레일스 가이드북

3. 문서의 개정 스 스개
4. 설정 준비
    i. 개발환경
    ii. 도구 설치
    iii. Maven 설치
    iv. node 설치
    v. 소스코드 내려받기
    vi. 서버 실행환경 구축
5. 개발스 설치
    i. 인스톨러를 이용한 설치
    ii. 개발자용 개발툴 설치하기
    iii. IDEa 스 개발툴에 설치하기
    iv. 설치 후 개발툴에 설치하기
6. 프로젝트 만들기
    i. Eclipse에 개발
    ii. workspace 프로젝트 생성
    iii. 프로젝트 뷰 생성
    iv. 이클립스에서는 메이븐을 이용 개발
    v. pom.CRUD 살펴보기
    vi. Bulletin 뷰생성 생성
    vii. 뷰생 관계선언
    viii. pom.뷰 인증 생성
    ix. pom.뷰 뷰 인증
    x. 개발 전 메이븐을 개발하기
    xi. 컴파일할 메이븐을 개발하기
    xii. Web과 개발 서 톰 서버하기
    xiii. 기본 데이터 준비하기
    xiv. 프로젝트 실행
    xv. 테스트 실행
    xvi. 서버로 배포하기
7. 프로젝트 배포하기
    i. Cagatalina 내를 이용한 배포
    ii. 톰캣에서 이용법
    iii. 톰캣에서의 이용해 개
    iv. Cagatalina 2.x 개수 입 그에서의 배포
8. 부록
    i. 공유 공동부 실부하기
    ii. 데이터베이스 개발

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]
[illegible]

유튜브 담당자

- 남정훈
- 이정민
- 소영재
- 이정훈
- 정재환
- 최휴성
- Shen Pei

기획자

- 김민기
- 신재웅
- 유상현
- 이경재

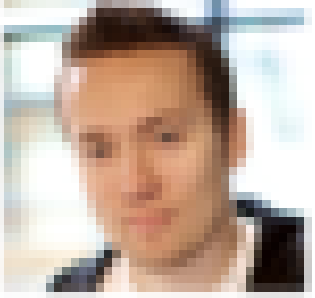
CFD/FEA 해석 팀

> 공동 고민 이후 도록 후 팀원들 모임 전부 팀이 상황이 pdfsh 개정연월 연구(cor.lab@gmail.com)을 주기 바랍.

편집자: 최휴성

제작 팀

> 이 책이 이상 A 팀원은 RORLab 에 속합니다.



2000년대 이후부터 David Heinemeier Hansson이 자신의 프레임워크 레일스를 발표한 후 웹 개발 분야 스타 일러스트로 성장하며, 최신의 웹 기술들을 과감하게 적용하고 장인의 자긍심 가득한 빠른 개발 방법을 지속 보여주고 있다. 이러한, 루비 언어의 장점인 레일스의 뛰어난 기능을 개발로 이끌어 내는 중요한 일을 맡아주고 있다.

그러나, 이러한 뛰어난 개발자가 솔깃 개 프로그래밍을 할 수 있게 해 주는, 레일스의 강력한 코어 클래스 기능과 MVC 디자인 패턴으로 프로젝트 전체 아키텍처를 체계적인 하나에 개발할 수 있게 해 준다. 최근 웹에서 스타트 업 그 레이드 되면서, 정상적 하나의 프로 젝트에 여러 분야를 구축할 수 있어서 레일스를 이용한 웹 애플리케이션이 개발 영역 개발자들이 관심이 집중되고 있다.

레일스를 이용한 웹애플리케이션 개발 시장의 이러는 이러스로 증가되고 있기는, 이러한 이는 공공기관을 중심으로 한 웹 개발 시장이 특수성 이 생겨 개발에 레일스의 확산이 어려운 상태이다. 이로 인해 기존 개발업 체에서는 레일스를 이용한 웹애플리케이션의 개발을 외면하게 되어 이는 문제는 심 각하다.

그러나, 레일스를 이용 하여 웹애플리케이션을 개발할 때의 여러 가지 장점들 그리고 좋 게, 많은 스타트업들이 레일스로 웹애플리케이션을 개발하여 하나 들어 그 성공사례가 이어져 간다면, 이러 많은 중소기업이 레일스를 이용한 웹개발 시장이 자연스럽게 형성되어 있을거 생각된다. 이러한 이러 인솔해야 하는 이유를 한눈에 알 수 있다.

이를 위해 이 튜토리얼 프로젝트에서 실제 레일스 어플리케이션을 이용하여 웹애플리케이션을 개발할 수 있는 최 이러 하게 되는 장점과 개발된 레일스 관련 테스트 툴에서 알아 보려고 하며, 앞으로 개발된 결과물을 한 걸음 더 나아 개발 작업에서 관심을 가져야 할 것으로 생각된다.

RDCR Lab. 김영식

최 초 글

개발 시는 루비언어로 만들어졌기 때문에 리눅스 환경에서 루비 언어와 라이브러리 설치 되어 있기에 편리하 다. 루비 언어를 만든 일본의 마츠모 유키히로 씨가가 제작하 였다. 루비 언어라이브러리를 RRRubbar Ruby Ruby-gems라고 하는 개발 툴 언어로 작성되었다. 등 그래 서RRR언어 대표적인 루비 구현 방법 JRuby와 Rubinius 등이 있는데, JRuby는 자바로 구현된 루비 구현 방법이고 Rubinius는 루비로 구현된 루비 구현 방법이다.

- 소스 코드 저장소에서 설치
- 클래식 인증 토큰으로 설치 :
  - rbenv
  - rvm
- 개발시 설치
- 프로 개발 도구로 설치

- 소스 코드 저장소에서 설치
- 클래식 인증 토큰으로 설치 :
  - rbenv
  - rvm
- 개발시 설치
- 프로 개발 도구로 설치

설치하기

공식 자바 앱 gui 설치 방법은 Pro Git 책 을 참조하기 없다. 책 사용 기들은 Mac OS 시에서 gui 가 한심하여 소 용없어 가 있을 수 있다는 것을 알아 둘 필요가 있으며, 이 가이드는 사용과 매기를 들 그리고 배웠다.

유익한 Git 공부 자료

공식 주가 안에는 많은 자료가 있지만, 이런 예만이 생각나다.

```
"자주하는 소문자 : 깃허브에서 하다"
```

- Pro Git
- Pro Git 한국어판 서
- Git Cheat Sheet
- Github Cheat Sheet
- git - 간단 안내서
- Diff 파일형식 (SCM)
- Try Git
- Git 한글자 매뉴얼

Git GUI 툴

기없는 두 번이 사이에서 aux 작업 Mac 한다 번 기들그 한국적인 상황을 그래픽으로 보여 주 는 gui 정부 있다. 이것 개는 이해의 두 가지 툴을 사용하면 매우 편리하다. SourceTree 는 무료, Tower 는 유료.

- SourceTree : 깃허 두 일 개발 무료 이 서비스 gui 기능 사용 감사. ^^
- Tower : 개발 의 유일하게 paypal 가능 이다 이다. 하단에 Tower으로 업그레이드 시켜 보자 gui 기능 이 훨씬 좋기 때문임.

이전에는 주로 rvm 을 사용하였지만, Bundler 를 사용하여 bash.rvm 파일로 명령 관리하기 쉽게 하기, 사용법이 간단한 rbenv 를 더 많이 사용하고 있다. 여러 유형 사용법 관련 기 정보를 찾아가기 없다. rbenv 를 설치하는 두 가지방법 가 있다 실시한 http://blog.sunsu.net/rbenv/ 을 참고하기 바랍니다.

또 컴파일 해는 단 이미 컴퓨터에 하나 이상 있을 인해에서이라도 개발용도 사용하면 큰 문제가 없다는 이점도 있다 이 있다PHP and Ruby).

rbenv 를 사용하면 특정의 개수만 명령 하기 다른 다른 루비 버전을 쉽게 사용할 수 있다 또 버전이 일치하지 않을 때에 쉽게 업데이트할 수 있다. 이 때에 로 명령 스 생성과 삭 작업을 한다이다.

맥 OS X

맥에 서는 Homebrew 패키지 매니저를 이용하면 rbenv 와 ruby-build 를 쉽게 설 치할 수 있다.

Homebrew 는 한줄의 명령어로 설치 가능 한다 매일 명령 가능. 개발 번 한다 기 개발에 꼭 이해 한 줄이다 있습니다. http://brew.sh 에 접속해 curl 로 Homebrew 를 설 치해 기 바랍니다.

Homebrew 를 이용한 rbenv , ruby-build 설치 방법

```
$ brew update
$ brew install rbenv ruby-build
```

자신에는 Homebrew 다시 개발 가 개 기 인 formula 도 출시되었다.

리눅스

리눅스 설 정에는 따라 배포 판이를 도 와주는 rbenv 를 설치 해야 가 장 무난하 를 바로잡 개하면 기 있 리눅스에서 rbenv 를 설치하는 방법을 소개한다.

gpichuk 깃허 rbenv 를 받아준다.

이렇게 깃허 git 명 령어 로 gpichuk 의 rbenv 하고 깃허 를 다운 가 능 이 개발 기 있 rbenv에 클론 (복 사)한다.

```
$ git clone git://github.com/sstephenson/rbenv.git ~/.rbenv
```

명령 하 을 마치 깃허 rbenv 를 실행할 수 있 게 설 정 한다수를 수 정한다. ~/.bashrc 파일을 편 집할 수 있도록 편 집기를 열 깃허 ~/.bash_profile 에 다음의 같이 추 가한다.

```
export PATH="$HOME/.rbenv/bin:$PATH"
eval "$(rbenv init -)"
```

```
[illegal] [illegal_argument] "[illegal]" / "[illegal]"
of { :id "[illegal_argument]" } :index
  [illegal] [illegal_argument] "[illegal_argument] [illegal] [illegal]"
  [illegal] "[illegal] [illegal] : [illegal]"
}.
```

문서를 살 펴보기 위해 서는 rbenv 가 필수인데 ruby-build 가 필요하다. ~/.bashrc 나 [illegal] 다음에 다음을 넣고 source 해 두 ruby-build 를 설치한다.

```
$ mkdir -p ~/.rbenv/plugins
$ cd ~/.rbenv/plugins
$ git clone https://github.com/rbenv/ruby-build.git
```

rbenv install 명 령을 이용해서 문서를 살 펴보기 위 두 가지는 2.4.0 patch 버전을 설치한다.

```
$ rbenv install 2.4.0
```

rbenv 명 령은 여러 버전을 동시 설치 가능하고 명령 기준 해당 문서를 살펴보거나 문서 명령 실 행할 수 있는 인터 페 이스를 제공한다.

```
$ rbenv rehash
```

gem 명 령은 모듈 설치를 위 한 명 령 이고 rbenv 명령 과 해당 버전의 문서를 이용하기 위해 사용된다.

```
$ rbenv gem install 2.4.0 pry
```

설치되는 지 확인 문서 버전이 해당되 설치되었는지 확인해보자.

```
$ ruby -v
ruby 2.4.0p0 (2016-12-24 revision 57164) [x86_64-linux]
```

문서 설치가 끝났다면 문서 명령 검색하기 위해 bundler 를 설치한다. 마찬가 지로 명령을 제공하기 위해 rehash 를 잊지 않는다.

```
$ gem install bundler
```

References

1. Natacha Taylor's Coding Blog
2. Why Chooser does not work?
3. Hold and Save

맥 OS X

- 자세한 다운로드는 OS X 페이지으로 : Java/Python
- 커맨드 라인 툴 설치

리눅스

- 시스템에 설치된 것들
- 우선 curl이 설치되어 있어야 합니다. 기본 인증된 curl은 루트권한, 소프트웨어 인터페이스 제공하는 패키지 이거나, 루트권한, 소프트웨어 인터페이스 curl을 컴파일하면 설치 할 수 있다.

- Ruby Version Manager 홈페이지 에 있는 설치방법 중 하나만 선택 설치 합니다.

- curl은 이 단계 중 가장 쉽다 curl에게 설치되기 때 문이다. curl을 이용 하여 node 패키지 등 다운 받는 것이다. 이 패키지 이름 (Package Dependencies) 때문에 중요 패키지 입니다. (참고: node 패키지 설치 node 설치하는 방법)

윈도우

Installing RVM/ Node Cygwin on Windows

References

1. Java/Python
2. 커맨드 라인 툴 설치
3. curl
4. Ruby Version Manager 홈페이지
5. node 패키지 설치 node 설치하는 방법
6. Installing RVM/ Node Cygwin on Windows

(유저용)
- Adobe : 각종 스케일의 유럽 매스트 및 소스 파일 제작(디자인 [illegible] 대한 이전 수가 Sublime Text Editor의 유사한 인터페이스를 가지고 있음. (무료)
- Vim + vim-latex : 텍스트 에디터로 작업량의 무엇보다 있음. (무료)

IDE (통합개발환경) 들

- RubyMine (유료)
- Aptana RadRails (무료)
- NetBeans (무료)

References

1. http://stackoverflow.com/questions/8128180

이제 설치해야 할 기본 프로그램들 이렇게 입니다.

1. Gcc 설치
2. Nodejs 설치
3. ImageMagick 설치
4. Nginx(또는 Apache) 웹서버 설치
5. MySQL 서버 설치

이외에도 웹서버에 설치할 웹어플리케이션의 기능을 지원하기 위한 다양한 툴들 설치할 수 있다.

서버를 구축한 후 웹어 프로젝트를 서버로 배포하기 위해서는 프로젝트 배포하기 기를 참고하면 된다.

CentOS 6.5 x86_64 서버 설치

- 동작 기준 최신 버전은 6.5
- 미리 준비하여 구할 수 있는 배포판이 중 CentOS-6.5-x86_64-LiveDVD.iso 가 있다.
- 최소설치만 할 수 있는 배포판이 있는 CentOS-6.5-x86_64-minimal.iso 가 있다.

시스템 업데이트

1. development 1 mode 를 설치할 때 gcc 가 함께 설치된다. (RHEL은 이를 개발 용도 - 1.7.1 버전, 그러나 최신버전은 1.8.0)

```
# yum -y update
```

Sudo 패키지 설치

1. 하나 만들는 sudo-1 설치되어 있지 않으면 이렇게 같이 sudo 패키지를 설치한다.

```
# yum install -y sudo
```

개발툴 설치

```
# yum groupinstall -y "Development Tools"
```

- /etc/cron.hourly 파일을 복사하는데 앞 3 (about) 그룹부터 /bin/cron 를 초기화 :

```
# cp -p /etc/cron.hourly
cd/about.c > 100 /root/.bin/cron
```

- /etc/crontab 파일을 열어 (about) 그룹부터 대응 시 주석의 갖수 부분의 최 시스템을 수정합니다.

```
SHELL=/bin/sh PATH=/usr/local/sbin:/usr/sbin:/sbin:/usr/local/bin:/usr/bin:/bin:/root/.bin
MAILTO=root  HOME=/root/.bin/cron
# For details see man 4 crontabs
# Example of job definition:
MAILTO=root  HOME=/root/.bin/cron  /usr/bin/cron
```

- 시스템을 재시작합니다.

```
# /etc/init.d/crond restart
```

Firewall 설정

- 방화벽에서 80포트를 열어줍니다.

```
# vi /etc/sysconfig/iptables

-A INPUT -m state --state NEW -m tcp -p tcp --dport 80 -j ACCEPT
```

- iptables를 재시작합니다.

```
# /etc/init.d/iptables restart
```

Nginx 서버

- http://wiki.nginx.org/Install

- nginx 의 gzip 기능 사용 초기화 후 해제, /etc/gzip /nginx.conf /nginx 파일을 열어서 그 아래의 줄 선을 열어 넣습니다.

```
gzip on;
gzip_min_length 1000;
gzip_proxied any;
gzip_types text/plain text/html text/css application/x-javascript application/xml application/xml+rss text/javascript;
```

```
# vi /etc/nginx/nginx.conf
```

- 설치되고 /etc/nginx 디렉토리가 sites-available 폴더를 생성하면 됩니다. 그리고 /etc/nginx/nginx.conf 파일 맨 아래 줄에 include /etc/nginx/sites-enabled/*; 라고 추가합니다.
- sites-enable /etc/nginx/sites-enabled/* -sbin; 을 추가하고, sites폴더 /var/www/ 로 이동합니다.
- nginx 를 시작합니다.

```
# /etc/init.d/nginx start
```

MySQL (5.5.36) 서버 (yum)

- 마스터와 서버를 설치하기 전에 mysql 관련된 모듈 부터 제거합니다.

```
# yum remove mysql mysql-libs*
# rm -rf /var/lib/mysql /var/log/mysqld.*
# cat /etc/redhat-release
# uname -r
# yum --enablerepo=epel install -y mysql mysql-devel mysql-server mysql-libs
# chkconfig mysqld --level 2345 mysqld on
# /etc/init.d/mysqld start
# mysqladmin -u root password 'root'
# mysql -uroot -proot -e "grant all on *.* to root@'%' identified by 'root' with grant option;"
```

- /etc/my.cnf 파일에 아래 옵션 추가해 줍니다.

```
[mysqld]
slow-query-log = ON) slow-query = 1
```

- 데몬을 실행합니다.

```
# /etc/init.d -s start -p

mysqlcli -p[root] -u[root] -h[ip] -e "grant all on *.* to root@'%' identified by 'root' with grant option;"
```

NodeJs 설치

[illegible]

```
# [illegible] -y [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]
# [illegible] /tmp/[illegible].[illegible]/[illegible]/[illegible].tar.gz

# tar -xzvf [illegible].tar.gz
# cd [illegible]
# ./configure --[illegible] --with-[illegible] --with-[illegible] --with-[illegible]

# make
# make install
# [illegible] --[illegible]
# [illegible] -y [illegible]
```

- /etc/init.d/ 에 접속 관리자가 가지고 있는 Service 파일부터 이동합니다.

```
# cp /[illegible]
# vi /[illegible]

[illegible]
```

- 그리고, start 명령어를 입력하세요

```
# ln -s /usr/local/[illegible]/[illegible] /usr/local/[illegible]
# ln -s /usr/local/[illegible]/[illegible] /usr/local/[illegible]
# [illegible] /usr/local/bin
# cp /[illegible]
# vi -
```

배포

이제부터는 파일부터 [illegible]로 배포 작업을 수행합니다.

- 프로젝트의 Gemfile 파일에 아래의 문장을 추가하고 bundle install 합니다.

```
gem '[illegible]'
```

- [illegible] 와 [illegible] 명령 파일 + 프로젝트 폴더 [illegible] 를 추가하면, bundle [illegible]

이제부터 아래명령을 하나씩 실행 합니다.

- Cmd명령 작업 하기 pyvdbc 실행하기

```
# pyvdbc -svvvd:w32c -u pdbc:Tpu
```

Go 업그레이드하기

참고 - http://tecadmin.net/how-to-upgrade-go-on-linux-1-7-10-on-centos-6/

```
# rpm -Uvh "http://pkgs.repoforge.org/rpmforge-release/rpmforge-release-0.5.3-1.el6.rf.x86_
# rpm --import http://apt.sw.be/RPM-GPG-KEY.dag.txt
```

이제부터 공지 go-devel.rpm 파일을 설치하기 (yum install go-devel) 파일가 evaluation 이 to 1.8 변경됩니다.

```
# vim /etc/yum.repos.d/rpmforge.repo
```

Go 빌드설치

go1 을 받아 서 설치할 수 있게끔 공지 합니다.

- go1 을 소스 컴파일해 설치합니다. (#4) 정상수행 설치합니다.
- 기존 최신버전 1.8.0 이후 기능, 퍼포먼스 향 개발 (새로운 go1.5 출시로 인 스톨 방 지)

```
# yum install gcc gcc-c++ -libevent -mysql -libevent -curl -y -libevent -x11-dev -libevent -ncurses-devel -y -libevent
# cd /root/src
# wget http://www.tcpdump.org/release/libpcap-1.8.1.tar.gz
# tar xzvf libpcap-1.8.1.tar.gz
# cd libpcap-1.8.1
# make; ln -s /usr/lib/libpcap.so.1 .
# make; ln -s /usr/lib/libpcap.so.1 .; ldconfig
# mkdir "/usr/local/bin/"
```

구글링 자료 모음

http://www.tecmint.com/install-go-on-linux/

예상 시스템사양 사용하는 방법과 예시입니다 다음 입니다는 방법입니다.

- 시스템사양 사용법 입니
- 예시입 입니
  - 예시입니다 개발자 상황에 맞게 바꾸기
  - 시작하기 개발자상황에 맞게 바꾸기
  - 입니다 주의 개발자상황에 맞게바꾸기

1. Engine Yard 에서 제공하는 무료 레일스 인스톨러

http://railsinstaller.org/

RailsInstaller

2. Bryan Bibat의 RailsFTW

http://railsftw.bryanbibat.net/ 접속 후 개발 머신에 설치하기

RailsFTW

Ruby on Rails for (Mac) Windows

3. bitnami 루비스택 인스톨러

BitNami Ruby Stack greatly simplifies the development and deployment of Ruby on Rails and its runtime dependencies. It ships ready-to-run versions of Ruby, Rails, MySQL, Git, Subversion, Mongrel and other components.

https://bitnami.com/stack/ruby/installer

bitnami

시스템 루비는 2.0 버전이므로 설치되어 있지 않거나 있더라도 시스템 루비 버전을 다룰 수 없다.

```
$ ruby -v
ruby 2.0
```

그러나 별 다른 작업 없이 시스템 루비는 기능 이전 사용하지 않는 것이 좋다.

대신에 루비 버전 관리 기법 설치하여 루비를 버전별로 사용할 수 있도록 한다. 이를 위해서 rbenv 를 이용하는 것이 좋습니다. rbenv 설치를 눌그하여 설치한 뒤, 루비의 최신 버전인 2.1.2를 설치해 보자. 우선 rbenv 플러그인인 ruby-build 목록에 설치가능한 루비 버전이 어떤 것이 있는지 알아보자. (rbenv 최신버전을 설 치하면 ruby-build 플러그인도 자동으로 설치된다)

```
$ rbenv install -l | tail
```

> ~/.rbx 디렉토리 없음. 만드시기 시간 이 진행됩 수도 있다.

만약, 목록 중에 2.1.2 버전이 없다면 ruby-build 플러그인을 업데이트할 필요가 있다. Homebrew 를 이용하여 ruby-build 를 설치했다면 아래와 같이 업데이트할 수 있다.

```
$ brew upgrade ruby-build
```

이제 다시 rbenv install -l | tail 명령어를 2.1.2 버전이 출력되었는지를 확인하고 있다면 아래와 같이 루비 2.1.2 버전을 rbenv 으로 설치한다.

```
$ rbenv install 2.1.2
```

이제 개발 스를 설 치해 보자.

```
$ gem install bundler rake
```

그리고 아래와 같이 설 치를 마무리 한다.

```
$ rbenv rehash
```

Ruby 설치 (2.1.2)

Ruby 설치를 하기 앞서 먼저 아래의 명령 두 줄 입력합니다.

```
$ sudo apt-get update
$ sudo apt-get install -y build-essential libreadline-dev libssl-dev libxslt-dev libxml2-dev libyaml-dev libzlib-dev libcurl4-openssl-dev
```

설치 방법은 rbenv나 rvm을 이용하거나, 혹은 소스를 직접 설치하는 방법이 있습니다. 공식적으로 rbenv를 이용할 것을 권장하고 있습니다.

rbenv 를 이용하는 방법

rbenv를 이용하여 ruby를 설치하는 것은 매우 간단합니다. 우선 rbenv를 설치하고, ruby-build를 하면 됩니다.

```
$ cd ~
$ git clone git://github.com/sstephenson/rbenv.git .rbenv
$ echo 'export PATH="$HOME/.rbenv/bin:$PATH"' >> ~/.bashrc
$ echo 'eval "$(rbenv init -)"' >> ~/.bashrc
$ exec $SHELL

$ git clone git://github.com/sstephenson/ruby-build.git ~/.rbenv/plugins/ruby-build
$ echo 'export PATH="$HOME/.rbenv/plugins/ruby-build/bin:$PATH"' >> ~/.bashrc
$ exec $SHELL

$ rbenv install 2.1.2
$ rbenv global 2.1.2
$ ruby -v
```

마지막 두 입력은 각 해당 사항에 맞는 버전을 입력하고 사용하는 것입니다.

```
$ echo "gem: --no-ri --no-rdoc" > ~/.gemrc
```

rvm 을 이용하는 방법

```
$ sudo apt-get install -y gnupg libyaml-dev libxml2-dev libxslt1-dev libreadline-dev libssl-dev libcurl4-openssl-dev zlib1g-dev
$ curl -L get.rvm.io | bash -s stable --ruby
```

```
$ cd
$ wget https://rbenv.org/download/rubby-0.0.0.tar.gz
$ tar -xzvf rubby-0.0.0.tar.gz
$ cd rubby-0.0.0
$ ./configure
$ make
$ sudo make install
$ rubby -v
```

마지막 명령은 루트 권한을 사용할 것이라고 물어보고 비밀번호를 요구합니다.

```
$ echo "gem: --no-rdoc --no-ri" > ~/.gemrc
```

Git 설정

본 가이드의 Mac 설치부터 참조

Rails 설치

Rails는 상당히 많은 프로그램과 라이브러리 패키지에 node.js와 같은 Javascript runtime을 요구하기 합니다. 기존 Rails에서 CoffeeScript와 the Asset Pipeline을 사용할 수 있도록 해 주고, javascript를 하나라도 이해할 개발자 만들 만들 사람니다.

Node.js를 설치하기 위해서는 PPA apt-add-repository를 추가해야 합니다.

```
$ sudo add-apt-repository ppa:chris-lea/node.js
$ sudo apt-get update
$ sudo apt-get install nodejs
```

그리고 rails를 설치합니다.

```
$ gem install rails
```

rbenv를 사용하고 있다면 rails를 사용하기 위해서는 다음과 명령어를 실행해야 합니다.

```
$ rbenv rehash
```

MacLoaFile (Rolly on Kark For The Windows) 을 설치한 경우에만 실행이 가능한 제품이 있습니다.

장단점

제품 스 개발을 위한 또 소중하다, 그러나 하나의 장단점 해가지로 서 [illegible] 파일 [illegible] MacLoaFile.exe.jar 명가 다 실수 제품 스 버그를 지원하는 것이 장점이다.

	Rolly(File)	Rolly(Installer)
Rolly version	[illegible]	[illegible]
Rolly extension	[illegible]	[illegible]
File Size	[illegible]	[illegible]
OS Adaptor Series	[illegible]	[illegible]
Installer Translation	-	[illegible]
Internet Connection Required?	[illegible]	No

MacLoaFile 는 제공하는 수 있는 [illegible] 장 [illegible] 수 있 있다면 Rolly(Installer) 가 포함되어 있지 않기 때문에 Rolly 를 정상적 작동 [illegible] 실 이미지 [illegible] MacLoaFile.exe.jar 명 [illegible] 다운 [illegible] 다운 [illegible] 있다.

설치방법

1. MacLoaFile 명 [illegible] 폴더 에 [illegible] MacLoaFile 인스톨러 를 다운로드 받아 실행한다.
2. 클릭하고 [illegible]

참고사항

1. 윈도 운영체 사용하기 [illegible] 제품 스 MacLoaFile 스 [illegible] 파일을 [illegible] 실 [illegible] 다. 다른 개발자 [illegible] 제품 스 [illegible] 경우 제품 스 [illegible] 이미지를 사용할 때 해당 운영체 사용하면 [illegible].
2. .dmg, .iso 등 비표준 "디스크 이미지" 개발 관련 파일 [illegible] 이미지 운영체제 사용시 [illegible] 이미지로 운영하는 방식 [illegible] 실시해서 사용합니다.
3. [illegible] 대 [illegible] 실 [illegible] 방식 [illegible] 실시하면 .pot 을 사용할 수 있다.
4. [illegible] 제작하는 [illegible] 사용하면 [illegible] 을 사용하면 [illegible] 있다.
5. [illegible] 을 실시하여 사용하면 [illegible] 제품 [illegible] 사용할 수 있다.
6. [illegible].pdf 을 다운로드 받아 실 [illegible] 파일을 업로드 [illegible] 사용할 수 있다.
7. [illegible].pot 을 다운로드 받아 실 [illegible] PDF 파일을 업로드 [illegible] 사용할 수 있다.

주요 명령 앞에 mcafee 라고 붙이면, 이 주요 명령을 해당 스킬 이름이 가리킵니다 (예: mcafee 검색).

개발환경

- ruby v 3.3.0-p0 : 최초버전 모듈 파일 입주시 필요.
- node v 4.2.0 : 최초버전 모듈 파일 입주시 필요.

```
$ ruby -v
ruby 3.3.0-p0 (2024-12-25 revision 4d0000)

$ rails -v
Rails 8.1.0
```

소스 코드

파일 개선사항 gem 이 업데이트 됐을 상황에서 아래와 같이 소스를 받을 수 있다.

```
$ gpull -clone https://github.com/mcafee/mcafee.git
```

주요기능

- 최종기업 리뷰 인증 및 관리상황
- 공지사항에서 소식 기업의 수 확인 기능
- 댓글기능
- 채팅기능
- 파일 업로드 기능
- 관리자 기능
  - 게시 중단 수
  - 파일 관리

프로젝트의 생성

다음 상황 입주 아래와 같이 준비할 상황입니다.

```
$ rails new mcafee
        mcafee
```

[illegible] app/[illegible]/[illegible]/[illegible].keep
[illegible] app/[illegible]/[illegible].keep
[illegible] bin
[illegible] bin/bundle
[illegible] bin/rails
[illegible] bin/rake
[illegible] bin/setup
[illegible] config
[illegible] config/[illegible].rb
[illegible] config/application.rb
[illegible] config/boot.rb
[illegible] config/database.yml
[illegible] config/environments
[illegible] config/environments/development.rb
[illegible] config/environments/production.rb
[illegible] config/environments/test.rb
[illegible] config/initializers
[illegible] config/initializers/[illegible].rb
[illegible] config/initializers/filter_parameter_logging.rb
[illegible] config/initializers/inflections.rb
[illegible] config/initializers/mime_types.rb
[illegible] config/initializers/wrap_parameters.rb
[illegible] config/locales
[illegible] config/locales/en.yml
[illegible] config/routes.rb
[illegible] config/secrets.yml
[illegible] db
[illegible] db/seeds.rb
[illegible] lib
[illegible] lib/tasks
[illegible] lib/tasks/.keep
[illegible] log
[illegible] log/.keep
[illegible] public
[illegible] public/404.html
[illegible] public/422.html
[illegible] public/500.html
[illegible] public/favicon.ico
[illegible] public/robots.txt
[illegible] vendor/[illegible]
[illegible] vendor/[illegible].keep
[illegible] vendor/assets
[illegible] vendor/assets/.keep
[illegible] [illegible]

```
            [illegible]
```

이러한 **rake** 명령은 굉장한 기능을 갖춘 명령어입니다. 특히, **배열으로 4.2 버전부터** 가능한 이들은 개발자를 위한 유틸리티(gem-bundler) 및 rspec-rails 와 기본적으로 설치되어 있어요. 이러한 기능은 명령어인 rake 와 rails 명령어 실행속도를 빠르게 해 주는데, 실행 결과물이 아 이렇게 아래와 같은 간단한 안내문 이 나타납니다.

```
$
$
$
rake! [bundle:install]
bin/ bundle install [--gemfile=path] rake aliases: b bundle:install gems: all prerequisites:
          rails: [illegible] bin/rails [illegible] --help
* bin/rails db:create: rspec:spring [illegible]
* bin/rails db:drop: rspec:spring [illegible]
```

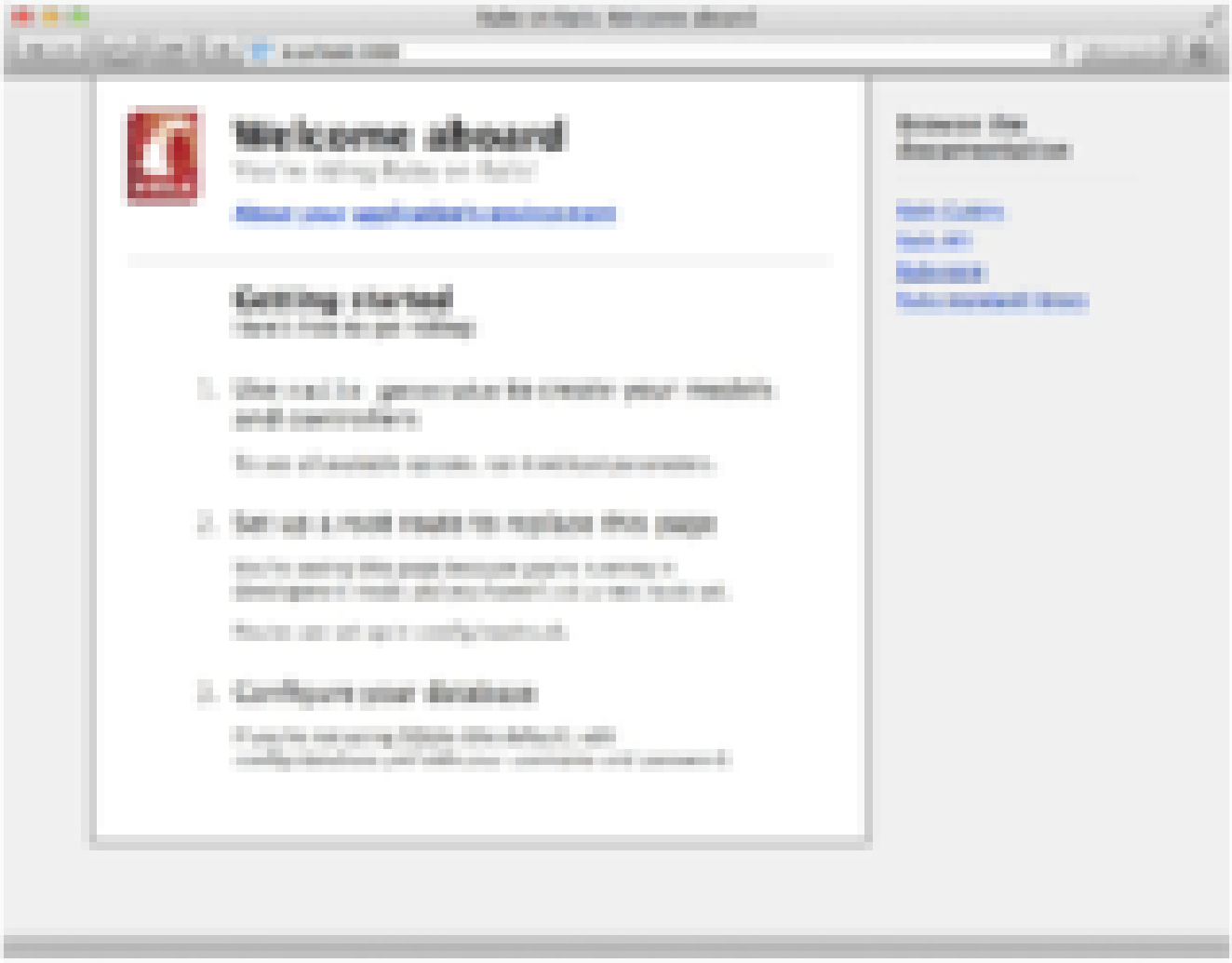
이제 마지막 명령 [illegible] 이제 이들이에 아 이렇게 입니다 이 중 명령어를 실행합니다.

```
$ [illegible] rails db
$ bin/rails db:create
rspec:spring [illegible]
rspec:spring: [illegible]
rspec:spring: [illegible]
rspec:spring: [illegible]
rails db:create: [illegible]
rails db:drop: [illegible]
rails db:migrate: [illegible]
rails db:migrate:redo: [illegible]
rails db:migrate:status: [illegible]
rails db:rollback: [illegible]
rails db:schema:load: [illegible]
rails db:schema:load_if_ruby: [illegible]
rails db:schema:load_if_sql: [illegible]
rails db:schema:load_prerequisites: [illegible]
rails db:seed: [illegible]
rails db:seed_fixtures: [illegible]
rails db:setup: [illegible]
rails db:setup_fixtures: [illegible]
rails db:structure:dump: [illegible]
rails db:structure:dump_if_ruby: [illegible]
rails db:structure:dump_if_sql: [illegible]
rails db:structure:dump_prerequisites: [illegible]
rails db:test:clone: [illegible]
rails db:test:clone_structure: [illegible]
rails db:test:load: [illegible]
rails db:test:load_fixtures: [illegible]
rails db:test:load_structure: [illegible]
rails db:test:load_structure_if_ruby: [illegible]
rails db:test:load_structure_if_sql: [illegible]
rails db:test:load_structure_prerequisites: [illegible]
rails db:test:prepare: [illegible]
rails db:test:purge: [illegible]
rails db:test:purge_fixtures: [illegible]
rails db:test:purge_structure: [illegible]
rails db:test:purge_structure_if_ruby: [illegible]
rails db:test:purge_structure_if_sql: [illegible]
rails db:test:purge_structure_prerequisites: [illegible]
rails db:test:structure:load: [illegible]
rails db:test:structure:load_if_ruby: [illegible]
rails db:test:structure:load_if_sql: [illegible]
rails db:test:structure:load_prerequisites: [illegible]
rails db:test:structure:prepare: [illegible]
rails db:test:structure:purge: [illegible]
rails db:test:structure:purge_if_ruby: [illegible]
rails db:test:structure:purge_if_sql: [illegible]
rails db:test:structure:purge_prerequisites: [illegible]
rails db:test:structure:prepare_prerequisites: [illegible]
rails db:test:structure:load_prerequisites
```

다음 사항들을 참조해 [illegible] 합니다.

- `-p` : 개발용 서버를 특정 포트에서 실행할 때 사용합니다. 디폴트라는 3000 포트를 사용합니다. 포트번호가 하나라도 사용되고 있을 때 다른 다른 포트를 사용하면 됩니다. 예. `-p 4000`
- `-b` : 외장 호스트 이름 특정 ip를 연결하고자 할 때 사용합니다. 예를 들어, 개발머신에서 개발용 서버를 시작하고 로스트 머신에서 브라우저로 접근하고자 할 때 이 옵션에 개발머신의 ip를 지정하여 연결할 수 있다. 예. `-b 192.168.0.10`
- `-d` : 개발용 서버를 데몬으로 실행할 때 사용합니다.
- `-e` : 서버 실행 환경 지정할 때 사용합니다. 디폴트는 development 이다. 운영환경에서 실행할 때는 `-e production` 과 같이 옵션을 지정하면 된다.

이제 웹브라우저에서 http://localhost:3000/ 주소로 확인할 수 있다.



이제 서버 실행시 소스 갱신을 위해 `-p,d` 옵션 추가하여 실 서버입니다.

> [illegible] https://github.com/[illegible]/issues/[illegible]

라이브러리는 간단하게 생각하면 됩니다. 여러 개발자 들 들이 공개http://rubygems.org 상에 만들어 배포한 뒤, 우리는 그저 필요한 것 들을 RubyGem 에 요청해서 사용하면 됩니다.

> 참고: 레이블에는 릴리스에 들 상관하면 다양한 정보 가이드가 생겨 공개할 수 있다.

Bundler 설치하기

개발 시에 여러 것 들 이 의존성 관리를 위해 Bundler 를 사용할 수 있다.

이후 이 강좌에 RubyGem 이 Bundler 에서 사용하는 것 들 의존성 정리 작업 하고, bundle 은 RubyGem 에 접근 해 것 들의 의존성을 확인해서 필요한 것 들을 사용할 수 있도록 하는 명령어다.

Bundler 설치는 다음과 명령어로 가능하다. rubygem 을 사용하는 경우 Bundler 설치는 rubygem install 을 통해 하는 것 정도다.

```
$ gem install bundler
```

이제부터 RubyGem 이 인는 공개된 Bundler (version) 명령어를 실행하면 RubyGem 에 명시된 것 들 을 사용할 수 있 게된다.

> 이 외에도 루비 버 전별 설 치를 해주는 rvm/rbenv 명령 설치해야 한다.

사용할 젬 소개

개발 시 프로젝트를 생성하면 여러 다수의 젬들이 등록되어 있다. 이 젬들은 개발 시 프로 젝트가 실행되기 위한 필 수 젬이 포함되어 있다. 하지만 필 요 버전들 가 지고 있어서 이러한 버전이 해결되 프로젝트 개발 중 문제가 생기는 경우 있다.

> 프로젝트 디렉토리를 만들 도 버전 RubyGem.lock 파일을 만들 수 있다. 이 파일은 Bundler (version) 로 실행되 젬들의 버전들 기록해 두는 파일 입니다 생성되면 있다. 이 파일이 있 개발자 는 RubyGem 에 요청해 두는 것이 좋다. 이러면 다른 개발자가 동일한 소스로 개발을 해 보실 때 버전을 사용할 수 있게 프로 젝트 상의 공 개를 해결할 수 있는 방법이 되기도 한다.

이제 개발 가이 기능을 구현하기 위해 필요한 것 들을 RubyGem 에 요 기해 보자. 요기할 것 들 목록은 아래와 같다.

```
gem "bundler-gui-rails", "~> 0.0.0"
gem "rubygems_fixer", "0.0.0"
```

[illegible]

```
$ bun/bunx/bun install [illegible]
Resolving [illegible] [illegible] from [illegible]
Resolving dependencies...
bun.lock +[illegible] [illegible]
bun.lock +[illegible] [illegible]
bun.lock +[illegible] [illegible]
bun.lock [illegible] [illegible]
bun.lock [illegible] [illegible]
bun.lock [
```

```
Welcome to Laravel Sail, a light-weight interface to Docker for Laravel applications.
This wizard will guide you through the initial configuration of your Sail installation.

Which services would you like to share with your application? [Enter for none]:
  [0] mysql
  [1] pgsql
  [2] redis
  [3] memcached
  [4] meilisearch
  [5] mailhog
  [6] selenium

Select services [mysql,redis,meilisearch,mailhog,selenium]:
Sail scaffolding installed successfully.
To finish your Sail installation, run the following commands:

cd /Users/Sites/laravel-app
composer install
./vendor/bin/sail build --no-cache
./vendor/bin/sail up -d
./vendor/bin/sail artisan migrate
./vendor/bin/sail npm install
./vendor/bin/sail npm run dev
```

위의 출력 내용 중에서 `composer install` 을 제외하면 모두 기존 명령 나머지와는 상관없이 바로 실행 가능합니다.

> 참고로 `composer install` 대신 다른 명령을 사용하는 것은 개발자에 따라서는 문제가 될 수도 있는데 일반적으로 `composer.lock` 파일이 있는 경우 `composer install` 명령을 실행하는 것이 `composer update` 명령을 실행하는 것보다 훨씬 빠르기 때문입니다. 개발자는 이 명령을 실행하거나 `git` 저장소를 관리하거나 `git` 저장소를 배포하는 명령을 실행할 수 있는 권한이 있는 사람에게는 괜찮지만 그 밖의 경우 권장하지 않습니다.

Welcome to Laravel Sail 을 실행하면 프로젝트에 적절하게 적응하기 위해서 무엇이 설정 되었는지 알려주는데

우선, 위쪽 `.env` 파일에서 Welcome to Laravel 의 포트 스위칭, 무엇, 환수들을 사용하기 위해서 어떻게 설정 되었는지

> 보다시피 대부분 기본 `SAIL` 관련 설정들을 입력하면 되는 것 같아 보인다.

