Netty-Starter-With-Spring-Boot (Day1)

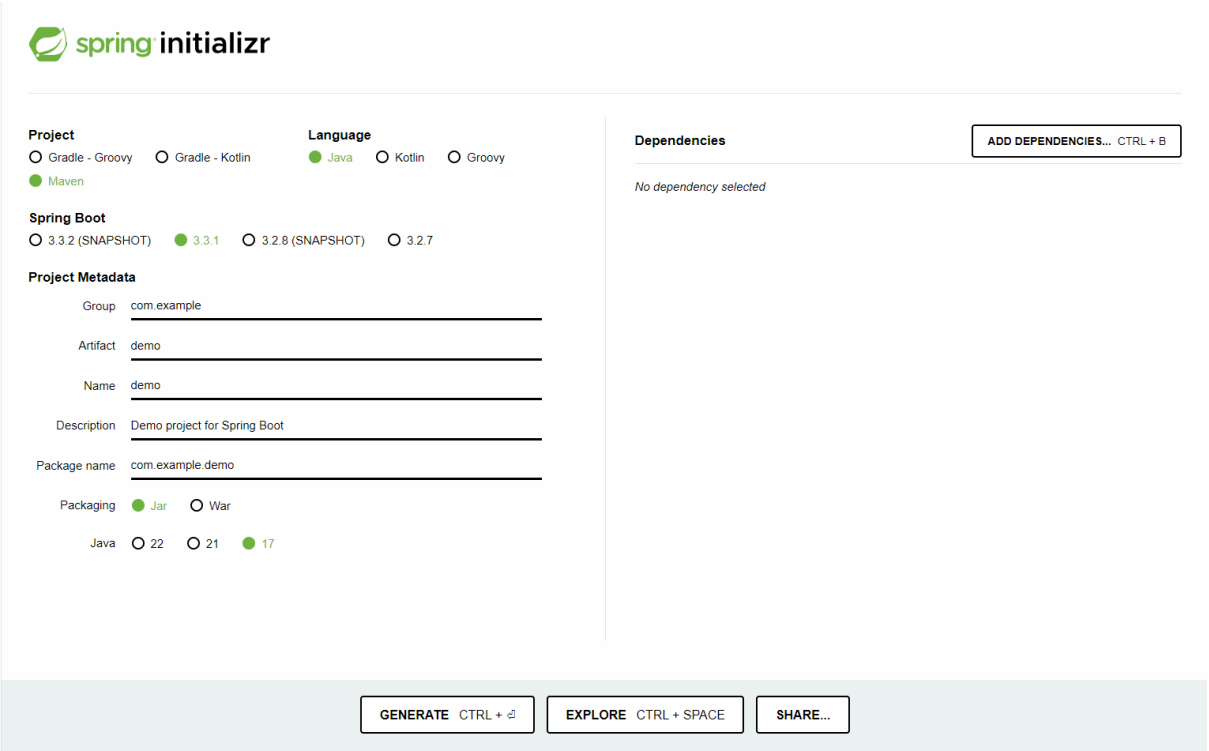
오늘의 목표

1. **Spring Boot 프로젝트를 생성할 수 있다.**
2. **GitHub Repository를 생성할 수 있다.**
3. **Spring Boot 프로젝트를 원격 저장소에 연결할 수 있다.**
4. **Netty Dependency를 추가하고 Discard Server를 구현할 수 있다.**

1. Spring Boot Project Created

1.1 start.spring.io 링크 접속

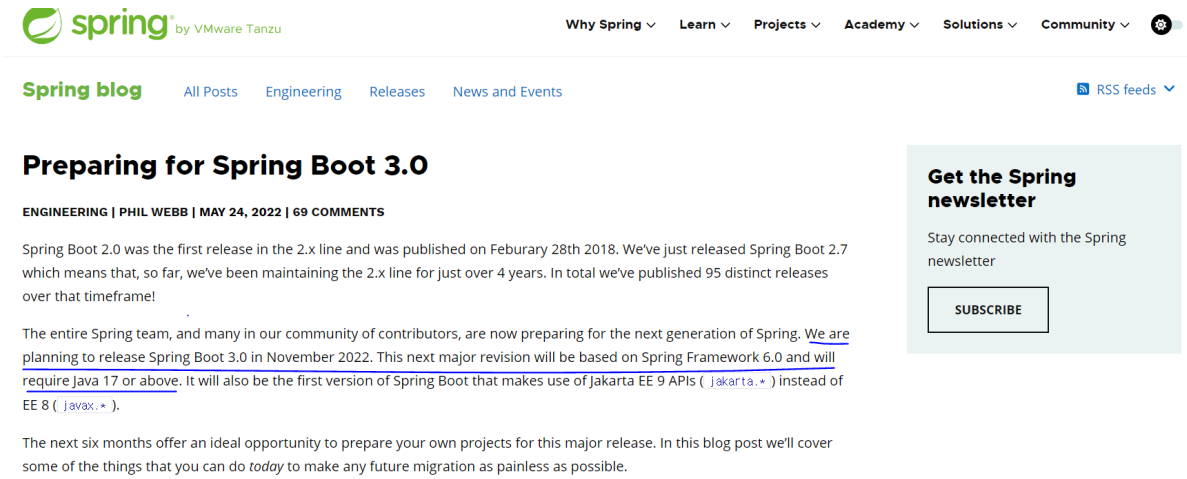
- **Intellij Ultimate가 아닌 경우 해당 사이트에서 만들어야 합니다.**
- **Ultimate인 경우 프로젝트 생성에서 만들 수 있습니다.**
- **접속 후에는 아래와 같은 창이 나옵니다.**



1.2 Maven Or Gradle

- **Maven, Gradle 둘 중에 하나를 선택하시면 됩니다.**
- **저는 해당 프로젝트에서는 Maven으로 진행 하겠습니다.**

1.3 Spring Boot Version & JDK Version



- 기본적으로 Spring Boot 버전이 올라가면서 요구하는 JDK 버전이 올라갔습니다.
- 3.2.7을 선택, JDK 버전은 기본적으로 17을 사용하게 되었습니다.
- 위에 사진을 보시면 Spring Boot 공식 사이트에 있는 내용입니다.
- Spring Boot 3.0부터는 JDK 17을 필요로 한다고 적혀있습니다.
- JDK 11을 사용하고 싶다면 Spring Boot 최신 버전인 2.7.x를 사용하기를 추천합니다.

