 Javascript 32 줄, Java 6,546 줄, JSP 15,600 줄, XML 1,740 줄의 결과가 산출된 것을 확인할 수 있습니다. 확장자 별 라인 수가 어떻게 산출되었는지 아래 로그를 통해 확인할 수 있습니다. 이 결과를 통해 GitFlow 분석 결과를 신뢰할 수 있습니다.

다음으로는 라이브러리를 제외하지 않은 결과를 보도록 하겠습니다.



[2-3-9. 라이브러리를 포함한 Repository 분석 결과]

위 오픈 소스의 분석 결과, Text 136 줄, XML 1,740 줄, CSS 528 줄, Javascript 60,936 줄, Java 6,562 줄의 결과가 산출된 것을 확인할 수 있습니다. 위 오픈 소스는 GitFlow 와 다르게 View 파일에 대한 코드 라인 수는 산출하지 않는 것을 확인할 수 있습니다.

또한, jQuery 와 같은 라이브러리를 제외하지 않았기 때문에 Javascript 의 라인 수가 약 60,000 줄로 굉장히 높은 수치가 산출된 것을 확인할 수 있습니다.

위 프로젝트는 Spring Boot 을 활용한 것인데, 분석 결과 Java 라인 수의 차이가 있는 이유는 GitFlow 는 Spring 프로젝트를 생성했을 때 최초로 만들어지는 ApplicationTests 키워드가 붙은 파일은 제외하고 결과를 산출하기 때문입니다. (정확히 16 줄 차이가 납니다.)

위 비교를 통해 GitFlow 가 기존의 GitHub API 를 활용한 오픈 소스보다 한층 더 세부적인 결과를 산출한다는 것을 알 수 있습니다.

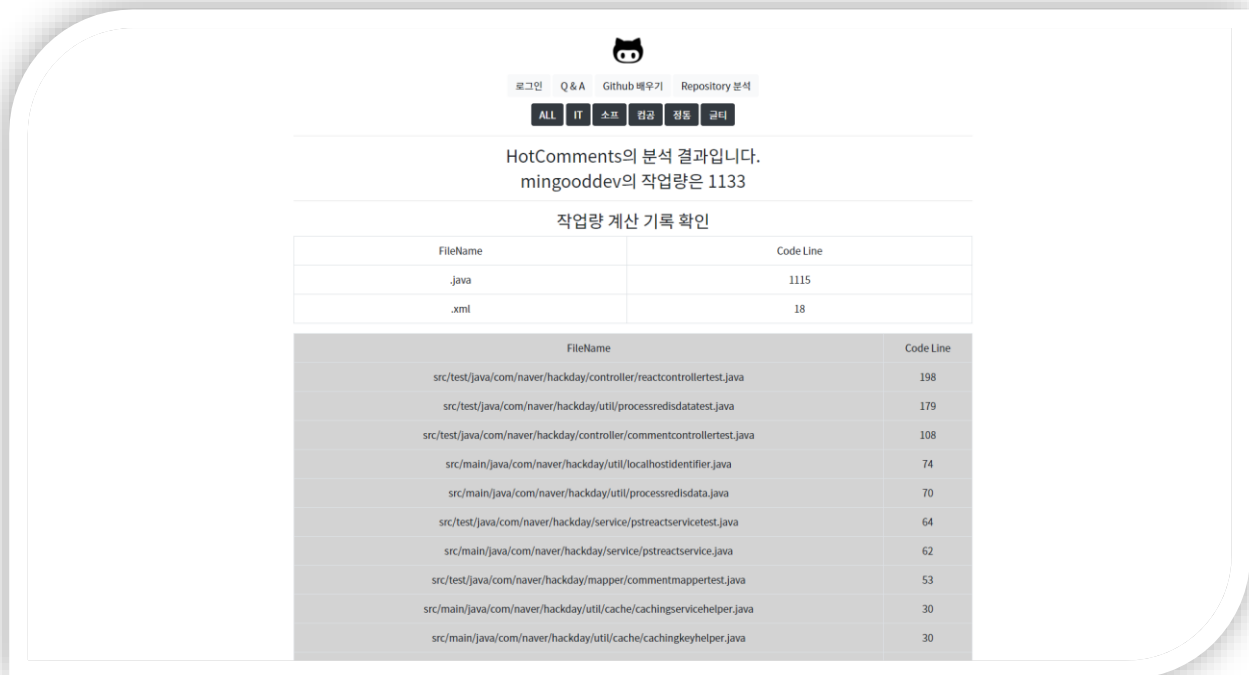
[2-3-10. 협업 Repository 분석]

GitFlow 는 해당 Repository 가 협업으로 이루어져 있는 경우도 생각했습니다. 그 협업 Repository 를 분석하고 싶은 경우, Repository 의 소유자(repoOwner), Repository 의 이름(repoName)과 작업량을 알고 싶은 사용자의 UserName(committer)을 입력하면 됩니다.