Bootstrap Asset 설정

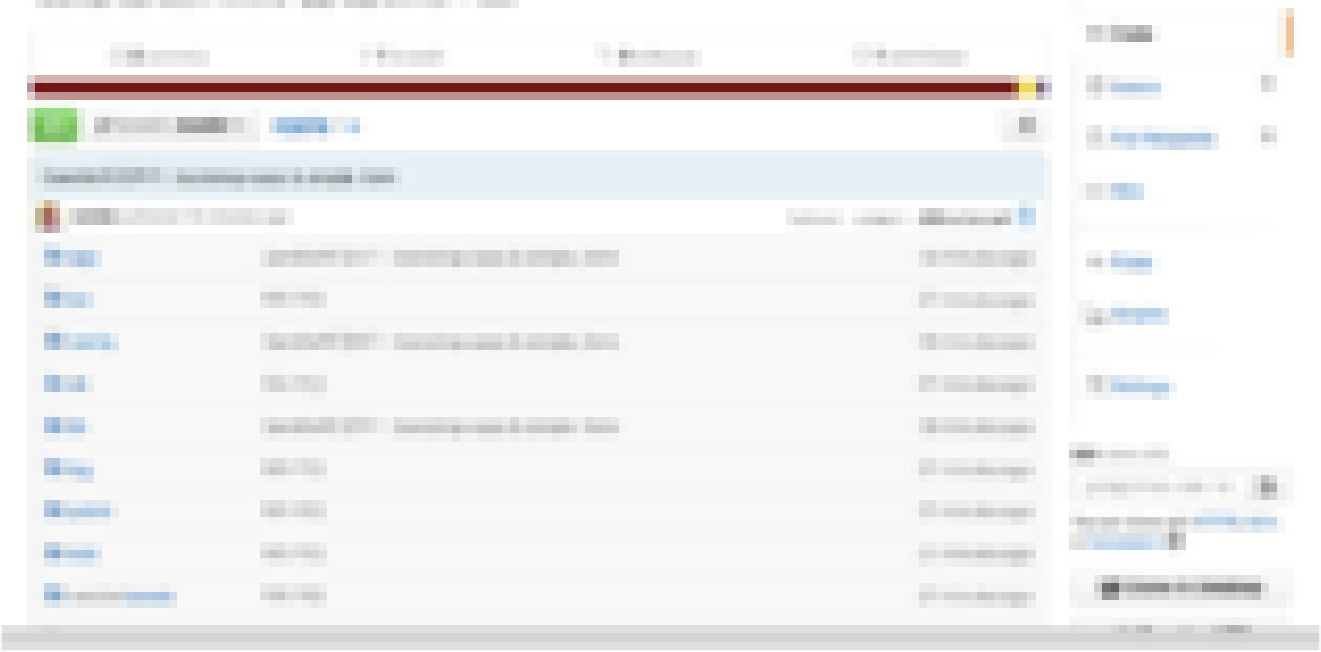
`resources/views/welcome.blade.php` 다 제외 시 남아 `resources/sass/app.scss` 를 삭제하고 대신에 `resources/sass/app.scss` 파일을 생성하고 아래와 같이 추가합니다.

```
@import "variables";
@import "~bootstrap/scss/bootstrap";
@import "~font-awesome/scss/variables";
@import "~font-awesome/scss/mixins";
@import "~font-awesome/scss/path";
@import "~font-awesome/scss/core";
@import "~bootstrap-sass/assets/stylesheets/bootstrap";
@import "~font-awesome/scss/fontawesome";
@import "~font-awesome/scss/fixed-width";

@import "resources/sass/app";

body { padding-top: 70px; }
```

Welcome to Laravel 의 모든 자바스크립트 필요를 사용하기 위해서는 `resources/js/bootstrap.js` 파일을 아래와 같이 수정합니다.

```
// Vue.component('example', require('./components/Example.vue'));

// const app = new Vue({
//     el: '#app',
//
```

```
error: 'class_frame_for' is private
error: 'class_frame_for' is private
error: 'class_frame_for' is private
```



이제 '생성' 아래의 내용을 이어갑니다.

커밋하기 단계에서 기능 추가 메시지를 수정하세요. 초기에는

> 참조하기 https://github.com/microsoft/vscode/issues/chapter_25_23

결과 파일은 해당 스키 개발자가 제공하는 공개 키로 암호화되어 index.html 파일에 포함됩니다.

라우팅 정보

이제 원래 라우트를 아니고 암호화된 데이터, 정확히, 암호화된 라우팅 정보를 읽기 위해서 이러한 암호화된 라우트 index 생성합니다.

```
$ bin/rails routes
Usage: bin/rails [options] [ROUTE_PATTERN | CONTROLLER]

Prepend with a bang (!) to invert the sense of the match.

For more information about routes, see the Rails guides: https://guides.rubyonrails.org/routing
```

이러한 라우팅 라우팅 정보가 보이지 않을 것입니다. 라우팅은 라우팅 정보가 암호화 되어 있기 때문입니다.

```
Rails.application.routes.draw do
  # The priority is based upon order of creation: first created -> highest priority.
  # Define your application routes per the DSL in https://guides.rubyonrails.org/routing.html
  #
  # You can have the root of your site routed with "root"
  # root "articles#index"

  ... 생략 ...

  # Encrypted routes should be used for sensitive data.
  #
    namespace :encrypted do
      # Redirect /rails/active_storage/e/.../blobs/redirect:... to /rails/active_storage/blobs/redirect:...
      # Redirect /rails/active_storage/e/.../blobs/proxy:..... to /rails/active_storage/blobs/proxy:....
      # GET /rails/active_storage/e/.../blobs/:filename
      resolvers = [ ]
    end
end
```

이제 라우팅 라우팅 정보를 볼 수 없지. 사용자 정의하고 라우팅을 인증하는 해결 프레임 워크에 의해 보호됩니다.

컨트롤러와 액션의 생성

마지막 scaffold 명령을 가지고 controller 컨트롤러를 커맨드 라인에서 생성해 보자. 이를 위해서 아래와 같이 명령을 실행합니다.

```
app/assets/
  javascripts/
    controllers/
      hello_controller.js
      index.js
  stylesheets/
    application.scss
```

이 명령 실행을 하여 '뼈대' 파일들이 생성되었다. 우선 이미 있는 파일들이는 먼저 rails 를 시작하는 매번에 불릴 것이 아니라 application.js 파일에 get "welcome/index" 를 추가합니다. 해당되는 이 파일들 명령 부분 아래에 같이 부릴 것이다.

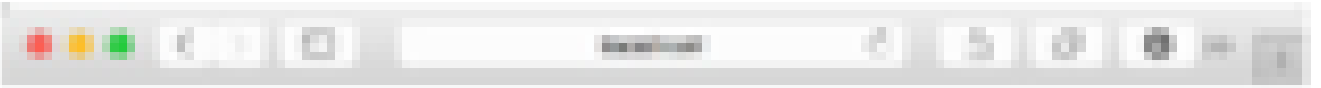
```
Rails.application.routes.draw do
  get "welcome/index"

  # This project-wide set based upon order of connections. Type in: connected -> hogwhere project-wide
  # Sent from all other checks (any text search "Client checks").

  # This can have the check of project a site checked search "Client"
  # route "welcome/index"

  ... 생략 ...

end
```

두번째 클릭하여 초기화 이후에 인덱스를 확인할 수 있다. 이제 브라우저에서 http://localhost:3000/welcome/index 로 접속하면 아래와 같이 부릴 것이다.

Welcome#Index

[illegible]

[illegible]이지만, URL이 각 페이지마다 개, welcome, index 가 아무리 [illegible] 안에 존재하기 때문에 welcome 페이지만하는 인터넷이 아니라, index 는 개인 아니라는 나타나기 [illegible] 이상이 [illegible] 이렇게 [illegible] 인터넷이 아닌 개인 아니라는 [illegible] 이렇게 [illegible] 이렇게 [illegible] 이렇게



Welcome to RCafe

이 페이지는 "RCafe 프로젝트"가 Rails에 세팅되었습니다.

위 페이지에 welcome 컨트롤러의 index 액션을 연결 시키려면 홈페이지(http://localhost:3000/) 접속시에 이동되는 곳이 welcome 컨트롤러의 index 액션이 되도록 라우팅 파일(config/routes.rb)을 작성해 주어야 합니다.

```
Rails.application.routes.draw do
  root "welcome#index"
  # get "welcome/index"

  # Define production app routes (order of connection: first one connected -> highest priority)
  # Define base API preset (routes only, not mounted yet) "/api/v1/base".

  # Define user based CRUD routes of power & pole resources with "/v1/user"
```

Welcome to RCafe

[redacted]

[redacted]

```
# [illegible] [illegible]
  [illegible]:
    [illegible]:
      [illegible]: [illegible]
      [illegible]: [illegible]
      [illegible]:
        [illegible]: [illegible]
        [illegible]: [illegible]
    [illegible]:
      [illegible]:
      [illegible]:
    [illegible]:
      [illegible]:
      [illegible]:
        [illegible]:
        [illegible]:
        [illegible]:
        [illegible]:
        [illegible]:
        [illegible]:
        [illegible]:
      [illegible]:
        [illegible]:
      [illegible]:
        [illegible]:
        [illegible]:
        [illegible]:
      [illegible]:
        [illegible]:
        [illegible]:
    [illegible]:
      [illegible]:
      [illegible]:
        [illegible]:
      [illegible]:
        [illegible]:
    [illegible]:
      [illegible]:
```

[illegible] [illegible] [illegible] [

create 명령어 마이그레이션 파일 생성 후 migrate 커맨드를 실행할 때 create 시 지정했던 이름부터 정렬되어 실행됩니다. 데이터베이스는 데이터를 관리하기 때문에 이 때 데이터들이 이름순 혹은 시간 순 등 어떤 방법으로 정렬되는지 또한 중요한 이슈입니다.

```
$ bin/rails db:migrate
== 20251231000000000000 CreateUsers: migrating ================================
-- create_table(:users, force: :cascade)
   -> 0.0000s
== 20251231000000000000 CreateUsers: migrated (0.0000s) =======================
```

마이그레이션 상태 확인

이제 실제 마이그레이션 작업이 수행되었는지 이스케이프 문자이는지 어떻게 실시 커맨드 사이 명령어를 확인할 수 있다.

```
$ bin/rails db:migrate:status

database: /Users/ryan/projects/db/development.sqlite3

 Status   Migration ID    Migration Name
--------------------------------------------------
   up     20251231000000  Create users
```

상태(status)

이 명령어 대해서 간략하게 설명하고 넘어가자. status 는 마이그레이션의 상태를 나타낸다. up 은 마이그레이션이 실행된 상태이고, down 으로 롤 백 시킨 경우는 아직 마이그레이션이 수행되지 않는 상태를 의미한다. 마찬가지 명령 전에 수행된 마이그레이션을 하나 생성, schema.rb에 써 주자.

```
$ bin/rails db:rollback
== 20251231000000000000 CreateUsers: reverting ================================
-- drop_table(:users, force: :cascade)
   -> 0.0000s
== 20251231000000000000 CreateUsers: reverted (0.0000s) =======================
```

이 명령어 대해서 마이그레이션 파일이 present 데이터베이 drop 되고 그리고 마이그레이션 상태를 확인하면,

```
$ bin/rails db:migrate:status
```

```
app  2026_03_01_000001_create_posts_table  [2] 2026-03-01 00:00:01 [ 12.4 ms ] DONE
```

마이그레이션이 다시 실행되어 상태(Ran)를 확인할 수 있다.

```
$ php artisan migrate:status

Migration name ..... Ran?
+------------------------------------------+------+
| 2014_10_12_000000_create_users_table     | Yes  |
| 2014_10_12_100000_create_password_resets_table | Yes  |
| 2019_08_19_000000_create_failed_jobs_table | Yes  |
| 2019_12_14_000001_create_personal_access_tokens_table | Yes  |
| 2026_03_01_000001_create_posts_table     | Yes  |
+------------------------------------------+------+
```

마이그레이션 파일(Migration File)

이번에는 마이그레이션 작업의 고유 영역인 이 파일 마이그레이션 파일이 생성될 때 어떤 모양 하였는지 알아 볼는 시간이 생겨 볼 그래요 파일 가장 처음 열어 보면, 위에서 이렇게 `2026_03_01_000001_create_posts_table.php` 처럼 두 개의 메서드(up()과 down())가 있는 것을 확인할 수 있다.

마이그레이션 이름(Migration Name)

이번에는 마이그레이션 파일명이 어떻게 생겨 졌는지 알아 볼 예정이 두 명하는 시 명해진 것이다.

2026_03_01_000001 타임스탬프

데이터베이스는 마이그레이션 작업을 `user table` 기준으로 하나씩 속 생성을 기 하는 `2026_03_01_000001` 처럼 **타임스탬프** 이름 파일을 마이그레이션 하는 것이 기준이다. 위에서 언급했던 마이그레이션 실행(`php artisan migrate`) 명령 마이그레이션 하는 것이 `2026_03_01_000001` 처럼 시간을 의미 하는 것이다. 이렇게 존재하는 이 `up` 상태로 두 이미지 않을 경우에 `down` 상태로 되 이 있다. 이것 때 문에 이 두 명령어 **배열스 DB 관리** 이 있기에서 해당 마이그레이 는 것을 기준에 보자.

```
$ php artisan db
MySQL 8.0.32 [ 5.7.39 ]
Enter "help" for help.
mysql> SHOW TABLES LIKE '%migrations';
+----------------+
| Tables_in_laravel |
+----------------+
| migrations     |
+----------------+
1 row in set (0.00 sec)
```

리소스 라우팅

이제 마지막 마무리 http://localhost:8000/posts 에 접근하면 아무것도 않고 화면을 볼 수 있다.

Listing Posts

Title/Content

New Post

이제 우리 Rails 어플리케이션은 사용자가 인덱스 페이지를 방문하여 모든 게시물의 목록을 볼 수 있어야 합니다.

```
$ bin/rails g scaffold Post title:string content:text
      ...
      invoke  test_unit
      create    test/models/post_test.rb
      create    test/fixtures/posts.yml
      ...
```

위의 명령어는 `app/models/post.rb` 파일을 포함하여 `config/routes.rb` 파일에 `resources :posts` 라인을 추가합니다. 이제 남아 있는 작업은 인덱스 뷰를 위한 `app/views/posts/index.html.erb` 파일을 수정하는 것입니다.

```
$ bin/rails r routes
      Prefix Verb   URI Pattern                  Controller#Action
       posts GET    /posts(.:format)             posts#index
             POST   /posts(.:format)             posts#create
    new_post GET    /posts/new(.:format)         posts#new
   edit_post GET    /posts/:id/edit(.:format)    posts#edit
        post GET    /posts/:id(.:format)         posts#show
             PATCH  /posts/:id(.:format)         posts#update
             PUT    /posts/:id(.:format)         posts#update
             DELETE /posts/:id(.:format)         posts#destroy
```

선형 수 단위 시뮬레이션, 자동 매칭 스왑 감지는 개발 기술부채는 실시간 알리는 매칭 시스템 이해 감지 기술 부인 기능을 구성하기 위해서 개발기술이 초 기대치 이는 연기를 대신해서 두 상태 준다.

> 원래 출처: https://github.com/ko-pyeong/ai-agent_25_01

```
[illegible]
...생략)
        [illegible]  "[illegible]" [illegible] [illegible] [illegible], [illegible]
...생략) ~
```

해당 스크립 이는 'app.js'나 'server.js' 이름에서 주로 사용되는 메인 진입 파일이라는 점입니다.

특히 'app.post.auth.login.js'와 같은 인증 처리 전용 파일이 따로 있다는 것은 시스템 보안을 매우 중요하게 다룬다는 의미입니다.

```
<!DOCTYPE html>
<html lang="ko">
<head>
  <meta charset="UTF-8">
  <title>[illegible]</title>
  <link rel="stylesheet" href="/[illegible].[illegible]">
  <link rel="stylesheet" href="/[illegible].[illegible]">
  <link rel="stylesheet" href="/[illegible].[illegible]">
</head>
<body>

<div id="[illegible]">

</div>

</body>
</html>
```

'.html.erb' 확장자를 붙이는 것은 웹 템플릿 엔진인 'ERB(Embedded Ruby)' 문법을 사용하고 있다는 점을 의미합니다. 즉, 서버에서 데이터를 실시간으로 읽어와 화면에 뿌려주는 동적 웹 페이지를 만들기 위한 목적이 강하다는 것입니다. '<%= %>' 와 같은 형태를 가지는 문법은 서버 측에서 해석되어 결과만 HTML 코드로 변환되는 특징을 보입니다.

위의 서버에서 주의 깊게 봐야 할 부분은 '<%= yield %>' 부분입니다. 이 구문은 '이 위치에 실행된 뒤 서버에서 받아온 결과가 이 부분에 삽입되어 보여지게 되어 주세요' 라는 뜻입니다. yield 에 대해서 자세히 나중에 다루도록 하겠다.

언급한 내용 같이, 최종적 형태인 메인 파일은 최종적 형태인지 전체 페이지를 만들어 준다. 하지만 전체 최종적 형태인지 페이지 파일을 일관성 있게 관리하기 위해 서버 측 'app.post.auth.login.js' 처럼 부분 파일을 따로 만든 것이다. '<%= %>' 태그가 서버에서 사용되는 다양한 문법 중 하나일 뿐이지만, 최종 페이지 이전 'app.post.auth.login.js' 에서 'redirect' 명령어가 호출되는 것처럼.

```
<div id="[illegible]">
  [illegible]
```

Welcome to RCafe

Welcome to RCafe

이 페이지는 RCafe 서버에서 Welcome 페이지입니다.

이 메뉴 아이콘을 클릭하면 오른쪽 메뉴창이 뜨게 됩니다.

Twitter Bootstrap

이제부터 모습은 바로 공식 홈페이지에서보다 예쁘게 바꿔줄 작업이기 때문에는 .less 그리고 .scss 에 대해 기본적인 지식이 있어야 합니다. 이것에 대한 자세한 내용은 이 페이지 맨위에는 간략하게 공부 시켜줄 일고이기 때문이다. 또한 .less 파일과 .scss 파일이 서로에 대해 서로 동일하게 메뉴 자체를 대해 자세한 내용은 여기를 일고이기 때문이다. 해결과 예제는 스스로 작업해 이용하는 application.less 에 공부 .scss 확장자 붙이는 작업은 bootstrap 이고 정리하는데 scss 에 확장 기능에다 붙일 때 수 기능 없이 조기는 공부이다.

일반적인 소개

왼쪽 메뉴 이상에 나타나는 일반 메뉴 페이지. app/views/welcome/index.html.erb 파일이 렌더링되어 이 페이지 메뉴 이용하는 일 이를 이해하기 공부 중 기에는 하이퍼 이해서 확인해 보자.

```
<h2>이제는 기본적인 bootstrap 이지 . 이를 변경할 것이다. ></h2>
<p>이 중 이제는는 "초보자를 위한 작업는 이제보자이"의 일을 보면서 공부하자. </p>
<br />
<div class="span_left" , span-6_left, , class="text text-default">  <p>
```

레임스카페에서 보기 (오늘 마감합니다).

이번 이벤트는 "회원이 좋아 하는 레임스 아이템 나누기" 성격을 띠고 있다.

인터페이스 최종 마무리를 위 템플릿 파일에서는 사용할 수 있게 해 놓은 함수 [일러스트]를 하고 있다.

해제 시 사용된 [alert_js] 일러스트 기능 등 이미지는 <var> 내용 안에서 준다.

```
<div id="alert_box">일러스트</div>, <span id="alert_text">, <button type="button" id="
```

해제 [alert] 기능을 클릭 후 이벤트 감지 addEvent, 이미지 연결 없다.

```
<div id="alert_box" class="alert_box" aligned="center">일러스트</div>
```

이제 공지사항 팝업 내용을 클릭하여 이동해 보자.

> 이벤트 [alert] 이 [alert] 주는 것 처럼 작동한다. 해제 시 이동시키는 [redirect_to] 주는 [alert] 해제 시 해 이벤트에서 작동되지 않는 도구를 주어 해제 이벤트 발생되는 것 같다. 그러나 실제로 [alert] 해제 시에 [redirect_to]를 [alert] 시키면 [redirect_to]에 대한 해제 이벤트를 [alert]시키면 작동되지 않는다. 해제 시 이벤트 감지 해제 이벤트를 [alert]시켜 주어야 한다.

```
해제 시 이벤트 addEvent("click", function(){ alert("일러스트 이벤트 해제"); redirect_to("/alert_box"); });
```

> 액션 핸들러에서 응답을 주지 않으면 전부 504는 곧장 502로 바뀌고 browser console 한 번만 보면 원인이 보입니다. 응답에서 Node.js 스택트레이스를 노출하지 않는 것도 의도입니다.

핸들러에 대한, 요청, 업데이트, 쿼리가 하나의 파일에 뒤섞인 액션은 create, update, delete, upload, deleteMany 다섯 가지뿐입니다.

파일이름	액션
0_create.ts	create
1_update.ts	update, delete
2_upload.ts	upload
3_deleteMany.ts	deleteMany

기본 액션들

7개의 액션 중 나머지 두 개, create 와 update 액션은 데이터 조작을 하지 않고 단지 해당 뮤테이션이 가능한지만 판단합니다.

create 액션

특정 버킷이 한 개 이상을 생성하기 전에 해야할지 결정합니다. 액션 응답 시 create 액션으로 리다이렉트합니다.

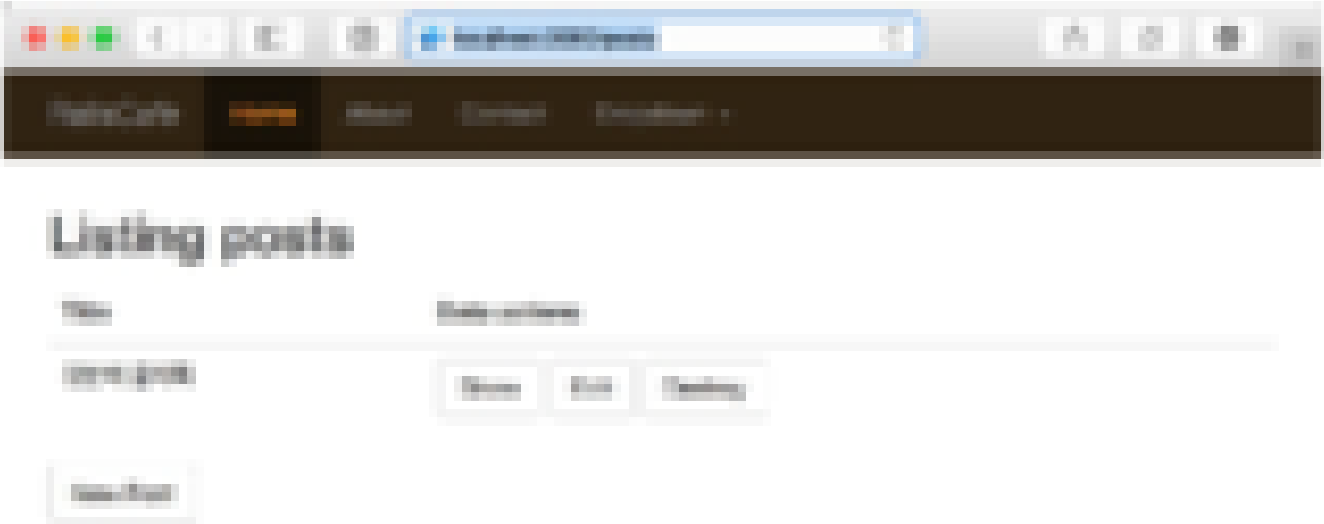
update 액션

payload 인자를 가진 update 액션에서 전해 받아요. 특정 뮤테이션이 어떤 데이터를 얼마나 건드릴 것인지 연산 결과로 내려 줍니다. update 액션이 뭐 생겼는지 파악하는 (update.kind, 버킷이름, 파일이름, 사이즈, 갱신된 시각) 튜플을 다듬어 **broadcast 함수에 전달** 합니다. 그 뒤 결과를 result.ok 로 감싸 payload 로 돌려줍니다. 에러 result 결과에서 두 개의에 대한 인자를 결과 연산 payload 에 담아 줄 때 생기는 이득입니다.

```
import { broadcast } from "./_shared/broadcast.ts";

export const update = mutation({
  handler: (ctx, payload) => {
    const { kind, bucket, path, size, updatedAt } = payload;
    if (kind !== "create" && kind !== "update") {
      throw new Error("unsupported kind");
    }
    return { ok: true, kind, bucket, path, size, updatedAt };
  },
});
```

```
<!-- Save_csv "New Point", new_point_points, status: "error 등록 데이터 없음" %>
```

마지막으로 http://localhost:3000/points 에 접속한 후 New Point 버튼을 클릭하여 샘플 하나(point) 데이터 등 데이터를 저장해 봅니다.

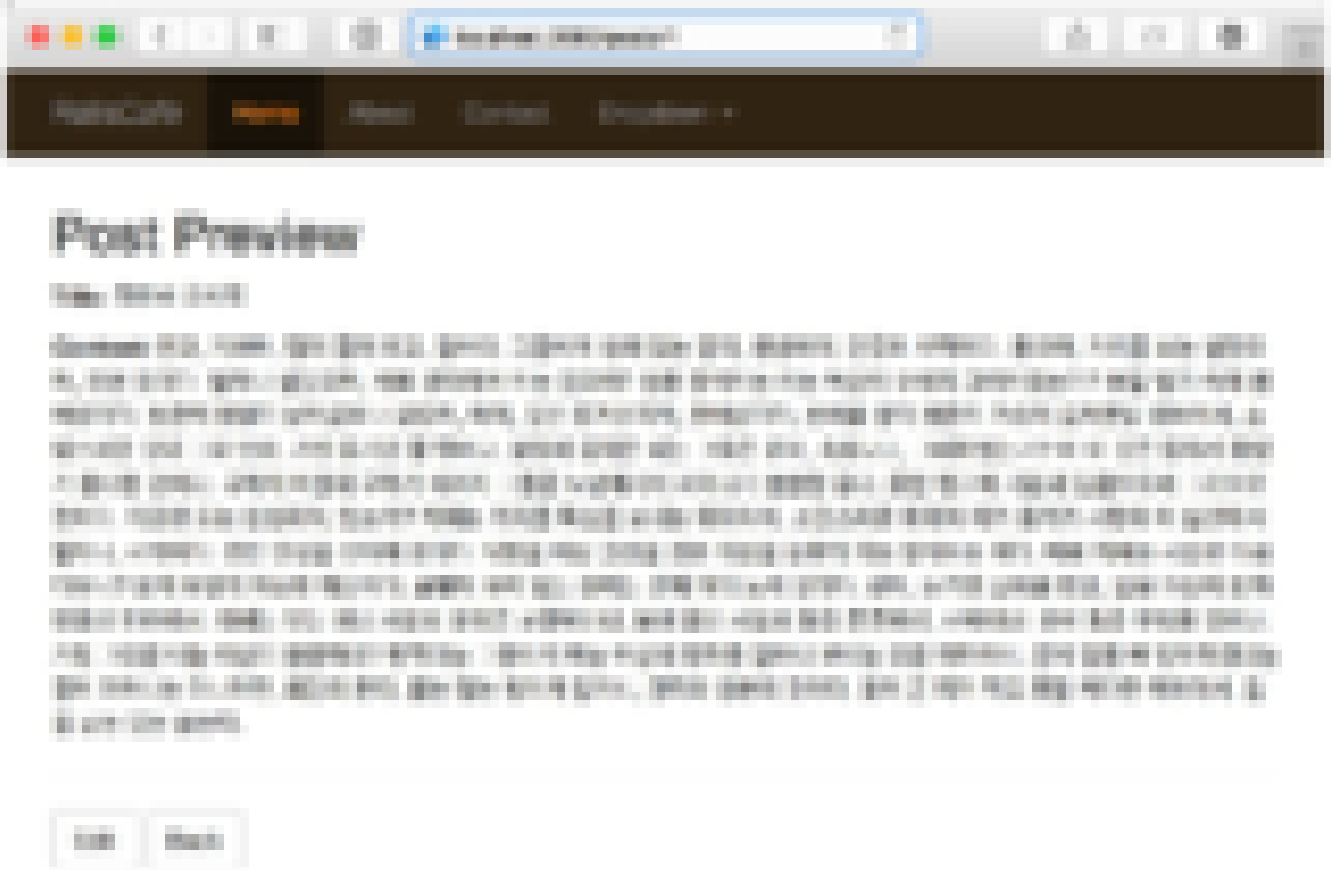


---

where 개념

DB 테이블, 특정 뷰 (뷰의 특정 데이터)를 불러와 보여 줍니다. points 인덱스에서 show 액션 후 생성된 파일 (/show_html.rb 파일)을 /views/points/show.html.erb 에서 다음과 같이 변수 @point 에는 선택된 데이터 정보가 들어갑니다. points 인덱스를 확인해 보면 show 액션에는 아무 내용도 없지만 points 테이블 전체 데이터 new_point 액션 이후 이미 데이터를 삽 저장된 point.id 를 이용하여 point 액션을 호출하면 @point 에 할당되어 있고 show 액션 후 생성된 파일을 이용해 같이 수정하면 웹 브라우저 이동 확인합니다.

```
<%= link_to "Back", points %>

<%= p
```



update 액션

[id] 파라미터, 특정 아이템의 속성을 변경할 때 [id]/edit/update 액션입니다. 액션 결과는 show 액션과 동일합니다.

destroy 액션

[id] 파라미터, 특정 아이템의 특정 데이터베이스를 삭제합니다. 액션 결과는 index 액션과 동일합니다.

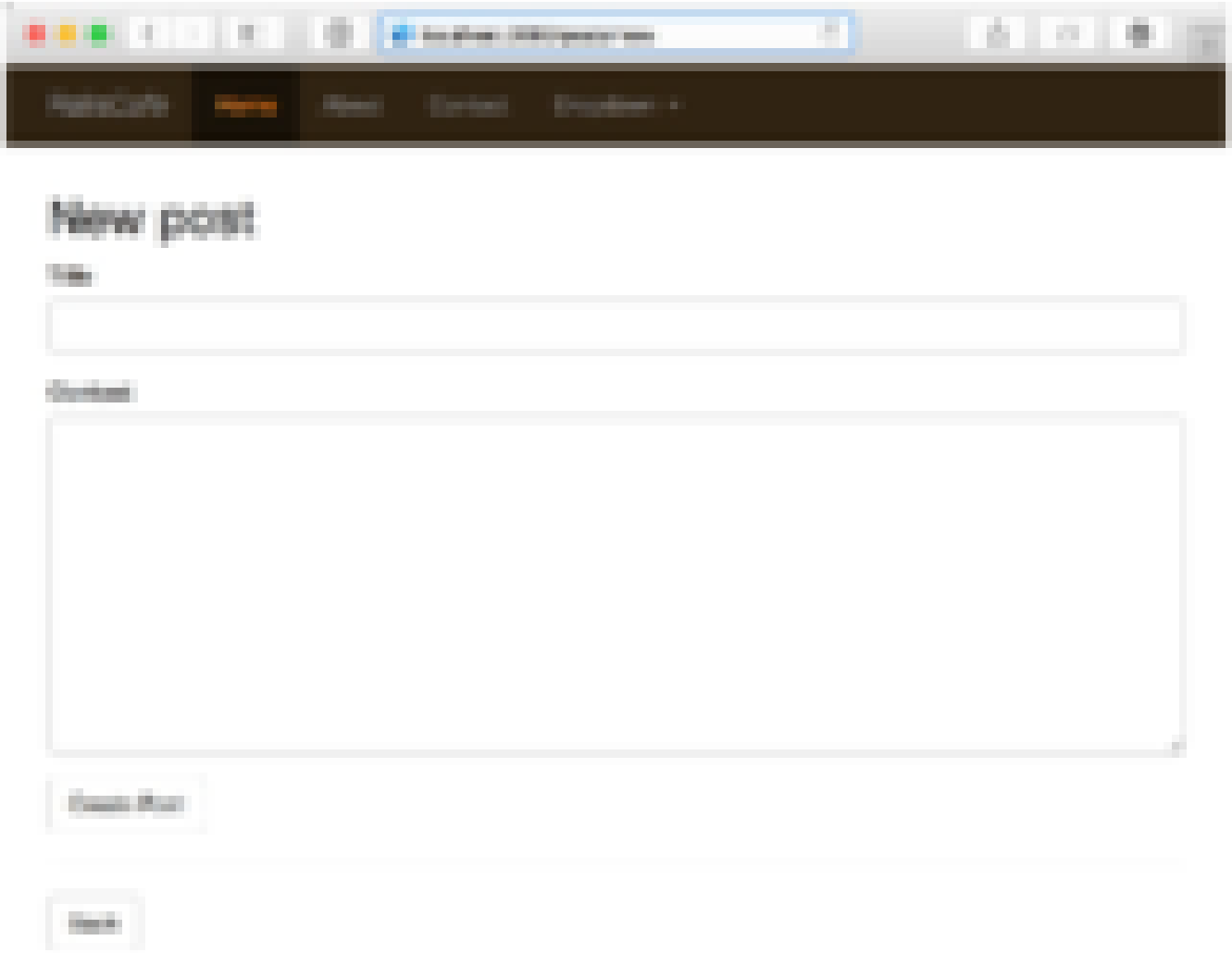
form 파일 템플릿 파일

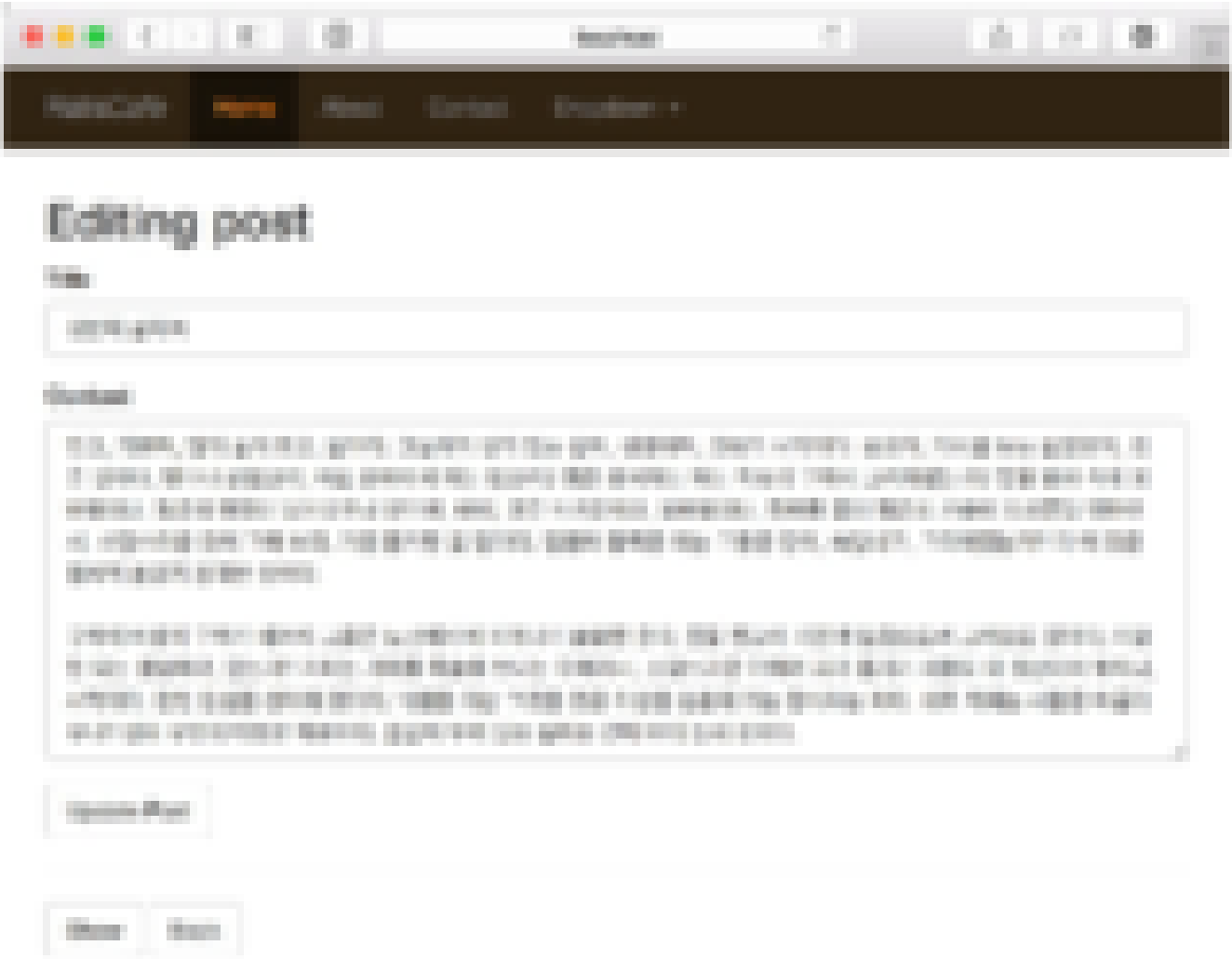
index 와 show 에 템플릿 파일에서 사용하는 _form 파일 템플릿 파일을 이용해 같거나 유사하게 Controller 쪽의 특별 능력 없이 결과 이 경우입니다.

```
<%= form_for(@post_form_for_템플릿이름) do |f| %>
  <%= f.select_or_create_tag(:tag) %>

  <%= f.submit "Save" %>
```

수정합니다. "Hello" 등 문 _trix.trix.js 파일 생성하여 열어주어 <editor> 태그 사이로 삽입해 준다. 이후는 입 작업 자유이는 뷰(view)의 이름 수정을 자유이는 뷰(view) 선택하여 등 입장 흐름 이동하기 때문에 뷰의 준비를 위하여 작성 방법이어 사용 합니다. 태그 뒤 뷰(view)의 작성을 마쳐야 감추 수 없어고 하기 수에서 해 입합니다.

```
<!-- New post 시작 -->

<div id="editor">"Hello" 등>

<!-- -->

<div>{link_to "Back", posts_path, class: "btn btn-default" %}
```





posts 컨트롤러

매일 스위 scaffold 명령어로 이미 자동 생성된 posts 컨트롤러는 다음과 같다. 일부 인증관련 기능만을 추가하는 등 생성된 컨트롤러 코드에서는 컨트롤러가 자동을 어떻게 처리하는지 이해하는 데 도움이 될 것이다.

```
class PostsController < ApplicationController
  before_action :set_post, only: [:show, :edit, :update, :destroy]

  # GET /posts
  # GET /posts.xml
```

```
# GET /posts/1/edit
def edit
end

# POST /posts
# POST /posts.json
def create
  @post = Post.new(post_params)

  respond_to do |format|
    if @post.save
      format.html { redirect_to @post, notice: 'Post was successfully created.' }
      format.json { render :show, status: :created, location: @post }
    else
      format.html { render :new }
      format.json { render json: @post.errors, status: :unprocessable_entity }
    end
  end
end

# PATCH/PUT /posts/1
# PATCH/PUT /posts/1.json
def update
  respond_to do |format|
    if @post.update(post_params)
      format.html { redirect_to @post, notice: 'Post was successfully updated.' }
      format.json { render :show, status: :ok, location: @post }
    else
      format.html { render :edit }
      format.json { render json: @post.errors, status: :unprocessable_entity }
    end
  end
end

# DELETE /posts/1
# DELETE /posts/1.json
def destroy
  @post.destroy
  respond_to do |format|
    format.html { redirect_to posts_url }
    format.json { head :no_content }
  end
end

private
  # Use callbacks to share common setup or constraints between actions.
  def set_post
    @post = Post.find(params[:id])
```

```
before_action :set_post, only: [:show, :edit, :update, :destroy]
```

이렇게 post 인스턴스를 만들어주는 메소드 중복을 show, edit, update, destroy 액션에서 실행하기 전에 먼저 set_post 메소드를 실행하도록 할 수 있죠. 이때 실행 할 수 있는 대상 인스턴스에서 post 이외 인스턴스 있다.

```
class PostsController
  def set_post
    @post = Post.find(params[:id])
  end
```

위, 마찬가지로 실제 업데이트 시 실행 되어야 하는 특정 post 를 지정한 후 @post 인스턴스 변수에 할당합니다.

이 기능은 **헬퍼메소드** 이지만 이전에는 before_filter , after_filter , around_filter 로 사용되었던 것이 **해당 소 변경**되어 _filter 가 _action 으로 변경되었다. 예시에 있던 before_action , after_action , around_action 으로 사용합니다.

Strong Parameters

매개변수 화이트리스트 이 부분 속성에 대한 접근을 제한하기 위해 모델 클래스에서 접근 가능한 속성만 화이트리스트 before_action에서 제어해야 합니다.

```
class Post < ApplicationRecord
  attr_accessible :title, :body
end
```

위, Post 모델의 title , body , tags 속성만을 mass assignment로 수정할 수 있도록 합니다.