1.4 Group 설정 Ex → com.dev.vihaan

- 보통 기업의 도메인명을 기입하여 사용합니다.
- 개인 프로젝트일 경우 자유롭게 만들면 됩니다.

1.5 Artifact 설정 Ex → netty-starter-with-spring-boot

- 빌드된 결과물 이름을 의미합니다 (.jar)

1.6 Description

- Ex → Netty Practice Project with Spring Boot
- 프로젝트에 대한 설명을 적을 수 있습니다.

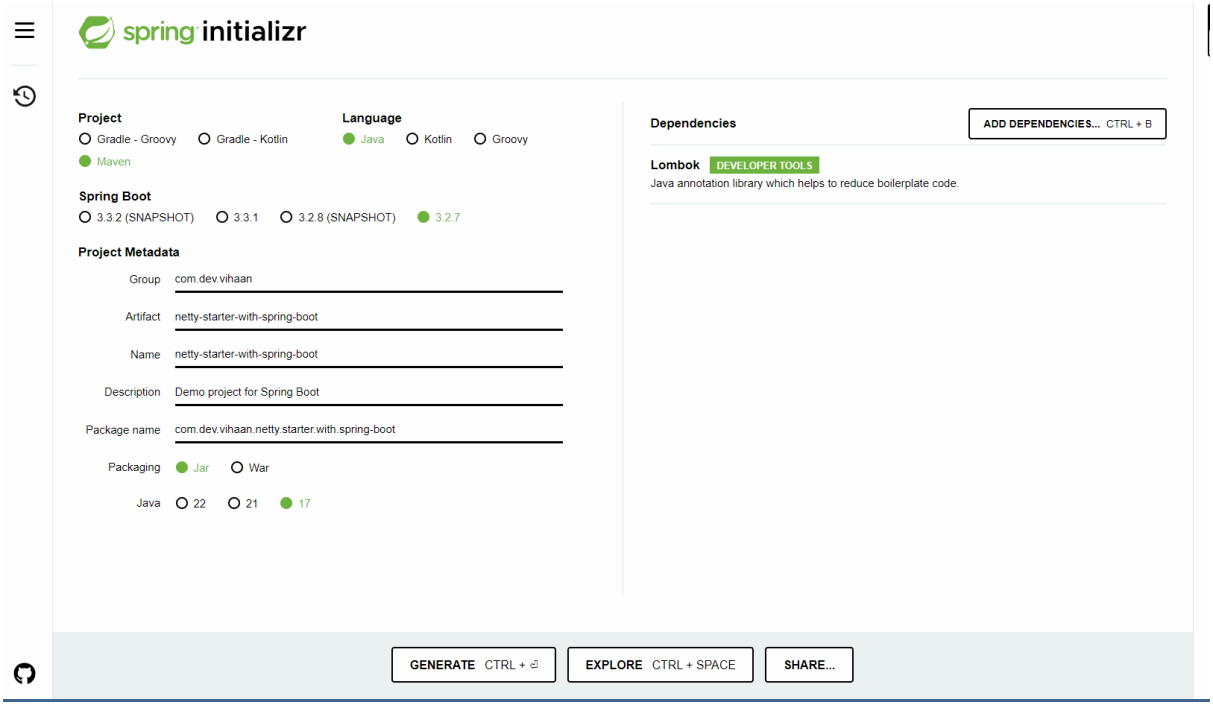
1.7 package

- java의 패키지 name을 의미합니다.

1.8 Dependency 선택

- 위와 같이 작성을 끝냈다면 오른쪽에서 Dependencies를 클릭하여 Lombok을 선택해 줄 수 있습니다. 단축키 (CTRL + B)

- 설정이 끝났을 때는 아래와 같이 나옵니다. (package 명만 살짝 수정하였습니다.)



1.9 Intellij에서 프로젝트 열기

- Generate를 누르면 압축 파일이 나오고 압축을 풀면 됩니다.
- 압축을 푼 파일을 Intellij를 통해 열 수 있습니다.