이 협업 Repository 분석 기능은 협업한 기록을 분류하여 해당 사용자의 작업량만을 산출해야 하므로 GitHub API 를 활용하여 구현되어 있습니다. 그러므로 사용자가 GitHub 을 제대로 활용하지 못하여 Commit 계정 설정을 본인으로 하지 않은 경우, Committer 를 찾을 수 없어 정확한 결과를 얻지 못합니다. 하지만 이 문제는 GitFlow 의 Logic 문제라기보다는 사용자의 GitHub 활용 방법의 문제이기 때문에 GitHub 의 정확한 활용 방법을 통해 극복할 수 있습니다.

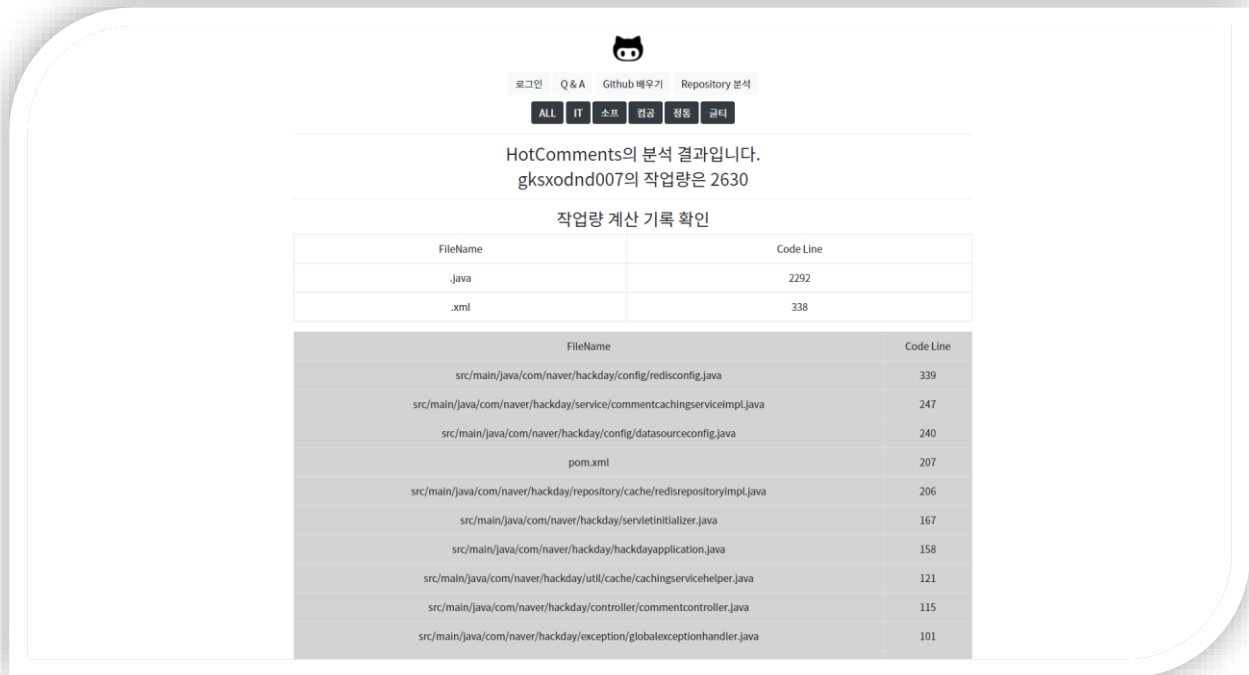
아래 협업 Repository 분석 결과를 확인해보도록 하겠습니다. 이 협업 Repository 의 committer 로는 크게 mingooddev, gksxodnd007, masssidev 3 명이 존재합니다. 원래 Repository 분석 기능과 마찬가지로 읽어 들인 결과 로그를 보여줍니다.

다음은 mingooddev 의 작업량 분석 결과입니다.