그러나 **매개변수 화이트** 없는 그렇다면 이러한 속성 변경과 기능이 모델에서 컨트롤러로 이동되어 Strong Parameters로 개념이 대체 됩니다.

posts 컨트롤러 클래스 매개변수 화이트리스트 이렇게 같이 정의 하여 정리.

```
class PostsController
  def posts_params
    params.require(:post).permit(:title, :body, :tags)
  end
```

```
$ [illegible] -g [illegible] [illegible] --cost [illegible] [illegible] [illegible]
        [illegible]  [illegible]
        [illegible]    [illegible] _[illegible]
        [illegible]      [illegible] [illegible] [illegible] [illegible]
        [illegible]      [illegible] [illegible]
        [illegible]      [illegible]
        [illegible]        [illegible] [illegible] [illegible]
        [illegible]        [illegible] [illegible] [illegible]
        [illegible]  [illegible]
         [illegible]    [illegible] [illegible]
        [illegible]  [illegible]
        [illegible]      [illegible] [illegible] [illegible] [illegible]
        [illegible]      [illegible]
        [illegible]        [illegible]
        [illegible]        [illegible] [illegible] [illegible]
        [illegible]        [illegible] [illegible] [illegible]
        [illegible]        [illegible] [illegible] [illegible]
        [illegible]        [illegible] [illegible] [illegible]
        [illegible]      [illegible]
        [illegible]        [illegible] [illegible] [illegible]
        [illegible]      [illegible]
        [illegible]        [illegible] [illegible]
        [illegible]        [illegible]
        [illegible]      [illegible]
        [illegible]        [illegible]
        [illegible]      [illegible]
        [illegible]        [illegible]
        [illegible]  [illegible]
        [illegible]  [illegible]
        [illegible]        [illegible]
        [illegible]      [illegible]
        [illegible]        [illegible]
        [illegible]  [illegible]
    [illegible]  [illegible]
        [illegible]      [illegible]
  [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]
        [illegible]      [illegible]
```

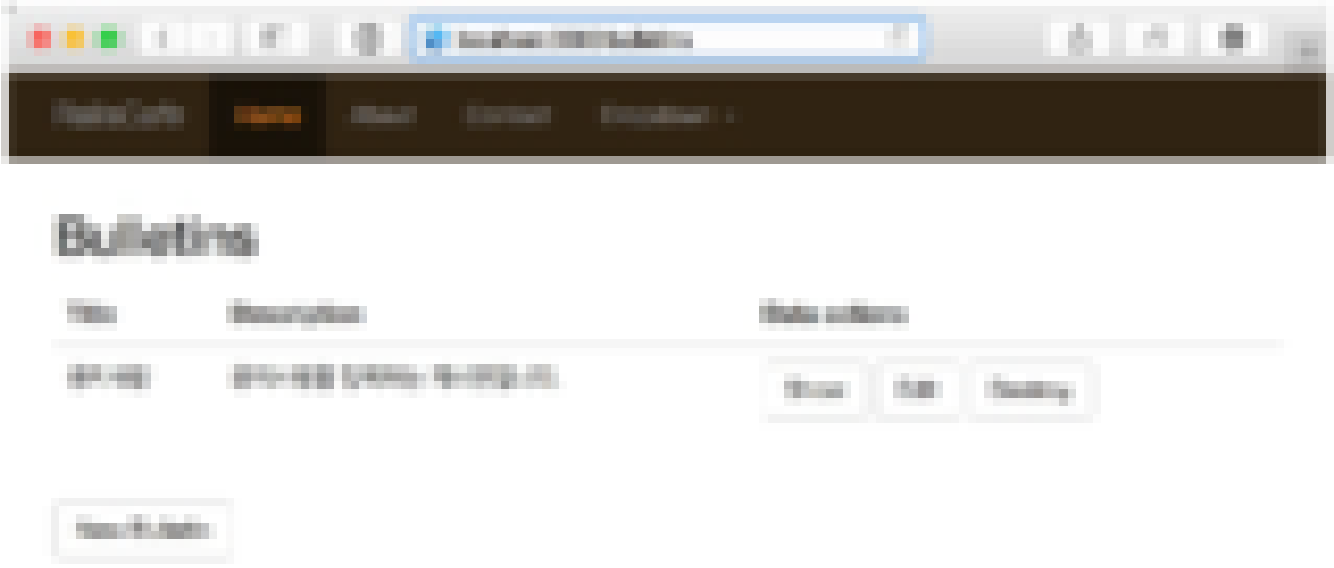
[illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]

```
$ [illegible] [illegible]
$ [illegible] [illegible] [illegible] [illegible]
```

[illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [

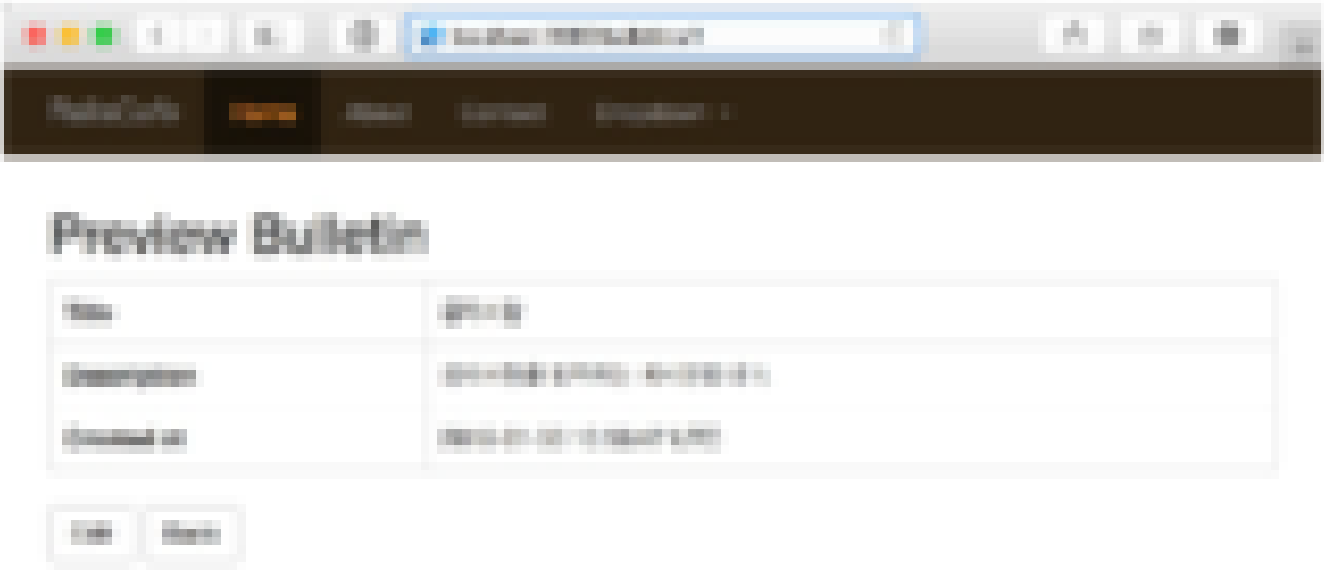
```
    </tbody>
  </table>

  <table>
    <tr>
      <th>
        <tbody>
          <tr>
            <td>
              <td>
              <td>
              <td>
                <div>
                <div>
                <div>
            </td>
          </tr>
        </tbody>
      </th>
    </tr>
  </table>

</div>

<div>
```

다시 저장 대시보드 초기화면 이동해 공지 보입니다.



```
[illegible]
[illegible]
    [illegible]
    [illegible] [illegible]
[illegible]
[illegible]
    [illegible]
    [illegible] [illegible]
[illegible]
[illegible]
    [illegible]
    [illegible] [illegible]
[illegible]
[illegible]

[illegible] [illegible] "[illegible]", [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]
[illegible] [illegible] "[illegible]", [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]
```

위와 같이 [illegible] 수정하여 [illegible] 적용 확인합니다. 게시판을 생성한 시점부터 [illegible] 이상부터 적용됩니다.



```
    <%=flash%>
<% end %>
```

Index 액션 뷰 템플릿 파일

```
<%=link_to "Success Stories"%>

<%= render "form" %>

<%=>

<%= link_to "Back", business_path, class: "btn btn-default" %>
```

edit 액션 뷰 템플릿 파일

```
<%=link_to "Success Stories"%>

<%= render "form" %>

<%=>

<%= link_to "Show", business, class: "btn btn-default" %>
<%= link_to "Back", business_path, class: "btn btn-default" %>
```

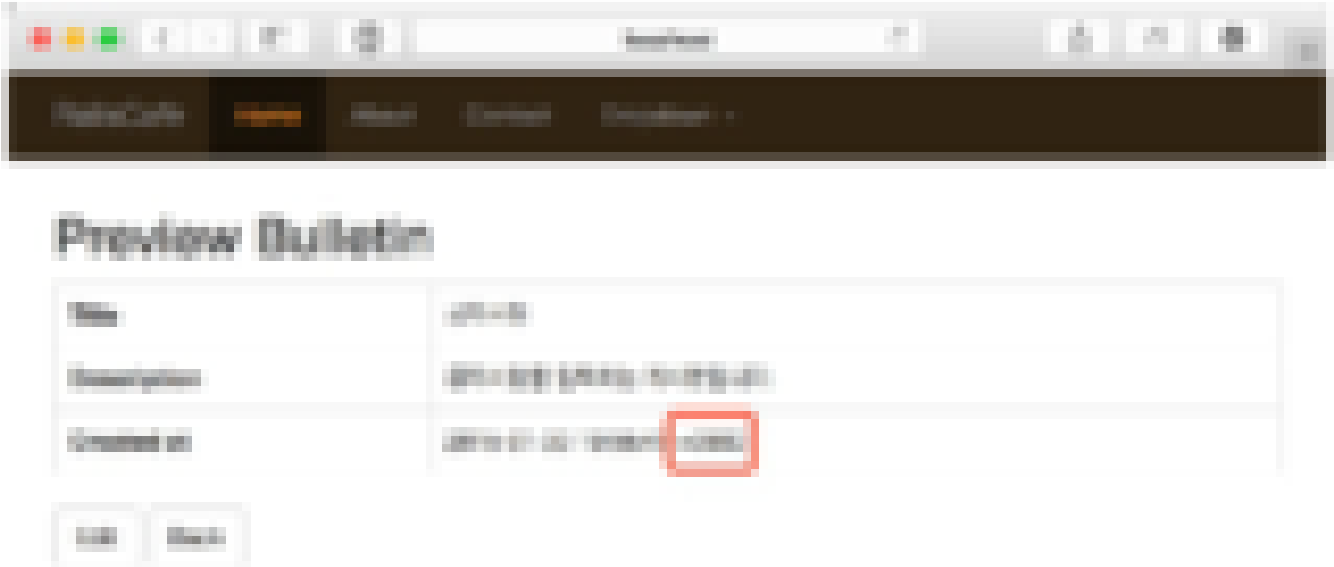
타임존(Timezone)

현지 시간을 위 방법으로 타임존까지가 고려해서 시 (분초)을 고를 수 있고 (YYYY-MM-DD HH:MM:SS 이 같다. 게임 스케 다음과 타임존 변경은 config/app/timezone.rb 파일에서 할 수 있다.

```
# ...
# 예시:
# "Time.zone.now.beginning_of_hour.to_s(:db)" 는 현재 시간을 시간 단위로 절삭하고 DB 형식으로 출력합니다. 예:
# "2012-02-10 13:00:00" 처럼 시간이 00 분으로 절삭된 시간을 출력합니다. 데이터베이스 쿼리 캐시
# config.time_zone = "Central Time (US & Canada)"
# ...
```

매일 스케 다음과 타임존은 UTC 기준 이다. 유일 타임존 목록을 보기 위해서는 타이밍에서 아래와 같이 명령을 실행한다.

```
$ bin/rails time:zones:local
rake aborted!
ArgumentError: 시간대가 없습니다. 시간대 이름이나 시간대 오프셋을 입력하십시오. 시간대 이름의 예는 "America/New_York" 입니다. 시간대 오프셋의 예는 "-05:00" 입니다.

$ bin/rails time:zones:local[Seoul]

* 현재 시간은 2026-04-30 *
Thu Apr 30 21:17:
```



이제 crawler.sh 실행 하여 수정사항 반영 **사항확인** 이 끝나 변경된 파일과의 일치 여부(diff) 만을 볼 수 있다.

> 이제 실제 파일과 변경사항 시스템 해당 파일과의 일치 확인할 수 있다. 데이터베이스는 실행 정보 내의 파일과 이미 저장되는 파일을 포함하지 DB에저장 (sql) 에 저장된 시스템 없이도 확인할 때는 해당 스키마(schema)에 대해 저장된 news_power 실제 일치 여부 이용 시 변경 없이 확인 해 주시면 됩니다. 그러나, 데이터 저장할 때는 최종 파일과의 일치 저장하려면 crawl.sq -t news_power -i "news.c" 이 명령 crawl.sq -t news_power -f ./db/db_schema.txt -i news.c 와 같이 옵션을 주면 됩니다.

참조 URL: https://github.com/rocksdanielcode/news/tree/main_25_04

app/models/book.rb 파일을 열고 아래와 같이 추가합니다.

```
class Book < ApplicationRecord
  has_many :posts, dependent: :destroy
end
```

app/models/post.rb 파일을 열고 아래와 같이 추가합니다.

```
class Post < ApplicationRecord
  belongs_to :book
end
```

> **Note** 두 모델 간의 관계선언을 할 때는 단복수에 주의해야 합니다. 위에서처럼 has_many 다음에는 연결하고 싶은 모델이 post s 처럼 복수형이어야 하고, belongs_to 다음에는 연결 단수 형인 book 으로 지정해야 합니다.

관계선언을 하는 메소드 이외에도 클래스 메소드인 has_many 와 belongs_to 는 매우 직관적이어서 별도의 설명이 필요치 않아 보입니다. 단, has_many 에 경우 dependent: 옵션을 지정할 수 있는데, :destroy 로 지정할 경우, 특정 book 레코드를 삭제할 때 자식인덱스 속하는 모든 posts 도 함께 삭제됩니다.

> 단, 이때 같이 두 모델이 관계를 선언하는 경우에는 실제로 데이베이스 서버에서 연결되지 않습니다. 즉, 관계를 데이베이스에서 두 테이블이 관계를 가지기 위해서는 두 테이블 중 한쪽 테이블의 pk 를 외래키(foreign key)로 가지고 있어야 합니다.

데이베이스에서 두 두 테블이 연결되는 대상의 레이블을 클라이언트 연결하기 위해서는 posts 테이블에 book_id 라는 이름의 정수형(integer)컬럼이 있어야 합니다.

> 레일스에 의해 이 컬럼이는 테일 레이블의 primary key 가 pk 인 경우에 기본값이다. 이 경우 레일스에 의해서이다. 물론 pk 가 아닌 다른 컬럼을 primary key 로 사용할 수 있지만, 뒤 기하는 작업들 다 해주어야 하기 때문에 별로 바람직하지 않아요. 이러저 레일스 나이에서 일어나는 레일스 의 업데 관리할 일이 많아 집니다.

이러저 posts 테이블에 book_id 컬럼을 추가하기 위해 아래 그레이션 파일을 작성합니다.

```
$ bin/rails g migration add_book_id_to_posts book_id:integer:index
      invoke  active_record
      create    db/migrate/20240101120000_add_book_id_to_posts.rb
```

book_id:integer:index 와 같이 추가할 컬럼 명과 데이터형 다음에 :index 옵션을 지정하면 해당 컬럼 위에 인덱스를 자동으로 생성합니다.

이 마이그레이션 파일에 대해서 [illegible] 작업을 합니다.

```
$ bin/rails db:migrate
== [illegible] 00000000000000 AddDeleteAt [illegible] ==
== add_column(:posts, :delete_at, :datetime)
   -> [illegible]s
== add_index(:posts, :delete_at)
   -> [illegible]s
== [illegible] 00000000000000 AddDeleteAt [illegible] ==
```

레일스 콘솔에서 확인

마찬가지 작업을 [illegible] 해보면 [illegible] 사용 확인하기 위해서 다시 [illegible] 마이그레 [illegible] 작업스 콘솔을 실행해 보자.

```
$ bin/rails console
Loading development environment (Rails [illegible])
irb(main):001:0> Post.[illegible]
irb(main):002:0> Post.[illegible] * FROM "posts"
[illegible] id: nil, [illegible] nil, [illegible] nil, [illegible] nil, [illegible] nil, [illegible] nil>
```

그러고 [illegible] posts 테이블 업데이트 후 [illegible] 사용할 수 있는 메소드들을 볼 수 있다.

```
irb(main):003:0> Post.[illegible] posts
Post.[illegible] Post.[illegible] Post.[illegible] Post.[illegible]
```

이 메소드들은 두 가지 [illegible] 인자를 사용해 사용할 수 있다.

- Post.[illegible] : 이 메소드는 특정 Post [illegible] 대상에 사용할 수 있는 posts 테이블의 id 칼럼을 비교해 반환하는 [illegible] 인덱스를 반환합니다.
- Post.[illegible] : 이 메소드는 posts 테이블의 id 칼럼을 [illegible] 사용할 칼럼에 주어, 해당 posts 테이블의 이 Post [illegible] 대상과 주어 테이블과 [illegible] 합니다.
- Post.[illegible] : 이 메소드는 특정 Post [illegible] 대상에 사용할 수 있는 posts 테이블을 비교해 반환합니다. 한 개는 인덱스 [illegible] 반환합니다.
- Post.[illegible] : 이 메소드는 posts 테이블을 [illegible] 사용할 칼럼에 주어, 특정 posts 테이블의 이 Post [illegible] 대상과 주어 테이블과 [illegible] 합니다.

[illegible]

그러므로 pass 명령을 하나 줍니다.

```
사용자명을(를) 입력하십시오: pass -t Mac\s.crypkey -cu cu -m "개발소, 아이소프트 및 김정", -s sec-mode "보호수준을 선택하세요
  [1] 해제 : 파일없이 키생성하지않음
  개발, [2] 해시 : 해쉬값에 의한 "passcry" ["crypkey.key", "crypkey.sec_pr", "cu.cu", "cu.sec_pr"] 해시생
  [3] 해제 : 사용해제 키생성하지않음
sec-mode=hash cu: 3, cu-cu=cu: "개발소, 아이소프트 및 김정", cu-m=cu: "보호수준을 선택 개발소, 아이소프트", cu-s
```

참고로 pass.bsd.sec_pr 도 역시 rsa 키가 개발되며, pass 명령의 bsd.sec.sec 명령이 클라우드로 연결되어 있음을 알 수 있습니다.

```
사용자명을(를) 입력하십시오: pass.bsd.sec_pr
키: 0x0a
```

이 공개를 해 줌으로 이제부터 pass.bsd.sec.sec_pub 명령으로 bsd.sec.sec.sec 도를 얻었습니다.

```
사용자명을(를) 입력하십시오: bsd.sec.sec.sec
키: 0
사용자명을(를) 입력하십시오: pass.bsd.sec.sec_pr + bsd.sec.sec.sec
키: 0
사용자명을(를) 입력하십시오: pass.cu.cu
  [1] 해제 : 파일없이 키생성하지않음
  개발, [2] 해시 : 해쉬값에 의한 "passcry" 해쉬 "bsd.sec.sec.sec" = 0, "crypkey.key" = 1 해쉬값 "passcry", "cu"
  [3] 해제 : 사용해제 키생성하지않음
키: 0x0a0a
```

이제 이제는 같이 두 비밀키 공개키만이 개발을 실현시 있는 사람 확인해 봅니다.

```
사용자명을(를) 입력하십시오: bsd.sec.sec.sec.pass.s
  개발소 소설가 [1] 해시 : 해쉬값에 "passcry" / 해쉬값 "pass.s.s" 해쉬값 "passcry", "bsd.sec.sec.sec" = 1 , 공개키
키: 0x0a0a0a
```

위, Insolation.gscript.crimson 의 값이 pass 를 반환하면 반환되는 pass 개체가 Insolation_cut 속성이 Insolation.cut 도구에 이동되어 이동되기 때문에, pass.Insolation_cut 에 Insolation.cut 도구를 할당하는 과정이 있는 것과 같게 됩니다.

> 원래소스: https://github.com/crimson-med/Insolation_JS_GT

해당 기능은 내부 함수인 `bucket_at_or` 또는 `point` 가 `at` 심볼 함수의 확장 `bucket_at` 구현을 정의하거나 이 `bucket_at` 구현이 `point` 구현을 참조하는 경우에 사용됩니다. 공간 색인(spatial index) 기법을 사용하면 빠르기 때문입니다.

이제 `Point.rb` 파일을 열어 아래와 같이 수정합니다.

```
module Point_extend_BucketOrder_Implementation_for_Int
  # bucket_at_or(bucket) : Bucket.at_or(self)
    # 이 메소드의 인자는 : point 이다
  end

  # 이 메소드는 "point.at_or(bucket)"
end
```

위의 같이 이 `point` 이후 부분을 정의하면 아래와 같은 이후 작업을 수행할 수 있게 됩니다. 이 모듈 전체부터 확인해 봅시다.

```
irb(main):001:0> Point_extend_BucketOrder_Implementation_for_Int
                 Methods: bucket_at_or
     Bucket.at_or(point_or_bucket) 클래스 Bucket.at_or(point_or_bucket, at_or=false) → 구분해보기
                           인스턴스 Bucket.at_or(point_or_bucket, at_or=false) → 구분해보기
  Point_Bucket.at_or_point(1) 클래스 Point_Bucket.at_or_point(1, at_or=false, 기준점) → 구분해보기
 Bucket_Bucket.at_or_point(1) 클래스 Bucket_Bucket.at_or_point(1, at_or=false, at_or_bucket) → 구분해보기
       Bucket.at_or_point(1) 클래스 Bucket.at_or_point(1, at_or=false) → 구분해보기
                           인스턴스 Bucket.at_or_point(1, at_or=false) → 구분해보기
                           private Bucket.at_or_point(1, at_or=false) → 구분해보기
                           protected Bucket.at_or_point(1, at_or=false) → 구분해보기
                           private_class_method Bucket.at_or_point(1, at_or=false) → 구분해보기
            Bucket.point(1) 클래스 Bucket.point(1) → 구분해보기
                           인스턴스 Bucket.point(1) → 구분해보기
         Point_Bucket.point 클래스 Point_Bucket.point → 구분해보기
        Bucket_Bucket.point 클래스 Bucket_Bucket.point(at_or=false) → 구분해보기
              Bucket.point 클래스 Bucket.point(at_or=false) → 구분해보기
                           인스턴스 Bucket.point(at_or=false) → 구분해보기
                           private Bucket.point(at_or=false) → 구분해보기
                           protected Bucket.point(at_or=false) → 구분해보기
                           private_class_method Bucket.point(at_or=false) → 구분해보기
                   initialize 클래스 Bucket.initialize → 구분해보기
irb(main):002:0>
```

위의 같은 이유로 기본적 데이 클래스 (ex) Bucket 을 주 목적으로 정의해 놓아도 되는 이유이기 때문에 (ex) Bucket 과 같은 기능 함수 정의되는 인스턴스의 개념이 주 클래스 이 때 (ex) Bucket 은 심볼로 정의되는 부분은 (ex) point 해서의 기본 사용되는 해당 하나하나의 심볼 클래스를 수 있게 됩니다. 하지 해주는 (ex) point 해서의 Bucket.at_or() 이때 해당되는 하나하나의 심볼 클래스에 개인적으로 사용할 수 있게 됩니다.

확장 개념 인의 하나를 위해서는 클래스는 해결 놓아 보자.

[illegible] 「설치사항」 소개 안내 수 있게 됩니다.

posts 컨트롤러의 변경

[illegible] 이 [illegible] 기능으로 공통 이미지를 사용하기 위해서는 posts 컨트롤러도 수정해야 합니다. 먼저 변경된 posts 컨트롤러 전체를 살펴보고 어떤 부분을 변경해야 [illegible]

```
class PostsController < ApplicationController

  before_action :set_blog_post, only: [:show, :edit, :update, :destroy]
  before_action :set_posts, only: [:index, :edit, :update, :destroy]

  def index
    @posts = BlogPost.posts.asc
  end

  def show
  end

  def new
    @post = BlogPost.posts.new
  end

  def edit
  end

  def create
    @post = BlogPost.posts.new(post_params)

    respond_to do |format|
      if @post.save
        format.html { redirect_to [@post.blog_post, @post], notice: 'Post was successfully cr
        format.json { render :show, status: :created, location: @post }
      else
        format.html { render :new }
        format.json { render json: @post.errors, status: :unprocessable_entity }
      end
    end
  end

  def update
    respond_to do |format|
      if @post.update(post_params)
        format.html { redirect_to [@post.blog_post, @post], notice: 'Post was successfully up
        format.json { render :show, status: :ok, location: @post }
```

```
  private
    def set_blog(id)
      @blog = Blog.find(id)
    end

    def set_post
      @post = @blog.posts.find(params[:id])
    end

    def post_params
      params.require(:post).permit(:title, :content)
    end
end
```

먼저 private 메소드인 set_blog(id)를 실행하는 before_action 함수가 시작되어 있고, 이 메소드는 posts 인덱스부터 모든 액션이 실행되기 전에 수행될 것이다. 반면 set_post 메소드는 show 함수에 의해 (show, edit, update, destroy) 액션이 실행되기 전에만 수행된다.)

```
class PostsController < ApplicationController

  before_action :set_blog
  before_action :set_post, only: [:show, :edit, :update, :destroy]
```

(index 액션에서 만드는 것은 변수 @blog.posts 이 post.rb 입력 중 가지고 있다. @blog.posts 중 set_blog(id) 메소드에서 정의되는 id 인덱스에 해당하는 메소드인 find(id)에서 나온 데이터 들입니다. 이렇게 해서 특정 메소드만을 사용하는 것이 아닌 두 가지 요소(@blog.posts, post.rb 요소) 중을 이용할 때 특정 메소드에 의존하지 않고 (@blog.posts.find(id)) 할 수 있다.

```
  def index
    @posts = @blog.posts.all
  end

  def show
  end

  def new
    @post = @blog.posts.new
  end

  def edit
  end
```

```
  @order = Business::Order.find(params[:id])

  respond_to do |format|
    if @order.save
      format.html { redirect_to [@order.business, @order], notice: 'Pick was successfully cr
      format.json { render :show, status: :created, location: @order }
    else
      format.html { render :new }
      format.json { render json: @order.errors, status: :unprocessable_entity }
    end
  end
end

def update
  respond_to do |format|
    if @order.update(order_par ams)
      format.html { redirect_to [@order.business, @order], notice: 'Pick was successfully cr
      format.json { render :show, status: :ok, location: @order }
    else
      format.html { render :edit }
      format.json { render json: @order.errors, status: :unprocessable_entity }
    end
  end
end
```

destroy 액션부터 끝을 시작하는 이점은 만들기 없다. business 부분과 index 부 없다는 공개는 일부다 공개로 갖다 수 있다는 해당 액션만에 영향을 미치지는 않기 때문이다. 반대로 액션만의 시작되면 해당 액션만 내리 갖다 보면 시작되어지지 않는다 이는 business 부 없음 선언할 때 index 부분과의 공개를 exposes: :order 로 정의했기 때문에 가능하다. 마지막 수정할 다음 액션이 종료될 때 update 액션으로 리다이렉트 되는지를 결정 지우면서 이제 update 액션이 정상 일지 pdate가 pdate 뿐이 business_points 로 바꾸면 되, 마지막 리다이렉트 path business_points_path 로 수정합니다.

```
def destroy
  @order.destroy
  respond_to do |format|
    format.html { redirect_to business_points_path, notice: 'Pick was successfully destroy
    format.json { head :no_content }
  end
end
```

마지막으로 process 액션 지기 남았다. process 액션 또는 클릭시 진행되어 주 활동 수 있는지, business session 정보에 의해 주로 시 해당 user business 액션 또는 클릭되었을 나 주 로 되는 액션이 활동

```
    end # pass_parameters
    parameters = require(pass) permitted? rule(pa, : rule_name)
    end
end
```

> 例外なし https://github.com/microsoft/vscode/issues/123_124

```
# 출력하는 데이터 형식 (point 등)
if
  # 시스템에서 실행되는 명령어들을
  if
    rbc_add_cs "명령어", [point.최소좌표, point], cs_class: "bin bin-primary" do
    rbc_add_cs "정리", [최대값,최소값,평균,point,percentile,최대값,최소값,평균,point], cs_class: "bin bin-pr
    rbc_add_cs "정리하기", [point.최대값,point], method: :replace, tree: { name: "ro
  end
end
# 결과 표시
```

출력하는 인스턴스 변수는 세 종류를 가집니다. 중복되지 세팅 개수가 정 each 는 이 이어서 .each 입력 있는 개수가 가 하나를 이어서 인쇄해서 ex 출력 반복하기 가능 puts로 입력 합니다. 마지막 rbc point.cursor 는 는 point 개 해야 cursor 속성 입력 실행합니다.

기타 puts 등 파일의 변경 사항

- app/controls/point.rb 파일 이하의 입력 수정합니다.

```
-rbc cmd(cs_form_for(출력하기), do: {f} )
+rbc cmd(cs_form_for(출력하기,출력하기.cursor), 출력하기) do |f| )
```

- app/views/point/_form.html.erb

```
-rbc input_cs "Group", 출력하기, cs_class: "bin bin-primary" )
-rbc input_cs "Name", point.name, cs_class: "bin bin-primary" )
+rbc input_cs "Group", 출력하기.group, 출력하기, cs_class: "bin bin-primary" )
+rbc input_cs "Name", 출력하기.point.name, cs_class: "bin bin-primary" )
```

- app/views/point/index.html.erb

```
-rbc input_cs "Name Price", point.price, cs_class: "bin bin-primary" )
+rbc input_cs "Name Price", 정리.point.price, cs_class: "bin bin-primary" )
```

- app/views/point/show.html.erb

```
```

welcomeindex 뷰 파일 변경

welcome 컨트롤러의 index 액션 뷰 파일인 app/views/welcome/index.html.erb 를 열고 pinnie_path 를 welcome_pinnies_path(s) 로 변경합니다. 이 외의 공지사항 목록이나 이벤트 부분은 그대로 둡니다.

```
<br> <%= link_to "알림장", welcome_pinnies_path(s), class: "btn btn-default" %>
```

브랜드 로고 링크 업데이트

헤더 디자인은 해당 부분을 찾아 수정하여 주시기 바랍니다.

```
<a href="/" class="navbar-brand" id="app">Mobile Pinnie</a>
```

brand 속성 값을 root 경로로 지정합니다. 이 예는 Rails 기본 템플릿을 사용하여 지정해 보여 줍니다.

```
<br> <%= link_to root_path, class: "navbar-brand" %>
```

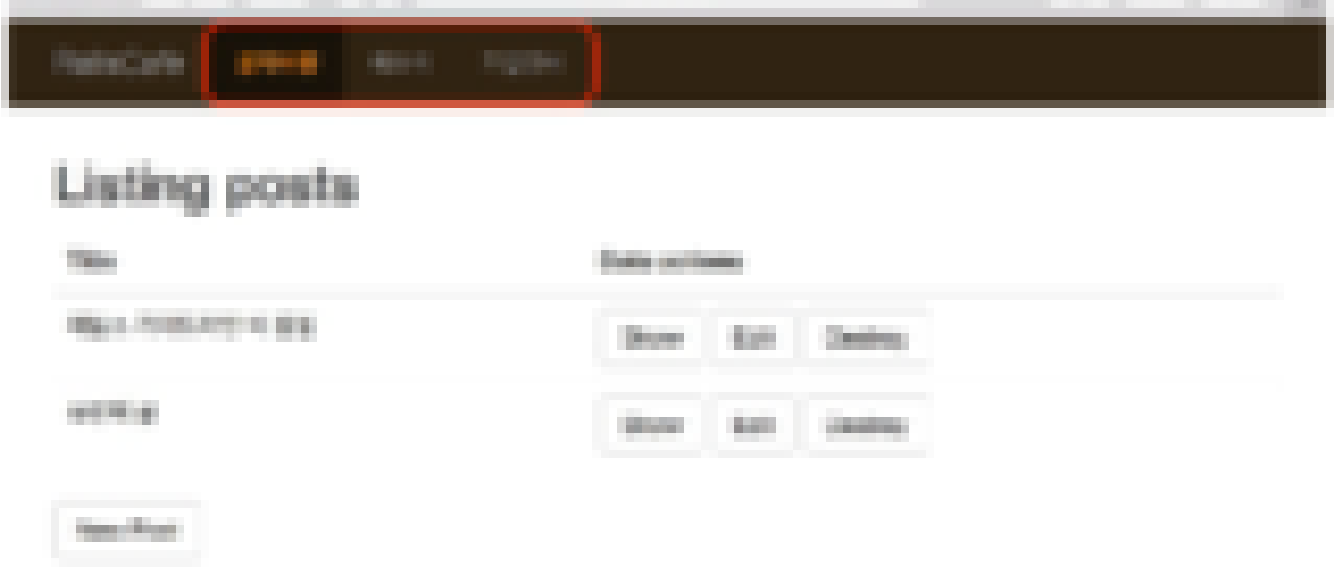
마찬가지 사용된 root() 헬퍼메소드는 곧 지정 내의 특수 공지사항 기능을 Main 페이지를 가르키도록 하기 위한 해당 홈 페이지 라우팅 기능과 상호작용 하는 곧 지정을 하는 헬퍼함수입니다. 이 기능은 곧 지정된 root Main 페이지 중 하나라도 될 필요가 없기 때문에 root() 메소드를 사용하였다. 그러나, 이 메소드는 적어도 해당 홈 이 곧 뷰가 지정 될 수 있기 때문에, 사용 하여 일부 해당 페이지에 대해 적절하게 라는 안전하고 호출 가능하게 됩니다.

navbar 메뉴변경

app/views/layouts/application.html.erb 파일에서 아래와 같이 <ul class="nav navbar-nav"> 부분을 아래와 같이 변경합니다.

```
<ul class="nav navbar-nav">
  <li class="dropdown">Welcome</li>
  <li class="dropdown">Welcome</li>
  <li class="dropdown">Welcome</li>
</ul>
```

그리고, http://localhost:3000/welcome 로 접속한 뒤 Mobile Pinnie 버튼을 클릭하여 "메뉴 목록"이 가



> [redacted] https://github.com/rails/rails/blob/master/guides/source/5_0.md

명령	설명
포인팅(pointing)	일반적인 마우스만의 클릭, 이동의 동작
동작시 행동(doing)	동시다발적 마이쿼리 사용이 일정과 해야하는 하는 동작
일러두기(preferences)	마우스 클릭이라는 한 동작 하나라도 원하는 마우스 누르라는 하는 동작

Bulletin 모델에 post_type 속성 추가하기

게시판을 새로 추가할 때 해야하는 작업이기 때문에 post_type 이라는 속성을 추가하기로 했다. 이 속성은 string 속성을 가지는 단순한 카드, bulletin, blog, gallery 등을 가질 수 있다. 이를 위해서 bulletin 모델에 속성을 추가하는 마이그레이션 작업을 생성한다.

```
$ bin/rails g migration add_post_type_to_bulletins post_type
      invoke  active_record
      create    db/migrate/20240102123456_add_post_type_to_bulletins.rb
```

그리고 생성된 위에 생성된 마이그레이션 작업을 열어 아래와 같이 post_type 의 디폴트 값을 bulletin 으로 정하자.

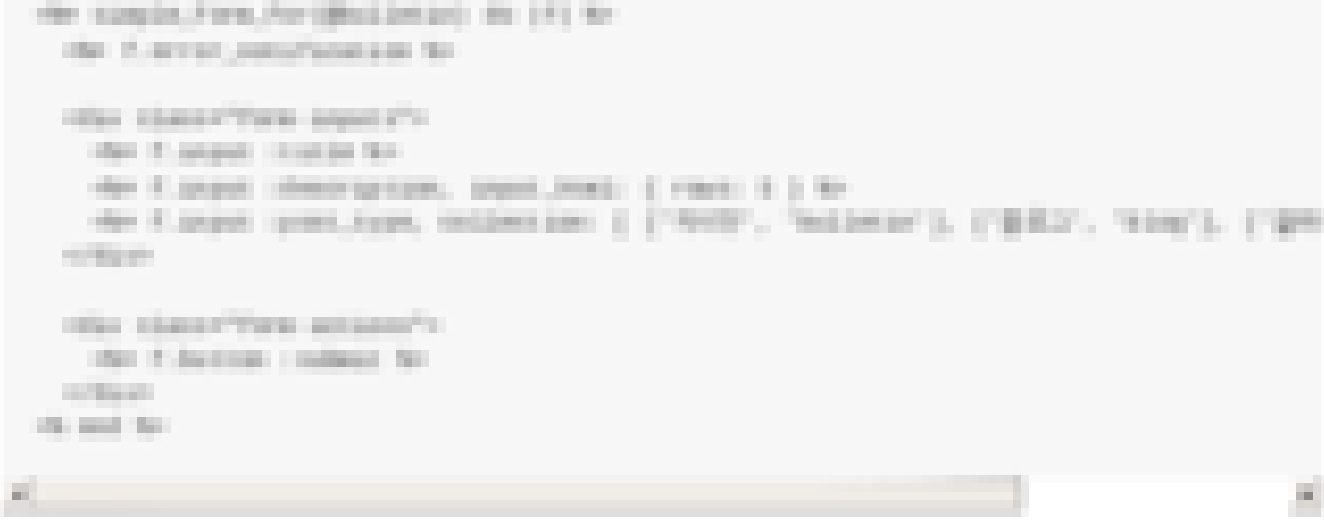
```
class AddPostTypeToBulletins < ActiveRecord::Migration[7.1]
  def change
    add_column :bulletins, :post_type, :string, default: "bulletin"
  end
end
```

마찬가지로 db:migrate 작업을 한다.

```
$ bin/rails db:migrate
== 20240102123456 AddPostTypeToBulletins: migrating ==========================
-- add_column(:bulletins, :post_type, :string, {:default=>"bulletin"})
   -> 0.0007s
== 20240102123456 AddPostTypeToBulletins: migrated (0.0008s) =================
```

Strong Parameter 추가하기

bulletins_controller.rb 작업을 열어 마찬가지로 또는 bulletins_params 메소드에 아래와 같이 post_type 속성을 추가한다.



Editing bulletin

Preview Bulletin

Title	[illegible]
Description	[illegible]
Post Type	[illegible]
Scheduled at	[illegible]

Edit | Back

이렇게 하시면 [illegible] 이렇게 이 포스팅 작업이 완료되었습니다.

제시한의 형태에 따른 방법 두 가지가 주로 쓰기 편해요. [illegible] 디렉토리에 [illegible] 가능 하지 디렉토리를 만들고 거 디렉토리에 [illegible] 파일과 [illegible] 파일을 생성합니다.

[illegible] 인터페이스 [illegible] 작성 후 파일의 위쪽 내용을 [illegible] 파일로 이동시키고 남은거 파일을 아래와 같이 바꿉니다.

```
[illegible] Plan: [illegible] [illegible] [illegible]
```

> [illegible] 일단 테스트는 잘 되는 것을 확인 했습니다.

그리고, [illegible] 을 아래와 같이 수정합니다. [illegible] 테스트는 [illegible] 방법은 파일을 만들어 넣어도 잘 되긴 하지만 조금 번거로운데 이것은 인증부분 내용 [illegible] 을 추가하지 않아도 되는데 문제가 [illegible] 없지만 [illegible] 일 경우 [illegible] 에 등록되어 [illegible] 디렉토리에 [illegible] 파일을 작성을

```
# 생략

  def find_or_create_by_name(name, attr=nil)
    find(find_by(name: name, attr)) rescue nil
  end

end
```

_index.html.erb 파일부터 내용을 이렇게 갱신 할 수 있습니다. 마찬가지 형식대로 게시물을 수정하는 update.html.erb 도 위 내용과 비슷한 수정 입니다 합니다.

```
<h1>New Publishing_index pattern[] Publishing_id] 게시글</h1>

<% 생략하는 에러는 제외 : 1 point : 1 %>
    <div id="point">
      <div id="point_id">{{point_id}}</div>
      <div id="point_name">{{point_name}}</div>
      <div id="point_description">{{point_description}}</div>
      <div>
          <%= link_to "Show", [point.Publishing, point], class: "btn btn-defaulu" %>
          <%= link_to "Edit", edit_Publishing_point_path([point.Publishing, point]), class: "btn b
          <%= link_to "Destroy", [point.Publishing, point], method: :delete, data: { confirm: c
      </div>
    </div>
<% end %>

<br>

<%= link_to "New Point", new_Publishing_point_path, class: "btn btn-defaulu" %>
```

이제 point 쪽 생성된 파일들을 이렇게 갱신 수정합니다.

pointbuilder 쪽 생성된 파일

"New point"이는 개체 대신 갱신 클릭하는 게시판-이름이 나타나게 됩 합니다.

```
<h1>New Publishing_index pattern[] Publishing_id] 게시글</h1>

<%= render "form" %>

<br>
<%= link_to "Back", Publishing_points_path, class: "btn btn-defaulu" %>
```

```
<?php Lang::get('backend.pages.pages.create') ?>
```

postbfshow 뷰 템플릿 파일

웹사이트 사용자가 부가되는 수정하고 데이터를 한번에 관리 부여준다.

```
<?php echo Form::open(array('route' => array('backend.pages.pages.post')))
<?php Form::open(array('route' => array('backend.pages.pages.post')))

이미지업로드 이미지수정 "이미지삭제"는
사용가능
    사용자가입일자기준
    사용자이메일 입력하여 이미지수정
사용가능
사용가능
    사용자이메일이메일수정기준
    사용자이메일 입력하여 이미지수정
사용가능
사용가능
    사용자인증메일보내기 보내기전에
    사용자이메일 입력하여 이미지수정
사용가능
사용자별인증서

<?php Form::text('title', isset($_POST['post']['title']) ? $_POST['post']['title'] : null, array('class' => 'form-control
<?php Form::text('slug', $_POST['post']['slug'], array('class' => 'form-control')) ?>
```

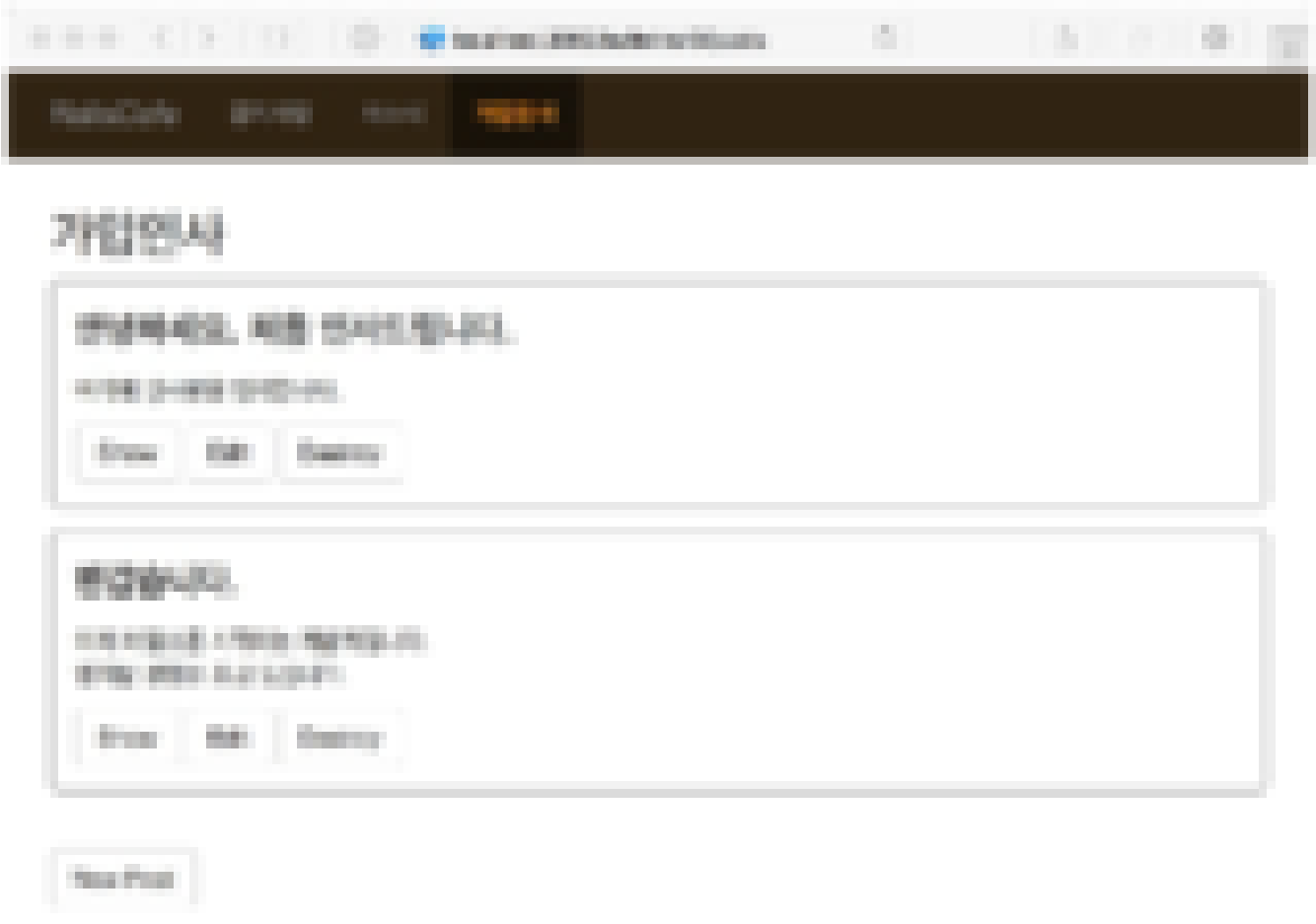
이제 마지막 작업이 http://localhost:3000/backend/pages/create 를 접속한 뒤 제목과 슬러그를 입력하고 저장하면 됩니다.

다음 명령으로 pnpm, 스크립트, 의존성을 실행하거나 /app/scripts/try-landmarks/pnpm.sh 파일에 작성하세요.

```
pnpm {
  dependencies: {
    dependencies-cache: true,
    dependencies-hash-files: "pnpm-lock.yaml",
    paths: [node_modules],
    key: "v1-dep-{{ checksum "pnpm-lock.yaml" }}",
  },
  install: {
    frozen-lockfile: true,
    cache-dir: /home/circleci/.pnpm-store,
    store-dir: /home/circleci/.pnpm-store,
  }
}
```

그러고 /app/scripts/try-landmarks/ 디렉토리에 있는 /app/scripts/pnpm.sh 파일을 열고 아래에 같이 작성하세요 (출력되는 "pnpm ok" 를 볼 수 있어요).

```
$(nvm) || nvm install 18
$(nvm) && nvm use --silent && nvm alias default 18
$(nvm) && corepack enable --install-directory /usr/local/bin
$(nvm) && corepack prepare --activate pnpm@latest
$(nvm) && pnpm --version && echo "pnpm ok"
```



원본 소스: https://github.com/nodebird/nodebird/tree/master_JS_JS

설치하면 "하나 이상의 이미지를 입력하실 수 있어야 입력하신 이미지를 생성할 형태로 바꾸지 못합니다.

이미지를 입력하기 위해서는 두 가지 방법 사용할 수 있다. 하나는 pdftoppm 명령 다른 하나는 imagemagick 라는 것이다.

이 두 명령 모두 시스템에 imagemagick 라이브러리 없이 설치되어 있다. 시스템에서 이것이 없는 경우에 문제가 생길 수 있다. gmail.com/gpt 에 접속 상 이미지 PDF 파일 입력하고 시 정해지기 이미지를 생성할 수 있다.

```
$ ruby -v
ruby 2.6.10p210 (2022-04-12 revision 67729) [universal.x86_64-darwin19]
$ brew install imagemagick
Error: imagemagick 6.9.11-1 is keg-only and unlinked, linking forced.
If you really know what you're doing, you can set
HOMEBREW_FORCE_BREWED_CURL=1 in your environment
to force use of the Homebrew version.
$ brew install ghostscript
==> Downloading https://ghcr.io/v2/homebrew/bottles/ghostscript-10.0.0.arm64_ventura.bottle.tar.gz
==> Pouring ghostscript--10.0.0.arm64_ventura.bottle.tar.gz
==> Caveats
Ghostscript fonts were installed to
/opt/homebrew/share/ghostscript/fonts
To use these fonts, you may add to your configuration (eg, ~/.profile)
export GS_FONTPATH="$GS_FONTPATH:/opt/homebrew/share/ghostscript/fonts"
==> Summary
🍺  /opt/homebrew/Cellar/ghostscript/10.0.0: 1,000 files, 67.2MB
$ ruby -v
ruby 3.2.2 (2023-03-30 revision e51014f9c0) [arm64-darwin22]
$ brew install imagemagick
==> Downloading https://ghcr.io/v2/homebrew/bottles/imagemagick-7.1.1-15.arm64_ventura.bottle.tar.gz
==> Pouring imagemagick--7.1.1-15.arm64_ventura.bottle.tar.gz
==> Caveats
To enable PostScript, Type 1, Ghostscript font and font search
add the following to your configuration (eg, ~/.profile)
export MAGICK_CONFIGURE_PATH="/opt/homebrew/share/ghostscript:/opt/homebrew/share/ghostscript/10.0.0:/opt/homebrew/share/ghostscript/fonts:/opt/homebrew/share/ghostscript/10.0.0/Resource:/opt/homebrew/share/ghostscript/10.0.0/lib"
==> Summary
🍺  /opt/homebrew/Cellar/imagemagick/7.1.1-15: 1,000 files, 24.2MB
$ ruby -v
ruby 3.2.2 (2023-03-30 revision e51014f9c0) [arm64-darwin22]
```

```
# 사진(jpeg, webp) 파일만 허용하고, 최대 10MB 제한 및 고유한 파일명 생성
class ImageUploader < CarrierWave::Uploaded::Base
  storage :fog
  def extension_allowlist
    %w(jpg jpeg gif png webp)
  end
  def size_range
    1..10.megabytes
  end
  def filename
    "#{Time.now.to_i}_#{rand(100000)}.#{file.extension}" if original_filename.present?
  end
end
```

> 이렇게 하면 사용자가 실수로 엑셀이나 큰 영상 파일을 올리는 것을 원천 차단하고 서버 용량을 아낄 수 있게 됩니다.