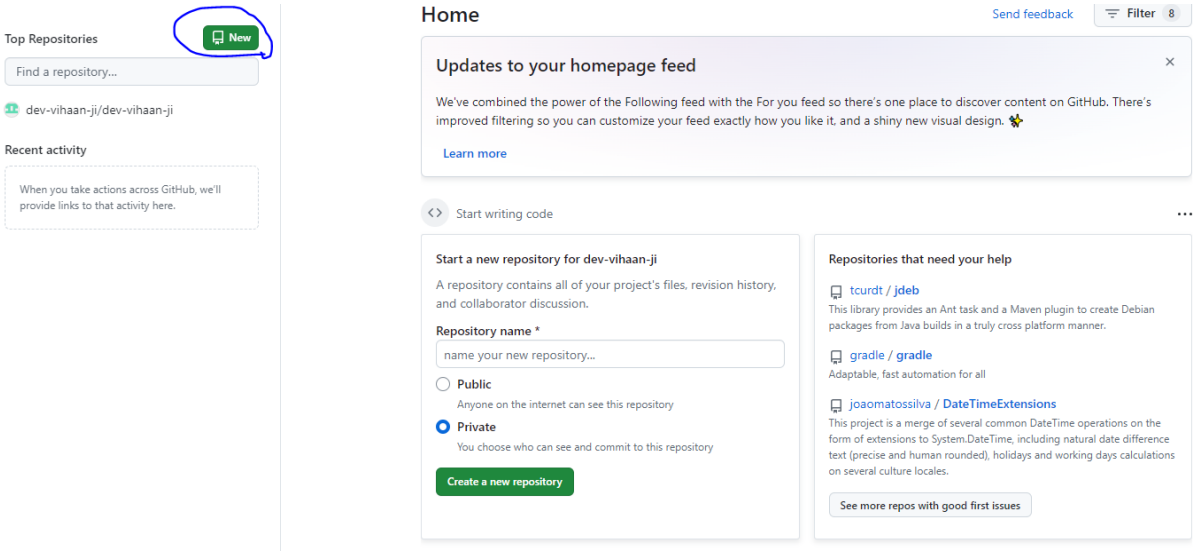
---

2. GitHub Repository Create!

2.1 GitHub login

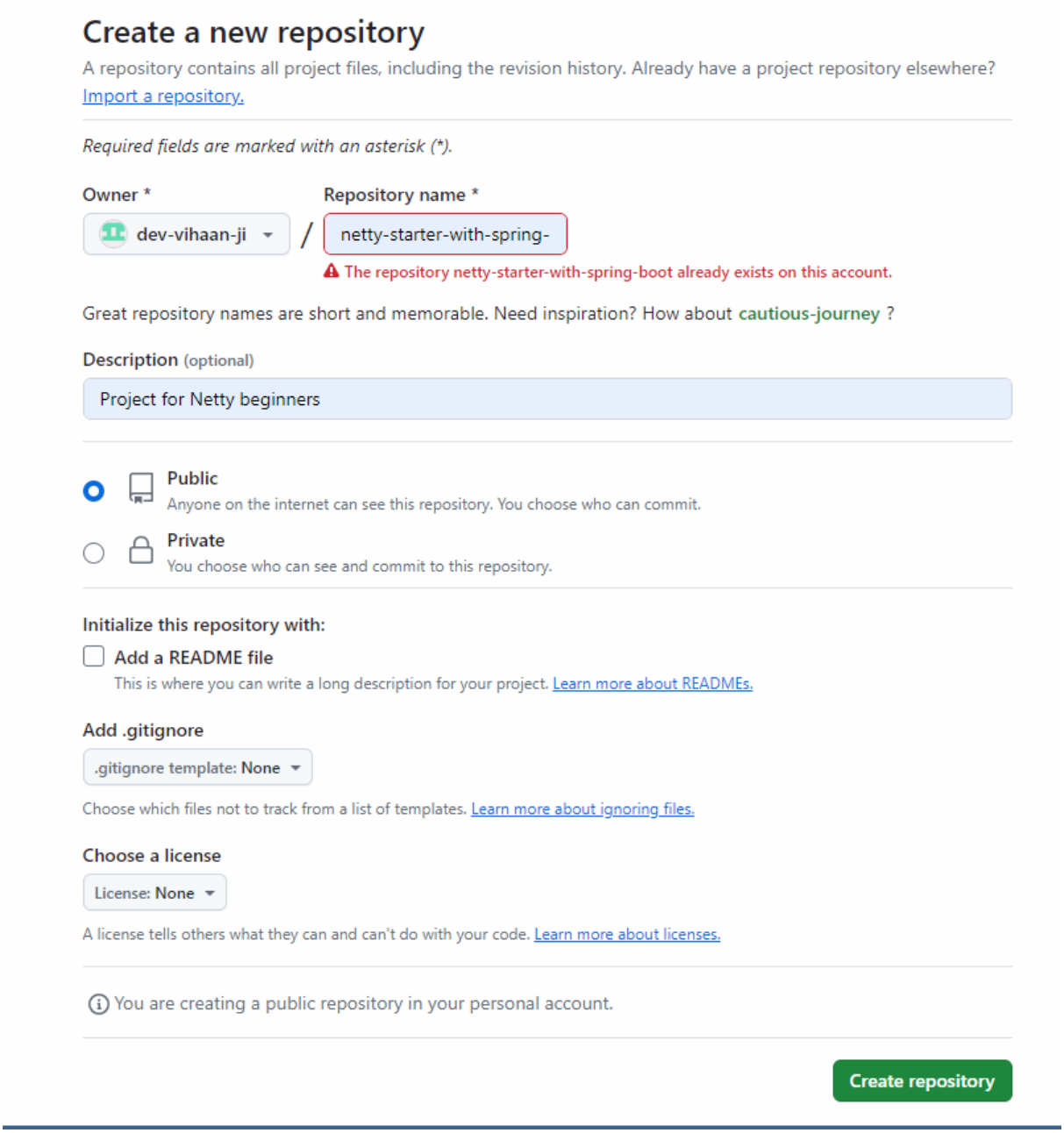
- GitHub 계정을 통해 로그인을 진행합니다.