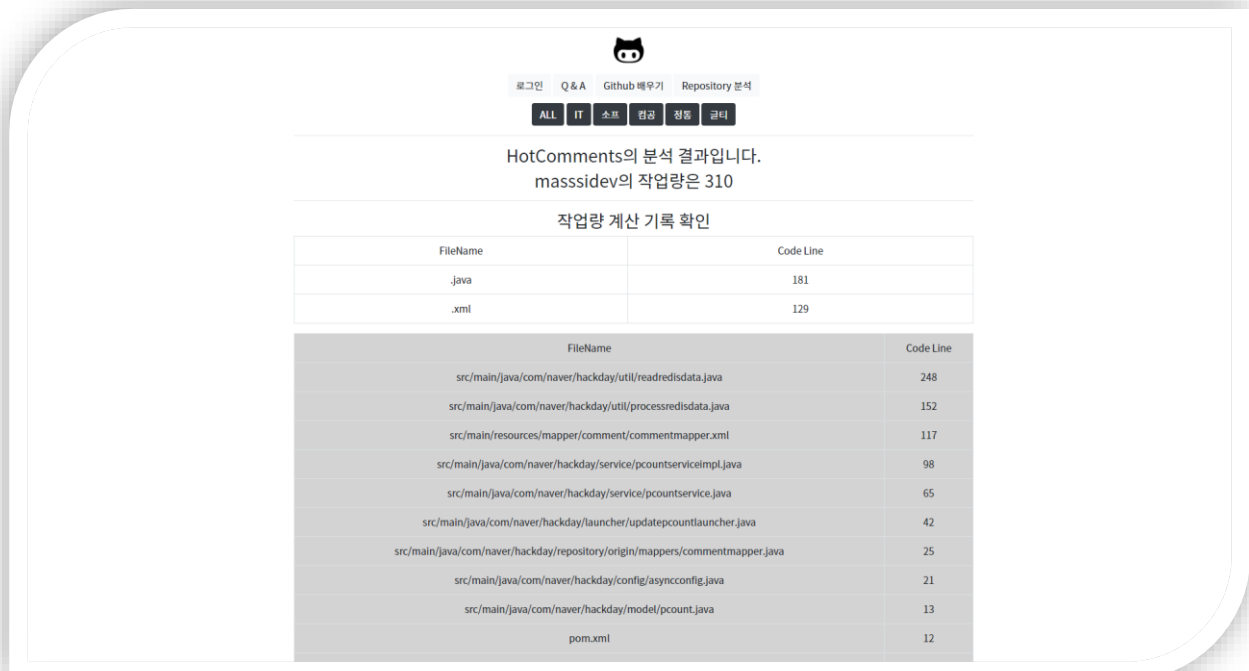


[2-3-11. 협업 Repository 분석 결과 mingooddev]

다음은 gksxodnd007, masssidev 의 작업량 분석 결과입니다.



[2-3-12. 협업 Repository 분석 결과 gksxodnd007]



[2-3-13. 협업 Repository 분석 결과 masssidev]

2-4. 개발 경과 및 완성도

- GitFlow 의 개발은 본래 이루고자 했던 기능들의 구현이 모두 완료되어 있습니다.
- GitFlow 는 실제 서비스를 목표로 하고 개발한 서비스이기 때문에 <https://www.gitflow.org> 라는 도메인을 구매하여 현재 어디서나 접속이 가능한 상태입니다.
- 앞으로 IT 학부 학생들의 GitFlow 를 통한 소통을 위해 메시지를 보내고 확인할 수 있는 기능을 추가할 계획입니다.
- 결론적으로 서비스의 핵심 기능은 모두 구현 완료되었다고 말할 수 있습니다.

2-5. GitFlow Repository Analysis & Scheduling Logic

- *Repository Analysis Logic (Input: GitUrl)*

첫 번째, 데이터베이스에 4 가지의 Table 이 존재합니다.

- Include: 읽어야 하는 확장자 목록
- Exclude: 읽지 말아야 하는 확장자 목록
- IgnorePath: 읽을 필요 없는 경로 목록
- MustContain: 읽지 말아야 하는 확장자에 속하지만, 결과를 산출할 때 포함했으면 하는 것들

두 번째, 확장자 별 라인 수, 파일을 읽은 로그를 저장할 2 개의 Map 을 생성하고, clone이 저장될 폴더명을 지정하기 위해 입력받은 Git URL 을 String Class 를 활용해 *UserNameRepoName* 의 형태로 변환합니다.

세 번째, 위 4 가지 DB Table 로 Boolean 결과 값을 만들고 이 결과를 &&, || 연산으로 조합하여 git clone 을 실행합니다.