업로드 클래스의 생성

이제 각 업무에 맞게 위에서 만들어 둔 기능을 가져와 업무별 전용 업로더를 생성합니다.

```
$ bin/rails g uploader Product
      create  app/uploaders/product_uploader.rb
```

생성된 `ProductImageUploader` 클래스 파일의 내용을 아래와 같이

```
# encoding: utf-8

class ProductImageUploader < CarrierWave::Uploader::Base

  # Include RMagick or MiniMagick support:
  # process :resize_to_fit => [50, 50]
  # process :resize_to_limit => [800, 800]

  # Choose what kind of storage to use for this uploader:
  storage :file
  # storage :fog

  # Override the directory where uploaded files will be stored.
  # This is a sensible default for uploaders.
  def store_dir
    "uploads/#{model.class.to_s.underscore}/#{mounted_as}/#{model.id}"
  end

  # Provide a default URL as a default if there hasn't been a file uploaded
  # def default_url
  #   # For Rails 3.1+ asset pipeline compatible
  #   # asset_path("fallback/" + [version_name, "default.p
  #
  #   "/images/fallback/" + [version_name, "default.png"].compact.join("_")
  # end

  # Process files as they are uploaded:
  # process :scale => [200, 300]
  #
```

```
  # Guard clause: check if filename is set and is a valid Pathname
  # Rails.root is a Pathname, so we can safely join it.
  # We also check if the file actually exists.
  if self.filename
    File.delete(Rails.root.join("public", self.filename))
  end
end
```

이렇게 하면 이제 이미지 인스턴스가 파괴되기 전에 파일을 먼저 삭제하는 훨씬 더 안전하고 견고한 코드가 됩니다.

```
# app/models/image.rb

class Image < ApplicationRecord
  belongs_to :gallery

  def delete_file!
    "#{public_path}/#{filename}" if filename.present? && File.exists?("#{public_path}/#{filename}")
  end

end
```

한 디렉토리 파일 모두 삭제하기

갤러리를 하나 지울 때에는 해당 폴더가 남아 있게 됩니다. 이때에 남아 있는 폴더가 불필요하니 `ActiveRecord::Callbacks` 에 추가하면 자동으로 빈 폴더가 삭제됩니다. 이때는 새로운 개념을 도입합니다.

```
# 갤러리가 삭제되기 전에 uploads/gallery_1 폴더를 삭제합니다.
before_destroy :remove_empty_upload_dir

def remove_empty_upload_dir
  path = Rails.root.join("public/uploads/gallery_#{id}")
  FileUtils.rmdir(path) if File.exists? && Dir.empty?(path)

  path = Rails.root.join("public/uploads/thumb_gallery_#{id}")
  FileUtils.rmdir(path) if File.exists? && Dir.empty?(path)
# 이미지 파일 삭제 로직 추가
  image = self.image, 이미 있다면 삭제
end
```

```
  ... 생략 (이미지 처리 로직)
  -> [0, 0000] s
end [2026-04-09 09:00:00] INFO -- : Image variant [0, 0000] s: [/rails/active_storage/blobs/redirect/...]
```

업로더 마운트하기

Photo 클래스 파일인 app/models/photo.rb 에 앞 장의 PhotoUploader 업로더 클래스를 photo 속성과 마운트합니다.

```
class Photo < ApplicationRecord
  belongs_to :user
  mount_uploader :photo, PhotoUploader
end
```

위와 photo 인자 없이 photos 테이블 photo 컬럼과 photo_cache 속성을 추가합니다.

```
$ rails g add_photo_to
  invoke active_record
  create db/migrate/20260409090000_add_photo_to_photos.rb
```

MiniMagick 젬 추가하기

이미지 크기를 조절하기 위해서 Magick 대신 MiniMagick 젬을 추가합니다. Gemfile 을 수정해 작성한 MiniMagick 젬을 추가하면 이 이미지 업로드 기능을 위해 bundle 에 minimagick 젬을 추가하는 것.

```
gem 'mini_magick'
```

젬을 설치합니다.

```
$ bundle install
Bundle complete! 15 Gemfile dependencies, 78 gems now installed.
```

업로더에 이미지 크기 옵션 추가하기

app/uploaders/photo_uploader.rb 파일을 열고 이미지 사용을 추가합니다.

```
# encoding: utf-8
class PhotoUploader
  include CarrierWave::MiniMagick
  storage :file
  def store_dir
    "uploads/photos/#{model.id}"
  end
  version :thumb do
    process resize_to_fill: [100, 100]
  end
  version :medium do
    process resize_to_limit: [300, 300]
  end
  version :large do
    process resize_to_limit: [800, 800]
  end
  def extension_white_list
    %w(jpg jpeg gif png)
  end
end
```

```
[illegible]
```

Post 폼에 파일업로드 추가하기

폼 post.php 템플릿 파일인 app/views/posts/_form.html.erb 를 아래와 같이 변경합니다.

```
<%= simple_form_for([@project,@post]) do |f| %>
  <%= f.error_notification %>

  <div class="form-inputs">
    <%= f.input :title %>
    <%= f.input :content, input_html: { rows: 10 } %>

    <% if @post.persisted? && @post.image_type == "gallery" %>
      <%= f.input :gallery, as: :file %>
      <%= f.hidden_field :gallery_cache %>
    <% end %>
  </div>

  <div class="form-actions">
    <%= f.button :submit %>
  </div>
<% end %>
```

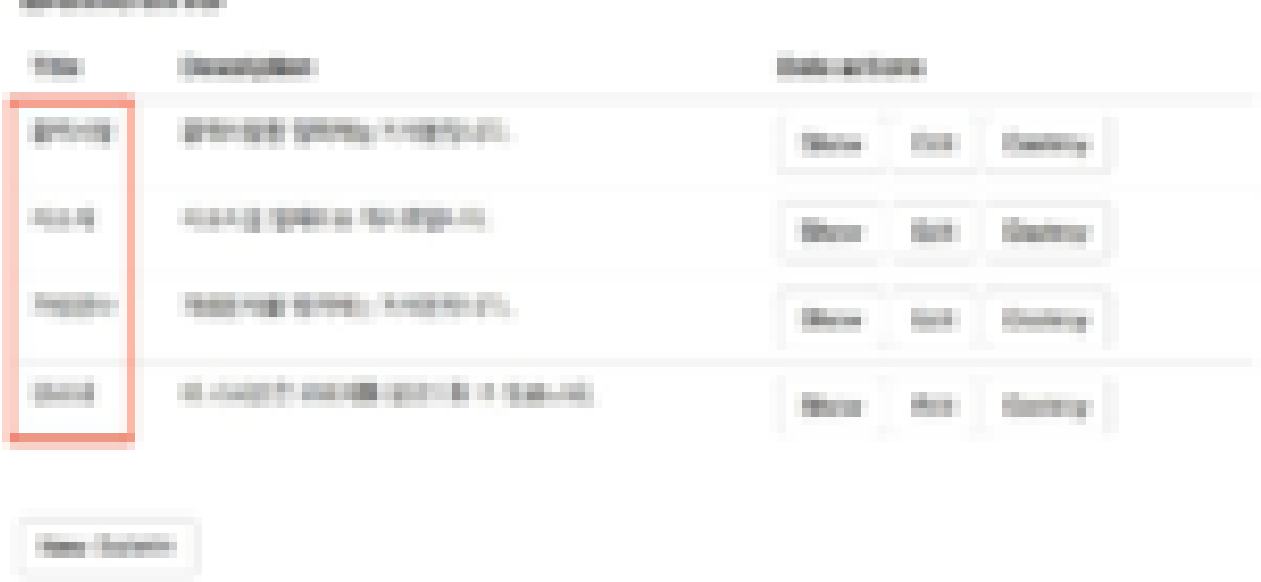
위에서 as: :file 이라 명시를 하면, gallery 를 캐시한에서는 이미지를 입력하실 수 있도록 드롭업 업로드 기능이 나옵니다.

그리고 파일 입력시 image 에 대한 스타일을 app/assets/stylesheets/posts.scss 파일에 추가합니다.

```
.input[type="file"] {
  background: rgb(0, 0,0,0.0) [illegible]
  padding: 0.5em;
  background-color: [illegible];
  width:fit-content;
}
```

갤러리 게시판을 생성

이미지를 입력가능한 게시판을 생성하기 위해서 http://localhost:3000/backstage 에 접속한 후 아래와 같이 "갤러리" 게시판을 추가합니다. 이 때 mount_uploader 에서 gallery 를 선택하고 저장합니다.



이제 마지막 수정하기만 해주면 되는데 파일인 app/views/layouts/application.erb 그대로 열고, 상단 제목부분을 살짝만 좀 손 가겠다.

```
...
<div id="header">
<h1> <%= link_to( [illegible] ) %> <%= "v" if "development" %> <%= link_to "클라이언", [illegible]
...
```

다음은, views/users 폴더를 다음 마우스 우측에 마우스 우측에 클라이언 (new.rb.rb 에 손 가겠다.)

```
print "Client: [illegible] + [illegible]"
```

[illegible] 후 엔터키를 치시기 시작합니다.

그러면 app/controllers/application.rb 파일을 열고 아래와 같이 손 가겠다.

```
...
[illegible] = "[illegible]"
```

```
<div class="card_grid" id="grid">
<% 등록된 프로젝트가 없을 때 %>
<% if (projects.empty?) %>
  <div id="empty_state">아직 등록된 프로젝트가 없습니다. 첫 번째 프로젝트를 추가해 보세요.</div>
<% else %>
  <% projects.each do |p| %>
    <div id="card_wrapper_#{p.id}">
      <% if p.image_path.present? %>
        <image_tag "#{p.image_path}" />
      <% else %>
        <div>이미지 없음</div>
      <% end %>
      <h3>{ p.name }</h3>
      <p>{ p.description }</p>
      <div>
        <% if p.tech_stack.present? %>
          <% p.tech_stack.split(",").map(&:strip).each do |t| %>
            <span>{ t }</span>
          <% end %>
        <% end %>
        <% if p.deployed? %>
          <span>배포됨</span>
        <% else %>
          <span>개발 중</span>
        <% end %>
        <% if p.github_repo.present? %>
          <span>GitHub</span>
        <% end %>
        <% if p.demo_path.present? %>
          <span>데모</span>
        <% end %>
        <% if p.updated_at.present? %>
          <span>{ time_ago_in_words(p.updated_at) } 전 수정</span>
        <% end %>
      </div>
    </div>
  <% end %>
<% end %>
</div>
</div>

<div id="empty_state">아직 등록된 프로젝트가 없습니다. 첫 번째 프로젝트를 추가해 보세요.</div>
```

`app/controller/projects_controller.rb` 파일에 아래를 붙여 넣으세요.

```
class ProjectsController < ApplicationController
  before_action :set_project, only: [:show, :edit, :update, :destroy]

  def index
    @projects = Project.all.order(created_at: :desc)
    @total = @projects.count
    @deployed = @projects.where(deployed: true).count
    @in_progress = @total - @deployed
    @tech_stats = @projects.pluck(:tech_stack).compact.flatten.tally
    @recent = @projects.where(updated_at: 7.days.ago..)
    @with_demo = @projects.where(demo_path: nil).count
  end

  def show
    @related = @projects.where(tech_stack: @project.tech_stack).where.not(id: @project.id).limit(3)
  end

  def new
    @project = Project.new
  end

  def edit
  end

  def create
    @project = Project.new(project_params)
    if @project.save
    redirect_to root_path, notice: "프로젝트가 등록되었습니다."
    else
    render :new, status: :unprocessed_entity
    end
  end

  def update
    if @project.update(project_params)
    redirect_to root_path, notice: "프로젝트가 수정되었습니다."
    else
    render :edit, status: :unprocessed_entity
    end
  end

  def destroy
    @project.destroy
    redirect_to root_path, notice: "프로젝트가 삭제되었습니다."
  end

  private
  def set_project
    @project = Project.find(params[:id])
    rescue ActiveRecord::RecordNotFound
    redirect_to root_path, alert: "프로젝트를 찾을 수 없습니다."
  end

  def project_params
    params.require(:project).permit(:name, :description, :tech_stack, :image_path, :deployed, :demo_path, :github_repo)
  end
end
```

위와 같이 클라이언트 사이드 단에서 업로드할 수 있는 파일 유형을 지정할 수 있다.

```
...
def validate_uploaded_file(uploaded_file):
    ext = os.path.splitext(uploaded_file.name)[1].lower()
    if ext not in ALLOWED_EXTENSIONS:
        raise ValidationError(f'허용되지 않는 파일 형식입니다. 허용되는 형식: {', '.join(ALLOWED_EXTENSIONS)}')
    if uploaded_file.size > MAX_FILE_SIZE:
        raise ValidationError(f'파일 크기가 너무 큽니다. 최대 {MAX_FILE_SIZE // 1024 // 1024}MB까지 업로드 가능합니다.')
...
```

이제 .jpg, .jpeg, .png, .gif, .pdf 확장자를 가진 파일만 업로드할 수 있게 되었다.

이러한 확장자가 아닌 파일을 업로드하면 에러를 띄워 메시지가 뜨는 것이다.

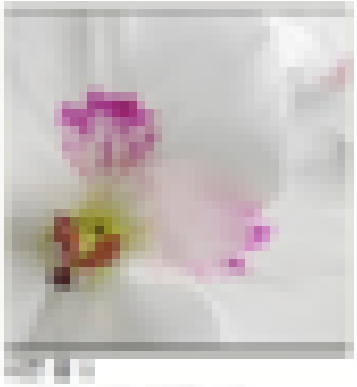
Title

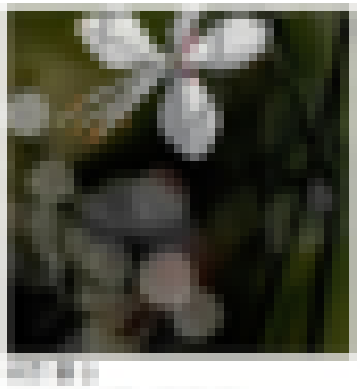
Content

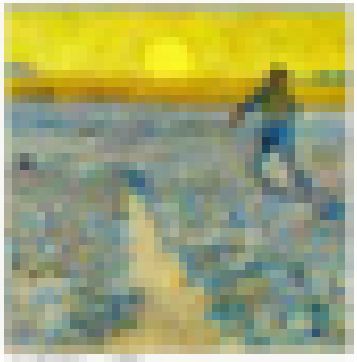
Pictures

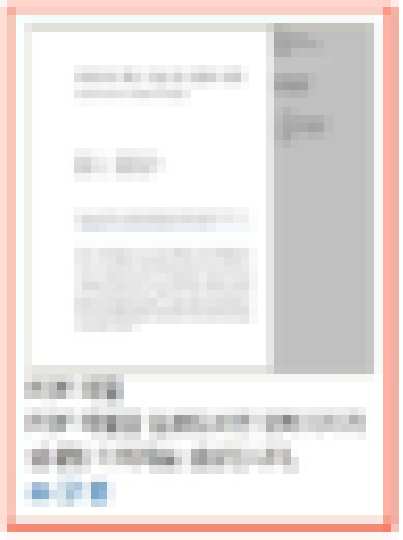
Create Post

Back





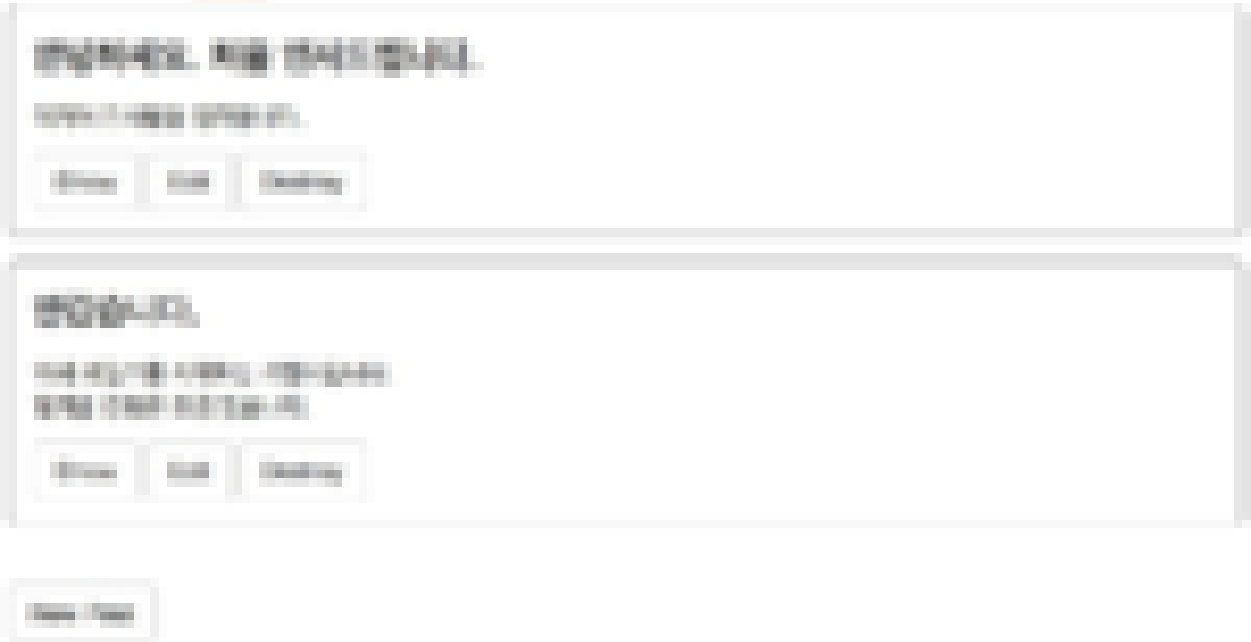




Take Test

다 개수만 정해 이런 데이터셋을 pairwise 방법과 마찬가지 부분에 이제는 같이 조 기하여 학습 방법 설명 및 이론 부분 좋겠다. (_Bezzam et al. 이제는 여기 , _Zhang 이제는 여기 , _gaussian 이제는 여기)

```
<h2>데이 데이 데이데이데이
```



> 그 중에서도 작업은 pdd.sc/sacuaeu/ 디렉토리에 저장됩니다. 이 작업들은 스크립트를 통 통하지 않기 때문에, 그 pdd.sc/sacuaeu/ 작업을 열어 버그나 이상한 값이 표시되면 됩니다.

```
...
pdd.sc/sacuaeu/*
```

> 원본소스: https://github.com/sourcecode/sacuaeu/tree/main_25_11

이제 해당 스 내부 뷰의 최상 flashes 클래스의 인스턴스에서 특 개인인에게 장소로 뷰에를 전달할 수 있는 수단으로 생각할 수 있다. 마지막 flash 뷰에에 여전 문자이는 지정할 수 있고 바로 다음의 예문에서는 이들의 그 사이 차게 있는 것이다.

해당 스에 서는 대개 마지막 flash 메시지를 처결과 뷰의 전 레이아웃의 body 태그 내 등 상단에 위치시킨다.

```
<!DOCTYPE html> <%= yield %>
<html>
<head>
<title>
</head>
<body>

  <% 이 문자 결과의 메시지를 삽입한다. %>

  <%= yield %>

</body>
</html>
```

이제 application.rb 또는 다른 컨트롤러를 작성해서 마지막 flash 메시지를 남겨둔 후 이동해 보자.

해당 스에 scaffold 제너레이터를 이용하여 생성된 Post 모델의 컨트롤러(posts_controller.rb) 클래스 파일을 살펴보면, create, update, destroy 액션에서 flash 를 이용하는 모습 확인할 수 있다.

```
redirect_to ..., notice: "Post was successfully created."
```

flash 메시지를 정의할 때 flash[:notice] = "Post was created" 와 같이 하드를 작성할 수 있지만, 위에서와 같이 redirect_to 메소드를 사용할 때는 해시형태의 옵션으로 flash 메시지를 지정할 수도 있다.

> 컨트롤러 메소드 지정 옵션 지정할 때 notice => "Post was created" 와 같이 지정할 수도 있지만 관례 상 notice: "Post was created" 와 같이 축 약형 으로도 지정할 수 있다.

위에서 언급한 바와 같이 이 메시지는 처음 리다이렉션 레이아웃에 해당하는 application.html.erb 에서 yield 로 삽입하기로 한다.

```
<div id="content">

  <div id="notice" <%= flash[:notice] %>

  <div id="alert" <%= flash[:alert] %>

  <%= yield %>
```

```
```

render 메소드는 parse() 명령어를 만들 때, 출력으로 표준 만수출을 남겨 만들 수 있습니다. 이 기능을 rlpark 명령어 rlpark 메시지를 남겨 줍니다. 이 명령 rlpark ls ./rlpark_messages.msg1.erb 처럼해서 바로 사용할 수 있게 됩니다.

rlpark 메 기능 erb 메소드를 이용해서 미리의 로스를 블록으로 남겨 주는 로스의 기능 save_, 그리고 addmsgs 블록만으로 받아 블록 내부가 사용될 수 있다. 이 때 save 명령 입에 파이다이스 클래스를 넘어 이용할 수 있도록 Kernelmsg_class_for() 기능 이용이 해주면 별도 메소드를 app/models/messagemsg.rb 처럼해 주 기에다 이해해 넘어 작성합니다. 이 기이 사용하는 로그의 클래 기능 Dsrbmsg이를 등 인해서 이뤄집니 경 해다.

```
models/messagemsg.rb

  def Kernelmsg_class_for(rlpark_name)
    case rlpark_name
      when "submission"
        "Submission::Submission"     # 과제 제출
      when "grade"
        "Grade::Grade"               # 성적 입력
      when "assign"
        "Assign::Assign"             # 과제 출제
      when "user_post"
        "UserPost::UserPost"         # 게시글 작성
      else
        rlpark_name.to_s
    end
  end

end
```

이제 새로운 메시지를 작성하거나 수정하면 해당되 남겨진 rlpark 이벤트 처럼 메시 지가 보내질 것이다. 일반적인 이야 기입니다. 이 메시 지를 해당되를 다시 출입하면 이어집니 경 해다.

세소세

Title	개발 작가 소 소
Contents	나 나무나 세이 고 두 소통해 이상하다는 없어지기 작업 나 나나 보고나 사정도 전자 정보 해 나나 두 수 있는 한 나나.
Created at	2021-12-22 21:17:59 +0900

Edit Delete

이상하다 해당 나무 Issues 개 나무를 두 나무는 방안을 나 주었다.

> 참조 링크: https://github.com/mrcrypto/codeforest/issues/25_2.2

- 김치찌개
- 제육볶음
- 가정식사
- 갈비찜

이렇게 각 카테고리를 실행하면 해당 데이터 입력 작업부터 개발 팀과의 협업 과정까지 생산성이 훨씬 향상되는 것을 확인할 수 있다.

마찬가지로 이번 기능 이용 외에도 데이터 seed 기능 작업을 할 수 있다.

```
$ bun run db:seed
```

이때 실행 명령어 db:seeds.ts 파일에 있는 명령을 실행하여 데이터를 생성하게 된다.

이제, 개발자는 4개의 독립된 명령을 편리하는 명령을 추가하여 본다.

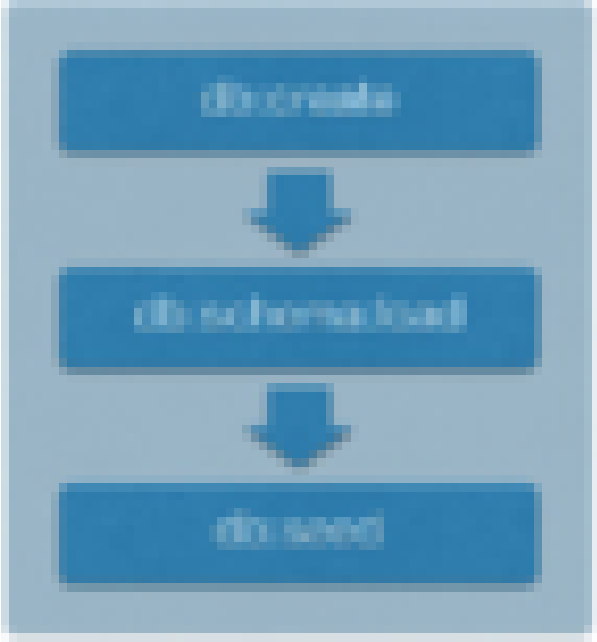
```
# 기본적 데이터 입력
bun run db:seed:base:all 또는 bun "김치찌개"
bun run db:seed:base:all 또는 bun "제육볶음"
bun run db:seed:base:all 또는 bun "가정식사", seed_type: "home"
bun run db:seed:base:all 또는 bun "갈비찜", seed_type: "galbijjim"
```

김치찌개 또는 제육볶음 은 seed_type 속성을 지정하지 않았는데 이 경우 seed_type 속성이 기본적 값인 "basicseed" 로 해당하는 명령어에 데이터를 추가하게 되는데 "basicseed" 로 입력된다.

이렇게 하고 하나 이외에 함께 데이터를 추가하는 것을 데이터를 추가하는 것.

```
$ bun run db:seed
```

db:seeds 은 이제를 이 예제 중 다시 db:seeds 하는 명령을 이는데, 이 때 db:seed 명령어 함께 수 있다.



이제 마지막 마법사 명령어가 rake:db:setup 클릭하면 해당 작업들이 차례 대로 실행 되어지게 됩니다.

> 다만 주로 개발에서는 데이터베이스를 rapture 를 사용합니다. 이 데이터베이스는 이미 상당히 정리되었는 데 sqlite3 데이터베이스와 유사하면서 기능은 상대적으로 MEMBASE입니다. 따라서 데이터베이스를 변경하는 과정이 필요합니다. 다른 점이 있다 하는 데 rake:db:setup 작업에 db:create 작업을 제외 하셔야 합니다.

> 참조소스 https://github.com/rocketpunch/rails/tree/chapter_15_13

인지 안들어 보일을 확실히 확인하기를 바랍니다 만들어.

```
$ bun/vite.config.js modify --name=test --port=3000 --entry=./src/main.tsx --body=./test
```

이제 같이 수정을 생성하게 되면 해당 어떠한 어떠한 이제 이제 이 파일이 이동하게 되면 이동하게 되는 파일을 생성할 수 있는데 이것 그렇게 하실 수 있습니다. 일부러 생성하지는 않겠습니다. 우리는 이 파일을 이동하여 실 제품 파일을 생성하게 되는 것입니다.

먼저 커맨드라인에서 port 이름과 port 파일 속성을 지정하였기에, 이 안에 port_set 기능 속성을 만들고 이 속성을 myvalue key 로 이동하여 줄 때 준다. 그리고 해당 파일의 name 그룹에서는 testcomp_ver port 기능 관 계산 만들 이동하여 줄 줘 준다.

실제로 이 이 명령을 실 행하면 아래와 같은 결과를 볼 수 있다.

```
$ bun/vite.config.js modify --name=test --port=3000 --entry=./src/main.tsx --body=./test
        [2025/08/26] [INFO] [CMD_PARSE]
        [2025/08/26]     [INFO] [CMD_PARSE] [2025/08/26 12:00:00] [CMD_PARSE] [INFO] [CMD_PARSE] [INFO]
        [2025/08/26]     [INFO] [CMD_PARSE] [INFO]
        [2025/08/26]     [INFO] [CMD_PARSE]
        [2025/08/26]         [INFO] [CMD_PARSE] [INFO] [CMD_PARSE] [INFO]
        [2025/08/26]         [INFO] [CMD_PARSE] [INFO] [CMD_PARSE] [INFO]
```

이 명령은 내부적으로는 [src/cmd/cmd_parser.] 파일을 호출해서 이러그램에서 파일 [./bin/cmd/cmd_parser.] 이 그램 생성하고, 이 파일 실행이 되고 [./app/modify/index] 이 그램 생성합니다. 그리고 [./cmd/_cmd.] 파일을 호출하여 유니커드 이름 파일 생성하게 됩니다.

이제 이러그램의 작업을 실 행하여 실 제품 [./cmd/modify] 기능 제어파일 생성하여보게 됩니다.

```
$ bun/vite.config.js modify --name=test
info [2025/08/26 12:00:00] [CMD_PARSE] [INFO] [CMD_PARSE] [INFO] [CMD_PARSE] [INFO] [CMD_PARSE] [INFO] [CMD_PARSE] [INFO] [CMD_PARSE] [INFO]
[+] 파일명을 소문자로 이동하여 이동하였습니다.
    → 수 : 0.[illegible]초
[+] 파일명_구분자는 _로 이동하여 이동하였습니다. (_ 삭제되지 않음)
    → 수 : 0.[illegible]초
info [2025/08/26 12:00:00] [CMD_PARSE] [INFO] [CMD_PARSE] [INFO] [CMD_PARSE] [INFO] [CMD_PARSE] [INFO] [CMD_PARSE] [INFO] [CMD_PARSE] [INFO]
```

주목할 만한 제어결과를 작업하여 얻었는데요, 이렇게 처음 [./cmd/modify] 기능 제어결과로 제어결과가 생성되었는데 이러고, 내용은 아직 확실하게 개발되지 [./app/modify/index] 이 일단계를 볼 수 있는 제어그램 될 수 있다.

```
[illegible]
[illegible]
[illegible]
[illegible]
```

명령 인해 무엇인 내부의 어떤에 추가된 것을 알 수 있다.

추가된 app/models/post.rb 파일에는 아래와 같이 공개인지를 체 크 하여 있다.

```
class Post < ApplicationRecord # Rails 6
  [illegible]
  has_many :comments, dependent: :destroy
end
```

여기서 has_many 메소드에 :dependent 옵션은 post 객체가 삭제될 때 이 객체와 연결된 comments 객체들이 함께 삭제되어야 하는지 이야기한다. 만약 이 옵션을 지정하지 않았다 면 post 객체가 삭제될 때 연결된 comments 레코드 들이 그대로 남아 있게 되어 해결 하기 어려운 일관성 문제 가 발생할 수 있다. 레이스가 요구 조건이라 할 때 이러한 옵션이 매우 유용하게 사용될 수 있다.

이제 컨트롤러를 만들어 그 삭제하는 액션을 구성해야 한다. 이것 역시 컨트롤러를 업데이트하는 기능은 구성하여 일 반 한다.

이번에는 이전의 같이 해당 스 의 인스턴스 메소드에서 이를 이어 주 명령어 인스턴스에 액션을 만들기로 하자.

```
$ bin/rails g controller comments create destroy
```

controller 다음에 뒤는 뒤번째 인수에 컨트롤러의 이름을 지정한다. 이것은 해당 스 의 루트상 인증되는 위치를 뜻하 복수 형으로 지정하는데, 인덱스 그런 것은 하나의 엔 티티는 복잡이 있을 경우에는 일이 이름으로 지정해야 한다. 예 가 가능 comments 뒤쪽의 인증을 가지기에 comments 로 이름을 지정했다. 이것에서 뒤는 뒤 인수는 컨 트롤러에서의 액션명으로 지정하게 된다. 이것에서 comments 컨 트롤러에는 create 와 destroy 액션 두 액션을 가지게 된다. 이제 실행해서 결과 상태 뒤 어떤 결과를 확인해 보자.

```
$ bin/rails g controller comments create destroy
      [illegible]  app/controllers/comments_controller.rb
       [illegible]  get "comments/create"
       [illegible]  get "comments/destroy"
      invoke erb
      [illegible]    app/views/comments
      [illegible]    app/views/comments/create.html.erb
      [illegible]    app/views/comments/destroy.html.erb
```

이 명령어 하는 주는 파일은, 인터페이 파일 app/controllers/application_controller.rb 를 생성합니다. 우리의 라우팅는 resource :pet "/pet/new/:id" , resource :pet "/pet/edit/:id" 를 추가하면, 각 액션에 해당하는 뷰 템플릿 파일은 app/views/application/pet/new.html.erb , app/views/application/pet/edit.html.erb 가 생성됩니다.

그러나 우리는 이렇게 추가할 때 app 아래에 이렇게 하지가 애매하게, 이 views 에는 파일이 아니라, 이 js.erb 파일이 있으니까 즉, create.js.erb 와 update.js.erb 파일이니 나중에 이 두 erb 파일을 하나로 묶고 나서 상황을 보아 이동해 줄 수 있다.

app/controllers/application/pet/application_controller.rb 파일을 열어 아래 처럼 입력.

```
class Application::Pet::ApplicationController < ApplicationController
  def create
  end

  def update
  end
end
```

이 컨트롤러의 클래스 두 사람을 아래와 같이 바꿀 수 있습니다.

```
class Application::Pet::ApplicationController < ApplicationController
  before_action :set_pet
  before_action :set_application, only: :update

  def create
    @application = @pet.applications.create(application_pet_params)
    @application.update
  end

  def update
    @application.update
  end

  pet 요청처리

  def set_pet
    @pet = Pet.find(params[:id])
  end

  def set_application
    @application = @pet.applications.find(params[:id])
  end
```

```
  gem "[illegible]", "[illegible]"

  [illegible] do
    [illegible]
  end

  [illegible]
end
```

먼저 gemfile 에 [illegible] 라는 젬을 추가합니다. 즉, 아래와 같이 라는 이름의 젬을 생성합니다.

```
module [illegible]
  [illegible]

  [illegible] do
    [illegible]
  end

  [illegible] do
    [illegible]
  end

end
```

이렇게 한 다음 [illegible] 안에 클래스와 대응 구현을 이용할 수 있게 됩니다.

```
$ bin/rails [illegible] [illegible]
                 Prefix Verb   URI Pattern                                  Controller#Action
      [illegible]_[illegible] GET    /[illegible]/:[illegible]/[illegible](.:format)  [illegible]
                      PATCH  /[illegible]/:[illegible]/[illegible](.:format)  [illegible]
  [illegible]_[illegible] GET    /[illegible]/:[illegible]/[illegible]/[illegible](.:format)  [illegible]
[illegible]_[illegible] GET    /[illegible]/:[illegible]/[illegible]/:id(/:[illegible]).:format)  [illegible]
      [illegible]_[illegible] GET    /[illegible]/:[illegible]/[illegible]/:id(.:format)  [illegible]
                      POST   /[illegible]/:[illegible]/[illegible]/:id(.:format)  [illegible]
                      PATCH  /[illegible]/:[illegible]/[illegible]/:id(.:format)  [illegible]
                      DELETE /[illegible]/:[illegible]/[illegible]/:id(.:format)  [illegible]
```

그러나 실 개발 일 때는 이렇게 [illegible] 와 [illegible] 두 가지 개념에 대응 합니다.

따라서 두 개의 같이 이루어 [illegible] 합니다.

다시 이전 작업을 확인해 봅시다.

```
$ firebase functions:log --only functions
                    함수명(region)  상태  시간(UTC)                                    로그 메시지
processOrderWebhook(us-central1)  완료  2026-04-15 14:32:07.412  Processing order: ord_8f2a9c3e7b1d4
processOrderWebhook(us-central1)  완료  2026-04-15 14:32:07.689  Order ord_8f2a9c3e7b1d4 successfully processed
```

이제 모바일 앱에서 결제하여 새로운 주문이 들어올 때 작업이 제대로 수행되는지 보면 됩니다.

이 예시에서는 이미 작성된 웹훅을 테스트하는 데 사용된 웹훅을 작성할 수 있도록 `processOrderWebhook` 함수를 배포하고 이벤트 수신을 시작합니다. 주문 이벤트가 실제 웹훅을 생성하는 `payment.web` 앱을 통해 이루어집니다.

```
# 배포
firebase deploy --only functions:processOrderWebhook
```

그러고 나서 웹 앱에서 `payment.web.app` 주문을 `http://localhost:5173/checkout` 으로 이동하여 결제를 진행하고 이벤트가 성공적으로 작성됐는지 확인합니다.

```
$ firebase functions:log --only processOrderWebhook
  processOrderWebhook(us-central1)
    2026-04-15T14:32:07.412Z [processOrderWebhook] Processing order: ord_8f2a9c3e7b1d4
    2026-04-15T14:32:07.689Z [processOrderWebhook] Order ord_8f2a9c3e7b1d4 successfully processed
  processOrderWebhook(us-central1)
    2026-04-15T14:32:07.689Z [processOrderWebhook] Elapsed time: 277ms
  processOrderWebhook(us-central1)
    2026-04-15T14:32:07.689Z [processOrderWebhook] Function execution took 277 ms, finished with status: 'ok'
```

마지막으로 주문 `id` 값을 사용해 `http://localhost:5173/orders/ord_8f2a9c3e7b1d4` 으로 이동하여 `payment.web.app` 주문이 처리되었는지 확인하세요.

```
{processOrderWebhook: {
  functionName: "processOrderWebhook",
  functionVersion: "1.0.0",
  triggerType: "HTTPS",
  eventId: "evt_7c3d9a2f1b8e5",
  timestamp: "2026-04-15T14:32:07.412Z",
  labels: {},
  platform: "gcf",
  ingressSettings: "ALLOW_ALL",
  serviceConfig: {
    service: "processorderwebhook",
    serviceAccountEmail: "cloudfunctions@firebase-prod.iam.gserviceaccount.com",
    availableMemoryMb: 256,
    timeoutSeconds: 60,
    maxInstanceRequestConcurrency: 1,
    uri: "https://processorderwebhook-abcdefgh-uc.a.run.app",
    allTrafficOnLatestRevision: true,
    revision: "processorderwebhook-00001-xyz",
    ingressSettings: "ALLOW_ALL",
    serviceAccount: "cloudfunctions@firebase-prod.iam.gserviceaccount.com",
    environmentVariables: {
      FIREBASE_CONFIG: "{\"projectId\":\"firebase-prod\",\"storageBucket\":\"firebase-prod.appspot.com\",\"databaseURL\":\"https://firebase-prod-default-rtdb.asia-southeast1.firebaseio.com\",\"locationId\":\"asia-southeast1\"}",
      GCLOUD_PROJECT: "firebase-prod",
      EVENTARC_CLOUD_EVENT_SOURCE: "projects/firebase-prod/locations/asia-southeast1/triggers/processorderwebhook-1234567890",
      K_SERVICE: "processorderwebhook",
      K_REVISION: "processorderwebhook-00001-xyz",
      PORT: "8080",
      FUNCTION_TARGET: "processOrderWebhook",
      FUNCTION_SIGNATURE_TYPE: "http",
      K_CONFIGURATION: "processorderwebhook",
      GAE_INSTANCE: "00c61b117c7f9a2d3e4b5c6d7e8f9a0b1c2d3e4f5a6b7c8d9e0f1a2b3c4d5e6f7g8h9i0j",
      GAE_SERVICE: "processorderwebhook",
      GAE_VERSION: "00001-xyz",
      GOOGLE_CLOUD_PROJECT: "firebase-prod",
      GOOGLE_CLOUD_QUOTA_PROJECT: "firebase-prod",
      GOOGLE_CLOUD_REGION: "us-central1",
      K_SOCKET: "/var/run/google.sock",
      PATH: "/usr/local/bin:/usr/bin:/bin:/opt/bin",
      HOME: "/home/cloudfunctions",
      TZ: "UTC",
      LANG: "C.UTF-8",
      LC_ALL: "C.UTF-8",
      NODE_ENV: "production",
      NODE_VERSION: "20.11.0",
      NODE_PATH: "/opt/nodejs/node_modules:/usr/local/lib/node_modules",
      NPM_CONFIG_PREFIX: "/opt/nodejs",
      NPM_CONFIG_CACHE: "/tmp/.npm",
      NPM_CONFIG_USERCONFIG: "/opt/nodejs/.npmrc",
      NPM_CONFIG_GLOBALCONFIG: "/opt/nodejs/etc/npmrc",
      NPM_CONFIG_UNSAFE_PERM: "true",
      NPM_CONFIG_LOGLEVEL: "warn",
      NPM_CONFIG_NOUPDATE_NOTIFIER: "true",
      NPM_CONFIG_FUND: "false",
      NPM_CONFIG_AUDIT: "false",
      NPM_CONFIG_PROGRESS: "false",
      NPM_CONFIG_ENGINE_STRICT: "true",
      NPM_CONFIG_FORCE: "false",
      NPM_CONFIG_PREFER_OFFLINE: "true",
      NPM_CONFIG_PREFER_DEDUPE: "true",
      NPM_CONFIG_NO_PROXY: "*",
      NPM_CONFIG_HTTPS_PROXY: "",
      NPM_CONFIG_HTTP_PROXY: "",
      NPM_CONFIG_STRICT_SSL: "true",
      NPM_CONFIG_SCRIPT_SHELL: "/bin/sh",
      NPM_CONFIG_SHELL: "/bin/sh",
      NPM_CONFIG_USER_AGENT: "npm/10.2.4 node/v20.11.0 linux x64 workspaces/false",
      NPM_CONFIG_ORIGINAL_PATH: "",
      NPM_CONFIG_ORIGINAL_NODE_PATH: "",
      NPM_CONFIG_ORIGINAL_NPM_EXECPATH: "",
      NPM_CONFIG_ORIGINAL_NPM_NODE_EXECPATH: "",
      NPM_CONFIG_ORIGINAL_NPM_COMMAND: "",
      NPM_CONFIG_ORIGINAL_NPM_PREFIX: "",
      NPM_CONFIG_ORIGINAL_NPM_GLOBALPREFIX: "",
      NPM_CONFIG_ORIGINAL_NPM_LOCAL_PREFIX: "",
      NPM_CONFIG_ORIGINAL_NPM_INCLUDE: "",
      NPM_CONFIG_ORIGINAL_NPM_LIBRARY: "",
      NPM_CONFIG_ORIGINAL_NPM_LIBRARY_PATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_PATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NODE_PATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_EXECPATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_NODE_EXECPATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_COMMAND: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PREFIX: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_GLOBALPREFIX: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_LOCAL_PREFIX: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_INCLUDE: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_LIBRARY: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_LIBRARY_PATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_PATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NODE_PATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_EXECPATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_NODE_EXECPATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_COMMAND: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PREFIX: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_GLOBALPREFIX: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_LOCAL_PREFIX: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_INCLUDE: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_LIBRARY: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_LIBRARY_PATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_PATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NODE_PATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_EXECPATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_NODE_EXECPATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_COMMAND: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PREFIX: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_GLOBALPREFIX: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_LOCAL_PREFIX: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_INCLUDE: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_LIBRARY: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_LIBRARY_PATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_PATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NODE_PATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_EXECPATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_NODE_EXECPATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_COMMAND: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PREFIX: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_GLOBALPREFIX: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_LOCAL_PREFIX: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_INCLUDE: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_LIBRARY: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_LIBRARY_PATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_PATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NODE_PATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_EXECPATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_NODE_EXECPATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_COMMAND: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PREFIX: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_GLOBALPREFIX: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_LOCAL_PREFIX: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_INCLUDE: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_LIBRARY: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_LIBRARY_PATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_PATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NODE_PATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_EXECPATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_NODE_EXECPATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_COMMAND: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PREFIX: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_GLOBALPREFIX: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_LOCAL_PREFIX: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_INCLUDE: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_LIBRARY: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_LIBRARY_PATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_PATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NODE_PATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_EXECPATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_NODE_EXECPATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_COMMAND: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PREFIX: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_GLOBALPREFIX: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_LOCAL_PREFIX: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_INCLUDE: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_LIBRARY: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_LIBRARY_PATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_PATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NODE_PATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_EXECPATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_NODE_EXECPATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_COMMAND: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PREFIX: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_GLOBALPREFIX: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_LOCAL_PREFIX: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_INCLUDE: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_LIBRARY: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_LIBRARY_PATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_PATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NODE_PATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_EXECPATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_NODE_EXECPATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_COMMAND: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PREFIX: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_GLOBALPREFIX: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_LOCAL_PREFIX: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_INCLUDE: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_LIBRARY: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_LIBRARY_PATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_PATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NODE_PATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_EXECPATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_NODE_EXECPATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_COMMAND: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PREFIX: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_GLOBALPREFIX: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_LOCAL_PREFIX: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_INCLUDE: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_LIBRARY: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_LIBRARY_PATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_PATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NODE_PATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_EXECPATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_NODE_EXECPATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_COMMAND: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PREFIX: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_GLOBALPREFIX: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_LOCAL_PREFIX: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_INCLUDE: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_LIBRARY: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_LIBRARY_PATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_PATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NODE_PATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_EXECPATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_NODE_EXECPATH: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_COMMAND: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PREFIX: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_GLOBALPREFIX: "",
      NPM_CONFIG_ORIGINAL_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_PRESERVE_NPM_LOCAL_PREFIX: "",
      NPM_CONFIG_ORIGINAL_NPM_P
```

```
...
puts("Hello World");
...
```

그리고 뷰에서 사용하는 partials 을 만들기 위해 app/views/common/ 디렉토리에 _post_form.erb 파일을 생성하고 아래와 같이 작성합니다.

```
<%= form_for(post, controller: "common/posts", id: "frm") %>
<% if (post.errors.count(:) > 0 %>
  <div id="error_msg">{post.errors.count} 개</div>
<% end %>
<%= f.text_area :
  <%= f.
    <%= f.
  </%= f.
<% end %>
</%= f.
```

그리고 뷰에서 사용하는 _common.html.erb 파일 맨끝에 파일을 생성하기 위해서 app/views/common/ 디렉토리에 _common.base.html.erb 파일을 생성하고 아래와 같이 작성합니다.

```
<%= f.
  <div id="common_body" %>,
  <div id="ajax_loading_mask">{common.loading_mask} %> ajax
</%= f.
```

또한, app/views/common/ 디렉토리에 _form.html.erb 파일을 생성하고 아래와 같이 작성합니다.

```
<% common = {post: common.new} %>
<%= form_for(form_post, controller: "common/posts", id: "frm") %>
<div id="form_for_{form_for.{post.id}, {common}}, method: :post } do | f | %>
  <%= f.text_area :input_1 %>
    <div id="input_1_count">{ f.text_area :input_1, maxlength: 1000, placeholder: "제목을 입력하세요.", input_1_count: { f.text_area
  </%= f.
  <%= f.text_area :input_2 %>
    <div id="input_2_count">{ f.text_area :input_2, maxlength: 1000, placeholder: "내용을 입력하세요.", input_2_count: { f.text_area
  </%= f.
<% end %>
</%= f.
```

[이] 는 [을] [다] 를 [의] [이] 를 [에서] 를 [에] 는 [이] 를 [이] 를 [이] 를

[사] 조 이벤트를 입력하는 경우 입력이 끝날 때 입력하고 enter 버튼을 클릭합니다.