2.2 로그인을 한 후 왼쪽 상단에 New 버튼을 클릭



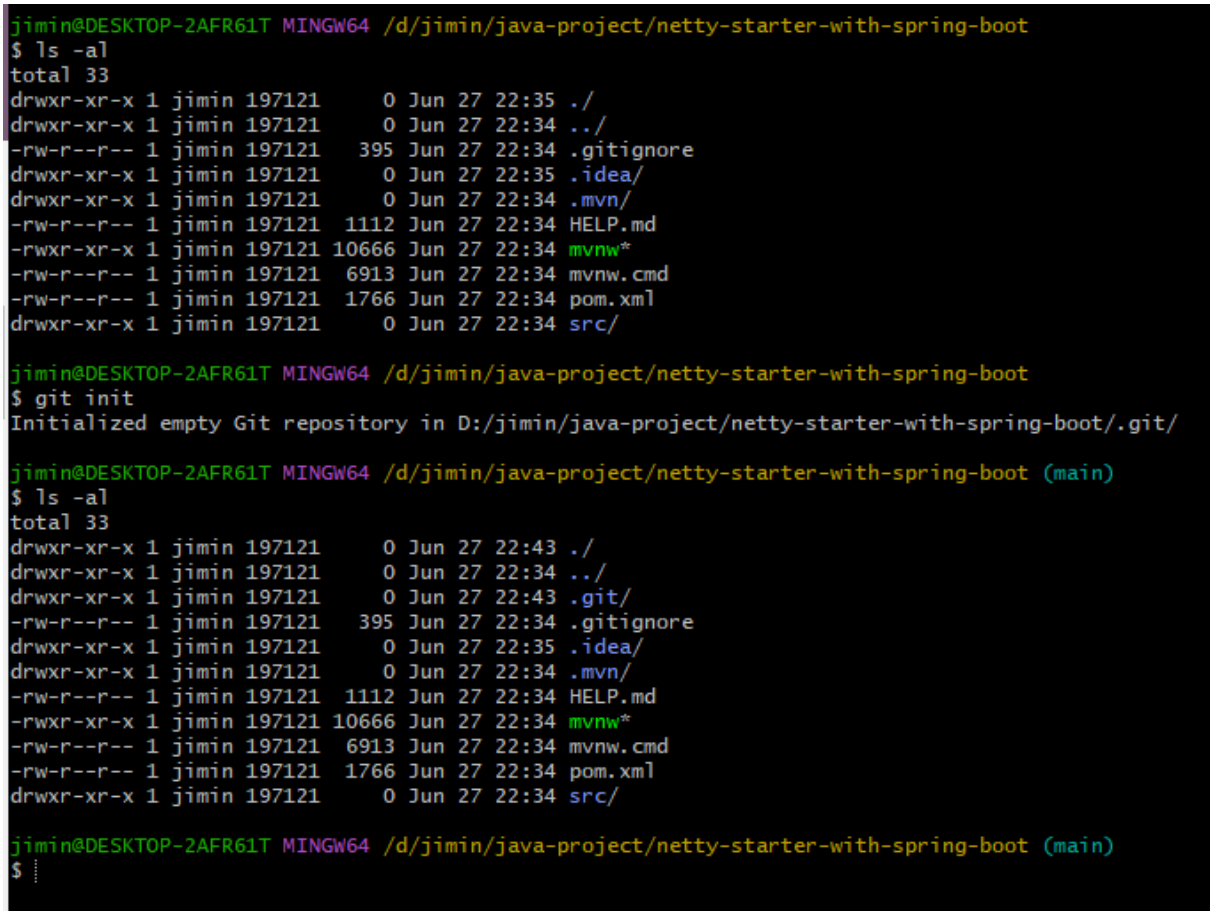
2.3 GitHub Create repository

- 아래 요소들에 대해서는 간략하게 설명만 하겠습니다. 자세한 내용은 찾아서 공부하시면 됩니다.
- Repository name : 원격 저장소 이름 (프로젝트 이름)
- Description (optional) : 원격 저장소 설명
- public / private :
  - Public: 누구나 저장소를 볼 수 있고, 복사할 수 있는 공개 저장소입니다.
  - Private: 저장소 소유자만 접근할 수 있는 비공개 저장소입니다.
- README file
  - 프로젝트에 대한 설명을 담는 파일입니다.
  - 프로젝트의 목적, 기능, 사용 방법, 설치 방법 등을 작성할 수 있습니다.
- .gitignore
  - Git이 특정 파일이나 디렉토리를 무시하도록 지정하는 파일입니다.
  - 빌드 결과물, 로그 파일, IDE 설정 파일, 중요한 보안 파일 등 프로젝트에 포함되지 않아야 할 파일들을 지정할 수 있습니다.
  - 프로젝트 시작 시부터 .gitignore 파일을 활용하면 효율적으로 프로젝트를 관리할 수 있습니다.



