**1. Git Flow 가이드**

Git Flow는 ‘Git에서 제공하는 브랜치 기능을 활용한 히스토리 관리 전략’ 중 하나다. Git Flow 이외에도 많은 히스토리 관리 전략들이 있으며, Git Flow도 개발팀마다의 사정에 맞게 권장된 방법 이외의 커스터마이징을 하여 활용되고 있다. 본 문서에서는 널리 권장되는 Git Flow 전략을 기반으로 해당 내용을 설명할 것이다.

**1-1. 개요**

Git Flow의 핵심은 ‘브랜치 별로 각각의 브랜치가 프로젝트 완수를 위해 기능하는 역할을 분배하여, 그 브랜치가 그 역할을 수행하게끔 한다’는 것이다. 따라서 프로젝트 완수를 위해 ‘배포’와 ‘기능 개발’은 필수불가결하므로, 해당 역할을 하는 브랜치는 반드시 존재하게 된다. 대개는

- master(or main) : ‘배포 브랜치’

- develop : 기능 개발 브랜치

로 브랜치를 명명하게 된다.

하지만 기능 개발 중 develop이라는 단일 브랜치에서 여러 개발자가 작업하게 되면 병합 충돌 등의 우려 사항이 걱정된다. 따라서 이를 피하기 위해, 각 기능의 개발을 위한 브랜치를 만든다. 예를 들어, 로그인 기능을 개발하기 위한 브랜치는

- feature/login : 로그인 기능 개발 브랜치

로 명명할 수 있다. 이러한 브랜치들은 생성 시점의 develop 브랜치를 클론하여 생성되고 로그인 기능 개발 완수, 즉 역할을 다할 때 해당 브랜치를 develop에 머지하고 삭제한다.

하지만 기능 개발을 완수했다고 해서, 해당 기능이 완벽히 동작하는 것은 아니다. QA 중 오류가 발생할 수 있고, 그 오류를 수정하기 위한 코드 작성을 위한 브랜치가 필요하다. 그러한 브랜치는

- bugfix/login : 로그인 기능 에러 수정 브랜치

로 명명할 수 있다. 이러한 브랜치 역시 feature 브랜치와 동일하게 역할을 수행 후, develop에 머지하고 삭제한다.

QA를 잘 해냈다면 해당 서비스가 배포되기 이전에 오류를 다 잡아낼 수 있겠지만, 배포 후에 발견된 오류가 있거나 배포 후 갑작스런 기획 변경 등으로 배포 버전을 기준으로 수정해야 할 사항이 생길 수 있다. 예를 들어, 배포 버전이 1.0일 경우,

- hotfix/1.0 : 배포 버전 1.0 긴급 수정 브랜치

로 명명할 수 있다.

혹은 hotfix와는 비슷하지만, “긴급한” 수정 사항의 반영은 아니지만, “필요하면서도 작은” 수정 사항의 반영을 위해서는

- support/1.0.1 : 배포 버전 1.0의 마이너한 추가 수정 사항 반영 브랜치

를 생성할 수도 있다.

마지막으로, 현 배포 버전이 1.0이라면 1.1 혹은 2.0 기능 개발을 위해 feature 브랜치를 통합한 develop 브랜치를 기준으로 QA를 할 브랜치가 필요할 수 있다. 예를 들어, 1.1 버전으로 배포 이전 최종 단계 브랜치를 생성하고 싶다면,

- release/1.1 : 배포 이전 최종 검수를 위한 1.1 버전 브랜치

로 명명할 수 있다.

**1-2. 시작하기**

Project Repository를 클론 받은 후, 다음의 명령어로 Git Flow를 시작할 수 있다.

- git flow init

그 후, 다음과 같은 질문이 터미널에 노출된다. 배포용 브랜치가 무엇인지를 물어보는 것으로, 기본적으로 master를 세팅해준다. Enter를 누르는 것을 권장한다.

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

이후, 배포 이전 release를 위한 기능 개발 통합 브랜치를 물어본다. 기본적으로 develop을 세팅해주기 때문에, Enter를 누르는 것을 권장한다.

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

이후, 앞서 설명했던 Git Flow에서 권장되는 브랜치 역할을 명명하여야 하는데, 기본적으로 모두 권장되는 명명을 세팅해주기 때문에, Enter를 누르는 것을 권장한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

**1-3. feature**

로그인 기능을 개발하기 위한 브랜치 생성은

- git flow feature start login

이라는 명령어로 생성할 수 있다. 현재 어느 브랜치에 있었든, develop 브랜치를 기준으로 feature/login 브랜치를 새로 생성하고, 해당 브랜치로 체크아웃한다.