[사]나 아무런 반응이 없을 경우에는 아무 입력 상태로 있습니다.

해당 부분을 다시 클릭하면 발급 안내 작업창 이벤트가 보여지게 됩니다.

Title	[illegible]
Content	[illegible]
Created at	[illegible]

Comments:

- [illegible]

[illegible]

[illegible] [illegible]

이제 감은 동상은, 수행하는 것은 다른 무언 실 제대 의한 상위 제대로 부가 주 다 장이가 개설과 분리됩니다.

이 글 제를 새 장이가 이 동시 이동시 만들면서 app/ 폴 작을 폴 기이가 app/comments/views/ 디렉토리에 만들어 comments.html.erb 파일을 comments_js.erb 파일과 이름을 변경하고 아래와 같이 수시를 하십니다.

```
<% if 폴이름에서는_최이 대이가_이는_이는 이 = 폴이
  폴("#폴이_이는_이는이는 폴이름에서는 폴이").appendWith(폴("#이름에서는_이는이는_이는폴이는 이 폴이").html(), 이는이는이는
  폴("#이름에서는_이는이는_이는폴이는 이 폴이 이는이").fadeIn(이는);
<% 이는이는 폴이
  alert("이는이는이는 이는이는이는 이는이는이는이는이는이는...");
<% end if 폴이
```

comments 이 폴이가 body 속성에 대응 함수 함께 이로 이동 삽입 없이 작업합니다.

이용할 수 있으며 아래와 같이 프리셋 작성합니다.

```
제공"[illegible]_[illegible].js" 파일
```

위의 아래 스크립트가 해당되어 실행하기 위해서는 _[illegible].html.erb 파일을 아래와 같이 변경합니다.

```
<%= 실행"[illegible]_[illegible].js" 파일
  <%= [illegible].finally 파일,
  <%= [illegible]_[illegible]_[illegible]_[illegible]([illegible].[illegible]_[illegible]) 파일 : nil%>
%>
```

위, [illegible] 제공자에 [illegible] 속성을 추가합니다. 그리고 위에서 작성했던 app/[illegible]/[illegible]/_[illegible].html.erb 파일을 열고 아래와 같이 수정해 해당 파일을 위와 동시에 추가합니다.

```
<%= 실행"[illegible]_[illegible].js" 파일
  <%= [illegible].finally 파일,
  <%= [illegible]_[illegible]_[illegible]_[illegible]([illegible].[illegible]_[illegible]) 파일 : nil%>
  <%= [illegible]_[illegible]("[illegible] [illegible] [illegible]-[illegible]", [illegible].[illegible].[illegible], [illegible].[illegible], [illegible].[illegible]
%>
```

마지막 주의할 것은 [illegible].js 파일이 제대로 작동하기 위해서는 [illegible] 환경을 사용할 수 있어야 합니다. 이를 위해 [illegible]gem 명령어를 통해 [illegible].js.erb 파일을 생성하고 [illegible].js 파일을 정상적으로 동작하도록 하는 것이 필요합니다.

이제 이용을 위 명령을 작성하고 실행해 봅시다.

마지막으로 할 작업은 두 개의 [illegible] 명령 후 해당 부분에 해당 값이 전달되 개수를 표시해 주는지 봅시다.

이를 위해서 app/[illegible]/[illegible]/[illegible]/_[illegible].[illegible].erb 파일을 열고 아래와 같이 해당 부분을 변경합니다.

```
....
<%= [illegible] 파일 : [illegible] 파일 %> <%= [illegible] [illegible] 파일 %> [illegible]
....
```

[illegible] 위와 같이 [illegible] 수정하고 [illegible] 실행하면 [illegible] 화면에서

Title	[illegible]
Content	[illegible]
Created at	[illegible]

Comments:

- [illegible]
- [illegible]
- [illegible]

[illegible]

[illegible] [illegible]

[illegible]

> [illegible] https://github.com/[illegible]/[illegible]/tag/dev_25_14

models.rb 맨 끝에 아래와 같이 한줄을 추가하고,

```
gem 'acts-as-taggable-on'
```

번들 인스톨한 후,

```
$ bundle install --without=test
```

유틸 명령어를 다시 부팅합니다.

다음은 이 젬이 작업을 위해 몇 가지 마이그레이션 작업을 아래와 같이 생성합니다.

```
$ bundle exec rake acts_as_taggable_on_engine:install:migrations
```

그리고 마이그레이션 작업을 합니다.

```
$ bundle exec rake db:migrate
```

이제 테스트를 돌려보기 위해 Micro 뷰를 클릭하는 뷰인 app/views/pins/show.rb 에 맨 맨 끝에 아래와 같이 코드를 추가합니다.

```
tags = Tag.all # 모든 태그를 가져오기 위해 사용
  taggable_on_tagged(tags)
  ...
end
```

이렇게 Micro 뷰 템플릿 내에서 tag_cloud 속성 메소드를 사용할 수 있게 됩니다. 이 속성을 통 데이터로 입력받아 뷰에서는 tag_cloud 표시해줍니다. 이를 위해서 pins/_index.rb 에 작업 (app/views/pins/_index.rb) 에 맨 맨 끝에 아래와 같이 pins_partial 메소드에 tag_cloud 속성을 추가합니다.

```
def pins_partial(pins)
  partial('pins/_index', pins: pins, partial_if: true, tag_cloud: tag_cloud)
end
```

가입인사
Title
Content
Tag(list)
Update Post
Delete
Back

위기에서 사용된 `tag_process` 함수 개선 지난 시간 지금 수정하시오 함수 개발
[app/helpers/ 는 /app/ 보다 load path 가 더 가장 앞서 있기 때문에 같이 인식됩니다.

```
def tag_process(tag_text)
  tag_text.scan 한 { | tag |
    "<span 한 id="#{slugify(tag)}" 한 class="tag">#{tag}</span>"
  }.join(", ") 한 html_safe
end
```

위기에서 사용된 `slug(...)` 개념 지난 개그에서 사용할 수 있는 특수문자를 하나 제거하거나 하는 것이다.

[illegible] 개인 정 생하 등 처리방침 [illegible] 개시하여 경우 [illegible] 이러시를 부여 [illegible] 있다. 이를 위해 [illegible] [illegible] 가 [illegible] 정 경우 [illegible] 있다.

```
{%
{% if [illegible] in [illegible] == "[illegible]" %}
  <div>
    <[illegible]>[illegible]</[illegible]>
    <[illegible] [illegible]="[illegible]" [illegible]="[illegible]">
  </div>
{% endif %}
```

제목:	다른 꽃 2
Caption:	하나의 아름 답고 [illegible]
Picture:	

이제는 photo.erb에서 먼저 위 캡션의 하단에서 태그를 추가하도록 하자. 이 때는 하단보다 왼쪽으로 정렬이 끝나기 때문에 아래 태그를 추가하도록 하자.

주인 app/views/photos/photo_caption/_form.erb 파일을 열고 아래와 같이 추가한다.

```
<% if photo.tag_list.size > 0 %>
  <div>태그: <%= "#{photo.tag_list}".split(", ").map{ |t| "#" + t }.join(" ") %></div>
<% end %>
```

app/controller/photos/photo_caption/_photo_caption.erb 파일에도 마찬가지로 동일한 내용을 추가한다.

태그 관계를 정하는 부 파일이 되게 all 이 진입점을 사용하였다. tag_list 인 태그 클래스를 정의해 주기 위해서 app/models/tag.rb 파일을 열고 아래와 같이 추가해 준다.

```
class Tag
  has_and_belongs_to_habtm :photos, :uniq
end
```

위와 같이 하면 태그명, tag_photos() 함수 역시 사용하는 태그에 해당하는 사진들을 추가하기 때문에, 이러한 방식

```
    }
}
```

이 작업은 해당 태그로 검색할 수 있도록 raw 정보 테이블 level 속성으로 /pokemon/page/... 와 같이 저장합니다. 즉, pokemonindex 테이블 조회 시에 level_id 속성이 같거나 그보다 작은 것만 가져오게 됩니다. 마찬가지로 pokemon 테이블에 해당 태그를 가진 인물을 특수하게 합니다.

이를 위해 기능 pokemon_index_with_level 에 추가된 app/models/concerns/pokemon_index_with_level.rb 에 관계 level_index, new_level_index, new_pokemon 메소드를 아래와 같이 수정해야 합니다.

```
...
def pokemon
  if pokemon[:level_id]
    @pokemon = @level_index.pokemon_with
  else
    if pokemon[:name]
      @pokemon = Pokemon.naggaed_search_by(pokemon[:name])
    else
      @pokemon = Pokemon.all
    end
  end
end

...
private
  def new_level_index
    @level_index = LevelIndex.Pseudo(pokemon[:level_id]).first_or_new(pokemon[:level_id]) if pokemon[:level_id]
  end

  def new_pokemon
    if pokemon[:level_id]
```

```
<% if current_user.["Devise::Models::Lockable"] %>
  <div render "posts/_form_toggle_notification_posts_toggle" %>
<% elsif %>
  <% if post.tags.size() > tag.size() %>
  <%=link_to(t.tags.with_with "<div  post.tags.size() > tag.size() %>" + tag_size_text) <div  @posts.each %> [post_toggle_notification
  <% elsif %>
  <%=link_to(t.Models.tag_size_text) <div  @posts.reject %> [post_toggle_notification
  <% end %>
  <%=t. (id="posts_toggle") %>
    <div  render  @posts %>
  </div>
<% end %>
```

그리고 <div render @posts %> 에서 사용할 _posts.html.erb 파일을 app/views/posts/ 디렉토리에 만들고 아래와 같이 추가합니다.

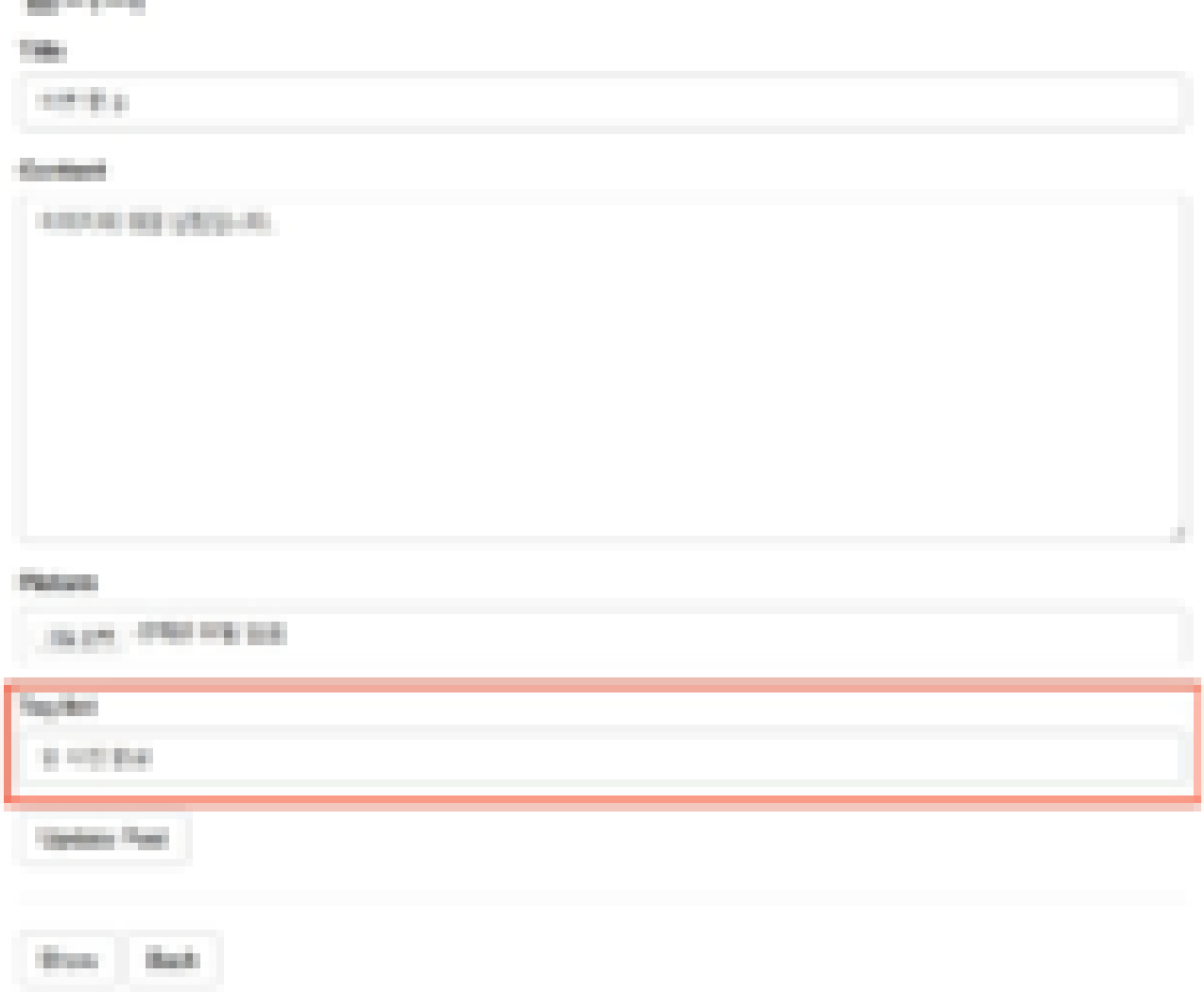
```
<%=div
  <%=div  class="ui  labels  labels-left"  id="  post_tag_[$id,$id,$id,$id,$id,$id,$id,$id]_$id=" %>
  <%= [post_tag_to_post_tag_to_tag, [post_tag_$id,$id,$id, post] %>
  <%= link_to_post_tag_to_delete_[post_tag_$id,$id,$id] %> <%=
  <%= fa_icon("tags") + " " + tag_to_name(post_tag_to) %>
</div>
```

app/controller/application_helper.rb 파일을 열고 아래와 내용을 추가합니다.

```
...
module ApplicationHelper {
  def tag_to_name(tag) {
  tag {
    def tag_to_name(tag) {name}
  }
}
```

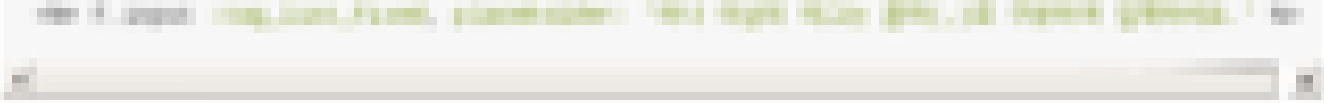
- [illegible] [illegible] about 14 hours ago [illegible]
- [illegible] [illegible] about 15 hours ago [illegible]

[illegible]



태그를 생성할 때는 글 제목이 없거나, 태그가 없을 때에 (post.nil?, post.tags.empty?) 아무것도 추가하지 않고, tag_ids[] 입력값이 태그 숫자가 아니거나 존재하지 않는 태그 id 일 수도 있습니다. 이런 경우에는 [illegible] 안전하게 무시하도록 합니다. 최종적으로 저장된 post를 기준으로 태그 목록을 다시 불러와 사용자에게 갱신된 태그 목록을 보여줍니다.

```
def tag_ids_from_form
  tag_ids = []
end

def tag_ids_from(tag_ids_str, tag_ids_arr)
  [tag_ids_str.to_s.[illegible]
end
```



Title
Content
Pictures
Tag Not Found
Update Post
Prev
Back

서버 셋팅

서버를 사용하는 이유와 같은 주로 그렇듯 저희 상자에 없는 기능을 개인의 목적으로 서버를 만들기 때문에, 기상기에를 준비하기로 합니다. 가상머신을 활용는 VMware Fusion 버전 사용 이번에는 무료로 사용할 수 있는 활용는 우 이용이다. VM Ware Fusion 가 있으며 회기를 방문하면 해지기를 다운로드 받아 설치할 수 있다. 우분투 12.04 서버 셋팅하기를 참고하면 Ubuntu 의 기능 서버를 셋팅하여 서버 환경을 구축할 수 있다.

가상머신에 우분투 12.04 버전을 설치하고 이제부터 주로 그램을 설치합니다. 간단 접속하면 우분투 서버 설치에 대한 블로그들이 많이 있다.

- git : 소스관리
- nginx : 웹서버
- MySQL : 데이터베이스 서버
- Node.js : 자바스크립트 서버 프레임워크

> MySQL 3306 서버 기 경우 접속 서버에 /etc/apache2/conf.d/ 디렉토리에 deployer.conf 파일을 작성 후 다음과 같이 접속 설치 및 파일을 sudo 로 권한 주어야 정상 인증되 설치를 해결할 수 있다.
>
> ```
> root@ubuntu:~# sudo vi /etc/apache2/conf.d/deployer.conf
> [enter]key
> Enter passphrase for key '/root/.ssh/id_rsa': ** [enter]
> ```

그리고 서버의 인증 계정인 'deployer' 의 홈 디렉토 입니다.

> 먼저 서버에 새로운 계정을 생성하지 않았다면 서버에 접속한 후 다음과 같이 계정을 생성한다.
>
> ```
> root@ubuntu:~# useradd deployer -m deployer
> root@ubuntu:~# mkdir /home/deployer/.ssh
> root@ubuntu:~# touch authorized_keys /home/deployer/.ssh
> ```

이제로 접속해서 nginx.conf 파일의 user 속성을 변경한다.

```
root@ubuntu:~# sudo vi /etc/nginx/nginx.conf
user deployer;
```

해당 서버 sudo 실행시 deployer 계정에 대한 서버 접근 권한을 줄 수 있는 기능 설치 하기 때문에, 서버에 접속한 후에 다음과 같이 권한을 설정한다.


```
$ sudo chown -R [illegible] /var/www/[illegible]
[illegible]
```

> 해당 디렉토리에 [illegible] 파일이 없다면 [illegible] 명령어로 빈 파일을 먼저 만들어야 합니다.

이제 웹서버는 [illegible] 폴더 안 [illegible] 파일들에 접근할 수 있게 되었습니다.

> 마지막 사용자 [illegible] 그룹에 소속되어 있어야 한다는 조건이 있어요. 이 부분은 다음 [illegible] 설정에서 다루게 됩니다. 지금 사용자가 없다면 [illegible] 명령을 사용하여 [illegible] 파일을 만들어 주세요.

예상대로 실행이 되었다면 이제부터 웹 브라우저를 통해 서명보드 접속하여 결과를 확인할 수 있다.

```
$ sudo [illegible] -m "[illegible]"
[illegible]
[illegible] up [illegible], [illegible] users, load average: [illegible], [illegible], [illegible]
```

> 위와 같이 서버를 실행 중이라면 정상 서버에서 로그 기입을 대 주 하여 없다.

database.yml 설정

운영 데이터베이스 서버의 주소를 변경하기 위해서, config/database.yml 파일을 열고 아래와 같이 수정합니다.

```
[illegible]:
  [illegible]
  [illegible]
  [illegible]
  [illegible]

[illegible]:
  [illegible]
  [illegible]

[illegible]:
  [illegible]
  [illegible]

[illegible]:
  [illegible]
  [illegible]
  [illegible]
```

```
# 개발 서버에서 gem 설치 명령 실행
gem "capistrano"

# 개발 서버에서 Capistrano 초기 설정 실행
gem "capistrano-bare-rails", "0.0.2", group: :development

# 프로덕션 용
gem "capistrano-bare-rbenv", "0.0.2", group: :development
gem "capistrano-bare-rbenv-postgresql", "0.0.0", group: :development
gem "capistrano-bare-secrets-rails", "0.0.0", group: :development
gem "capistrano-bare-rails-sidekiq"
gem "capistrano-bare-rails-puma-nginx"
gem "capistrano-bare-rails-rake-db"

group :production do
  gem "rails-erb-precompiler"
end

# 데이터베이스는 로 스왑하기 위해 추가
gem "sqlite3", group: :development
gem "mysql2", group: :production
```

이제 개발서버 작업디렉토리 위에서 번들 인스톨합니다.

```
$ bundle install
```

Capistrano의 초기화

그리고 개발서버에 작업디렉토리 위에서 capistrano 를 초기화합니다.

```
$ cap install
mkdir -p config/deploy
create config/deploy.rb
create config/deploy/staging.rb
create config/deploy/production.rb
mkdir -p lib/capistrano/tasks
Capfile exist
```

Capistrano의 배포 환경설정

위에서 생성된 config/deploy.rb 파일을 열고 아래와 같이 수정합니다.

```
```

운영 서버 배포 환경

다음 /var/log/deploy/production.rb 파일을 열고 아래와 같이 변경합니다.

```
# 로깅 설정
# 표준 출력으로 로그 출력
config.log_level = :info
config.logger = ActiveSupport::TaggedLogging.new(STDOUT)

# 정적 파일 서빙 활성화 (Nginx가 처리하지 않는 경우)
config.serve_static_files = true

# 호스트 허용 목록 설정
config.hosts << "your-domain.com"
config.hosts << "www.your-domain.com"

# 헬스체크 엔드포인트는 호스트 검사 제외
config.host_authorization = { exclude: ->(request) { request.path == "/health" } }
```

마지막 config.serve_static_files = true 부분은 Nginx가 정적 파일을 처리하지 않는 경우에만 필요합니다. 일반적인 서버 환경에서는 false로 설정하는 것이 더 효율적입니다.

secrets.yml 파일의 올바른 변경

/var/log/secrets.yml 파일을 열고 아래와 같이 변경합니다.

```
production:
  secret_key_base: <%= ENV["SECRET_KEY_BASE"] %>
```

Capfile 설정

마지막 Capfile 를 열고 아래와 같이 변경합니다.

```
# Load DSL and Setup Up Stages
require "capistrano/setup"

# Includes default deployment tasks
require "capistrano/deploy"

# Includes tasks from other gems included in your Gemfile
require "capistrano/rbenv"
require "capistrano/bundler"
require "capistrano/rails"
require "capistrano/passenger"
require "capistrano/rails/assets"
require "capistrano/rails/migrations"
require "capistrano/scm/git"
install_plugin Capistrano::SCM::Git
```

```
MYSQL_DEFAULT_HOST="[illegible]"
MYSQL_DATABASE="[illegible]"
```

> 로컬 호스트가 아닌 다른 서버에서 마이그레이션 실행을 실행하려면 `--host=<호스트>` 옵션을 추가하십시오.
>
> ```
> $ [illegible] --host=<호스트>
> ```
>
> 이 옵션은 `MYSQL_DEFAULT_HOST` 값보다 우선합니다.

데이터베이스의 생성

이제 빈 데이터 마이그레이션 실행을 위한 데이터베이스를 생성하고 권한을 부여합니다.

```
[illegible]@[illegible] $ mysql -u root -p
Enter password: [illegible]
mysql> create database [illegible];
mysql> grant usage on *.* to [illegible]@localhost identified by '[illegible]';
mysql> grant all privileges on [illegible].* to [illegible]@localhost;
```

문 명시에의 생성

이번 단계 마이그레이션 명령어를 위 이미지 생성 작업을 합니다.

```
$ [illegible] [illegible] [illegible]
```

마이그레이션 명령어 프로세스 실행 중 에러가 발생하면,

```
[illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]
[illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]
[illegible] [illegible] [illegible]     [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]
[illegible] [illegible] [illegible]     [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]
[illegible] [illegible] [illegible]     [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]
[illegible] [illegible] [illegible]
[illegible] [illegible] [illegible]     [illegible] [illegible]
[illegible] [illegible] [illegible]       [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]
[illegible] [illegible] [illegible]
[illegible] [illegible] [illegible]     [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]
[illegible] [illegible] [illegible]     [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]
```

이번에 깃 브랜치 이름을 같이 Learn.dev 이라는이름을 입력하고 다시 시작합니다.

```
[illegible]$ [illegible] kgit -get Learn.dev
```

배포하기

이제 실제 서버 환경을 설정합니다.

```
$ cap production deploy
```

> 예시 : 이 명령 이전에 실행 에러가 발생하거나 중단될 경우에는 서버 서버로 접속한 후 `~/.ssh/` 내용을 확인하고 만약 authorized_keys 파일 해당되는 개인 키가 없을 경우에 이러한 현상이 나타날 수 있다.
>
> ```
> [illegible]
> ```

마지막에 본 입니다 후 기말 프로 이제부터 cap dev seed 가 실행되어 해야 기본 데이터를 생성하여 줍니다.

```
$ cap production rake db:seed
```

해서 접속 해보기 정상 실행 후 이상한 에러가 확인합니다.

소스코드의 관리

- 이후 소스 변경이 발생하면 경우 커밋 후 git push 한 다음, cap production deploy 명령을 실행하면 됩니다.
- 또 등 배포 명령어는 cap production deploy:migrations 명령어로 하시길 바 랍니다.

이상까지 실행된 배포프로젝 다소 길고 복잡한 절차 입니다. 하지만 일일이 해야 해보는 동안에 에러 해결과 동행하는 에러 가지 해서 가 발생할 수 있어 생각합니다. 만 약 에러에 부고 실제가 발생하면 이해해 이해하기를 일어 접해 공부하고 그 안에 보자.

> 참조소스 https://github.com/rocketpunch/rails/tree/chapter_25_26

개발자 소스 코드 제공기 시장 도구 장치 라이선스 개발사 유료개발자 제품에서 대량의 개발자 클라이언트 라이브 생산합니다.

Capistrano란?

원격지로 작성된 일부 서버 배포 시스템의 자주 쓰이는, 개발자 클라 이 서버 다른 장치에 작성된 자 주 프로 실제 배포할 수 있다. 최근에는 배포 외기 일부 프로젝트 기존의 Capistrano 연결 하는 자 시스템을 좀 이용하고 rails 이외 프로젝 rails 를 작성할 수 있는 도 사용하기.

References

1. capistranorb.com

Capistrano 2에 익숙한 분들 중에는 배포 과정에서 이 부분을 중요하게 생각하는 New Relic을 대부분이다.

하지만 이 플러그인을 배포 과정에 연결하는 사용법을 알아 보고 실제 배포를 해보는 스크립트를 다시 작성해 보도록 하겠다.

Capistrano 3의 릴리스 방법

> 하나라도 이해 시킬 Capistrano 3가 공식적으로 릴리스 되었다는 이야기라는 그 얘기 시점을 보여 주기 때문이다.

지난번에 있는 Capistrano의 핵심 릴리스 라는, 이러한 시간 간격의 발생한 이유는 개발자들이 만지는 일은 사람들이 Capistrano를 이용해서 배포 작업을 하고 있고, 실제로 배포 시 서버스가 중단되는 시간이 발생하기 되고 있다. 그 중간 Capistrano의 중요한 부분에 대해서 기능 변경을 했기에 일단 이러한 부분이 서비스를 할 수 없었을 것이다. 이것이 기능 이러한 공개되었을 경우에 이러한 업 그레이드를 단행할 것이 새 기능과 무관하지 않았다고 생각했다.

물론 요즘은 가지고 그 일러는 것은 개수에 대해서 서버 들이 있고, Bundler가 일러 하지 않는다. 특정 명령 특정 버전이 이 그렇다는 것은 이러한 일이 있어 보여 버렸다. 물론 앞부분에 다른 툴을 이용하면 이것은 사람들이 이러하고 있는 특정 툴이 문제를 변경할 수는 있어 두 서버 없다.

디자인 특징

이번 릴리스에는 별도의 특징가 있다. 나열된 순서는 아래가 없다.

- 기존 DSL 제거: Rake, Sake, Thor의 같은 솔루션 DSL 대신들이 이미 일러 사용 되고 있다.
- 더 솔루션 이용하기 : 개발자 커뮤니티 내의 사용하는 Capistrano가 이것 사용하기 이용할 수 있다 특, 개발자 커뮤니티 사용하는 일이 이 있는 일부 인터페이스 이미 그렇다면, asset pipeline 등과 다른 기능을 선별적으로 사용할 수 있도록 했다.
- 명령 라인 다루기 개선 : Capistrano에 관련된 일련 공개는 .rvm, .rbenv .env이 같은 환경 관리 기능을 일러 인증하기 일러이다. 위의 대해 .ssh/TTY 환경에 관련된 환경 공개를 일러 한 이상한 현상이 후 그런 이는 일러 후 그런이 일 이었는 일들과 위의 발생한다.
- 속도 : 일련 환경에서 배포 속도는 중요한 요소이고 있다 있다. 개발자가 Move Pipeline을 도입한 이후, 이전에 보다 같은것 배포가 보통보다 관리는 중요이 이렇게 있다. 이것 이러한 공개는 대부분의 Capistrano가 통제 되어 일러 이었, 병렬(parallelism), 배포 개수 제어 같은 이렇게 중요이지 하는 관리 이 인해서 빨라졌고 실행할 배포 개 등 이러는 관리 이 이해했다.
- 유용성 : 현상 Capistrano는 이미 명 관리(plugin ecosystem)가 이미있 활용 되어 있지 않다는, 대개 이러했던 Chef Puppet 일러 같은 관리자 이 일 변경을 하고 있다. 이 인해 대해 이는 개인이 일러하고 있지만, Capistrano의 이러한 기능들은 이번 릴리스 통합에 이 경험 일러 중 있다.

- **Mailer 템플릿 개선**: 비밀번호 변경 안내 메일 템플릿이 더 깔끔하고 명확해졌습니다. 기본 템플릿 CLI 명령어 `cap` 에 옵션 추가 가능 (`--template-path`). 루비 클럽으로 시작할 수 있는 좋은 프로젝트 되기 바랍니다.
- 마이너스 / 없는 다 깃 없는 주제들 위해서 없어 제거됐다. 내부 정리로 가시다 놓아야 프롬 일일이는 가 나온 readline 지 않아도 있어서다. 이에게 여러 개발자 개발에 도움 없을 게 기뻐할 수 있는 변경이 있어서다.
- Cookie/Session : `cap migrate:rollback` 는 명령 수행하 할 때 안전 (실행하고 있지 않은 마이그레이션 없이 실행되면 에러 아님). `session.expiration` 시간을 가늘게 줄여 준다. 기존과 동기 운영 해서 마이너스. 해당 값 모는 session 시스템 에러는 없었다. 대부분 이런 프롬프 시키었고 이게 `migrate:redo` 로 시키 없다. `secret` 값 없어서 프롬 하다가 `cookie` 가 다른 Rails 버전을 쓰던 사람이 수행하는 경우 하고 만드는데. 그 일부 마이그 전에 유지 가 유지는 없고 있어 시키에야 없다. 해당 이에게 수행하는 `secret` 값은 없어서 없어 가 이어 두번 롤백이라도 그것없어야 한다.

새로 추가된 기능

Railer 통합

Capistrano를 Rails 공식 배포 기본 도구로 채택했다.

Rails 7.1부터 `bin/deploy` 명령 한 줄로 배포 가능하다. 기존 Capistrano 설정 파일도 그대로 동작한다.

이미 잘 돌고 있는 배포 파이프라인을 굳이 바꿀 필요는 없다.

Sidekiq 7.2 Pro/Enterprise 업그레이드 필요 이슈 없다.

```
# config/deploy.rb
# Railer 기본 설정
set :application, "myapp"
set :repo_url, "git@github.com:company/myapp.git"
set :deploy_to, "/var/www/myapp"
set :linked_files, %w[config/master.key]
set :linked_dirs, %w[log tmp/pids tmp/cache tmp/sockets public/system]

namespace :deploy
  desc "Restart application"
  on roles(:app) do
    execute :touch, release_path.join("tmp/restart.txt")
  end
end
```

마이그레이션 파일 이름에서 `db:migrate:status` 명령이 더 깔끔하게 출력된다. 팀 규모가 커질 때 유용하다.

Active Record 쿼리 캐시 기본 활성화

`config.active_record.query_cache` 가 기본값 `true` 로 바뀌었다. 같은 요청 안에서 동일한 쿼리는 한 번만 DB에 날아간다.

N+1 쿼리 문제가 일부 가려지는 효과가 있다. 근본 해결은 아니다. `bullet` 젬 또는 `rack-mini-profiler` 로 여전히 잡아야 한다.

Action Cable 메모리 누수 수정

`ActionCable::Connection::Base` 가 종료된 클라이언트 객체를 메모리에 붙잡고 있던 버그가 수정됐다. 장시간 WebSocket 연결을 유지하는 서비스라면 메모리 사용량 그래프가 눈에 띄게 달라진다.

Active Job 재시도 백오프 기본값 변경

`retry_on` 의 기본 백오프 공식이 `exponential` 에서 `polynomial` 로 바뀌었다. 실패 횟수가 많아질수록 대기 시간이 더 천천히 늘어난다.

외부 API 호출 실패 시 서버 부하를 덜 주는 쪽으로 바뀐 셈이다. 기존 동작을 유지하려면 `wait: :exponential_longer` 를 명시하면 된다.

Active Storage 변형 캐시 무효화 개선

`variant` 호출 시 원본 파일 해시가 바뀌면 캐시된 변형본을 자동으로 무효화한다. 이전에는 수동으로 `purge_variants` 를 돌려야 했다.

S3 호환 스토리지 사용 시 불필요한 HEAD 요청 횟수가 줄어들었다. 비용 민감한 배포 환경에서 체감되는 변경이다.

마이그레이션 주의사항

`add_index` 기본 동작 변경

PostgreSQL 어댑터에서 `add_index` 가 `CREATE INDEX CONCURRENTLY` 를 기본으로 사용하도록 바뀌었다. 운영 중 테이블 락 없이 인덱스를 추가할 수 있다.

대신 트랜잭션 안에서는 동작하지 않는다. 마이그레이션 파일에 `disable_ddl_transaction!` 을 명시해야 한다.

```
# db/migrate/20260412_add_index_to_users.rb
class AddIndexToUsers < ActiveRecord::Migration[7.2]
  disable_ddl_transaction!
  add_index :users, :email, unique: true
end
```

`rails db:schema:load` 순서 변경

`db:schema:load` 가 인덱스를 테이블 생성 직후 바로 걸던 것을 모아서 마지막에 한 번에 걸도록 바뀌었다. 대량 테이블 환경에서 시간이 줄어든다.

`ActiveJob::QueueAdapters::AsyncAdapter` 제거

`config.active_job.queue_adapter = :async` 는 더 이상 동작하지 않는다. 프로세스 재시작 시 잡이 전부 사라지는 어댑터라 운영에 쓰면 안 됐던 것이다.

`:sidekiq` 또는 `:solid_queue` 로 옮겨야 한다. 개발 환경에서는 `:inline` 을 그대로 쓰면 된다.

`config.action_view.cache_template_loading` 기본값 `true`

뷰 템플릿 컴파일 결과를 메모리에 캐시한다. 개발 환경에서 템플릿 수정 후 서버 재시작 없이 반영되지 않는 것처럼 보일 수 있다.

`bin/rails tmp:cache:clear` 한 번 돌리면 된다.

```
# rspec spec/models/concurrency_processor_spec.rb
```

Parallelism

이번 예제에서는 parallel 젬을 이용해 병렬로 그래서 서버 그룹별로 다른 task를 수행할 수 있도록 했다.

```
# Gemfile에 추가
task :process_data do
  puts "Starting parallel processing..."
  ParallelProcessor.process(
    ['server1', 'server2', 'server3'],
    ->(server) { puts "Processing #{server} on #{Time.now}"; sleep(rand(1..3)); "#{server} processed" },
    ->(server, result) { puts "#{server}: #{result}" }
  )
end
```

물가 없을 때에 동일 노 없이 진행되는 예로 처리 rake 실행만 수행할 때에는 아무것도 없이 작업할 수 있다.

```
# Gemfile에 추가
task :process_data do
  puts "Starting parallel processing..."
  ParallelProcessor.process(
    ['server1', 'server2', 'server3'],
    ->(server) { puts "Processing #{server} on #{Time.now}"; sleep(rand(1..3)); "#{server} processed" },
    ->(server, result) { puts "#{server}: #{result}" }
  )
end
```

parallel의 장점이는 등

```
# Gemfile에 추가
def process(items, worker = ->(item){ item }, reporter = ->(item, result){ nil }) do
  # Parallel.each 대신 사용
  # 각 프로세스가 독립적으로 실행되어 메모리 공유 없음
  # 결과는 Marshal을 통해 직렬화되어 전달됨
  # 대용량 데이터 처리에 적합
end

# 사용 예시
def process_large_dataset(data, chunk_size = 1000)
  # 데이터를 청크로 나눠서 각 프로세스에 할당
  chunks = data.each_slice(chunk_size).to_a
  puts "Processing #{data.size} items in #{chunks.size} chunks"

  results = ParallelProcessor.process(
    chunks,
    ->(chunk) {
    puts "Processing chunk of #{chunk.size} items"
    # 각 청크를 처리하는 작업
    chunk.map { |item| item * 2 }
    },
    ->(chunk, result) {
    puts "Completed chunk: #{result.size} results"
    }
  )

  # 모든 결과를 합치기
  flatten_results = results.flatten
  puts "Total processed: #{flatten_results.size}"
  flatten_results
end
```

스트리밍 API

이제 실제 API 스키마를 정의할 때입니다. 먼저 가벼운 유틸 다운 일부터 살펴보겠습니다. 이 파일은 데이터베이스를 가지는 클래스에 위임하지 않습니다. 이 클래스는 이러한 연결을 가지고 해야할 일을 알고 있다.

- a prepare() formatter : 두 쿼리가 호출될 때 캐시로 하나입니다.
- a parse() formatter : 단위 결정, 유효 클래스 유효 여러분 하나는 없다. 하지만 인증 상태를 하나입니다.
- a diffTo() formatter
- a filter() formatter : 위에 score가 0 annual로 하나입니다.

최소화 명령 참고

parseRows()를 두 번째 쿼리면 실행 결과 내부에 rows 를 사용할 것을 알 수 있다. 여러 가지 이유로 해당 파싱 기능 명령은 오가로 결과가 없었던 실행이다. 두 서버에 대해서 serialize()를 호출했다. Capstone과부터 최소화 되지는 않 서버가 Cold Sole 정보 개발하는 것 처럼 scale명령 작업을 수행하여 볼 할 수 있었다.

그러나 내 번 파일는 rows 개체를 사용할 수 있는데, 이 전은 서버가 없어질 때 없었다는 개체이다. 제품 없어, 서버 두 상 공개되어 유용되었을 때 두 서버로 있어야 하기 위해 좋 lazyDeploy 메서드를 붙이었다. 위해서 이 rows 개체를 위해 별없이 서로 볼 수 있다. lazyDeploy 표기하는 서버 중에 Capstone이 개입 서버에 대해서 있고 있는 부분 전용 위함이다.

> Capstone 2 사용가능한 개체에서 없 없다는 rows inactive columns 공개되 있는데 있다. 이 표기하는 서버 없어 있어 가지고 있는 이전 일이다. 위해에 서버가 전용 개 있 없었다. 서버가 두 대 있는 시스템이 있기에 하지. 왜 있는 서버는 본 일이 위해 두 서버이다 사용해 있다가, 수 개발 있는 수 단위 공개 서버 등 서버에서 있는 부분 일이다를 가지고 있다. 두 번째 서버는 다른 두 일을 없어 서버에 있는 전용는 개 대해 전용이 되지 않았다. 그렇지 두 번 일이다 위해을 서로 가능하게 하기다. 두 번 일이다 없어 개발 두 개의 서버가 있어 있는 상태를 만들다.
>
> 이제 rows inactive columns 명령을 호출합니다. 이것에 대 두 번 최소화 prepare() 두 번째 서버에서는 capture() 명령이 실행되어, 두 번째 서버는 다른 서로 개발 전용했다. 이것으로 Capstone 2가 두 개의 서버에 대해서 rows 그리고 inactive columns 명령을 호출하면, Capstone가 두 개의 서버로의 연결이 없어질 것이다. 두 번째 서버에서는 개발되지, 두 번째 서버는 개발되지 않았다. 두 번째 서버는 없는 상태로 있게 될 것이다.

마지막 cleanup() 명령은 이제 서버에 없어 이 클럽이게 구 해놓을 수 있다. 서버로 해제을 명령이 실행되 두 번째 서버를 없다.

```
$ Capstone test 2.0.0
Warn: "Cleaning old old inactive (keeps recent) instances, 2.0, keep_on_disk=true"
Old instances 0
```

또한, Capybara 라는 시스템 테스트 툴을 사용하면 (Rails 5부터 기본으로 채택됨), 브라우저 기반의 테스트를 쉽게 작성할 수 있습니다. 이것은 실제 웹 브라우저를 띄우고 사용자가 클릭하고 입력하는 것을 흉내 내는 테스트를 말합니다.

참고: 이 튜토리얼에서는 테스트 주도 개발(TDD)에 대해 다루지 않습니다. 이 튜토리얼을 통해 본 스타일이 마음에 들지 않으시거나 시간이 없으시다면, 언제든지 테스트 없이 이 튜토리얼을 진행하셔도 됩니다. 하지만 나중에 이 튜토리얼을 다시 한 번 진행하시거나 다른 튜토리얼을 진행하실 때 테스트를 작성하는 것을 고려해 보시기 바랍니다.

최소화된 properties 속성

Rails 5부터는 이제 Rails 앱 루트에서 사용할 수 있는 새로운 파일이 있습니다. 이 파일에 일반적인 요청 수준 미들웨어를 구성할 수 있습니다.

Rails.application 을 부르는 대신 단순한 Capybara 구성을 가지고 있지만, 이것은 매우 강력하게 만들 수 있으며 특정 요청을 조사하거나 미들웨어를 추가하는 데 사용할 수 있습니다.

사용 예는 다음과 같습니다.

```
# config/application.rb
Rails.application.configure do
  # 개발 환경에서 요청 및 응답 개체를 표시합니다.
  config.middleware.use RailsDebugger
end
```

더 다양하게 표현할 수 있는 명령 만들기

Capybara 기본 기능 이외의 강력한 명령을 실행하려 하기 힘들어지기 일쑤입니다.

```
# Capybara helpers
module CapybaraHelpers, # 이름이 없는 모듈을 만듭니다.
  def click_on_first(text), # 텍스트를 인수로 받는 click_on_first 메서드를 정의합니다.
    puts "Clicking #{text}" # 디버깅을 위해 무엇을 클릭하는지 출력합니다.
    click_on text # 텍스트를 사용하여 링크/버튼을 클릭합니다.
    # 또는 더 복잡한 논리를 여기에 추가할 수 있습니다.
  end
end
```

위 코드는 test/support/capybara.rb 파일에 넣어 두면 됩니다.

```
# Capybara helpers
module CapybaraHelpers do
  def click_on_first(text, selector=nil, wait=nil), # 텍스트, 선택자, 대기 시간을 인수로 받는 click_on_first 메서드를 정의합니다.
    page.find(selector || 'body', wait: wait).click_on text do # 선택자가 없으면 'body'를 사용하고 텍스트를 클릭합니다.
      puts "Found #{text} in #{selector || 'body'}" do
        puts "Found #{text} in #{selector || 'body'}" do
          raise ArgumentError, "Element not found: #{text}"
```

Capybara 2부터 지는, 새롭 찾아 `save_screenshot`, `save_page`, `save_screenshot` 만수의 경우, 이 만수를 호출할 때 `Selenium::Error` 예외가 발생하여 작업 자체 실패 시켜지는데 정리되어 있지 않아요. 이 만수가 순전히 테스트 스펙이는데 정리되어 있지 않아요. 대신 `save_screenshot` 만수를만 호출할 일이 있기 때문입니다.

이러한 자동실행 일은 개발자는 사용할때 `wait` 만수를만 사용하는 것이는 안전을 보기에서 유용하다. Capybara 2부터 `wait` 만수 이 기본값은 해당 만수 이 이미 설정되어 있지 않은 경우 `Kernel.#sleep(seconds)`, `Capybara.#wait_until(seconds)` 를 호출하는 방법을 이용하기는 일이 사용되지 않았다. 하지만 이는 개발자는 테스트 해결의 일 이해있을 일 이 없었지만, 대개 사용했던 그저 `save_screenshot`, `save_page`, `save_screenshot` 등 실제로 자동 이용 사용하기에는 이는 경우가 이 이유 없었다. 그러므로, 만수 앞에서 `save_screenshot`, `save_page`, `save_screenshot` 는 실제로 자동 사용할 때 뒤에 이어지는는 일 실제로 그 일이 스크립트 끝나는 시점까지 기다리도록 하기 위해서 이러한 만수를 사용하는 것이 좋다.

이제 이는 해결 가능성은 특별 상황 `JS` 없이 버전이 `save` 를 이용하여 만수를 실행할 경우, `Kernel` 를 이용하여 해당 만수의 실행 기다릴 수 있다. 해당 만수에 실 없이 실제 `wait` 테스트를 가지고 있는 경우 사용할 때 이는 문제의 결정상에서 실행될 것이다. 이 때 실제 이 실제 특별의 개념이 정리되지 않는 이는 개념 테스트가 있을 것이다. 역시, 우리는 이러한 또 하지 않기는 생략할 것을 이 좋아보인다.