네 번째, 파일을 읽으면서 위 2 개의 Map 에 데이터를 채우고, 라인 수를 계산합니다.

다섯 번째, 계산된 결과를 보여줍니다. (logMap 은 Comparator 를 활용 -> DESC)

- *Collaboration Repository Analysis Logic* (Input: repoOwner, repoName, committer)

마찬가지로, 위 *Repository Analysis Logic* 의 4 가지 DB Table 을 활용합니다.

첫 번째, 입력받은 repoOwner 와 repoName 으로 GitHub API 를 활용해 Repository 를 검색합니다. 그리고 확장자 별 라인 수, 파일을 읽은 로그를 저장할 2 개의 Map 을 생성합니다.

두 번째, GitHub API 를 활용해 Commit List 에서 committer 가 입력받은 committer 와 일치하는 Commit 을 검색합니다.

세 번째, 4 가지 DB Table 로 Boolean 결과 값을 만들고 이 결과를 &&, || 연산으로 조합하여 읽어야 하는 File 만 읽습니다. (Merge 내역은 제외됩니다.)

네 번째, 세 번째 단계를 거치면서 확장자 별 라인 수와 로그 기록을 Map 에 저장합니다.

다섯 번째, 계산된 결과를 보여줍니다. (logMap 은 Comparator 를 활용 -> DESC)

- *Scheduling Logic*

스케줄링 로직은 크게 2 가지로 분류할 수 있습니다.

- *notExistRepoDataRemove Scheduler.*

사용자가 기존의 Repository 이름을 변경 또는 삭제한 경우를 생각하여, 데이터를 갱신하기 전, 존재하지 않는 DB 데이터를 삭제합니다.

이 Scheduler Logic 은 다음과 같습니다.

첫 번째, DB 에 저장된 User List 를 가져옵니다.

두 번째, DB 에서 userId 에 해당하는 Repo List 와 GitHub API 를 활용해 userId 에 해당하는 실제 GitHub Repo List 를 가져옵니다.

세 번째, 두 번째의 2 가지 List 를 비교하기 위해 DB 에서 가져온 Repo List 를 parameter 로 받아 value 값을 전부 false(Boolean)로 하는 Map(compareMap)을 생성합니다.

네 번째, compareMap 의 데이터와 GitHub 에서 가져온 실제 Repo List 를 비교하여 존재하면 value 값을 true 로 수정합니다.

다섯 번째, compareMap 에서 value 가 false 인 데이터를 DB 에서 삭제합니다.

■ *repoAndGitUserDataUpdate Scheduler*

*notExistRepoDataRemove Scheduler*가 완료된 후, 본격적으로 사용자의 데이터를 갱신합니다.

이 Scheduler Logic 은 다음과 같습니다.

첫 번째, DB 에 저장된 User List 를 가져옵니다.

두 번째, GitHub API 를 활용해 userId 에 해당하는 실제 GitHub Repo List 를 가져옵니다. 그리고 변동 사항이 있는 Repo 를 저장할 *changedRepoList* 를 생성합니다.

세 번째, 두 번째에서 만들어진 GitHub Repo List 를 순회하면서 GitHub Repo 와 DB 상의 Repo 데이터를 각각 검색합니다.

네 번째, 세 번째에서 검색한 2 가지 Repo 의 commit 값을 비교하여 서로 다를 경우 DB 의 Repo 데이터를 갱신합니다. (DB 상의 repo 가 없는 경우 INSERT, 있는 경우 UPDATE) 그리고 변동 사항이 있는 Repo 를 *changedRepoList* 에 등록합니다.

다섯 번째, 일단 갱신된 DB 상의 Repo 데이터(Repo List)로 userId 에 해당하는 사용자의 주 언어 값을 산출합니다. (Map(languageMap)을 생성해 Repo List 를 순회하며 언어별로 count 를 증가시켜 최대치를 가져옵니다.)

여섯 번째, 변동이 있는 *changedRepoList* 를 순회하며, git clone 으로 파일을 읽어 코드 라인 수를 갱신합니다.

일곱 번째, 이제 DB 의 Repo 데이터가 모두 갱신되었으므로 이것을 토대로 User 의 전체 Commit 수와 전체 라인 수, 주 언어를 업데이트합니다.

여덟 번째, 데이터 갱신이 종료되었으므로, git clone 으로 생성된 파일들을 삭제합니다.