3. Spring Boot Project Connect GitHub Repository

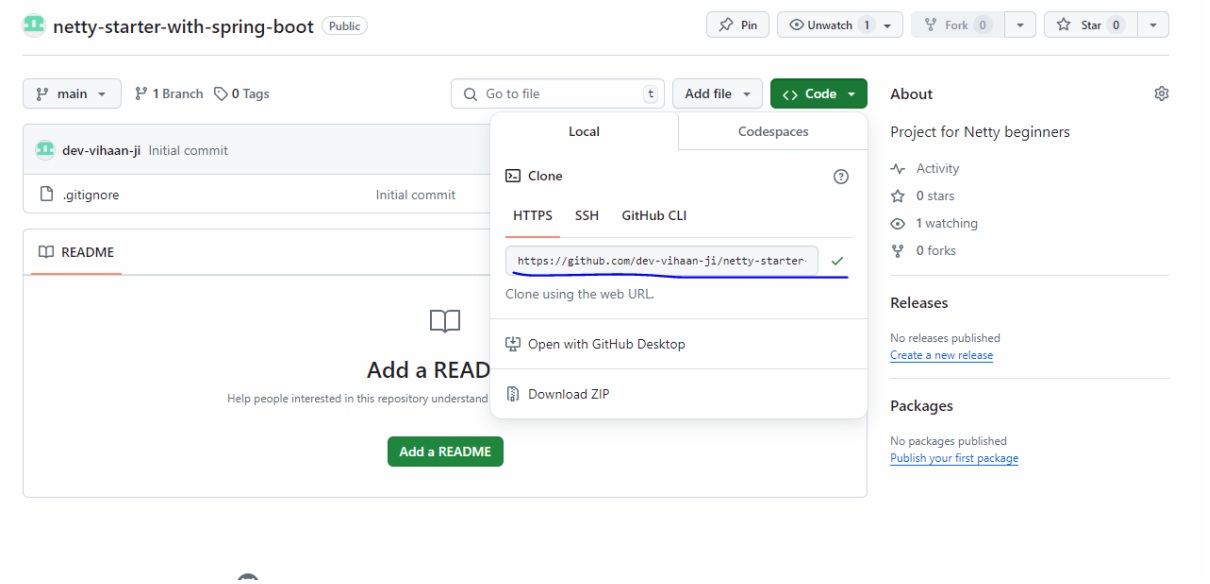
3.1 `GitBash` 를 통해 내가 올릴 프로젝트 경로로 접속



- 접속 후 `git init` 명령어를 사용합니다.
- `git init`이란:
  - Git을 로컬 저장소에 생성하기 위해 Git Bash 프롬포트를 통해 입력을 하면 `.git`이라는 파일이 생성됩니다.
  - 이 파일은 버전에 대한 정보를 담고 있는 파일이며, 평소에는 숨김 파일로 설정되어 눈에 보이지 않습니다.
  - `.git` 파일을 보기 위해서는 `ls -al` 옵션을 주게되면 `.git` 볼 수 있습니다.
  - 또한 해당 파일을 삭제하면 이전 버전에 대한 정보가 날아갈 수 있음으로 주의해야 합니다.

3.2 .git 파일 생성 후(git remote)

- 파일이 생성됐다면 아래와 같이 원격 GitHub Repository 주소를 복사합니다.




```
jimin@DESKTOP-2AFR61T MINGW64 /d/jimin/java-project/netty-starter-with-spring-boot (main)
$ git remote -v

jimin@DESKTOP-2AFR61T MINGW64 /d/jimin/java-project/netty-starter-with-spring-boot (main)
$ git remote add origin https://github.com/dev-vihaan-ji/netty-starter-with-spring-boot.git

jimin@DESKTOP-2AFR61T MINGW64 /d/jimin/java-project/netty-starter-with-spring-boot (main)
$ git remote -v
origin  https://github.com/dev-vihaan-ji/netty-starter-with-spring-boot.git (fetch)
origin  https://github.com/dev-vihaan-ji/netty-starter-with-spring-boot.git (push)

jimin@DESKTOP-2AFR61T MINGW64 /d/jimin/java-project/netty-starter-with-spring-boot (main)
$
```

- 그리고 위와 같이 내 원격 저장소에 연결을 할 수 있습니다. (`git remote add origin 저장소`)
- `git remote -v` 현재 프로젝트에 설정된 원격 저장소의 URL을 확인합니다.

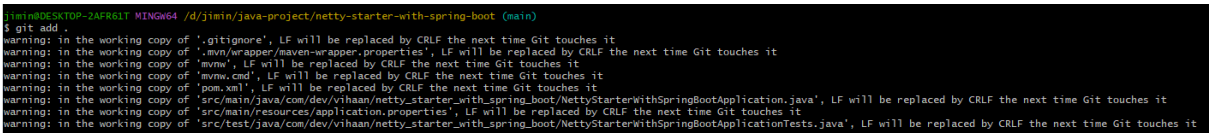
3.3 원격 저장소에서 처음에 파일 가져오기 (git pull)

```
jimin@DESKTOP-2AFR61T MINGW64 /d/jimin/java-project/netty-starter-with-spring-boot (main)
$ git pull origin main
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (5/5), 1.86 KiB | 158.00 KiB/s, done.
From https://github.com/dev-vihaan-ji/netty-starter-with-spring-boot
 * branch            main       -> FETCH_HEAD
 * [new branch]      main       -> origin/main
```

- git에 처음으로 연동할 때, 원격 저장소에 `.gitignore`, `README.MD` 파일이 이미 존재하는 경우가 많습니다.

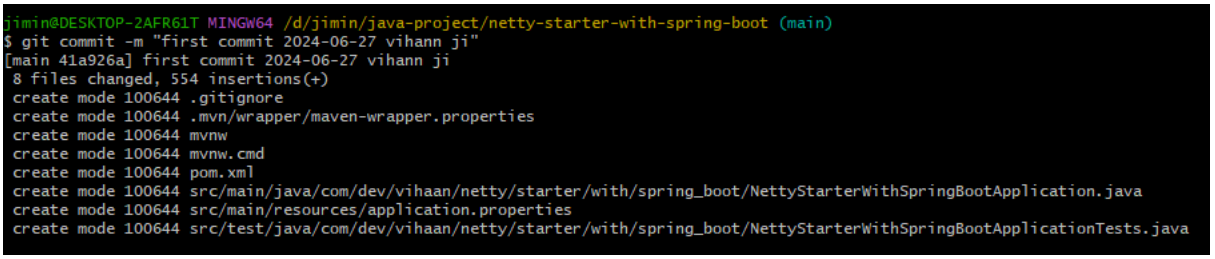
- 이 경우 먼저 `git pull 브랜치명` 명령어를 사용하여 원격 저장소의 파일들을 로컬 저장소로 가져와야 합니다.
- 만약 원격 저장소와 로컬 저장소의 내용이 일치하지 않는 경우, `git pull` 명령어를 실행하면 충돌이 발생할 수 있습니다.
- 이 경우 충돌을 해결한 후 `git add`, `git commit`, `git push` 등의 작업을 수행할 수 있습니다.
- 이렇게 하면 로컬 저장소와 원격 저장소의 내용이 일치하게 되어 이후 `git add`, `git commit`, `git push` 등의 작업을 수행할 수 있습니다.

3.3 원격 저장소에 올릴 파일 Git에 추가 (git add)



- (git add .) / (git add *) → 전체 파일 추가
- (git add 파일 명) → 선택한 파일만 추가
- 새로 생성된 파일 또는 작업 디렉토리에서 변경된 파일을 `Git의 Staging Area`에 추가하는 명령어입니다.
- 아래와 같이 warning 로그가 뜨는 이유는 운영체제가 다르기 줄바꿈 문자 특성이 다르기 때문입니다.
- Window OS에서는 줄바꿈 문자 CRLF `(4 byte 사용)`
- Unix OS에서는 줄바꿈 문자 LF `(2 byte 사용)`
- Git은 이 차이를 자동으로 처리하려고 시도할 때 이런 경고 메시지가 나타나는 것이며, 실제 코드나 파일에는 영향을 미치지 않습니다.