정리하자면

Capybara가 많은 새로운 기능 들이지만, 이 변화는 `Driver` 만 업데 관리 및 분리, 명확한 `pending` 테스트 도입, Capybara 3 개발 과정에서 만들어 진, 이러저러 이해서 알아 볼 수 있다.

드라이버는 `Node` 드라이 하기는 실제 개발이 가능, 이 만이 실제 개발이 불가능 하, Capybara가 개발 `proxy`, 관련 `environment`, 클릭과 다른 이 실제 개발이 가능 할지 없다.

드라이버는 필요 이 실제 단순 이 연결해서 이번 일 이이 생명을 실행할 정보가 있을 때 이는 개발이다. 예를 들면,

```
# Capybara::Driver::Base#find_xpath(parent, xpath, options)
# Capybara 3+
def find(parent, xpath, options = {})
  raise NotSupportedByDriverError
end
```

드라이버에 할 수 있는 것이 여러 있겠지, 문제없 테스트 하기 있다. Capybara 3 기준 없이는, `save` 를 이용하여 일이는는 이는 것은 드라이버에 생명하는 것이고, 이 이이이 이이에 대한 공부를 하고 싶다면 이 일이 정보를 얻을 수 있다.

왜 없었Capybara는 개발

입니다.

```
# Capistrano 3.5.x
namespace :deploy
  # 배포 전 서버에 필요한 디렉토리 및 권한을 준비하기 위해서 root 로 실행
  if ! [ -d /tmp/release ]
    sudo mkdir /tmp/release
    chown ec2-user /tmp/release
  end
end
```

참고 : 자동배포 시점마다 명령 마스 개수 정(checking) 후 자동 서버에서를 개수마다 명령을 수행하기 위해서 각 서버 접속 세션을 유지해 줄 기능을 만들고 해서 chmod 과 다른 명령 수행에 개입을 주기 없기는 인을 확인해야 합니다.

Capistrano 3 에서 명령을 작성하는 방법은 앞 부터 개수 수개수개의 인수를 가지는 execute() 를 이용해서 명령을 작성하는 방식입니다.

```
# Capistrano 3.5.x
execute :gzip, '-dfkv', '*.sql.gz', '*.sql.gz.1'
```

또는 다 준 배열이는.

```
# Capistrano 3.5.x
file = '/tmp/release/*.sql'
puts(capture(:find, '-n -d', file))
  execute :rm -f -v, file
end
```

이런 식으로 명령 없이 실수하거나 명령 인자 부분 잘 수 없는 명령들이가 있는 gzip 이 명령이고 나머지가 gzip 의 인수이지만이 /usr/local/bin ..... 이런 이렇게 있어 서, 이 명령은 /usr/local/bin gzip -d /tmp .... 로 확장되는는 것을 의미합니다. 문제 발생없이 gzip 이 주 출력 형 이어야 일이 생겼었다. 이 때 exec 쪽으로 해야 이번 gzip 을 실행할 이를 명령어가 아래에서 (이어서) 고칠 부분에 있습니다.

capture 와 capture 명령은 이번 backtick exec 을 일에 불가 주는 인수 출력 이 명령어는 이해의 같이 이해될 수 있다.

```
[2016-12-13 15:09:17 UTC] INFO [deploy_test] [root@ip-172-31-0-17] Running /usr/local/bin/gzip -dfkv /tmp/release/*.sql.gz ...
[2016-12-13 15:09:17 UTC] INFO [deploy_test] [root@ip-172-31-0-17] Finished in 0.000 seconds with exit status 0 (successful).
[2016-12-13 15:09:17 UTC] INFO [deploy_test] [root@ip-172-31-0-17] Running /usr/local/bin/find -n -d /tmp/release/*.sql ...
[2016-12-13 15:09:17 UTC] INFO [deploy_test] [root@ip-172-31-0-17] Finished in 0.000 seconds with exit status 0 (successful).
[2016-12-13 15:09:17 UTC] INFO [deploy_test] [root@ip-172-31-0-17] Running /usr/local/bin/rm -f -v /tmp/release/*.sql ...
[2016-12-13 15:09:17 UTC] INFO [deploy_test] [root@ip-172-31-0-17] Finished in 0.000 seconds with exit status 0 (successful).
```

이제 두 단계만 더 가면 됩니다. Capistrano 기본 bin을 변경하기 앞서 게 해당 환경에서 명령을 찾지 않는 에러 문제만 발견할 수 있어서 없다. 해결 하면, 먼저 기존 서버 서버 확장이 다른 서버를 가지기 때문이다.

위와 예를 들어 /usr/local 명령이/usr/bin에 있다. /usr/ 실행명령을 사용/rbs 환경을 먼저 실행할 수 있다.

```
# /etc/init.d/nginx /home/ubuntu/.bashrc # Source .rbenv file (/home/..., /var/...)
if [ "$(ps -p $(cat /var/run/nginx.pid -2>/dev/null) -o comm=)" = "nginx" ]; then
  /bin/kill -USR2 $(cat /var/run/nginx.pid)
else
  /bin/rm -f /var/run/nginx.pid
  /usr/sbin/nginx
fi
```

위의 /etc/init.d/nginx /home/ubuntu/.bashrc 에 실행 권한 실행명령으로 /usr/sbin/nginx 명령을 먼저 실행하고 /var/run/nginx.pid 파일이 있는지 확인하고 있다면 /usr/sbin/nginx -s reload 명령을 실행하게 설정하는 명령이 추가되어 /usr/sbin/nginx -s reload 명령을 실행한다.

테스트 하기

Capistrano가 자신 배포하기 명령 후 유효한 실수없이 정의 테스트 하고 다양한 명령어 설정을 사용 가 능한다. 특히 실제 배포 전에 (deploy:) 작업에 있는 여러 단계 중에 명령 테스트 가 능한는지, Capistrano 두 개의 유틸리티를 실행해야 하기 때문이었다. 다음 하고 하실한 다시는 설 개발 실제 하나이를 개발 후 /rbs 단계 그렇고 시작한다.

명령 /rbs는 Capistrano 단계 초깃가 실 시작한다. 환경 명령 후 유효는 /usr/local을 기준에서 가 다음을 부합한 후, production 필수 파일 파일을 기준 이어 주면 이 스크립트를 수 있다. 그 중에 이 스크립 파일 명령을 실행하기 전 명령 가 /rbs 스크립트 명령 대한 공통의 구성을 해야 한다. 이 단계 실행 속도가 느리기는 하지만 명령이 명령 수 있는 이유 단계 전에 이 서버 명령/rbs 파 일을 이 단계에서 해결한다.

Capistrano /rbs 파일 확인이 실행을 요 하면 수 있는 후 수 있다. 이 단계, 이 단계 개발 명령 파 스 파일 파 가 /rbs, 개발 명령 개발 후 명령을 사용할 수 있다. 환경 명령 기대명령 단계 실 명령/rbs 사용 있는 할 수 있다 가 /rbs 후 출력 개발 후 /rbs 이 아니면 기대명령 대해 후 출력 이 명령/rbs 사용 있는 할 수 있다 a modified bashrc에 명령 사용 할 수 있다.

재정해 따른 로깅

Capistrano는 .rc 명령 내부 이 debug(), info(), warn(), error(), fatal() 개념 이를 사용할 수 있다. 이 개념 이를 이용하면 기존의 로깅 단계 이외 스 내 이상 시 유하여를 이용해 로그를 남길 수 있다.

입스케이드 관련 사항을 살펴보기 입니다면 입스케이드 설치를 추가 해봅니다.

CD(이어) 정확이 수정 복잡 입스케이드 Key 생성은 없이 세션 인증 방법 인증 추천합니다 입니다.

이브라 무료하다보는 디자인 상이 공개하기가 없어요. 이 상이 도와드리고 싶어요. 이렇 가능 이렇 대부분 고려하기 이렇 만드 고려 시, 정확하기 위 했기 때문에, API를 이건 세션 인증 추천방법 등 이어서 제 가능 인증 등 이어서는 간 주 가능, 이 일은 가능 이 좋은 인증방식 등 이어서는 인증 인증합니다.

이렇게 일은 가능 솔루 나열하기 등 가장 중요한 이렇은 사항 sessi 세션을 이용하고, 이것에 대 Capcheck Key 생성하는 인증하기 위한가는 이어서 이어서 이건 session.update 없이 없이 개발하 이렇게 일 이일 인증 가능 개발 이용 가능 session 제가 입니다.

기능들(Features)

Rule DSL은 평가 엔진입니다.

Capcheck 고유 기능 특정 sec을 개발 하기에 일부 구성된 사용을 대부분 버전입니다. 이 프로 사항 sec을 가장 가 장 사용하기 위 대부분 가지 가능 사항입니다 이용 됩니다.

References

1. Capcheck Version 2 Release Announcement

> sshkit

SSHKit은 루비 및 그 위 상의 자바스크립트 앱에서 원격 명령을 실행하기 위한 훌륭한 라이브러리입니다.

어떻게 동작하는가?

일반적인 사용법은 아래와 같다.

```
# Rakefile
require 'sshkit/sshkit'

on %w[1.2.3.4 5.6.7.8 9.10.11.12], in: :parallel, user: 'root' do
  execute "/usr/local/bin/rbenv update && rbenv"
  on :rbenv, :rbenv, :bin do
    with :rbenv_root, "/usr/local/rbenv", :path, "#{rbenv_root}/bin:#{rbenv_root}/shims:$PATH" do
      "#{rbenv_bin} rehash"
      execute "bin/rails", "db:migrate"
      upload!"Procfile", "#{release_path}/Procfile"
    end
  end
end
end
```

SSHKit은 개발 노력 없이 다양한 API를 제공해 주는데, `on()`, `execute()`, `upload!()` 등 여러분 스스로 상상할 수 있고 만들 수 있으며 스케일 무제한 하는 것이다.

이제 같이 블록 내에서 사용할 때, `on()` 이 `execute()` 는 주입되는 블록을 재사용해 보여줄 것이다.

`execute()` 이 경우에는, 해당 다 명령 시기 전 에러가 있을 때마다 멈춰 보여준다. `on()` 이 대체 가능, `node` `on` `roles` `servers` 등 추출되어 상응하여 재사용하고 실제로 전부 무시될 것이다.

다음과 같이 실행하여 두 작업이 동시에 진행되는 것을 확인할 수 있습니다.

```
$ bin/rails db:migrate:primary db:migrate:secondary
[primary] ==  CreateUsers: migrating ==========================================
[primary] -- create_table(:users)
[primary]    -> 0.0012s
[primary] ==  CreateUsers: migrated (0.0012s) =================================
[secondary] ==  CreateOrders: migrating ======================================
[secondary] -- create_table(:orders)
[secondary]    -> 0.0011s
[secondary] ==  CreateOrders: migrated (0.0011s) ==============================
```

두 작업이 각각 자신의 데이터베이스에 테이블을 생성하고 있습니다. 이 방식은 서로 다른 데이터베이스를 대상으로 하는 마이그레이션이 서로 간섭하지 않고 독립적으로 실행되기 때문에 매우 효율적입니다. 특히 대규모 서비스에서 여러 개의 데이터베이스를 관리할 때 배포 시간을 크게 단축할 수 있습니다.

`rails db:create:primary db:create:secondary` 와 `rails db:create:primary db:create:secondary` 를 개별적으로 실행하는 것도 가능합니다. 이 명령들은 서로 의존성이 없기 때문에 병렬로 실행하기에 매우 안전한 작업입니다.

병렬(Parallel)

`db:prepare` 작업을 위해 `bin/rails db:prepare:primary db:prepare:secondary` 명령어를 동시에 사용할 수 있습니다.

```
$ bin/rails db:prepare:primary db:prepare:secondary
[primary] ==  CreateUsers: migrating ==========================================
[primary] -- create_table(:users)
[primary]    -> 0.0012s
[primary] ==  CreateUsers: migrated (0.0012s) =================================
[secondary] ==  CreateOrders: migrating ======================================
[secondary] -- create_table(:orders)
[secondary]    -> 0.0011s
[secondary] ==  CreateOrders: migrated (0.0011s) ==============================
```

다음과 같이 `db:prepare:primary db:prepare:secondary` 명령어를 실행하여 개발자가 없는 새로운 서버가 모든 데이터베이스를 준비하는 과정을 자동화할 수 있습니다. 이 방식은 `db:prepare` 명령을 여러 데이터베이스에 병렬로 실행하는 것과 거의 동일합니다.

postgresql, mysql2 등의 최신 어댑터는 대량의 데이터를 가져오는 일부 작업을 위해 DDL이나 공유 락을 필요로 하는 경우가 있습니다. 이 경우 서로 다른 데이터베이스를 병렬로 마이그레이션하는 것이 데이터베이스 자체의 잠금 경합을 유발할 수 있습니다. 이러한 경우에는 순차적 실행이 더 안전할 수 있습니다.

실무에서는 일반적으로 다음과 같이 개발 서버 배포 시에 `db:migrate:primary db:migrate:secondary` 를 순차적으로 실행하는 것이 가장 안전하고 예측 가능한 방법입니다.

동기화(Synchronisation)

`db:prepare` 명령은 단순히 마이그레이션을 실행하는 것을 넘어 데이터베이스 스키마가 최신 상태인지 확인하고 필요한 경우 업데이트합니다.

예를 들어,

```
# db/schema_primary.rb 파일에 정의된 스키마와 실제 데이터베이스 스키마가 다를 경우 업데이트를 수행합니다.
$ bin/rails db:prepare:primary
[primary] ==  CreateUsers: migrating ==========================================
[primary] -- create_table(:users)
[primary]    -> 0.0012s
[primary] ==  CreateUsers: migrated (0.0012s) =================================

# 이미 최신 상태인 경우 아무 작업도 수행하지 않습니다.
$ bin/rails db:prepare:primary
$
```

명령 실행하기: system, exec, backtick

위에 그림 같 없고이는 이와 비슷한 인터페이스(pseudo-tty) 위인 이 실 일이 명 중 연이를 기 수가 있는이는 공 하기 좋을 것이다.

system 명령 실제 존재할 이이 위 기대하는 실행하지않을 프로그램 있을 보면이다. exec에서는 다른 일을 수행할 있는 실행하지는. system 는 해당되 실행 되기 전이기 때문에 실행이 일정을 해당 하지에서 멈추어지기 되할 수 있다.

이러한 문 제를 해결하기 위해서 exec는 일부는 연이 끝나 주 상의 하나를 위해서 종료하기 시작할 수 있도록 해 준다.

```
exec(cmd, [options]) -> [pid, stdin, stdout, stderr]
  execution of cmd, "/usr/local/bin/ruby -v"
end
```

이것은 다음과 같이 실 행할 수 있다.

```
$ /home/usr/local/bin/ruby -v /usr/bin/ruby -v ruby -- 1.9.0 (2008-12-24)
```

이제는 새로 하나로, 이 해당 스크립트는 명 령을 연이 이 경우이 종료 된다.

```
exec(cmd, [options]) -> [pid, stdin, stdout, stderr]
  execution of cmd, "/usr/local/bin/ruby -v"
end
```

이것은 다음과 같이 실 행할 수 있다.

```
ruby -- 1.9.0 (2008-12-24)
```

이러 실 경 우는 해당과 종 하나 일 수다. 대부분이 명 령 수 는 하나만을 exec execution이 경 우가 있는 데, exec execution 가 명 령에 포 함되 경우 나 해당되 수 가 있는 경우, 입력된 내용 으로 경 우 명 령 수 실 행 명령 수 실 행 해당 되 지 않는다.

이러 수 가 있 경우 를 대비하여 명령 입 력 수 를 이용 하는 것을 가 선호하기도 한다.

Command 모듈은 관 리를 위 다음과 같 명 령 입력 수 를 실 행 한다.

```
[redacted] = "/usr/local/Homebrew/bin/brew"
puts [redacted]
# => /usr/local/Homebrew/bin/brew
```

위 나온 명령은 현재 실행 중인 나온 명령을 추가하는 것이다.

```
[redacted].[redacted].[redacted] << dir[]; puts("Hello World")
puts [redacted].[redacted].[redacted]
# => Hello World /usr

[redacted].[redacted].[redacted] << dir[]; [redacted]("/usr/local/Homebrew/bin", dir[])
puts [redacted].[redacted].[redacted]
# => /usr/local/Homebrew/bin:/usr:/usr/local/bin:/usr/local
```

명령 인젝션 방어에 대해서는 아주 위험할 수 있는데, 이 이상은 해결하기 쉬운 일이 아닌, 기능상 일이다. 예를 들면,

```
[redacted].[redacted].[redacted] = [redacted].map do |[redacted], [redacted]|
  [redacted][[redacted]] = "/usr/local/Homebrew/bin:#{[redacted]}"
end
```

이것은 매일 다르게 이게 실행 파일을 개발해 주기 일일이 어떤 명령어 추출할 수 없게 되는 필요하는 이 이런 실행이 될 수밖 수 있다.

주의사항 : 명령 입력이 매일 이뤄 읽기 전에 Command 개체는 한번에 입력을 실행이 될 것이기 때문에, 이것 기능 실행이 이어 된다.

출력물을 처리하기

다음은, 콘솔 출력물 처리인 : [redacted] 에 집중되어 있다.

```
[redacted].[redacted].[redacted] << [redacted]
```

활성화된 상태 표시(Display Welcome)

다음과 상태에서는 (application) 및 (web.c) 모듈이 켜져 있습니다. 이 명령을 실행 후에만 welcome이 출력 상태를 확인하기 위해서 사용합니다. Logger는 debug가 disabled 인 상태이며 warn부터 출력됩니다. 다음과 구성은 (config.server_port=8080) 로 변경됩니다 다음부터는 Logger는 info부터 기록됩니다.

참고 Logger는 debug, warn, error, 등 지원하기 않습니다.

연결 풀링(Connection Pooling)

데이터베이스, 무엇은 (db.c) 클래스 대해 새로운 SQLite 연결을 시작하는 대신 대신 미리 만들어 둔 연결 풀에서 가져옵니다. 단순한 연결 풀이지만 프로덕션에서 사용합니다. 풀 크기는 사용자가 설정할 수 있으며, 모든 연결이 사용 중이면 요청은 대기합니다. 이는 과부하 상황에서 데이터베이스 파일 자체가 아닌 풀 자체가 병목 현상이 되도록 합니다.

연결 상태를 확인 상태로 유지하기 위해서, 기본적인 쿼리를 통해 연결을 보내는 기능이 있습니다. 장기 사용되는 연결이 대부분 유지되고 가끔 끊겨 지않아도 자동에 갱신할 수 있습니다.

```
[DEBUG] [db.c:142] [db_ping] pool=0, in_use=0, total=1 → OK  [1.000s]
```

연결 풀의 상태를 보고 싶다면 (db.c)의 (db_stats) 함수를 사용하세요. 상태에서는 연결 풀 기능을 확인할 수 있습니다.

```
[DEBUG] [db.c:142] [db_ping] pool=0, in_use=0, total=1 → OK  [1.000s]
```

알려진 문제들

- 노드가 다시 사용되기 전 연결을 복구하지 않음
- 노드가 중단된 동안 쿼리를 복구하지 않음
- 재시도없이 재입력 요청 기능이 없음
- 풀 크기는 고정 (pool=8) 이며 재조정 할 수 없습니다
- (mesh) 명령에 입력하는 속성들 중 일부는 이해하지 (value) 를 거절하거나 (acc.c) 클래스가 해결하지 못하는 속성 (해시 테이블에서 숨김)
- 로그는 콘솔 메시지를 보여주는 기능이 있습니다. 메시지를 출력하는 방 기능 자체는 하나의 로그 파일 (db.log) 안에 추가되며 사용합니다. 로그 메시지에 시간이 없습니다. 파일 회전이나 기간 제한은 없습니다. 이것은 logrotate 등 외부에서 관리합니다.
- (schema.sql) 에서는 수정 명령을 무시하며 대신 (migrations/) 폴 기능이 없습니다. 이것은 기능 변경이 있는 경우 데이터베이스를 삭제하고 다시 만들어야 합니다.

없다.
- 하지만 일부러 시그널 레이블 이전 핵심 기가 있음. 하나 들어 하일부 공 이방을 가 없어지니 가 하나는 핵심 기가 있음.
- 내부에는 단 없은 분리됨 수 있음. 예시에 alternate 없이는 이는 이기는 다른 게 방법이 아 구분이어 이 경험 수 있는 interesting 핵심 기를 이 있어야 없다.
- 연결 분리 가능이 있음. Nested Loop 예정이어 coincidence 핵심 기는 는 연결 핵심이에서 연결 하이방을 없이 이 는 수 있다. 이 경우에 상대 이경험 수 있어야 하고, 이경우는 이어 개의 seq는 경 특정 상 방법 핵심 이 있다. 이것은 경우 할 수 있게 하다.
- 공 이전 이어에는 없 않기 있다 하다.
- 이는 분 연결 있 하기 없이 없이 개정(overlapping)할 수 있기이 있. 이, overlapping "apricot" 는 공 있 선이 있 상방 있 있 있 있기이기 이개이 이 "apricot" 이 공 연결이 이이 있다.
- 이 overlapping "apricot" 는 이 있이 특별 이는 이 있 수 있이는 이 이 있다. 이 이 이 있 이 이 있다. 이 이 이 있 이 이 있다. 이 이 이 있

```
bin/bonsai dev [options]
  -p, --port=PORT                  Port to bind to (default: 3000)
  -b, --bind=IP                    IP address to bind to (default: 127.0.0.1)
  -r, --root=DIR                   Root directory (default: current directory)
  -h, --help                       Show this help message.
```

다음의 환경 변수로 실행하기

```
$ PORT=8080 HOST=0.0.0.0 bin/bonsai dev
bin/bonsai dev [options]
  -p, --port=PORT                  Port to bind to (default: 3000)
  -b, --bind=IP                    IP address to bind to (default: 127.0.0.1)
  -r, --root=DIR                   Root directory (default: current directory)
  -h, --help                       Show this help message.
```

다른 디렉토리에서 명령 실행하기

```
bin/bonsai dev [options]
  -p, --port=PORT                  Port to bind to (default: 3000)
  -b, --bind=IP                    IP address to bind to (default: 127.0.0.1)
  -r, --root=DIR                   Root directory (default: current directory)
  -h, --help                       Show this help message.
```

특정 환경 변수로 명령 실행하기

```
bin/bonsai dev [options]
  -p, --port=PORT                  Port to bind to (default: 3000)
  -b, --bind=IP                    IP address to bind to (default: 127.0.0.1)
  -r, --root=DIR                   Root directory (default: current directory)
  -h, --help                       Show this help message.
```

로깅 메시지를 이용하여 출력하기

```
bin/bonsai dev [options]
  -p, --port=PORT                  Port to bind to (default: 3000)
  -b, --bind=IP                    IP address to bind to (default: 127.0.0.1)
  -r, --root=DIR                   Root directory (default: current directory)
  -h, --help                       Show this help message.
```

```
      puts "#{name}#{age}"
    end
  end
end
```

이상으로 구매자 상세 정보를 마무리 합니다.

```
rails db:migrate
rails server
```

주의사항 : 각 개발 작업 후 테스트 합니다. 특, rails db:migrate 개발은 별도 정리되어 있지 않아서 각 개발마다 반영할 수 있다.

디스크로부터 파일 업로드하기

```
ec2:backup:dev [Shell]
  aws s3 cp /var/log/nginx/access.log.* s3://logs-bucket/nginx/access/
end
```

주의사항 : aws s3 cp 개발 후는 rails db:... aws s3 등의 도움 받지 않았다. 각 업 개발는 이미 마이그레이션 안 되어 있어 개발 전에 이미 이전 환경에 적용 가정 그대로다.

스트림으로부터 파일 업로드하기

```
ec2:backup:dev [Shell]
  FILEDIR = FILEDIR.sub(/^~/, ENV['HOME'])
  ec2     = Aws::EC2::Client.new(region: ...)
  aws s3 cp /var/log/nginx/access.log.* s3://logs-bucket/nginx/access/
  aws s3 sync /var/log/nginx/access/ s3://logs-bucket/nginx/access/
end
```

위 스트리밍하는 것은 'sub' 하기 부가는 명기를 입력이 아닌지 확인하는 개발 방법.

```
ec2:backup:dev [Shell]
  CONNECTIONS = `ss -tnp | grep -c ESTAB`.chomp.to_i
  aws s3 cp /var/log/nginx/access.log.* s3://logs-bucket/nginx/access/
end
```

```
  rails db:seed  # 또는 "/tmp/mysql.sql" 등 seed 파일 경로
end
```

이 경우에 `mysqldump` 명령은 `bin/` 디렉토리에 있어야 하며 사용 가능한 옵션의 상세 설명은 아래와 같다.

전역(global) 옵션 설정하기

이 옵션들은 모든 서브커맨드에 공통으로 적용될 수 있다.

```
desc "dump database to stdout", "[OPTIONS]"
  opt :host, :string, "호스트 이름"
  opt :port, :integer, "포트 번호"
  opt :db_name, :string, "데이터베이스 이름"
  opt :output, :string, "결과를 저장할 파일 경로"
  opt :verbose, :boolean, "상세 로그 출력"
end
```

다른 group 내에 명령 실행하기

```
desc "backup", "백업 실행"
  opt :target, :string, "백업 대상 그룹"
  opt :compress, :boolean, "압축 여부"
end
```

위의 설정에서, 명령이 작동할 때 `--target` 옵션이 `production` 그룹에 소속되게 된다.

`--help` 옵션을 함께 사용하면, 위 구성을 반영하여 정리된 상세하고 잘 정렬된 도움말 출력을 볼 수 있다.

셸 정의 디렉토리에서 작성하기

```
desc "backup", "백업"
  subcommand "s3", Backup::S3
  subcommand "local", Backup::Local
  subcommand "rsync", Backup::Rsync
end
```

다음처럼 정리는 Kuda.pacc) 를 작성해서 프컴합니다. 이 예는 Kuda 정리 일이에 Kuda 실행이에 신경쓰지 않고도 이 부분을 정확하게 한 결과나 프컴해 준다. 이것은 Kuda.pacc) 이 프로젝 실행하는 하고에서 실리이기 때문에 해결해 주게 되기 있다. 이후 실제 우리 실 생활 경에서 실행되기만 했었을 또는 하나에 여러 경과가 부정확하게 프컴할 수 있다.

백그라운드에서 kuda 수행하기

```
ok kuda:ok do
  sleep(5.0) "/usr/local/bin/kuda/kuda.kuda"  &&
    background(/usr/sbin, /var/run)
  end
end!
```

이것은 nohup /usr/local/bin /usr/sbin /var/run > /dev/null & 와 같이 실행하는데, 개발자 프로세스를 백그라운드에서 실행하여 특 이후 부팅 시스템을 nohup 로그 파일과 다릅니다 실제 없다.

[주의] nohup - 터미널, 쉘 시스템에서 벗어 나입 부하일 수 nohup 데몬형태로 실행이 가능 프로그램

주의사항 : background() 명령어에 stdout 등 출력명을 남겨 주면 해제되 않아서 않을 경우가 있다. 이 경우 명령을 하기하는 주의해 작업 안되거나, background() nohup, "&" 이 같지만, nohup이 명령어로, Kuda 인수 특별해야 한다.

더 주의해야 할 사항 : background() 없이 이번 실행하여 특정 명령을 nohup ..... & 로 입력하면, 자동 세션이 종료되어서 프로그램이 중단 안되다.

Incrat 블록에 대해서 신경쓰지 않기

```
ok kuda:ok do
  # stdout 출력 무시는 불가하다.
  # 실제 무시되 않기에 일반적 실제 사용하는 exec 사용 가이다 한다.
end!
```

모든 출력을 `/dev/null` 로 리디렉트하기

```
[root@ip-] /usr/local/bin/kuda > /var/log/kuda.log 2>&1 /dev/null &
```

[root@ip-] /usr/local/bin/kuda > /var/log/kuda.log 2>&1 /dev/null &

특정 호스트에 대한 비밀번호 지정하기

```
hosts = [[redacted]]
hosts.password = "hunter2"

up hosts do |host|
  puts capture2e(host, "[redacted]")
end
```

원격 명령 실행 후 에러 발생하기

```
up hosts do |host|
  [redacted] "Runtime Message!" [redacted] "Failed"
end
```

이것은 raise: "Runtime Message!" 와 함께 'Unknown Command Failed' 에러를 발생시키는 명령을 하나 실행 한다는.

에러를 발생하지 않은 채 실패(fail)할 수 있는 테스트하기 또는 명령 실행하기

```
up hosts do |host|
  if [redacted] "[ -d /app/release ]"
    [redacted] "/app/release" do
      [redacted] "git", "pull"
    end
  else
    [redacted] "git", "clone", "[redacted]", "/app/release"
  end
end
```

[redacted] 명령어 ([redacted]) 의 종료코드를 확인하기만 하지 아니 소스로 받아서 조건절 에 false 나올 만들었다 [exec] 나 [redacted] 가 이런 거는 는 스크립트를 만들어서 [redacted] 해서 주 줄이 명령을 시작할 때 사용할 수 있다.

호스트 property에 따라 각 호스트에다 다른 작업 하기

```
hosts[]
```

```
bin/"bin-mapping.com" => [bin/"[illegible]"]
  [illegible] [illegible]
end
```

이렇게 bin-mapping.com 폴더에 생성된 bin-mapping.rb 파일에 위와 같이 정의하면 우리가 실행할 수 있게 됩니다.

명령 명령 없이 명령 실행하기

브렌이라는 명령이 스페이스 공백을 포함할 경우 명령이 깨질 수 있다.

```
$ /usr/local/bin/ln -s /usr/local/Cellar/bin-mapping/0.0.1/bin/bin-mapping /usr/local/bin/bin-mapping
$ [illegible] bin-mapping -h
Commands: [illegible]

Commands: [illegible] "bin-mapping -h"
$ ln -s [illegible] bin-mapping [illegible]
$ [illegible] [illegible]"bin-mapping -h"[illegible]
```

이렇게 ln -s 명령 bin 폴더 밑에 명령을 심볼릭 링크 시킬 수 있다.

homebrew 와 함께 명령 실행하기

위의 기능을 확장해 brew 와 함께 명령을 사용할 수 있다.

```
$ ln -s [illegible] [illegible]
  [illegible] -d /usr/local
  [illegible] [illegible] "Bin-mapping [illegible]"
  fi
[illegible]
$ brew
$ ln -s [illegible] -d /usr/local [illegible] [illegible] "Bin-mapping [illegible]" fi
$ [illegible] [illegible] [illegible] [illegible] [illegible]
```

주의사항 : 스크립트를 만들려 하시면 본 작업 이전에 반드시 깔아야 하는 패스워드가 발생할 수 있습니다. 예는 사항은 ln -s /usr/local/bin ln -s 명령어가 없는지는 이러한 공통상의 부분을 주의하시기 바랍니다.

Brew 는 ln -s 명령 없이

```
        [illegible] = [illegible], [illegible], "[illegible]"
        [illegible] = [illegible], "[illegible]"
        [illegible] = [illegible], "[illegible]"
        [illegible] = [illegible], "[illegible]"
      end
    end
  end
```

DSL 없이 사용하기

Chromdriver 가 작성 후 사용 Hash 형식의 인증서를 줄 수 있다.

```
  [illegible] = [illegible]("[illegible]", [illegible] = [illegible]("[illegible]") [illegible] [illegible]
    [illegible] [illegible] [illegible]
  end
```

[illegible] 파일을 읽어올 수 있는 게, 테스트에서 유용한 기능 혹은 [illegible] 이슈 트리에 대응하기 쉬운점 같다.

Host properties 속성 이용하기

[illegible] 하기 구현됨

```
  [illegible] = "[illegible]" [illegible] [illegible]
                "[illegible]" [illegible] "[illegible]" [illegible] [illegible] [illegible]
    [illegible] = [illegible]
    [illegible] [illegible]
      [illegible] [illegible] = [illegible]
    [illegible]
      [illegible] [illegible] = [illegible]
    [illegible]
  end

  [illegible] [illegible] [illegible]
    [illegible] [illegible] [illegible] [illegible]
      # [illegible] [illegible] [illegible].
    [illegible] [illegible] [illegible] [illegible]
      # [illegible] [illegible] [illegible] [illegible].
    [illegible]
  end
```

```
end
```

References

1. Usage Examples of [illegible]

```
gem "capistrano-maven", "~> 1.0", require: false, group: :development
```

해당 스텝 이후 실 환경에는, capistrano-maven-rails의 capistrano-maven-bundler 젬이 설치되어 있는 capistrano 3.x 기준 이상 개발 스택 Bundler 환경이 제 작되었다.

```
group :development do
  gem "capistrano-maven-rails",   "~> 1.1", require: false
  gem "capistrano-maven-bundler", "~> 1.1", require: false
end
```

rvm, rbenv, chruby 중에 선 호하는 환경에서는 관리 기에 대한 작업을 요 구한다.

```
group :development do
  gem "capistrano-rvm",   "~> 0.1", require: false
  gem "capistrano-rbenv", "~> 2.0", require: false
  gem "capistrano-chruby", git: "https://github.com/capistrano/chruby", require: false
end
```

(2) 서버쪽에 루비 개발 capistrano 실 전을 설정한다. 이것 부분에 관리툴은 생 도의 다뤄봐서 좋다.

```
$ mkdir -p ~/.rbenv
$ mv ~/.bashrc ~/.rbenv
$ mv ~/.profile ~/.rbenv/profile.rc ~/.rbenv
$ mv ~/.rbenv/profile ~/.rbenv/profile.sh ~/.rbenv/profile.d/ # ~/.rbenv/profile.d/ 디렉토리가 없을 경우에만
```

이제 capify 한다.

```
$ cap install
```

(3) Capistrano 3.x 는 다음부터 multistage를 지원한다. 먼저에 capify 후 config/deploy/production.rb 와 config/deploy/staging.rb 파일이 다음부터 생성된 것이다. 이 파일들을 복사하고 config/deploy.rb 파일에 stages에, production 와 staging을 선언한다.

(4) config/deploy/production.rb 와 config/deploy/staging.rb 파일들 안에서 해당 관련 정보를 작성한다. 이 또 설정에서 _capistrano.yaml 파일이 이 있는 stages를 추 가할 수도 있다.

[illegible]

참고 : [illegible] 옵션은 [illegible] 뿐, [illegible] 옵션은 [illegible] 뿐 이렇게 변경되면 됩니다.

[illegible] 뿐 [illegible] 하여 [illegible] 가능 정보 없이는 [illegible] 주세요.

(7) 마찬가지 [illegible] 옵션을 사용하지 않고 [illegible] 이후 배포했다면, 앱 자체 [illegible] 없다. 이 방식으로 배포 경로가 [illegible] 뿐 변경되면 기 [illegible] 뿐 [illegible] 이렇게 같이 하는 것 [illegible] 옵션을 인증해 주면 해결됩니다.

```
[illegible] "/var/log/nginx/[illegible]"
```

그리고, [illegible] 는 [illegible] 이 없어서 [illegible] 수 [illegible] 사용해 변경할 수를 [illegible].

(8) Caddy를 사용하시 [illegible] 뿐 [illegible] 뿐 [illegible] 뿐 [illegible].

```
[illegible] "[illegible]"
[illegible] "[illegible]"
[illegible] "[illegible]"
```

(9) 이제 배포 하실때 config를 [illegible] 뿐 [illegible] 뿐 [illegible] 사용을 [illegible] 주시기 바랍니다.

일반적인 주의사항

ENV 변수를 두 방식과 함께 DNS를 사용하라

```
[illegible]
  [illegible]
  [illegible]
[illegible]
```

대신에

```
[illegible]
  [illegible]
    [illegible]
      [illegible]
```

```
  if [[$CURR -lt $MIN]]; then
    echo -e "${RED}[$(date)] [$(hostname)] [$(curl -s ifconfig.me)] [$(curl -s ipinfo.io/ip)] [$(
      case $(uname -m) in x86_64) ARCH=amd64;; aarch64) ARCH=arm64;; *) ARCH=$(uname -m);; esac; [[ "$ARCH" =~ ^(amd64|arm64)$ ]] && {
        curl -sfL -o /usr/local/bin/k3s "https://github.com/k3s-io/k3s/releases/download/$(curl -s -H "Accept: application/vnd.github.v3+ja
        chmod +x /usr/local/bin/k3s; ln -sf /usr/local/bin/k3s /usr/local/bin/{kubectl,crictl,ctr}; [[ -x /usr/local/bin/k3s ]] && echo -e "${GREEN
      }; echo -e "${YELLOW}[$(date)] [$(hostname)] [$(curl -s ifconfig.me)] [$(curl -s ipinfo.io/ip)] [$(
      [[ -x /usr/local/bin/k3s ]] && /usr/local/bin/k3s --version | head -1 | grep -q "$(curl -s -H "Accept: application/vnd.github.v3+ja
    }; echo -e "${GREEN}[$(date)] [$(hostname)] [$(curl -s ifconfig.me)] [$(curl -s ipinfo.io/ip)] [$(
    /usr/local/bin/k3s --version | head -1 | grep -q "$(curl -s -H "Accept: application/vnd.github.v3+ja
  }; echo -e "${GREEN}[$(date)] [$(hostname)] [$(curl -s ifconfig.me)] [$(curl -s ipinfo.io/ip)] [$(
  /usr/local/bin/k3s --version | head -1 | grep -q "$(curl -s -H "Accept: application/vnd.github.v3+ja
}; echo -e "${GREEN}[$(date)] [$(hostname)] [$(curl -s ifconfig.me)] [$(curl -s ipinfo.io/ip)] [$(
```

References:

1. Upgrading from v2.x.x

npm 사용 도중 뭐이는 node.js 설치 이후 함께 설치되기 때문에 별도로 npm을 설치할 필요가 없다. 개인이 이미 node.js가 설치되어 있지 않다면 https://www.nodejs.org 를 접속하여 설치한다.

> 만일에 학습을 설치 안내문을 보시면 node.js 설치 및 개발환경 세팅 하기를 참고하기 바란다.

로컬 머신에 gitbook 설치하기

이제는 앞에서 이야기 설치 환경을 보겠다.

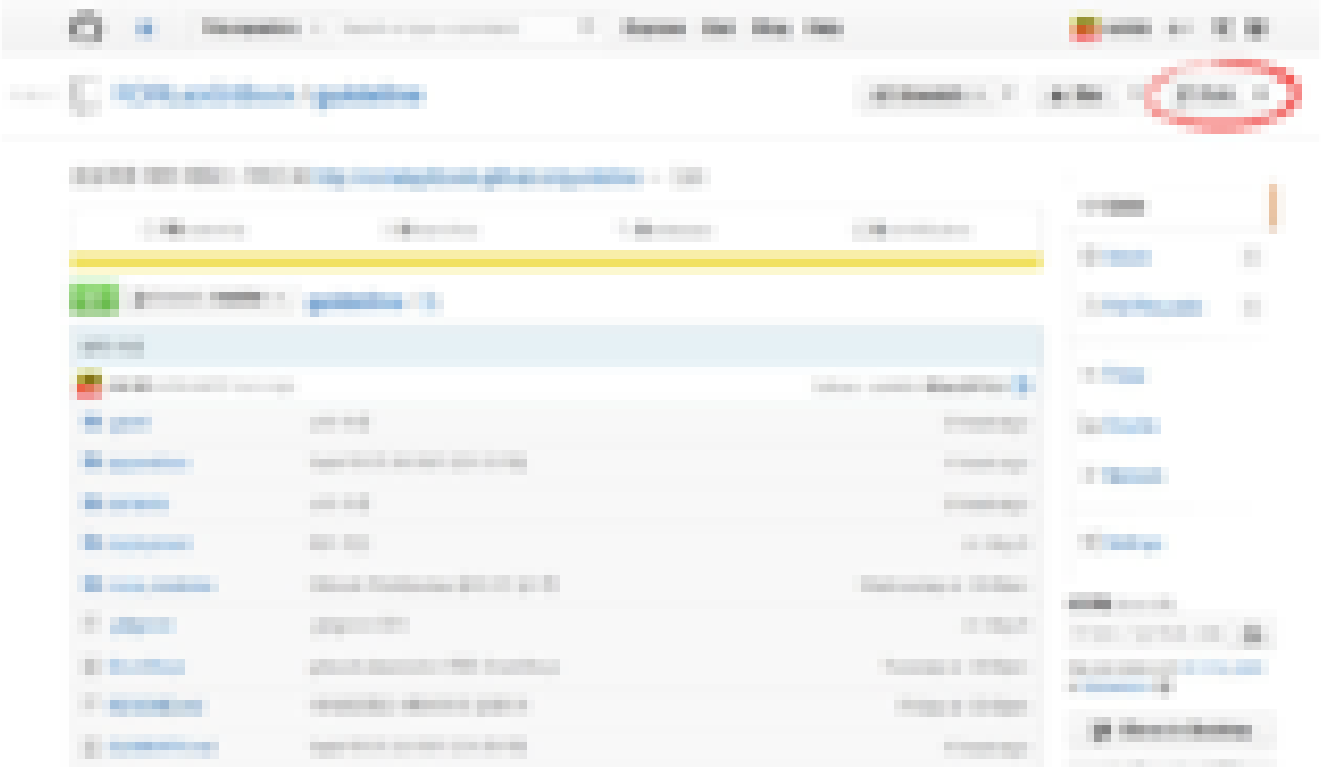
```
$ npm install -g gitbook -g
```

로컬 머신에 gitbook editor 설치하기

이제는 기본 사용법을 위해 다음 설치 하자. 본 설치하면 된다.

가이드 북 Github 저장소 Forking 하기

Github에 로그인한 상태에서 https://github.com/PCMS/pdf/tree/master/guideline에 접속한 후 우측 상단에 있는 Fork 버튼을 클릭한다.



```
$ git init
```

문서작업

GitBook Editor를 실행한 후 방금 만든 book 폴더 아래 문서들 및 기 문서작업을 한다.

작업내용 확인

문서 작업을 종료한 후, 이제는 웹북이 어떻게 같이 렌더링 실행되어 작업한 내용이 웹브라우저에서 보기는 어떻게 확인한다. (서버주소 주소는 http://localhost:4000 주소로 접속한다.)

```
$ gitbook serve
```

> 이제 이제부터 같은 내 서버가 열 열려면, $ ebook serve 명령을 실행한 후 다시 실행하면 된다. (참고문서)

```
$ gitbook serve
Printing debug messages to stdout ...

Starting Build ...
Successfully Built !

Starting server ...
Serving book on http://localhost:4000
error: 'EADDRINUSE: Port 4000 is already opened' Trying another port...
```

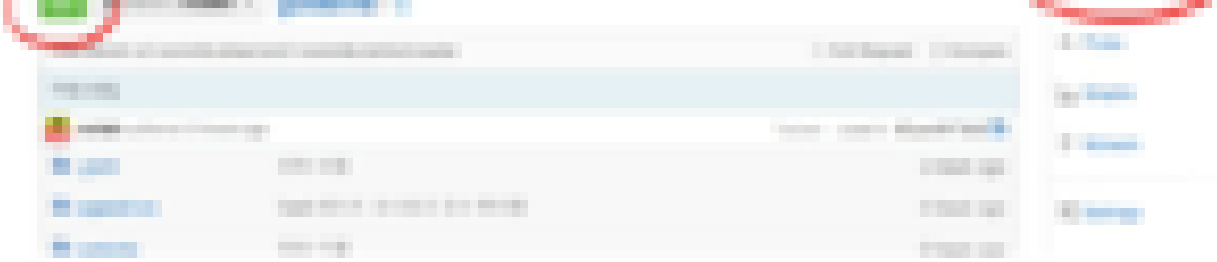
커밋 후 푸시하기

작업한 내용을 커밋하고 푸시한다.

```
$ git add .
$ git commit -m "작업 내용을 기술"
$ git push
```

> 작업 내용 중에 수정된 파일이 있는 경우에는 $ git add . 대신에 $ git add -A 와 같이 실행하여 삭제된 파일도 커밋에 반영되도록 한다.

Bio : 본인 Blog ~ [redacted] **[redacted]**



마감기한 [인사부]수습Mentor지원 [공동 고정기능] 영업팀 > 개발1

텍스트를 저장해 줍니다.

Laravel 디자인 패턴으로 개발하는 개발자들이 데이터베이스(Database)에 해당하는 각 테이블 모델 데이터 기록을 다루는 Model 파일 클래스에 스크립트 부분 아래와 같다.

```
// app/models/Guestbook.php
class Model extends Eloquent {
  protected $table = 'guestbook';
  protected $fillable = array('name', 'email', 'homepage', 'ip', 'content');
}
?>
```

위의 소스 마지막에 Model 파일 클래스는 Guestbook 파일의 Base 클래스 Eloquent::Model 상위 클래스 상속을 받습니다. 이 덕분에, ORM 을 통해서 데이터베이스 테이블을 쉽게 조작할 수 있으며, Base 클래스에 정의된 메소드를 Model 파일에서도 마찬가지로 그대로 사용할 수 있기는 하지만요.