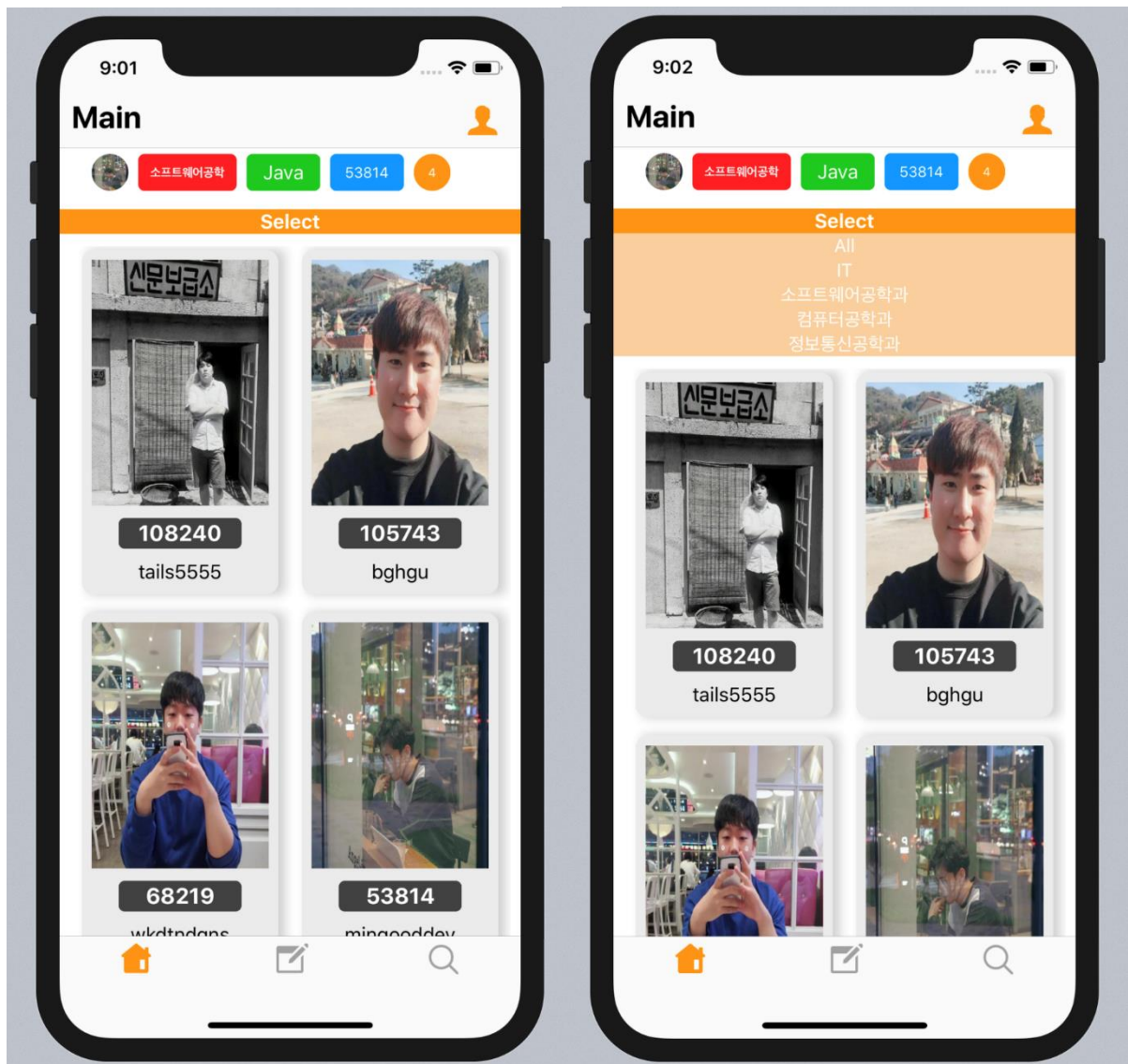
iOS 개발



[최초 실행, 로그인]