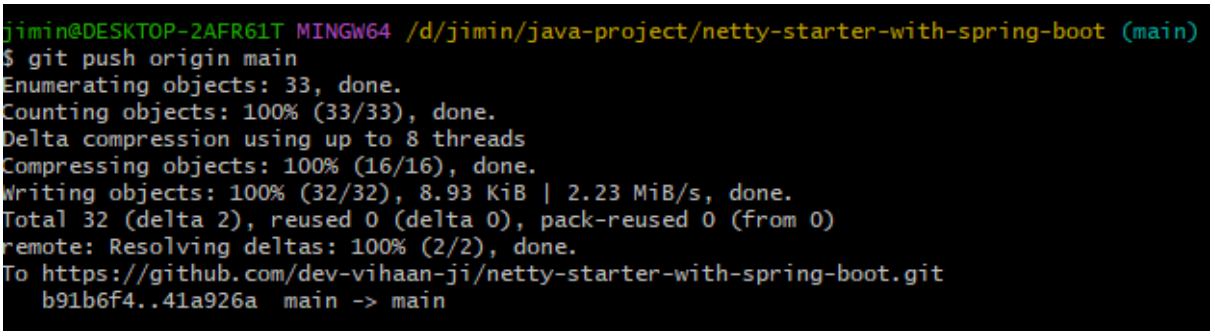
3.4 Staging Area에 추가된 변경 사항들을 커밋 (git commit)



- Staging Area에 추가된 변경 사항들을 하나의 커밋으로 기록하는 명령어입니다.

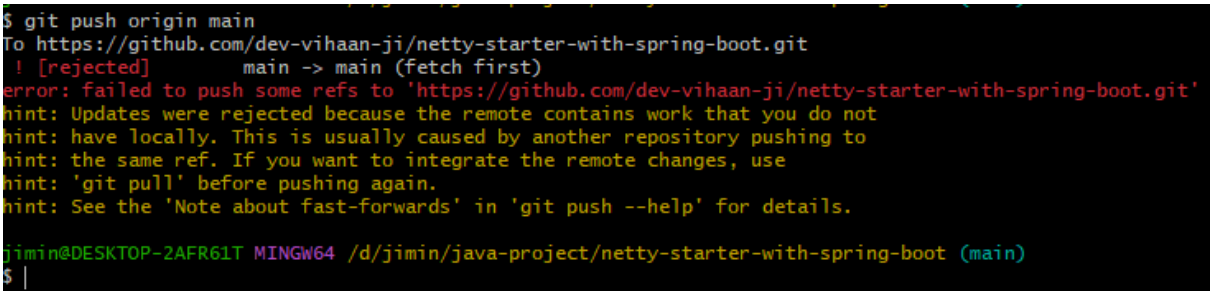
- 커밋 메시지를 작성하여 변경 사항에 대한 설명을 남길 수 있습니다.
- 커밋은 로컬 저장소에 기록됩니다.

3.5 원격 저장소에 업로드 (git push)



- 로컬 저장소에 있는 커밋들을 원격 저장소(예: GitHub)에 업로드하는 명령어입니다.
- 로컬 저장소의 커밋 내역을 원격 저장소에 반영합니다.
- 3.5.1 에러 참조

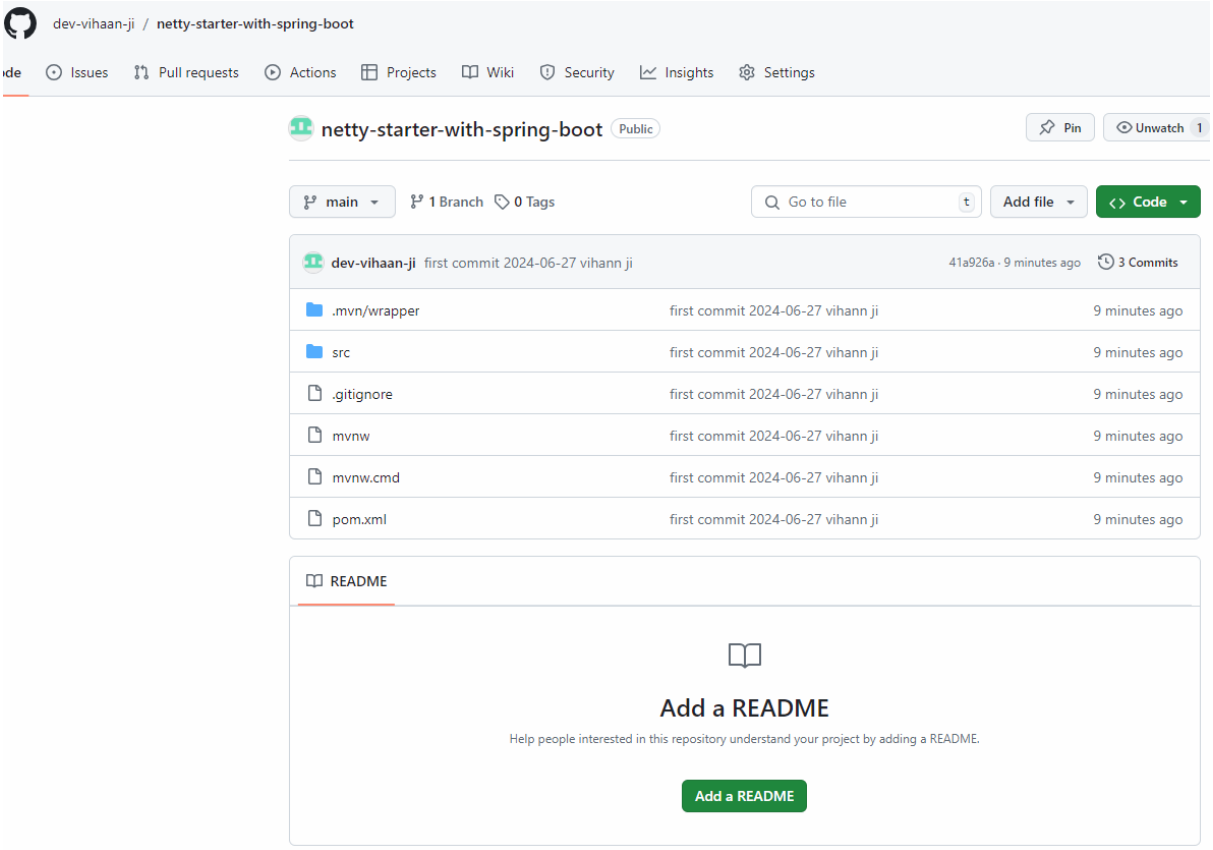
3.5.1 git push 에러 (자주 겪는 문제라 참조하라고 넣었습니다.)