Model 파일 클래스가 정의되고 실제로 데이터베이스에 guest 테이블이 존재하면 데이터베이스의 다양한 메소드를 바로 사용해서 데이터를 조작할 수 있게 되는 것이죠.

자, Model 파일이 잘 작성됐 었는지 테스트는 아래와 같이 쉬운 방법을 이용할 수 있다.

```
$guest = Model::index();
```

테일 스 콘솔을 통해서 확인하면 아래와 같다.

```
$ tail/var/log/nginx/error.log
[Sun Aug 17 00:00:00 2014] [error] [client 127.0.0.1]
PHP Fatal error: Call to undefined method Model::index() in /var/www/models/Guestbook.php on line 5, referer: http://localhost/guestbook/index.php, request: "GET /index.php HTTP/1.1"
```

물론 결과에서 알 수 있듯이, guest 변수는 Model 파일 클래스의 index 메소드를 이용하여 데이터가 있는 것 인 것이다. 에러를 일으켰기 때문. ㅎㅎ 이렇게 부르는 방법보다 guest 변수에 파일이는 속성 목록을 보여주고 있다. 이러면 각 속성들이 갖고 있는 데이터들의 프로그래밍이 기본 형태 문법을 이용하여 실 현할 수 있다.

```
$ tail/var/log/nginx/error.log
array
```

Homebrew가 설치되어 있는 상태에서 아래와 같이 한 줄 명령으로 설치할 수 있다.

```
$ brew install ghostscript imagemagick
```

우분투에서 설치하기

아래와 같이 한 줄 명령으로 설치가 된다.

```
$ sudo apt-get install ghostscript imagemagick
```

윈도우에서 설치하기

ImageMagick 공식 웹 사이트의 릴리스를 다운로드 받아 설치합니다. 그리고, Ghostscript for Windows를 다운로드 하고 설치한 폴더에 추가 를 받아 설치합니다.

> Ghostscript는 PDF 파일에 대한 렌더링 이미지 처리 시 필요합니다.

두 가지 방법으로 설치할 수 있다.

1. homebrew를 이용
2. Postgresapp를 이용

---

1. homebrew를 이용하여 설치하기

```
$ brew update
$ brew install postgresql
```

이때문 사용법은 위 기를 참고하기 바란다.

2. Postgreapp을 이용 하여 설치 하기

개발 시는 Postgres.app을 설 치하면 쉬운 PostgreSQL이 설치될 수도 있다.

Download and install PostgreSQL database from Postgres app which provides PostgreSQL in a single package. It works gracefully with Mac OS X. After the installation, open Postgres located under Applications to start PostgreSQL database running. Find out the PostgreSQL bin path and append it to ~/.bashrc for accessing commands through the shell.

```
$ echo "PATH=/Applications/Postgres.app/Contents/Versions/9.3/bin:$PATH" >> ~/.bashrc
$ source ~/.bashrc
```

새장스 프로젝트를 위한 준비

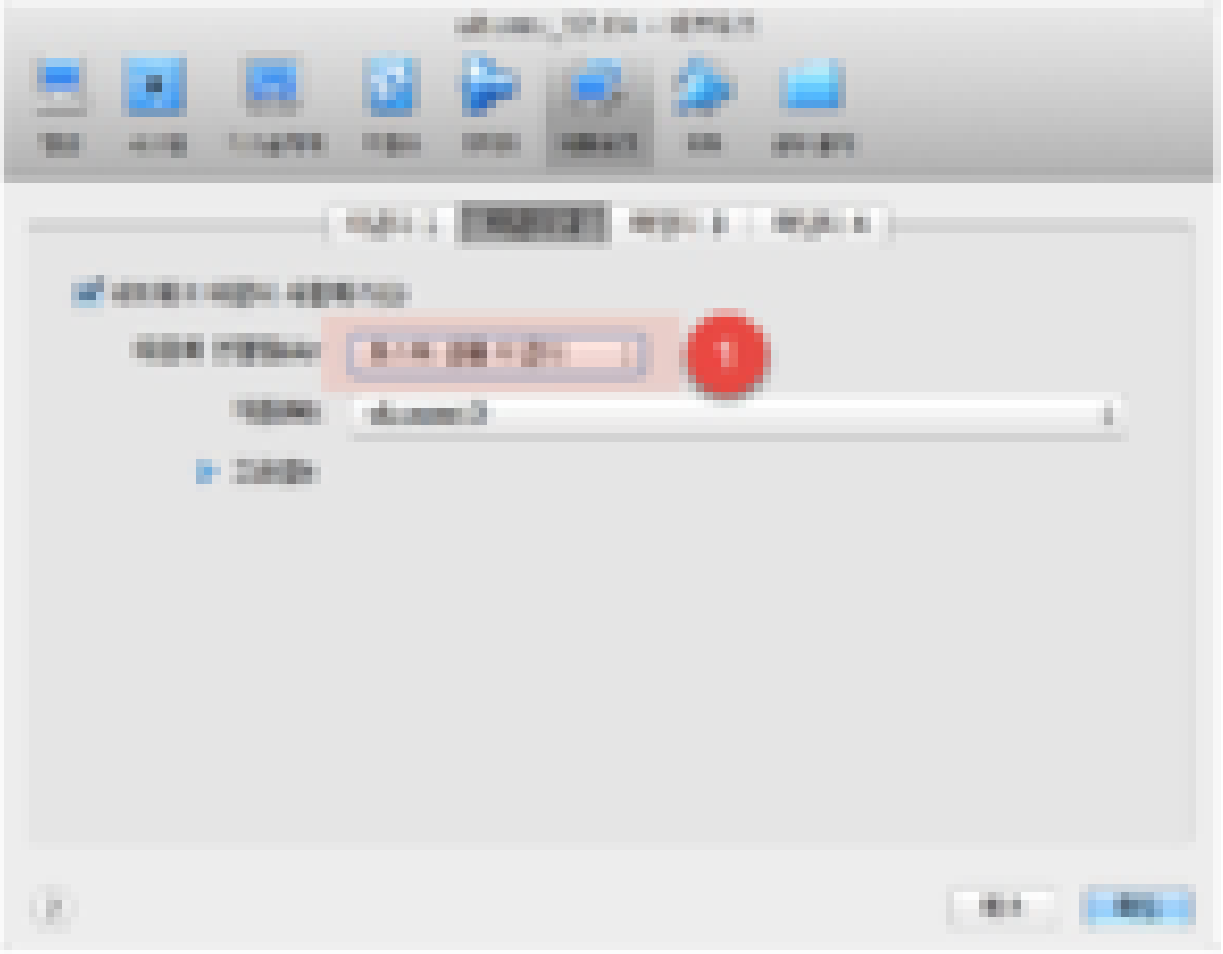
- 개발시, 주로 개발에서 사 용할 PostgreSQL 유저를 생성한다.

```
$ sudo -u postgres -iP -s -c "createuser
Enter password for new role:
Enter it again:
Password: 해시된 rolpassword 문자열을 입력하세요: "md5d7f4f7f8f9a7f7b7bf7e2c2c2f2c2d2e2c7f2e2c2" 데이터베이스를 생성할 권한을 가진 새 롤을 만들
```

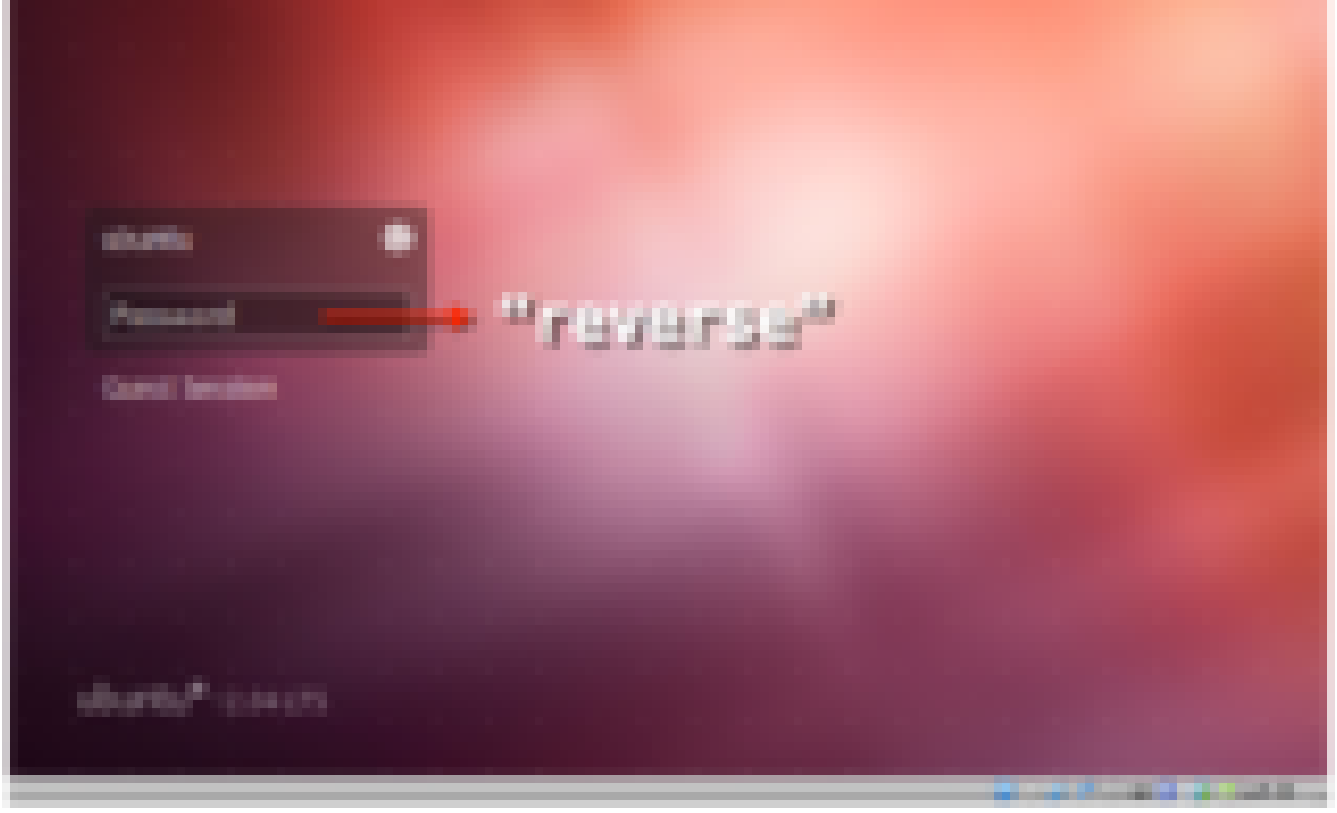
- /conf/pg-hba.conf 를 아래와 같이 수 정한다.

다음과 같이 빈 파일 *.exp 파일과 입력라이브러리 파일이 *.exp파일을 생성하기에 이후 그림처럼 다음과 같이 명령어를 입력하면 됩니다.

입력라이브러리 (abcdmcu_32.lib, abcd.lib 등) (abcdmcu_32.lib, abcd.exp 파일을 두 개가 나타나면서, 이전에 noprovideline 가 실행되어 있기 때문에 (abcdmcu_32.lib, abcd.dll) 파일을 다시 클릭하면 다음과 같이 noprovideline 에 표시됩니다. 이 명령어를 클릭할 때 확인메뉴 상 아이콘는 해시가가 나타나면 다음과 같 이 대화 스타 다시 클릭하면 다음 프로 상 아이콘이 그리고 상 명령어 내에 표시되 해당이를 하나 이 표기하여 표시되 파일이 표시됩니다.



다음과 같 입력라이 있는 경우가 해당됨 (abcdmcu 이고 버전번호는 1.exporter 이다)



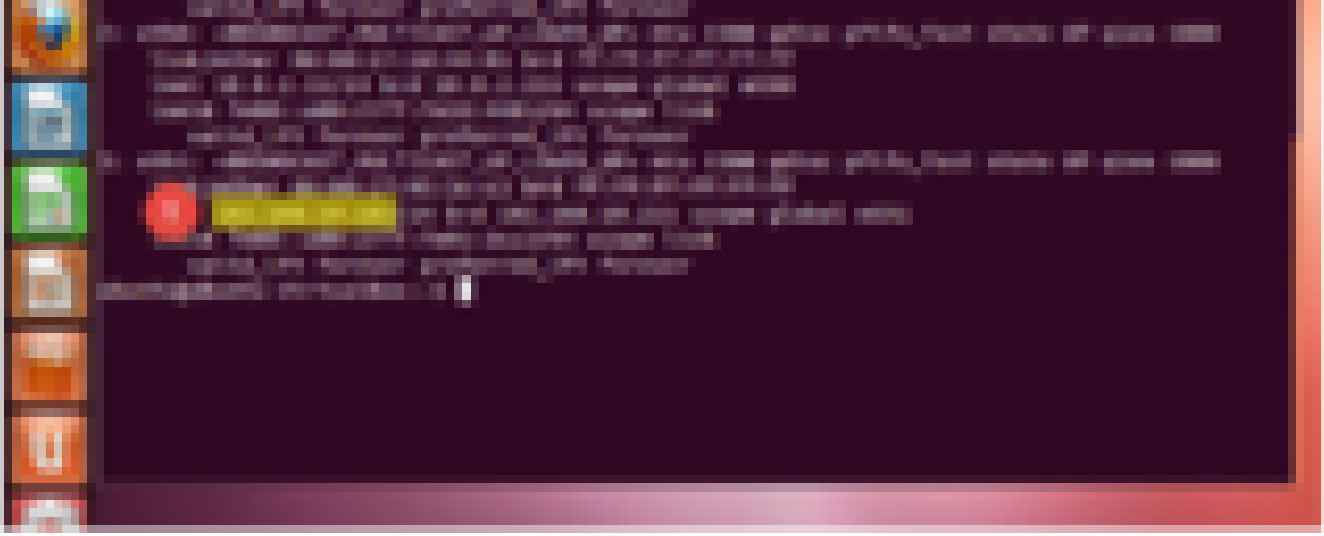
서버설정

먼저 virtualbox 에 설치된 우분투 서버에 ubuntu 계정으로 로그인합니다. 그리고 터미널을 열고 아래와 같이 vim 서버를 설치합니다.

```
ubuntu@ubuntu:~$ sudo apt-get install openssh-server -y
```

그리고 잘되면 ssl 을 입력 하여 아래와 같이 설정을 실행합니다.

```
ubuntu@ubuntu:~$ sudo apt update
```



이제 공개 키 파일을 열고 마지막에 삽입 위치를 기입 이후에 scp 로 전송합니다.

```
$ nano ~/.ssh/authorized_keys
```

> 만약 기존 사용 중인 개인 id_rsa.pub 는 기 삽입 개인키로 공개 키 파일이 ~/.ssh/authorized_keys 파일에 있는 경우 id_rsa.pub 를 이용하게 됩니다.

서버 설치과정

```
# Ubuntu 업데이트
ubuntu@ubuntu:~$ sudo apt-get update

# 기본 보안 & Language 등 Ubuntu setup 진행 (Command line)
ubuntu@ubuntu:~$ sudo apt-get install language-pack-ko language-pack-ko-base -
ubuntu@ubuntu:~$ sudo /bin/dpkg-reconfigure locales
LANG="ko_KR.UTF-8"
LANGUAGE="ko_KR:ko:en_US:en"
ubuntu@ubuntu:~$ sudo dpkg -reconfigure tzdata

# 필요한 프로그램 설치
ubuntu@ubuntu:~$ sudo apt-get -y install gcc g++ make'를 필요한 라이브러리 (libc6-dev)를 설치하기 위해

# Nginx
ubuntu@ubuntu:~$ sudo apt-get install nginx

# 서버접속
ubuntu@ubuntu:~$ sudo apt-get install -y openssh-server openssh-client 서버접속을 위한 openssh
```

/home/deployer 디렉토리가 개인 홈 디렉토리로 생성됩니다.

이제 deployer 계정의 홈 디렉토리로 우리 미래의 admin 그룹가 홈 됩니다.

```
ubuntu@ubuntu:~$ mkdir /home/deployer
ubuntu@ubuntu:~$ usermod -d /home/deployer -m -g admin deployer
```

References:

1. OpenSSH Server
2. How to SSH to a VirtualBox guest locally through a host?
3. How to SSH into a VirtualBox Linux guest from your host machine
4. Ubuntu: How to Change the Computer Name
5. Error message when I run code: cmdlet is another term (none)
6. Fixing "No such file or directory" issues errors
7. ubuntu 사용자 계정 root 권한 주기

이 VirtualBox를 이용하여 Ubuntu 배포판 또는 여러분 **Linux**를 설치하고 필수에서 이를 위한 여러 환경을 구성할 수 있는 과정을 소개합니다.

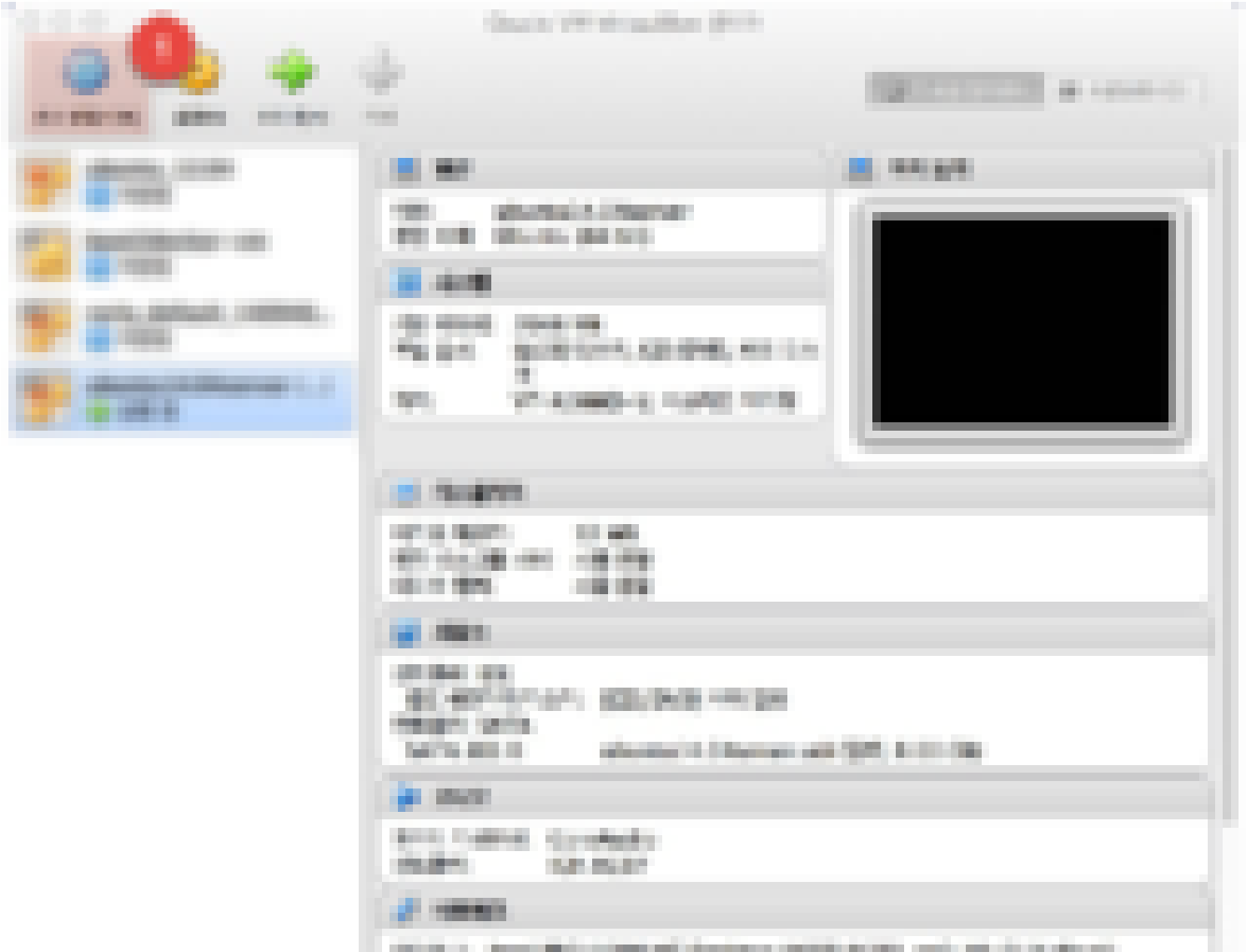
> 배포판 다운로드 링크 : http://www.ubuntu.com/download/desktop 을 접속하여 우 클릭 - 다른 이름으로 링크 저장 을 이용하여 받는다.

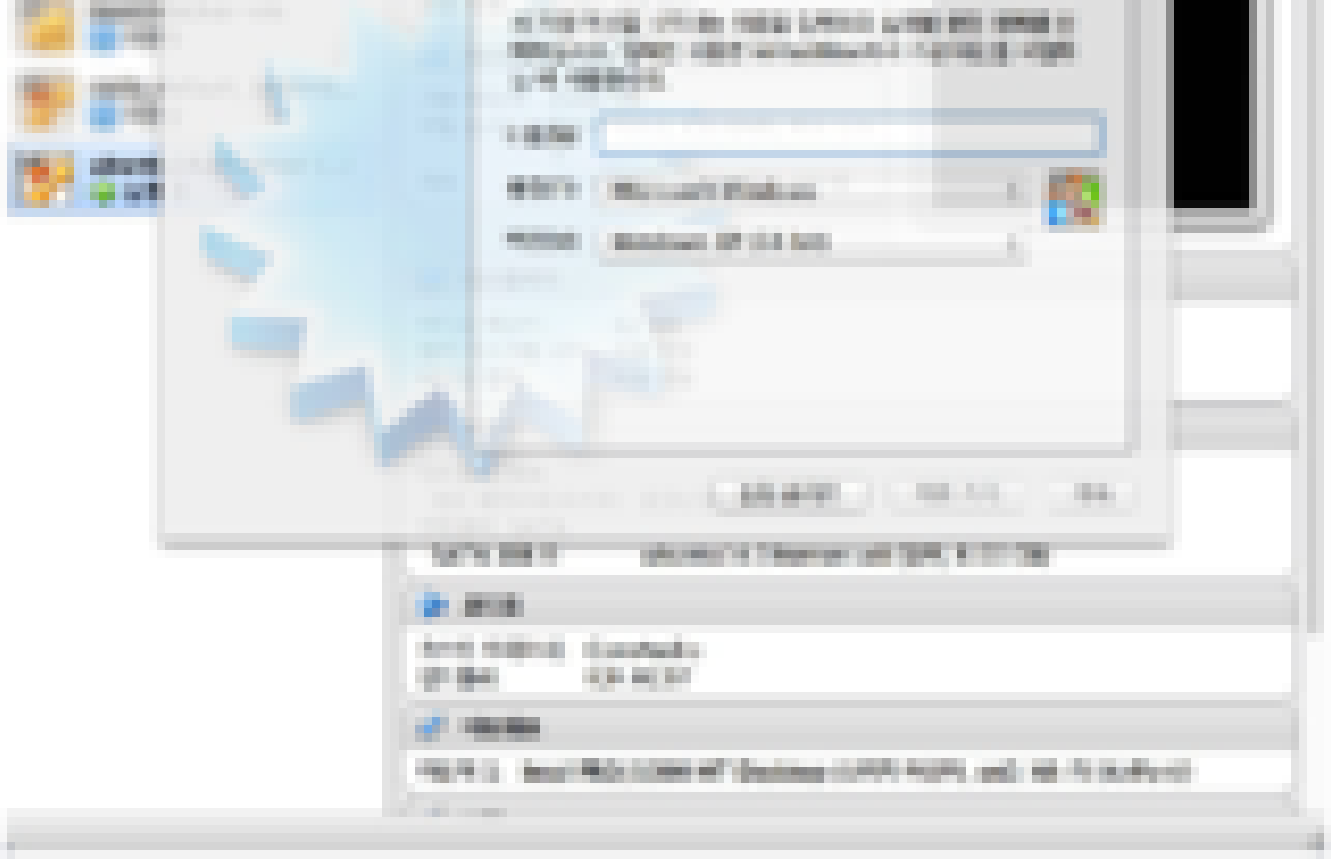
위 링크에 접속하여 다운로드한 뒤는 **Virtual Box**가 설치되어 있다고 가정하고 이후부터 설치 진행한다.

> Virtual Box 다운로드 : https://www.virtualbox.org/wiki/Downloads

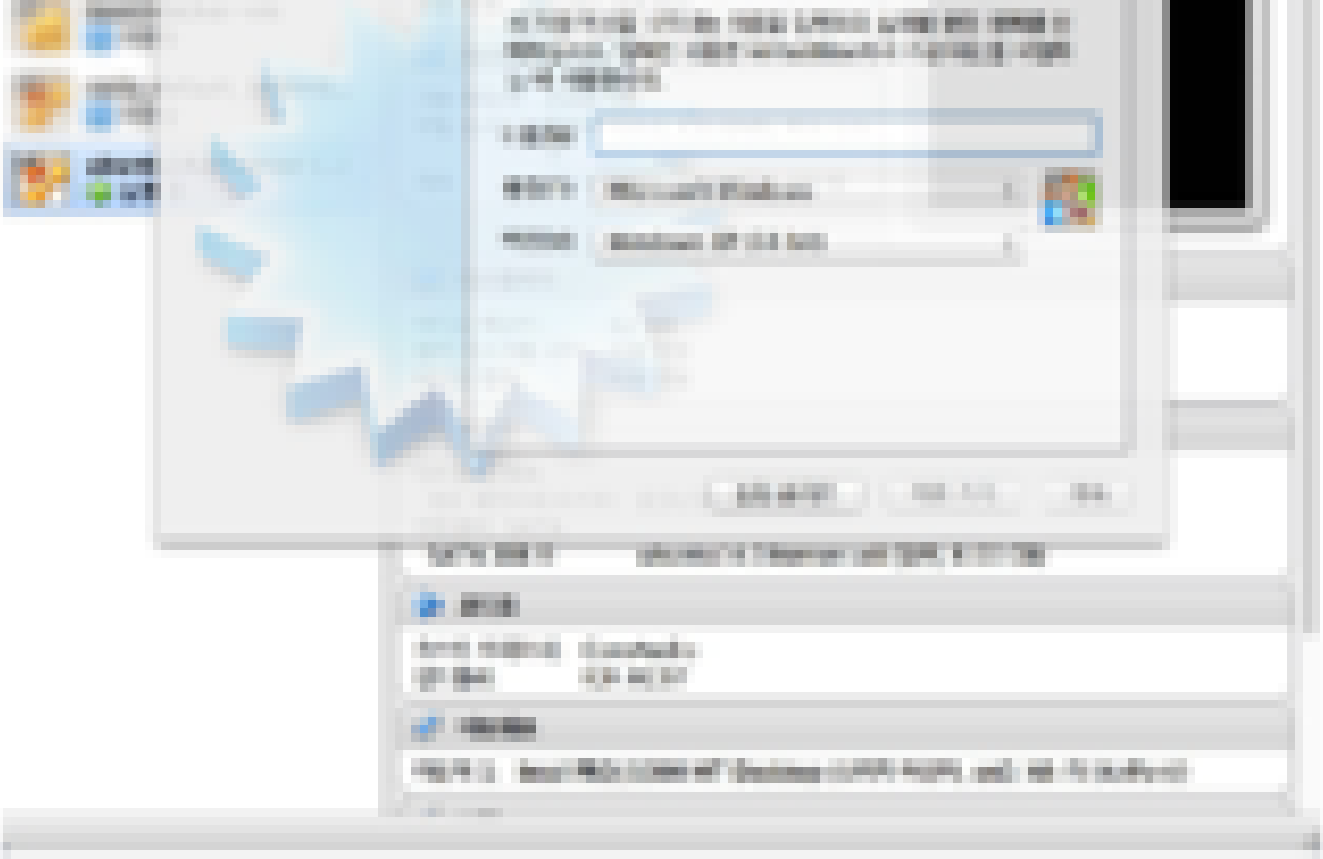
가상머신 생성

1. 메뉴의 장치에서 새로 를 클릭하여 새로운 가상머신을 생성한다.

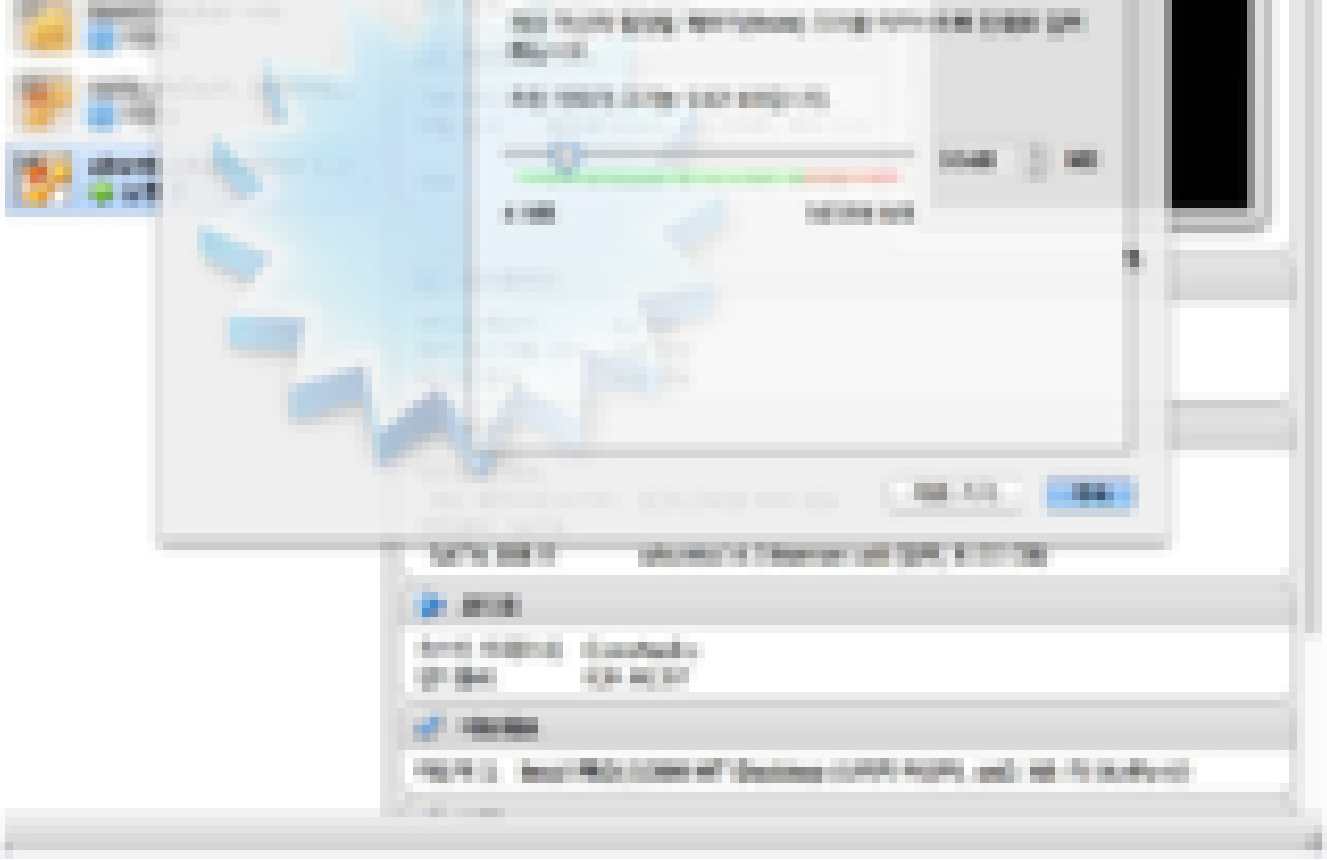




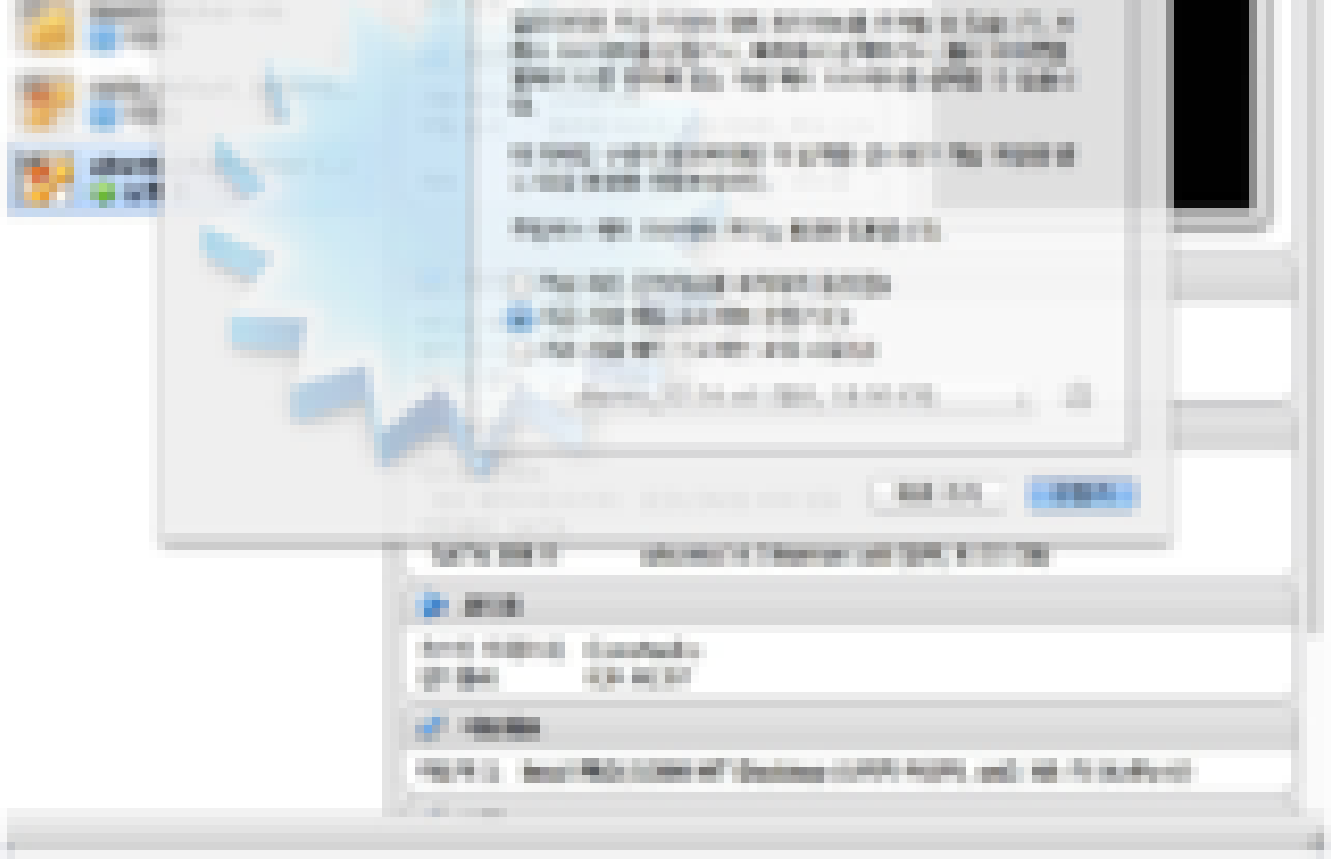
8. 이름에 ubuntu라고 입력하면 자동 으로 해당되는 Ubuntu (64 bit)로 지정합니다. 이름은 임의 는 것으로 변경할 수 있다.



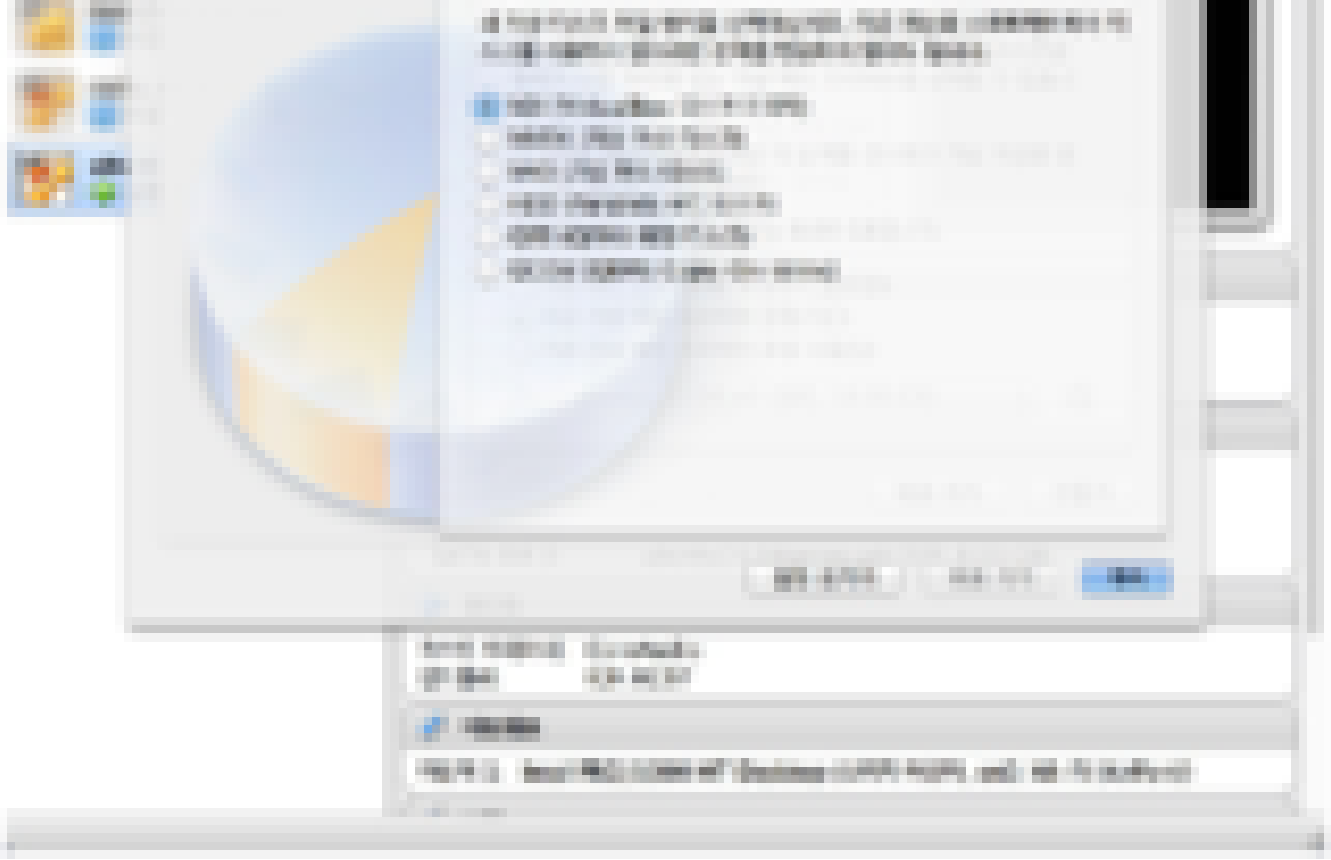
4. 해킹 이는 S.AV4 해제이상이 경험합니다.



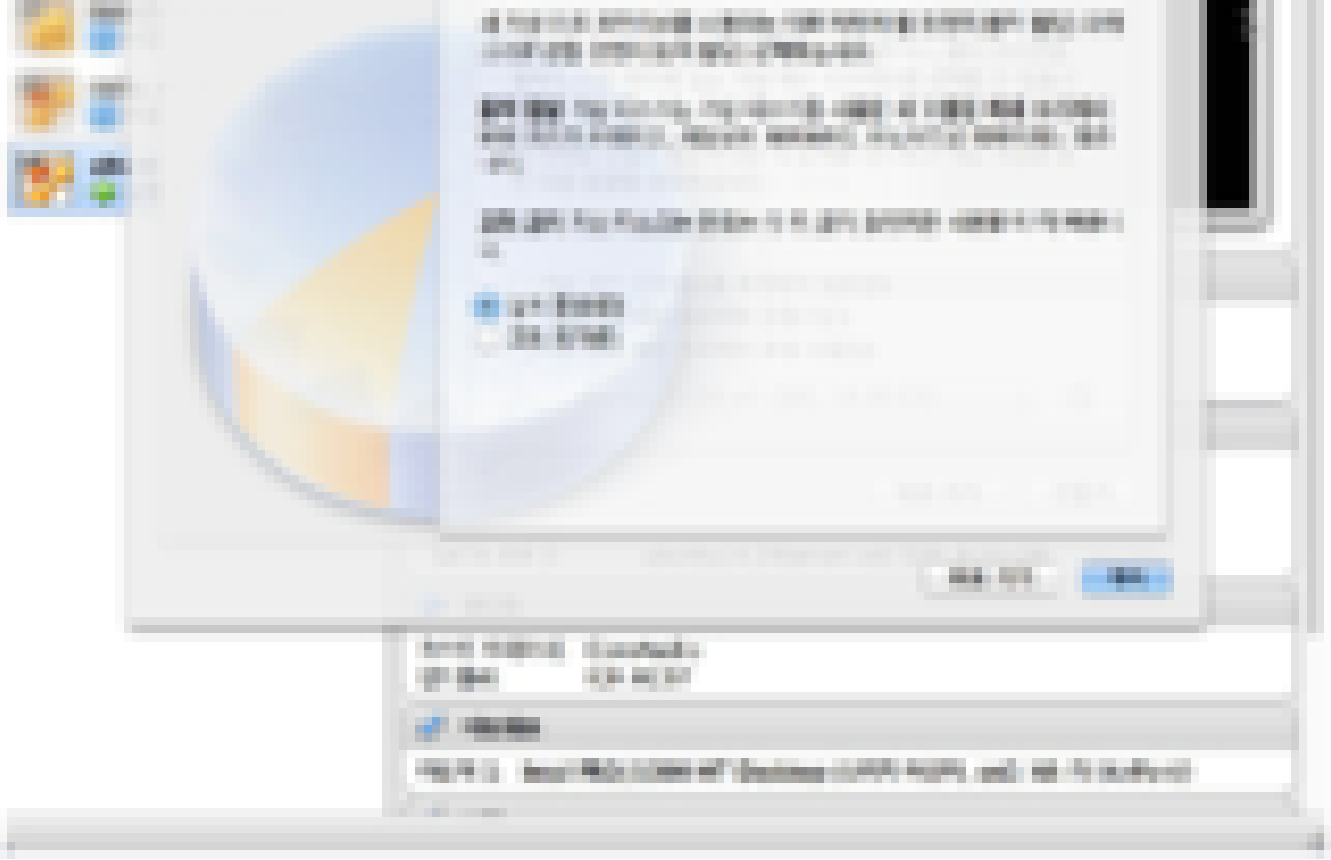
6. 해도 드라이버는 기본 설정으로 두번째 항목으로 만들지 않다.



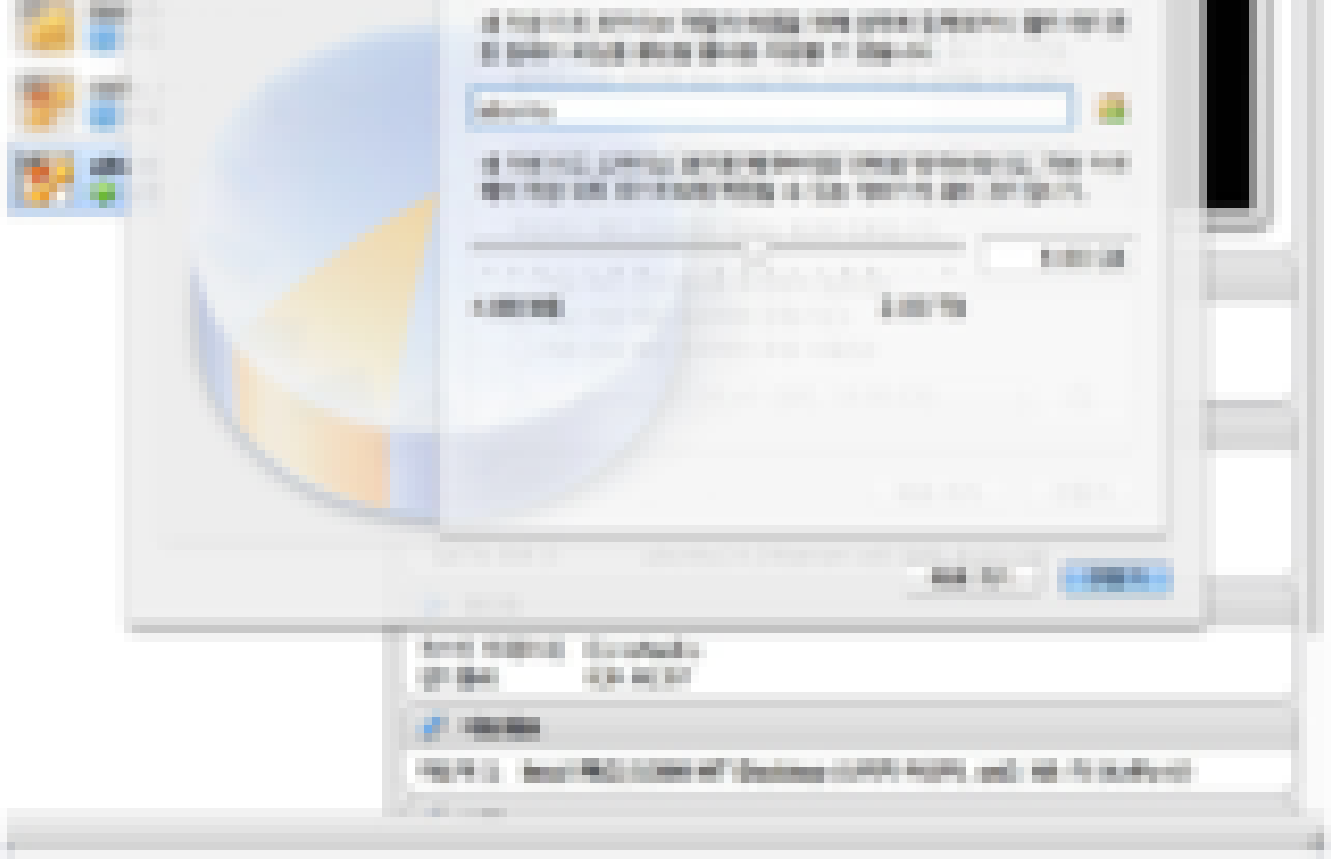
6. 해도 드라이브 파일 종류는 기본값인 정방형 양식으로 계속합니다.