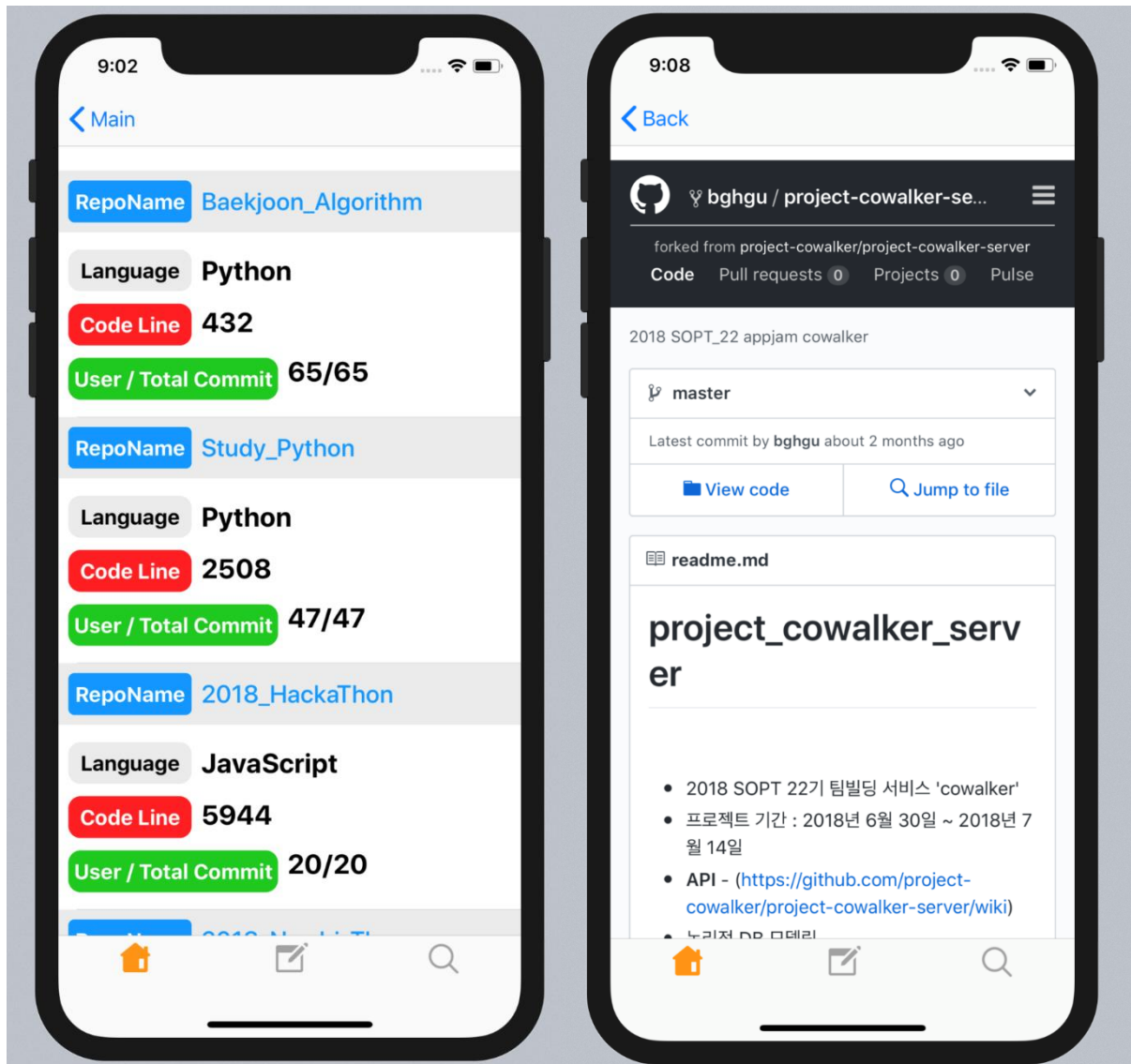
App 을 최초로 실행할 경우 가장 먼저 로그인, 회원가입으로 이동할 수 있는 Intro View 가 보입니다. 만약 로그인 후, 로그아웃을 하지 않는다면 자동 로그인 처리되어 Intro View 는 보이지 않습니다.

아이디와 비밀번호가 틀릴 경우, Main View 로 갈 수 없고 어떤 내용이 틀렸는지 메시지를 통해 알려줍니다.