- 이 오류 메시지는 원격 저장소에 로컬 저장소와 다른 커밋이 존재하여 푸시가 실패한 것을 의미합니다.
- 이러한 경우 일반적으로 아래와 같은 해결책을 사용합니다:
- 원격 저장소 변경 내용 가져오기: `git pull 브랜치`
- 명령어를 사용하여 원격 저장소의 변경 내용을 로컬로 가져옵니다.이 과정에서 충돌이 발생할 수 있으므로, 충돌을 해결하고 커밋합니다.
- 그래서 보통 git에 처음 연동할 때 원격 저장소에 `.gitignore, README.MD` 파일이 존재하는 경우에는 먼저 pull을 받고 작업을 해야 합니다.

3.6 원격 저장소 확인 성공

- 아래와 같이 정상적으로 GitHub Repository에 업로드한 것을 확인할 수 있습니다.



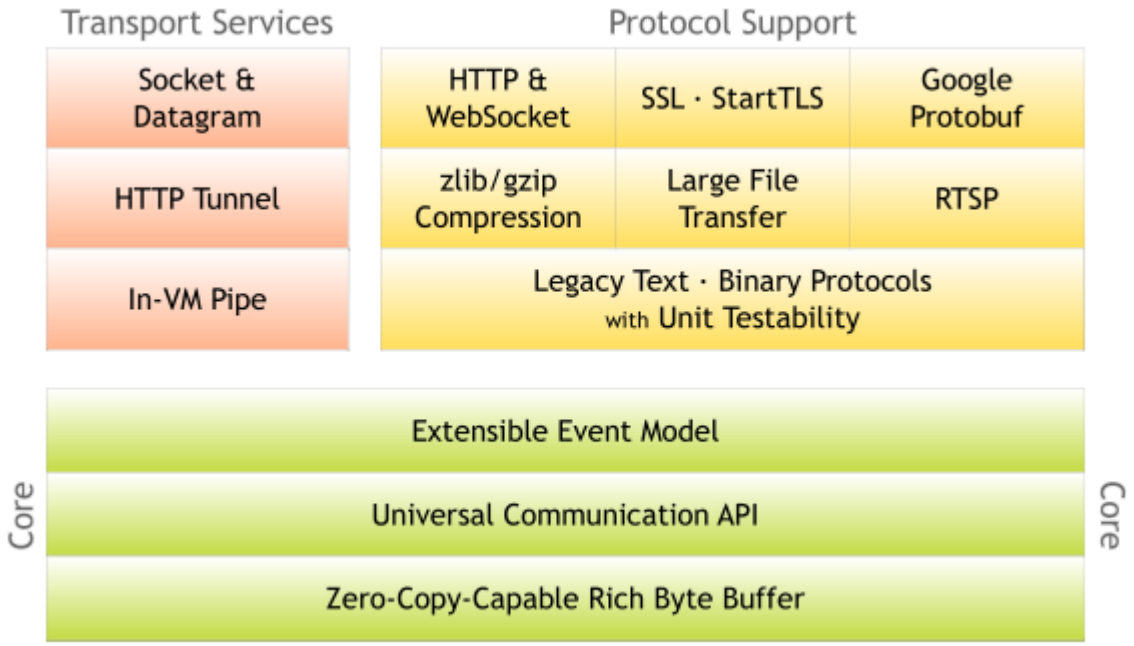
4. Netty Framework란 무엇인가?

4.1 What is Netty?

> 💡 **네티는 비동기 이벤트 기반 네트워크 애플리케이션 프레임워크로써 유지 보수를 고려한 고성능 프로토콜 서버와 클라이언트를 빠르게 개발할 수 있다.**

- 비동기 이벤트 주도 모델
  - Netty는 비동기 이벤트 주도 모델을 사용하여 네트워크 프로그래밍을 단순화합니다.
  - 이를 통해 서버와 클라이언트 간의 통신이 효율적이고 확장성 있게 이루어질 수 있습니다.
  - 비동기 방식으로 동작하여 블로킹 I/O 문제를 해결하고, 이벤트 기반 구조로 인해 코드 복잡도가 낮아집니다.