기능 개발이 완료된다면

- git flow feature finish login

이라는 명령어를 사용한다. 해당 브랜치가 develop에 머지되고, develop 브랜치로 체크아웃한다. 해당 브랜치는 로컬 및 원격 브랜치 모음에서 전부 삭제된다.

**1-4. bugfix**

기능 개발이 완료되어 develop에 머지된 로그인 기능에 에러가 확인되어 수정하기 위한 브랜치 생성은

- git flow bugfix start login

이라는 명령어로 가능하다. 현재 어느 브랜치에 있었든, develop 브랜치를 기준으로 bugfix/login 브랜치를 새로 생성하고, 해당 브랜치로 체크아웃한다.

에러 수정이 완료된다면

- git flow bugfix finish login

이라는 명령어를 사용한다. 해당 브랜치가 develop에 머지되고, develop 브랜치로 체크아웃한다. 해당 브랜치는 로컬 및 원격 브랜치 모음에서 전부 삭제된다.

**1-5. hotfix**

배포 중인 버전 1.0 서비스에서 에러가 확인되어 긴급한 수정이 필요한 경우,

- git flow hotfix start 1.0

이라는 명령어를 사용하여 hotfix/1.0 브랜치를 생성할 수 있다. 현재 어느 브랜치에 있었든 master 브랜치를 기준으로 hotfix/1.0 브랜치를 새로 생성하고, 해당 브랜치로 체크아웃한다.

에러 수정이 완료된다면

- git flow hotfix finish 1.0

이라는 명령어를 사용한다. 해당 브랜치가 master에 머지되고, master 브랜치로 체크아웃한다. 해당 브랜치는 로컬 및 원격 브랜치 모음에서 전부 삭제된다.

**1-6. release**

1.1 버전 배포 이전 기능 테스트를 확인하기 위해

- git flow release start 1.1

이라는 명령어를 사용할 수 있다. 현재 어느 브랜치에 있었든 develop 브랜치를 기준으로 release/1.1 브랜치를 새로 생성하고, 해당 브랜치로 체크아웃한다.

해당 버전의 QA가 종료된다면

- git flow release finish 1.1

이라는 명령어를 사용한다. 해당 브랜치가 master에 머지되고, master 브랜치로 체크아웃한다. 해당 브랜치는 로컬 및 원격 브랜치 모음에서 전부 삭제된다.

**2. RESTful API**

RESTful API는 ‘명시적 상태 전송(REpresentational State Transfer)의 규칙을 따르는 API’라는 의미로, HTTP의 메소드인 GET, POST, PUT, DELETE 등을 통해 자원을 처리하고, 통신하도록 설계되었다. 여기서는 RESTful API에 대한 추가 이론적인 설명을 줄이고, 각 메소드와 각 메소드의 응답 코드 별 의미를 어떻게 해석하여야 하는지에 대해 설명할 것이다.

**2-1. GET**

GET 메소드는 특정한 리소스를 가져오도록 요청할 때 사용한다. GET 메소드로 요청하는 API는 request body를 사용하지 않는 것이 일반적이다. 따라서 필터 등의 정보는 URI에 쿼리로 보내게 된다. 예를 들어, 서비스를 사용하는 사용자 계정의 정보들을 얻어오는 API

- GET /users

가 있다고 하자. 이 정보에서, 이름에 “김”이 들어간 사용자 계정만 얻어오고 싶다면

- GET /users?searchType=name&searchTerm=김

과 같이 나타낼 수 있을 것이다. 단, 이렇게 요청될 경우 쿼리 제한을 감안하여야 한다. 만약 필터할 수 있는 요소가 너무 많다면, 쿼리 사용이 권장되는 GET 메소드로 API를 구성하는 것이 아닌, request bod를 사용하는 것이 일반적인 POST 메소드로 API를 구성하는 것이 권장된다.

GET 메소드로 요청한 API의 Status Code는 다음과 같이 해석할 수 있다.