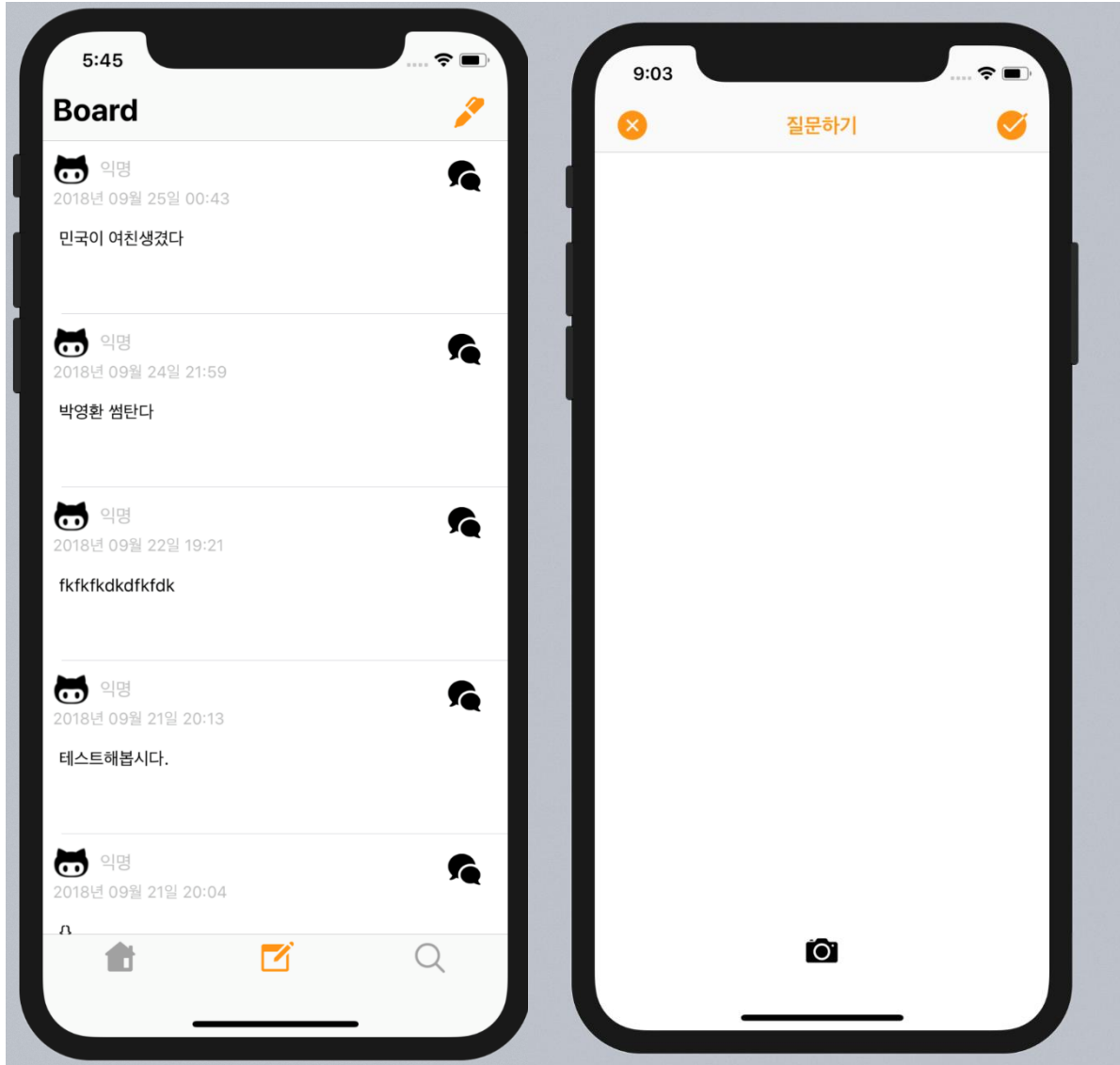
[Main View]

로그인 성공 시, 메인 화면으로 넘어옵니다. 메인 화면 상단에는 현재 로그인 계정 정보와 카테고리 별 등수가 보이고, Drop Down Menu 를 통해 보고 싶은 소속을 선택할 수 있습니다. 상단 네비게이션 바의 버튼으로 로그아웃을 할 수 있습니다.