- 높은 성능
  - Netty는 Java NIO(Non-Blocking I/O)를 기반으로 하여 높은 성능을 제공합니다.
  - 메모리 누수 방지, 빠른 직렬화/역직렬화, 최적화된 버퍼 관리 등의 기능을 통해 성능을 극대화합니다.
- 다양한 프로토콜 지원
  - Netty는 TCP, UDP, HTTP, WebSocket, MQTT 등 다양한 프로토콜을 지원합니다.
  - 이를 통해 개발자는 프로토콜 구현에 신경 쓰지 않고 비즈니스 로직에 집중할 수 있습니다.
- 확장성 및 유연성
  - Netty는 모듈화된 설계를 통해 확장성과 유연성을 제공합니다.
  - 개발자는 필요한 기능만 선택적으로 사용할 수 있으며, 사용자 지정 코덱, 핸들러 등을 쉽게 추가할 수 있습니다.
- 다양한 플랫폼 지원
  - Netty는 Java 기반으로 개발되었지만, 다양한 플랫폼(Windows, Linux, macOS 등)에서 사용할 수 있습니다.
- netty-core



4.2 Netty Features

- 설계
  - 다양한 전송 유형 지원
    - Netty는 blocking, non-blocking 소켓을 모두 지원하여 개발자가 필요에 따라 선택할 수 있습니다. 이를 통해 다양한 네트워크 환경에서 활용할 수 있습니다.
  - 이벤트 모델의 유연성
    - Netty의 이벤트 모델은 매우 유연하고 확장 가능합니다.
    - 개발자는 필요에 따라 이벤트 핸들러를 추가/수정할 수 있어 다양한 요구사항을 처리할 수 있습니다.
  - 스레드 모델의 사용자 정의
    - Netty는 단일 스레드, 하나 이상의 스레드 풀 등 다양한 스레드 모델을 제공합니다.
    - 개발자는 자신의 요구사항에 맞게 스레드 모델을 선택하고 구성할 수 있습니다.
- 사용 편의성
  - 풍부한 문서화
    - Netty는 Javadoc, 사용자 가이드, 예제 등 다양한 문서화 자료를 제공하여 개발자의 학습 및 사용을 돕습니다.
  - JDK 호환성
    - Netty 3.x는 JDK 5 이상, Netty 4.x는 JDK 6 이상을 지원합니다.
    - 이를 통해 다양한 Java 버전에서 Netty를 사용할 수 있습니다.
  - 추가 구성 요소
    - HTTP/2 등 일부 구성 요소는 추가적인 설정이 필요할 수 있습니다.
    - 이 경우 Netty 문서를 참고하여 올바르게 구성해야 합니다.
- 성능
  - 높은 처리량, 낮은 지연 시간
    - Netty는 높은 처리량과 낮은 지연 시간을 제공하여 고성능 네트워크 애플리케이션 개발에 적합합니다.
  - 자원 소비 감소

        - Netty의 효율적인 설계로 인해 CPU, 메모리 등 시스템 자원 소비가 감소합니다.
    - 메모리 복사 최소화
        - Netty는 불필요한 메모리 복사를 최소화하여 성능을 향상 시킵니다.
- 보안
    - SSL/TLS 및 Start TLS 지원
        - Netty는 완전한 SSL/TLS 및 Start TLS 지원을 제공하여 안전한 네트워크 통신을 구현할 수 있습니다.

4.3 Netty Dependency Maven 추가

```
<!--    Netty library   -->
    <dependency>
            <groupId>io.netty</groupId>
            <artifactId>netty-all</artifactId>
            <version>4.1.32.Final</version>
    </dependency>
```

- 위에 Dependency를 pom.xml 파일에 추가를 합니다.

4.4 Netty Discard Server 구성

- Netty를 사용하여 Discard 서버를 구현하는 부분입니다.
- Discard 서버는 클라이언트가 보낸 데이터를 버리는 간단한 서버입니다.
- Netty를 사용하여 서버를 구성하고 실행하는 방법을 보여줍니다.

4.5 Netty Discard Handler 구성

- Discard 서버에서 사용할 Netty 핸들러를 구현하는 부분입니다.
- Netty 핸들러는 클라이언트의 요청을 처리하고 응답을 보내는 역할을 합니다.
- 이 Discard 핸들러는 클라이언트가 보낸 데이터를 단순히 버리는 기능을 구현합니다.

4.6 Netty Disacrd Server TEST

```
// 8888 포트에서 서버를 시작합니다.
ChannelFuture future = bootstrap.bind(8888).syncUninterruptib
```

- 어플리케이션을 IDEA에서 실행 후에 TELNET 명령어를 통해 테스트를 진행합니다.
- TELNET을 할 때 localhost:port를 지정해줄 수 있습니다.
- 여기서 port는 내가 bind 수행 시 매개변수로 넣은 포트 번호를 통해 테스트를 할 수 있습니다.
- 아래와 같이 connect 된 것을 확인할 수 있습니다.

```
C:₩Users₩jimin> curl -v telnet://localhost:8888
* Host localhost:8888 was resolved.
* IPv6: ::1
* IPv4: 127.0.0.1
*   Trying [::1]:8888...
* Connected to localhost (::1) port 8888
```

- 그리고 TELNET을 종료하고 IDEA 서버 로그를 확인하면 아래와 같이 로그가 나옵니다.

```
java.net.SocketException Create breakpoint : Connection reset
    at java.base/sun.nio.ch.SocketChannelImpl.throwConnectionReset(SocketChannelImpl.java:394)
    at java.base/sun.nio.ch.SocketChannelImpl.read(SocketChannelImpl.java:426)
    at io.netty.buffer.PooledUnsafeDirectByteBuf.setBytes(PooledUnsafeDirectByteBuf.java:288)
    at io.netty.buffer.AbstractByteBuf.writeBytes(AbstractByteBuf.java:1132) <9 internal lines>
```

- 위에 절차를 정상적으로 수행했다면 Discard Server 테스트를 성공한 것으로 보입니다.
- 문제가 생겼을 경우 GitHub 저장소의 Issues 탭에 새로운 이슈를 생성하여 문제 상황을 자세히 설명해주시면 답변을 드리겠습니다.

4.6 GitHub Repository 참조

- 이 프로젝트의 코드는 GitHub에 공개되어 있습니다.
- **GitHub 저장소 주소:** https://github.com/dev-vihaan-ji/netty-starter-with-spring-boot
- **사용한 브랜치:** `day1`