- 200 : API 요청 및 처리 성공

- 202 : API 요청이 승인되었으나, 처리가 되지 않음

- 203 : API 요청 및 처리가 성공하였지만, proxy 등의 외부 요인에 의해 request 및 response가 수정되었을 가능성을 암시

- 204 : API 요청 및 처리가 성공하였지만, response가 없음

- 400 : API 요청 시 request가 유효하지 않음

- 403 : API 요청 권한이 유효하지 않음

- 404 : API URI가 유효하지 않음

- 500 : 서버에 에러가 있음

**2-2. POST**

POST 메소드는 서버로 데이터를 전송하거나, request body를 통해 특정한 응답을 요청할 수 있다. 다시 말해, 대부분의 API 메소드는 POST로 처리할 수 있으나, 이는 RESTful의 원칙에 위반되므로 자제하여야 한다.

POST의 첫째 예시로, 사용자 계정을 생성하는 API의 URI 및 payload를 다음과 같이 구성할 수 있다.

- POST /user

payload: {

userId: “test”,

userPwd: “test”,

}

또한 POST의 둘째 예시로, GET과 마찬가지로 사용자 계정의 정보들을 획득하고, 그 정보들의 필터를 적용하여 제한된 정보를 얻을 수 있다.

- POST /users

payload: {

searchType: “name”,

searchTerm: “김”

}

POST 메소드로 요청한 API의 Status Code는 다음과 같이 해석할 수 있다.

- 200 : API 요청 및 처리 성공(단, POST 메소드 API가 정보를 “획득”하는 요청을 할 때 사용함이 바람직)

- 201: API 요청 및 처리 성공(단, POST 메소드 API가 새로운 정보를 “생성”하는 요청을 할 때 사용함이 바람직)

(그 외는 GET 메소드와 동일)

**2-3. PUT**

PUT 메소드는 서버에 존재하는 데이터를 수정할 때 사용된다. PUT 메소드 역시 request body를 사용할 수 있다. 예를 들어, 내가 PUT 메소드를 활용하여 “안녕하세요!”라는 내용을 가진 id 값이 1인 게시글을 수정하려 한다.

- PUT /board/1

payload: {

…,

content: “안녕하세요?”

}

이렇게 요청을 보낼 때, id 값이 1인 게시글의 내용은 “안녕하세요!”에서 “안녕하세요?”로 변경될 것이다. 단, URI에 수정 요청할 게시글의 id를 보내는 것이 아닌, request body에 그 id값을 포함하여 보낼 수도 있다.

PUT 메소드로 요청한 API의 Status Code는 다음과 같다.

- 201: API 요청 및 처리 성공

(그 외는 GET 메소드와 동일)

**2-4. PATCH**

PATCH 메소드는 서버에 존재하는 데이터를 수정한다는 점에서 PUT과 동일하지만. PUT의 경우에는 해당 데이터의 일부를 수정하더라도 해당 데이터 전부를 변하게 한다면, PATCH는 해당 데이터를 유지시킨 상태에서 일부만 수정한다는 점이 PUT과의 가장 큰 차이점이다.

PATCH 메소드로 요청한 API의 Status Code는 다음과 같다.

(PUT 메소드와 동일)

**2-5. DELETE**

DELETE 메소드는 서버에 존재하는 데이터를 삭제하기 위해 사용한다. 일반적으로는 request body를 사용한 요청이 권장되지 않으나, 다중 삭제 건 등을 처리하기 위해 request body를 사용하기도 한다. 예를 들어, 단건 처리의 경우에는

- DELETE /user/1

과 같은 URI로 id 값이 1인 사용자의 단건 삭제 요청을 할 수도 있으며,

- DELETE /user?deleteIds=1

과 같은 URI로 쿼리를 통해 다건 삭제 요청을 받고, 그를 이용하여 id 값을 하나만 집어넣어 다건 삭제 요청을 할 수도 있다. 하지만 쿼리로 다건 삭제 요청을 구현할 경우,

- DELETE /user?deleteIds=1,2,3,4,5,6

과 같이 string의 형태로만 처리하게 되는 문제가 생길 수 있다. 따라서 DELETE 메소드는 일반적으로 request body를 통한 요청이 권장되지 않으나, 쿼리로 전달할 때의 불편한 점을 피하기 위해 request body를 주로 사용하게 된다.

DELETE 메소드에서 request body를 담는 것은, axios 기준으로 다음과 같다. 이미 axios request/response interceptors가 설정되어 있고, 그를 http라는 상수명으로 import 하고 있음을 전제로 설명한다.

- const deleteUser = (payload) => {

return http.delete(“/users”, {

data: payload

})

}

위와 같은 방법으로 DELETE 메소드에서도 request body를 담아 요청할 수 있다.

DELETE 메소드로 요청한 API의 Status Code는 다음과 같다.

(GET 메소드와 동일)