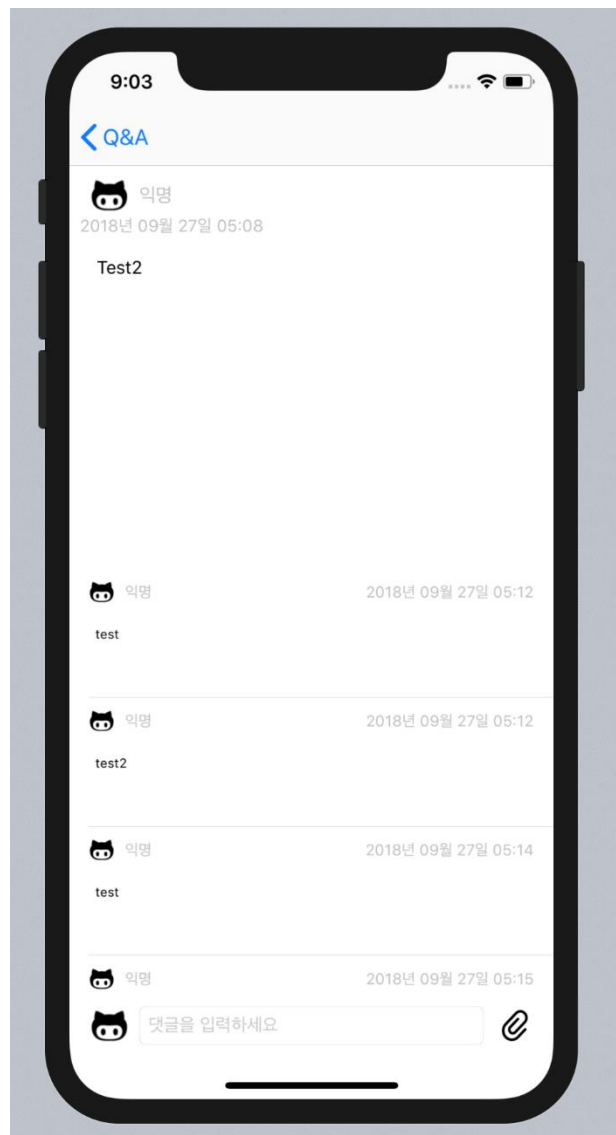
[Repository List View]

사용자의 Card 를 터치할 경우, 그 사용자의 Git Repository List 를 가져오고, Repository 의 정보들을 보여줍니다. 원하는 Repository 를 선택할 시 GitHub Page 를 불러와 더 자세한 내용을 볼 수 있습니다.



[문의 사항(Q&A) 게시판]

Q&A 익명 게시판입니다. 네비게이션 바에 버튼을 사용해 글을 작성할 수 있으며, 내용과 작성일을 확인할 수 있습니다. 또한, 원하는 게시글을 터치할 경우 댓글을 볼 수 있습니다.



[문의 사항 댓글]

보고 싶은 문의 사항을 선택하면 댓글을 볼 수 있는 화면이 나옵니다. 아래 댓글 입력 창을 통해 실시간 댓글을 작성할 수 있으며, 댓글 목록을 확인할 수 있습니다.

- iOS 개발에 사용된 라이브러리

- Alamofire
- SwiftyJSON
- RxSwift

- 디자인 패턴 및 아키텍처

- Singleton Design Pattern

- ◆ NetworkManager, NetworkProtocol 을 구현하여 네트워킹 관련 로직은 전부 한 곳에서 처리했으며, App 자체 성능을 위해 Background Thread 에서 통신했습니다.

- MVVM Architecture

- ◆ MVC 패턴에서 Controller 를 제외하고 View, Model 을 추가한 패턴입니다. 여기서 Swift 의 ViewController 가 View 가 되고, ViewModel 이 Controller 와 View 의 중간 다리 역할을 하게 됩니다.
- ◆ View 와 ViewModel 사이에 Binding 이 있어 Model 에 의한 ViewModel 의 변화가 있으면 Binding 으로 인해 View 도 업데이트됩니다. ViewModel 은 View 에 대해 아무것도 모르기 때문에 테스트가 쉽고 코드 라인 수가 상당히 줄어듭니다.

참고 문헌

[github api]

<http://github-api.kohsuke.org/>

[handlebars]

<https://handlebars-java-helpers.beryx.org/releases/latest/#introduction>

<https://allegro.tech/2015/04/spring-boot-starter-handlebars.html>

[Spring-Security]

<https://docs.spring.io/spring-security/site/docs/5.1.0.RELEASE/reference/htmlsingle/>

[MyBatis]

<http://www.mybatis.org/spring/ko/getting-started.html>

[Scheduler]

https://spring.io/guides/gs/scheduling-tasks/#_how_to_complete_this_guide

<https://docs.spring.io/spring/docs/current/javadoc-api/org.springframework.scheduling.support/CronSequenceGenerator.html>

[SSL]

<https://superuser.com/questions/1072408/how-to-install-lets-encrypt-on-amazon-linux>

[swagger]

<https://swagger.io/tools/swagger-ui/>

[travis-ci]

<https://travis-ci.org/>

[maven]

<https://mvnrepository.com/>

[Swift]

<https://developer.apple.com/kr/swift/>

[Alamofire]

<https://github.com/Alamofire/Alamofire>

[RxSwift]

<https://github.com/ReactiveX/RxSwift>

[MVVM pattern]

<https://academy.realm.io/kr/posts/doios-natasha-murashev-protocol-oriented-mvvm/>