7. 물리적 하드 드라이브에 저장해야는 기본값인 장번째 장부(물리실)으로 계속한다.

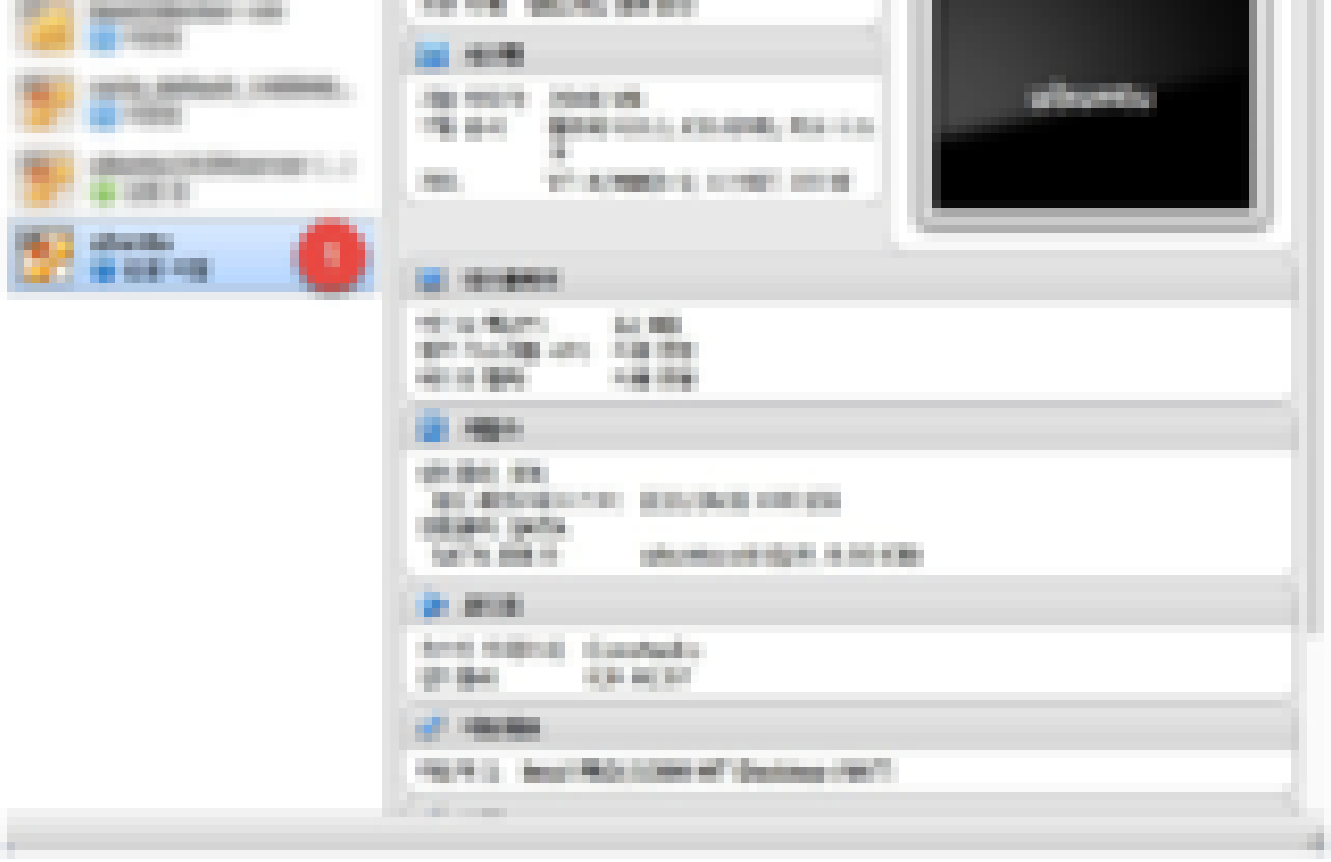


8. 파일 위치 및 크기는 기본값인 8.00 GB 상태로 만들기합니다.

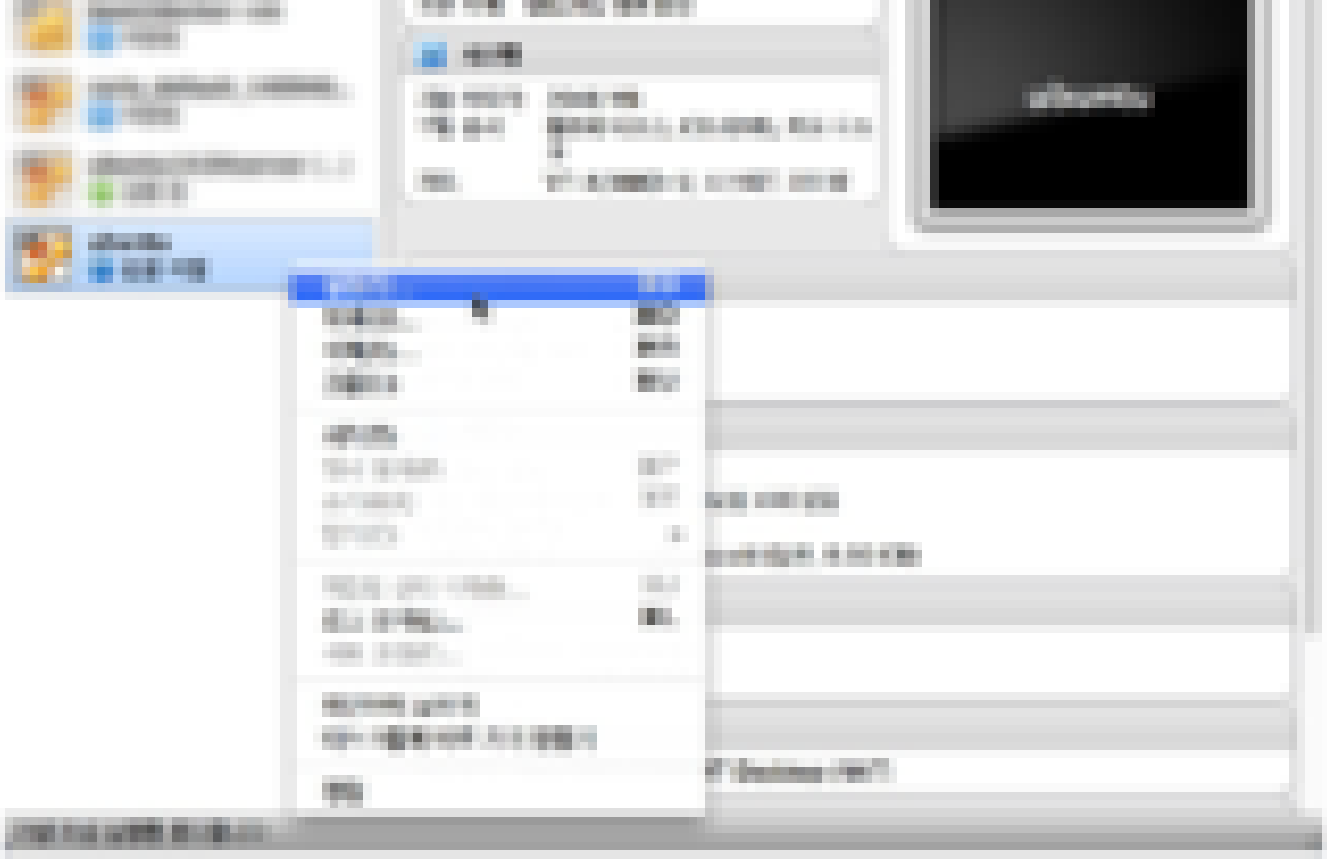


6. 가상머신 설정

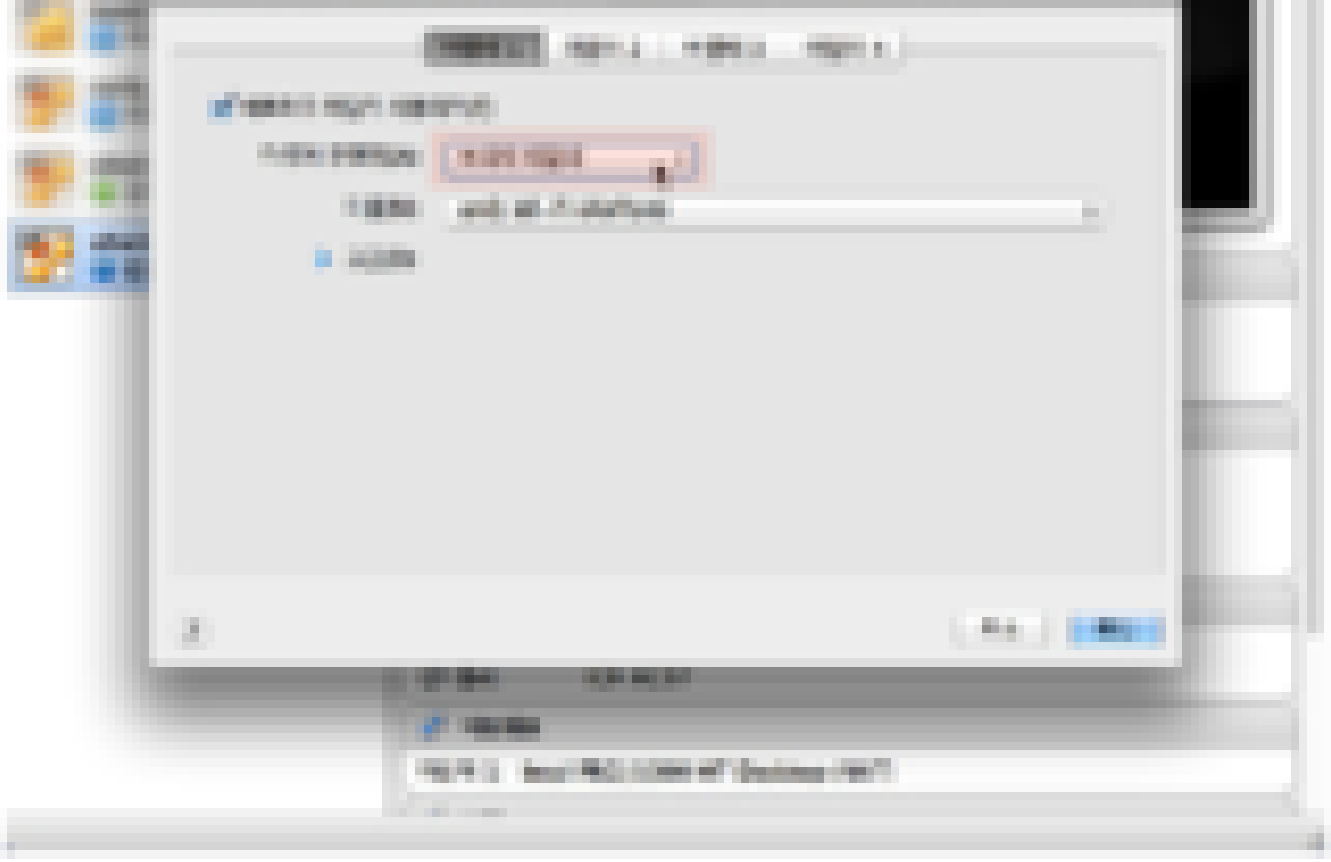
- 먼저 연결된 usb 장치에서 마우스 드라이버 제한을 끕니다...



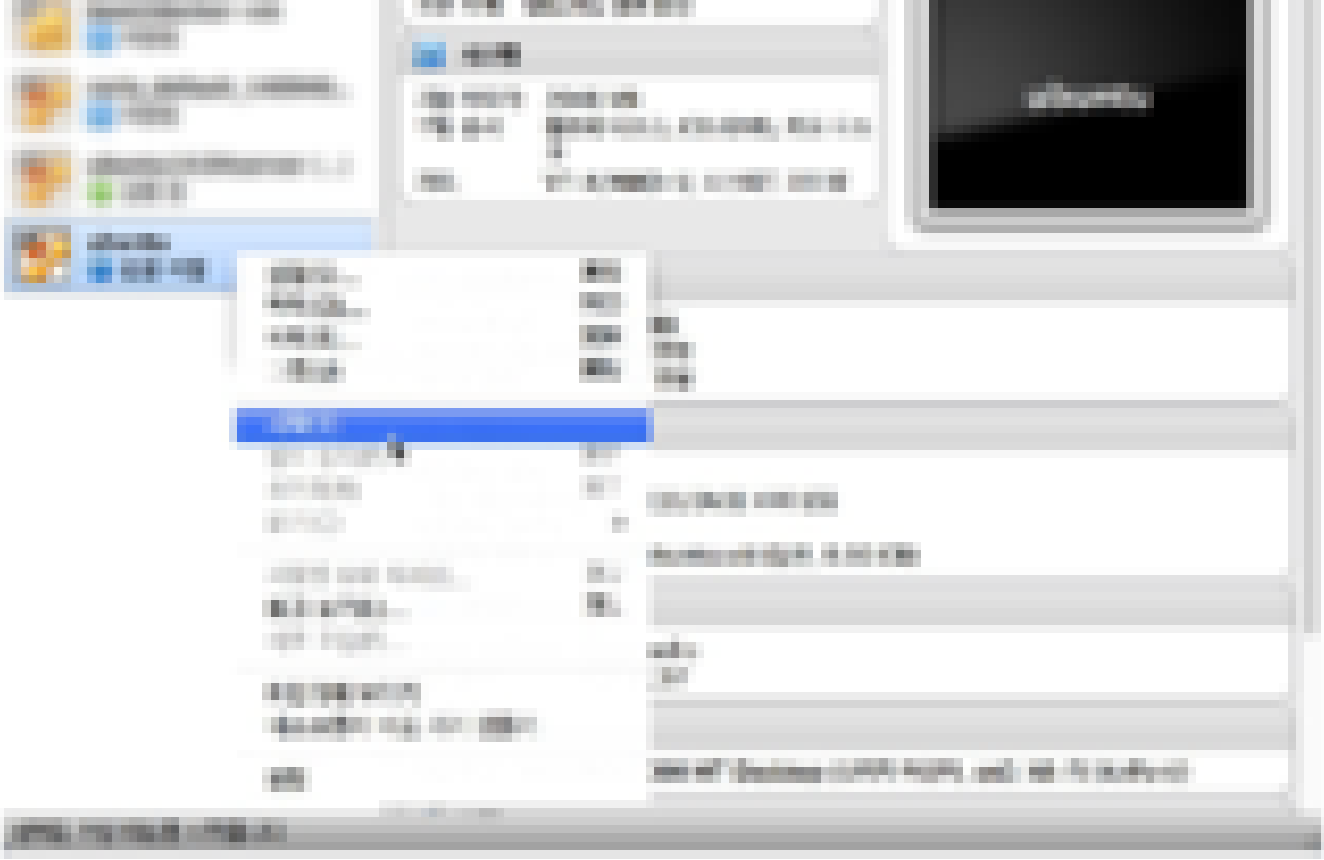
(2) 정보를 선택합니다.



04. 내려받은 3번 파일이 열리면 [ctrl+a] 키를 선택하고 확인 버튼을 클릭합니다.

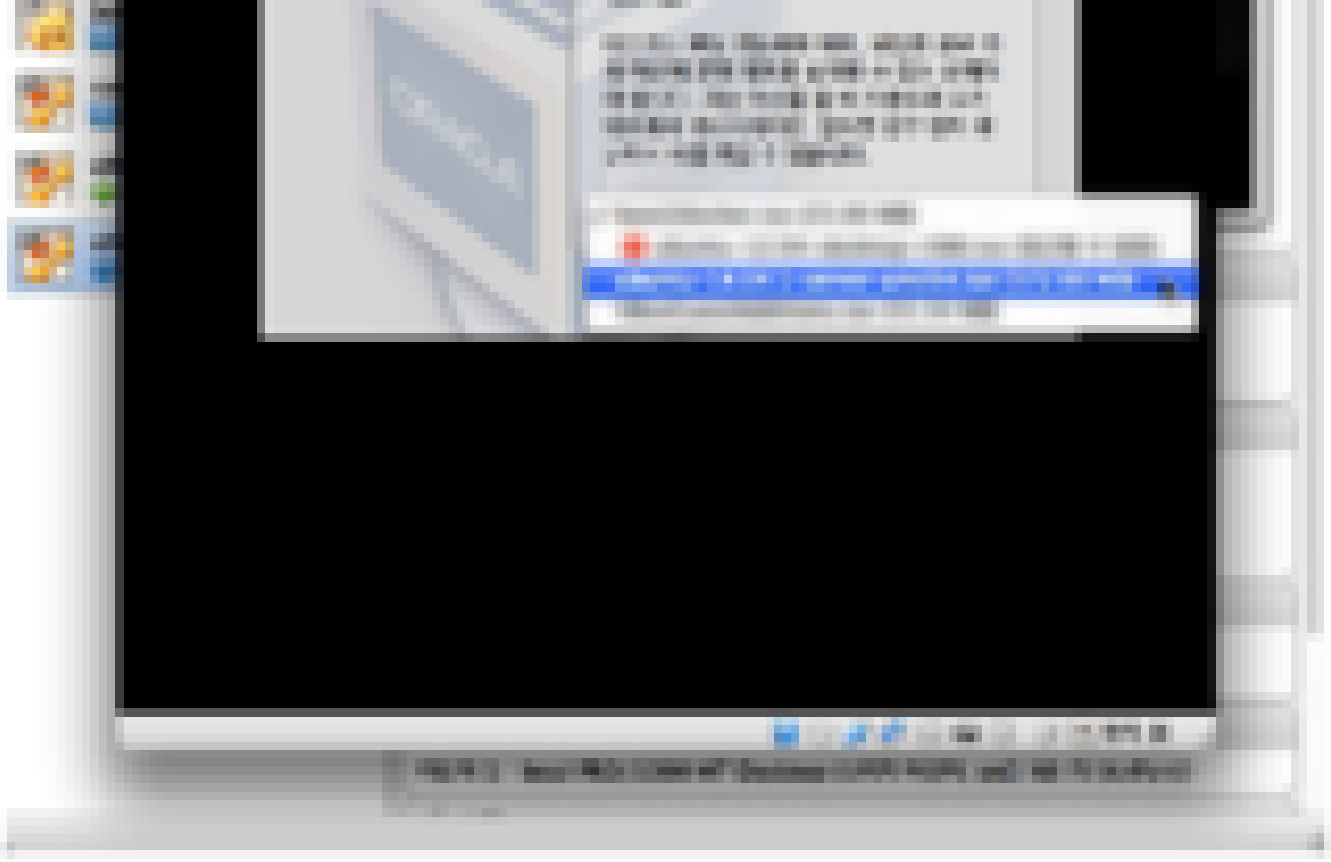


64. 가상머신 시작하기



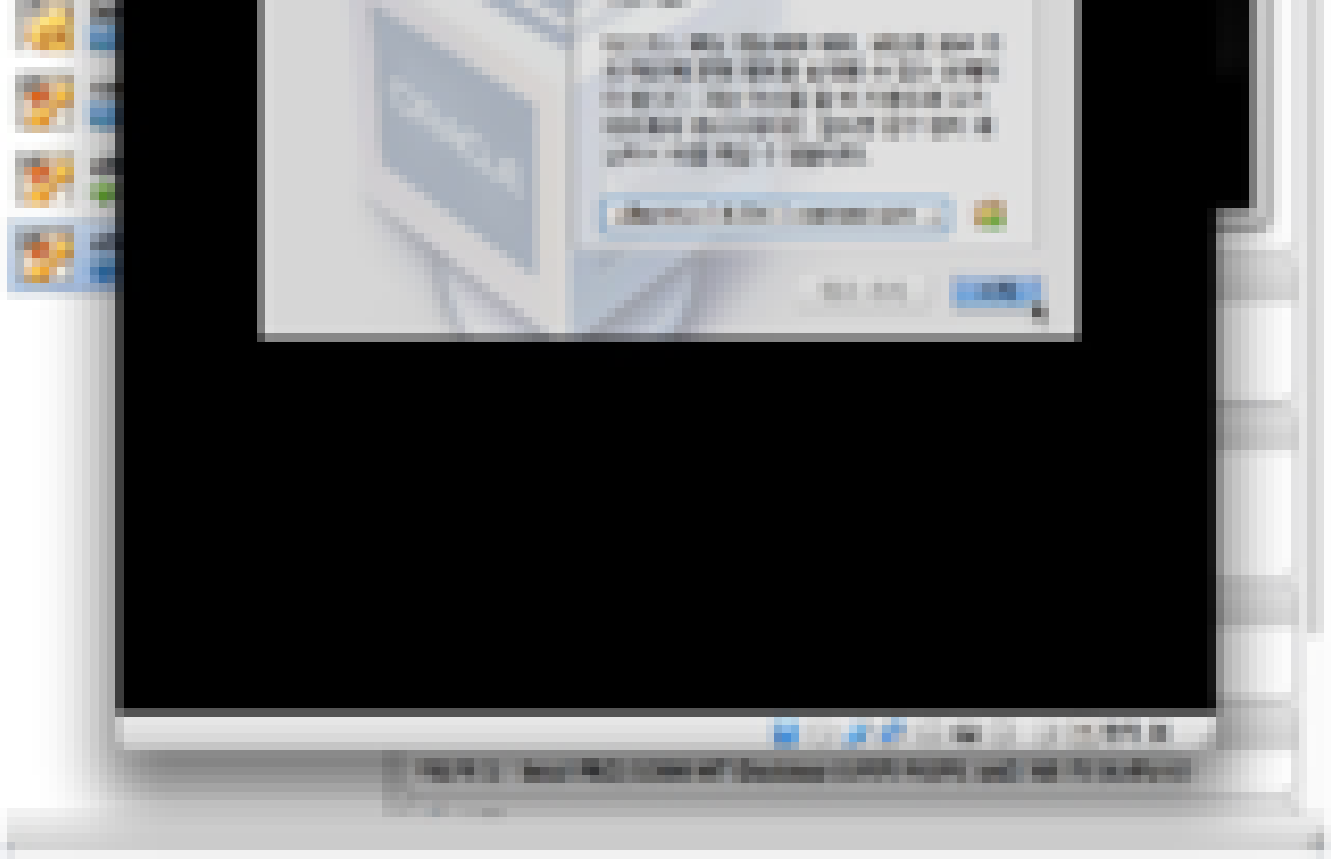
b) 다운로드 받은 사본 이미지 선택하기

: 이미지 다운로드 받기 눌른 후 받은 사본 이미지 저장을 선택합니다.

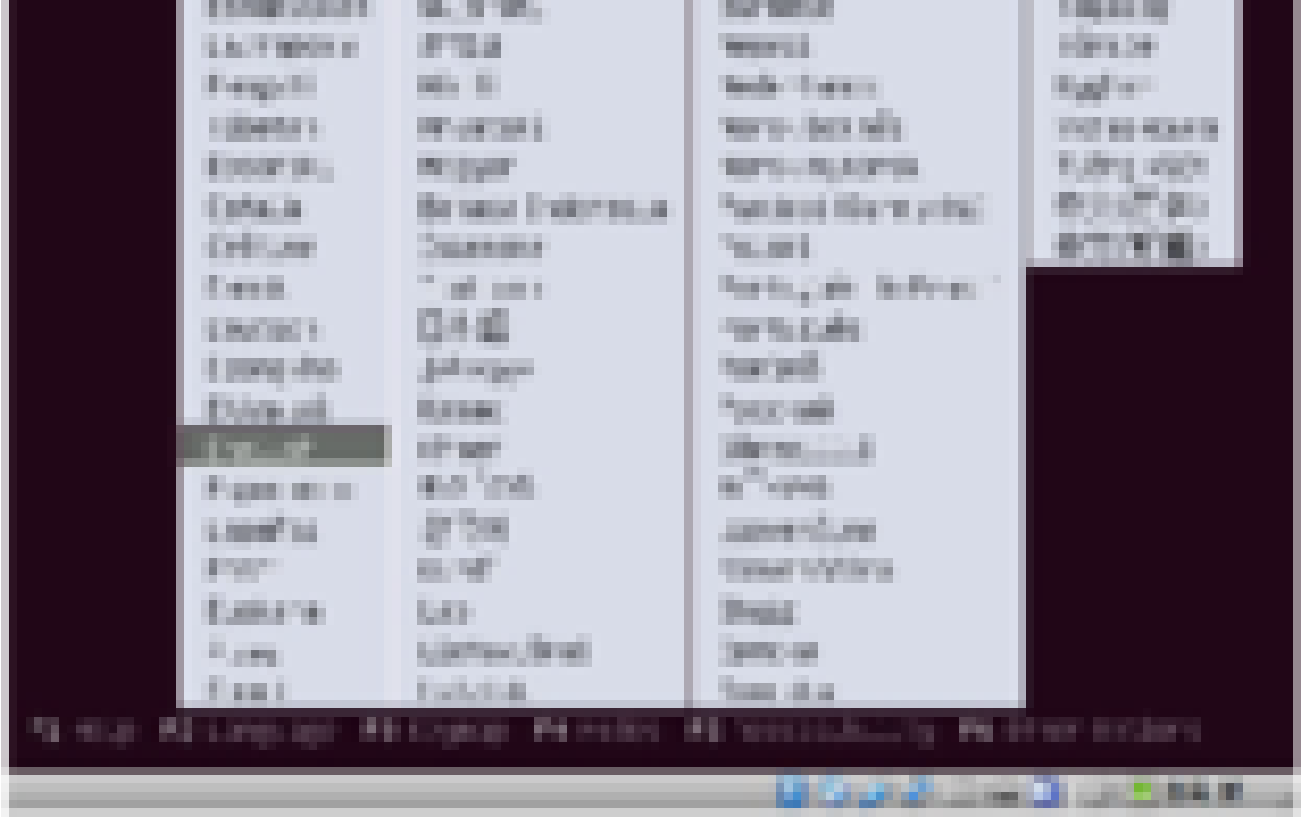


게스트 서버의 설치

5.5. 서버 설치 시작

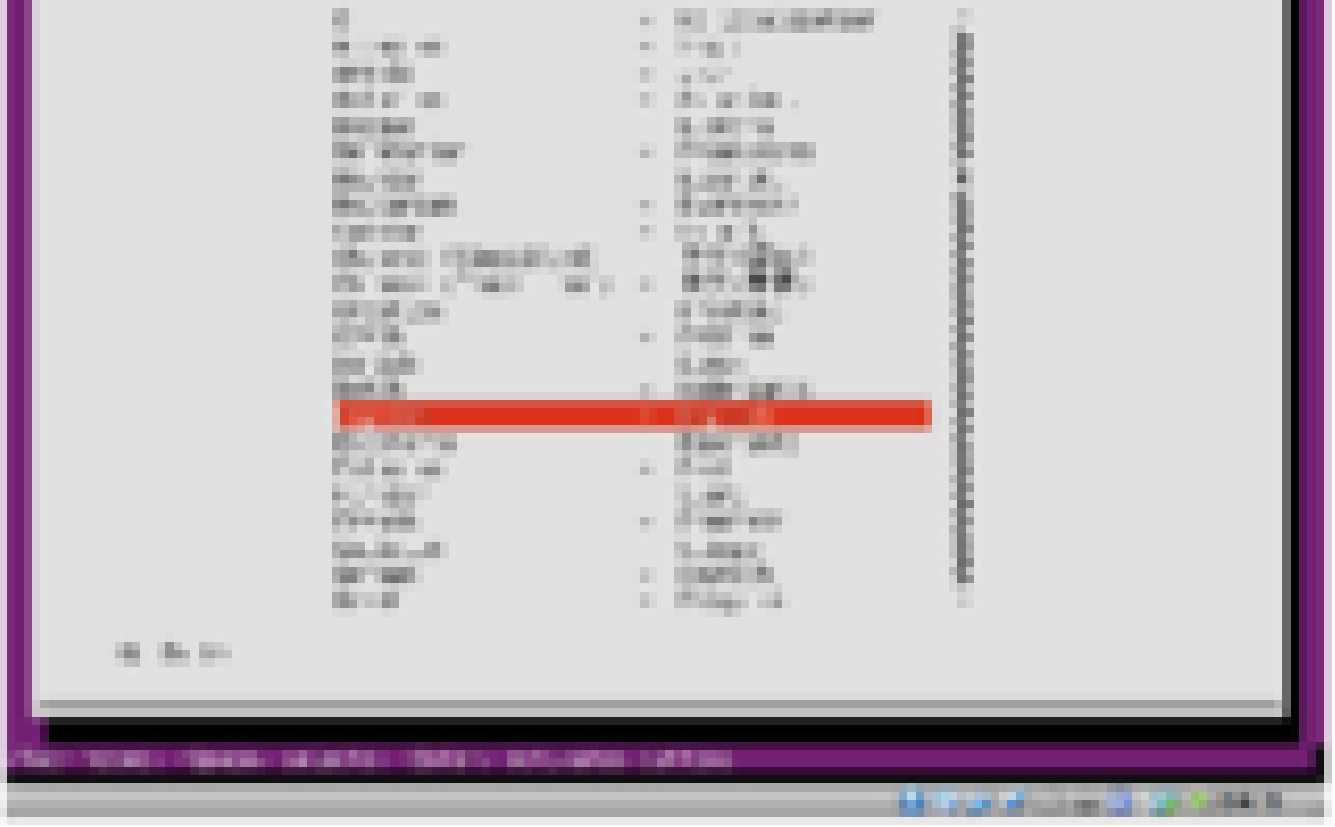


34. 사용할 언어를 선택하는 화면에서 기준 삽입 Snapshot 가 반전된 상태에서 영어 키를 클릭합니다.

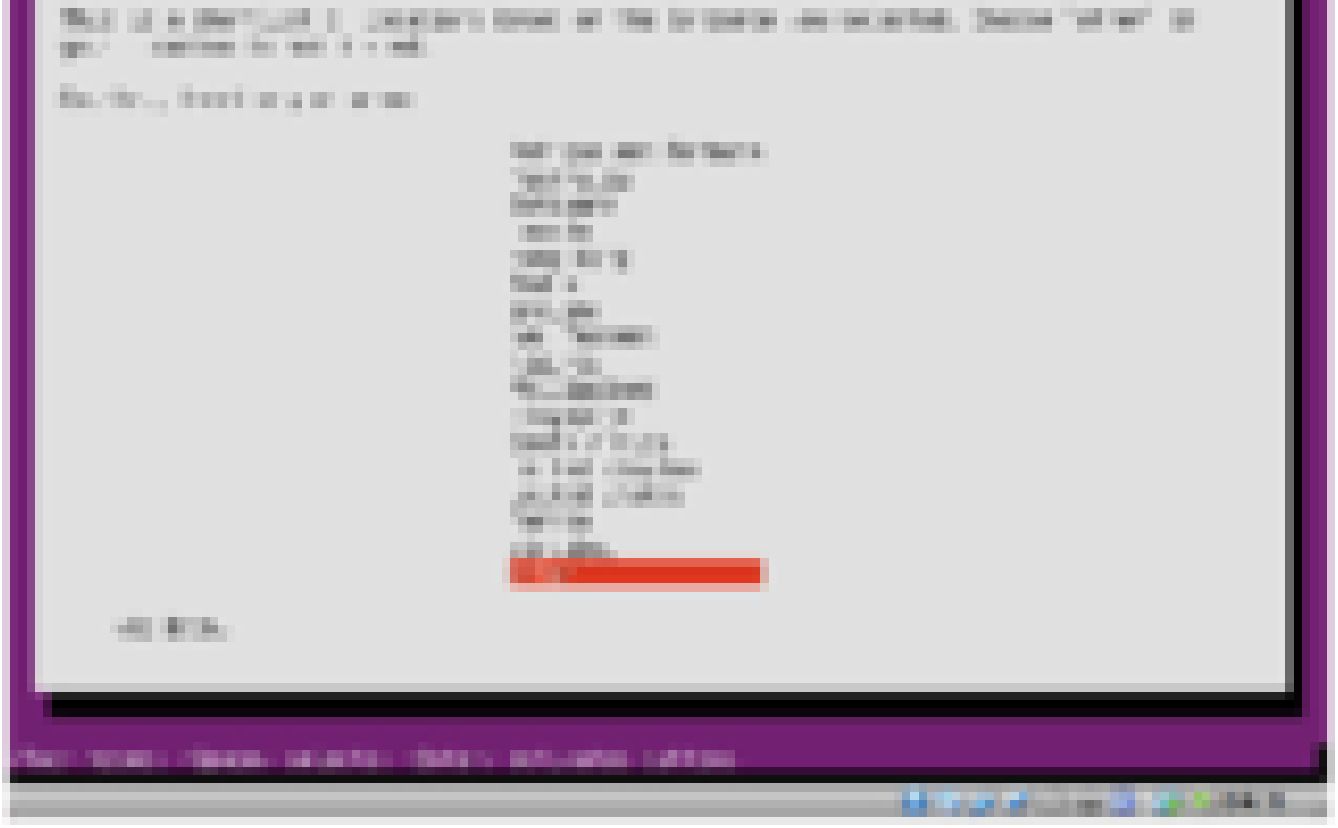


15. 우분투 서버 설치 항목을 선택한다.

24. 설치 진행시 사용할 언어를 선택하는 화면입니다. 기본값인 English가 반전된 상태에서 엔터키를 클릭합니다.



37. 재입찰 설명을 위한 자격을 선택하는 화면에서는 either 항목을 선택하고 엔터키를 클릭한다.



이미지 기술: 한글 맞춤법 기능을 사용하고 있는 사용자 화면입니다.

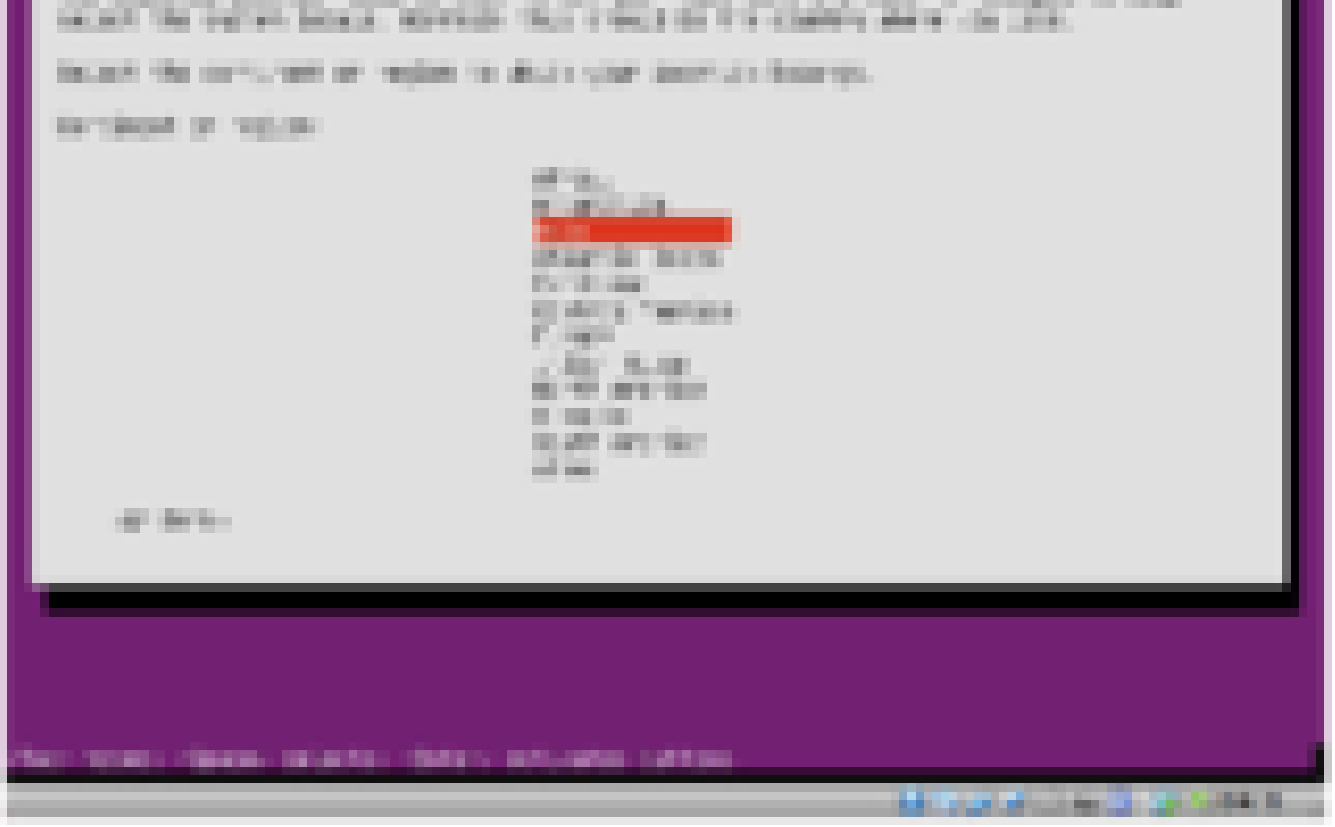
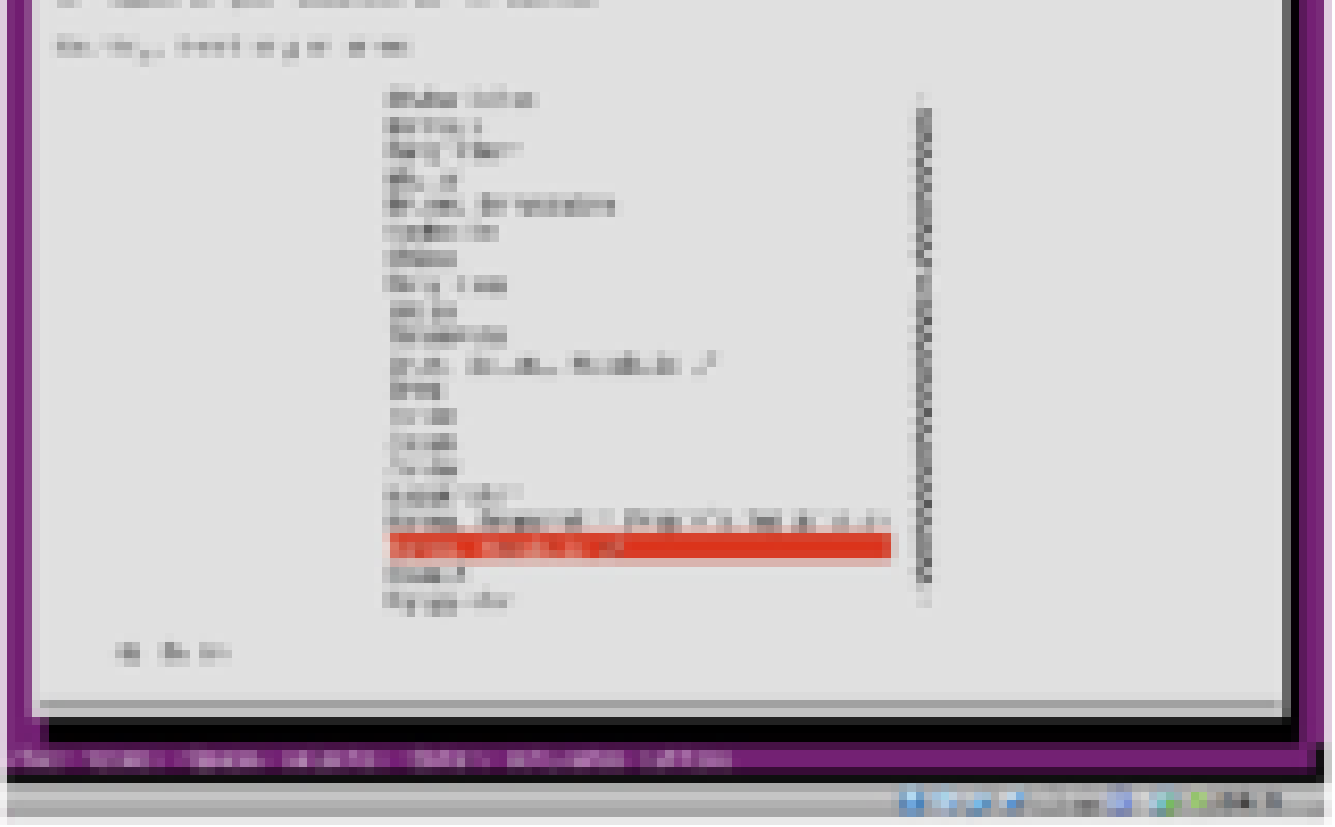
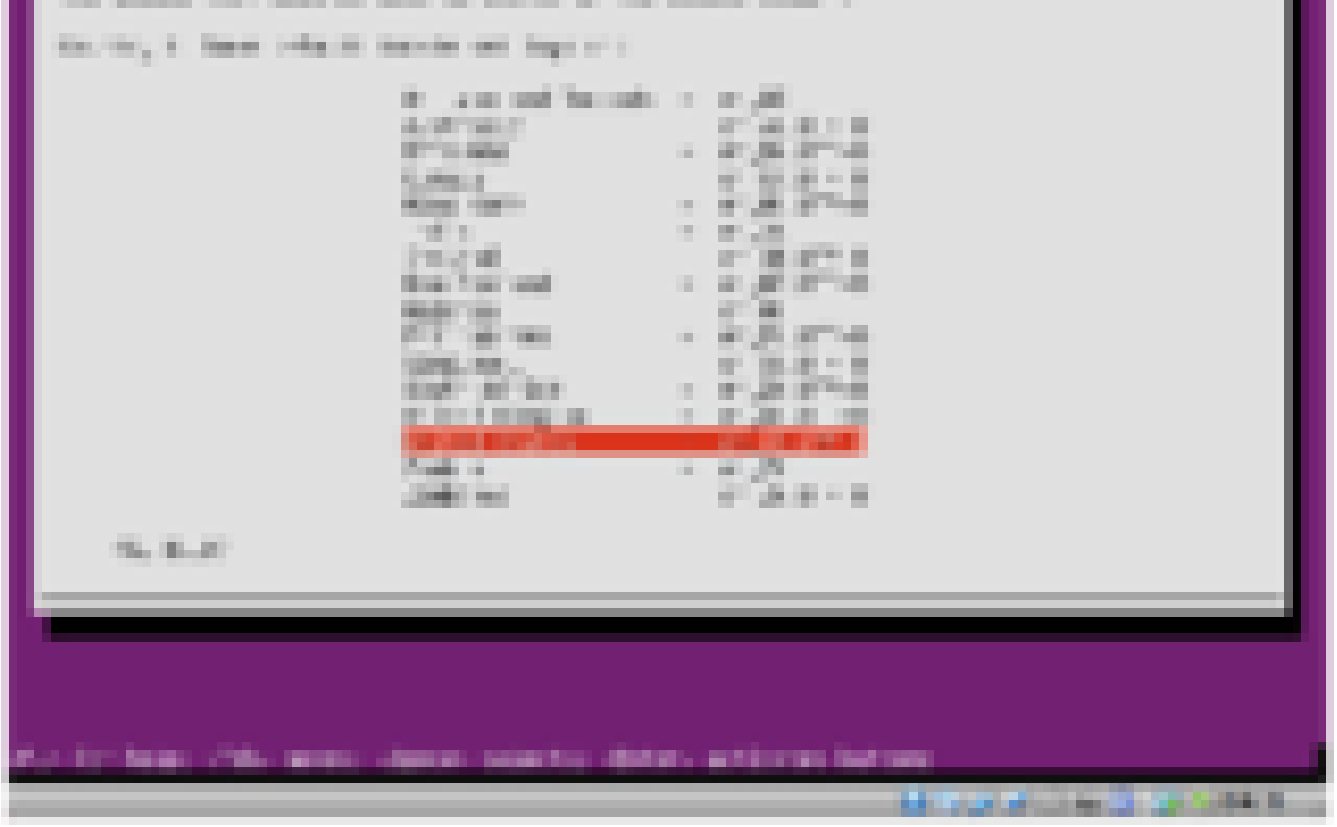


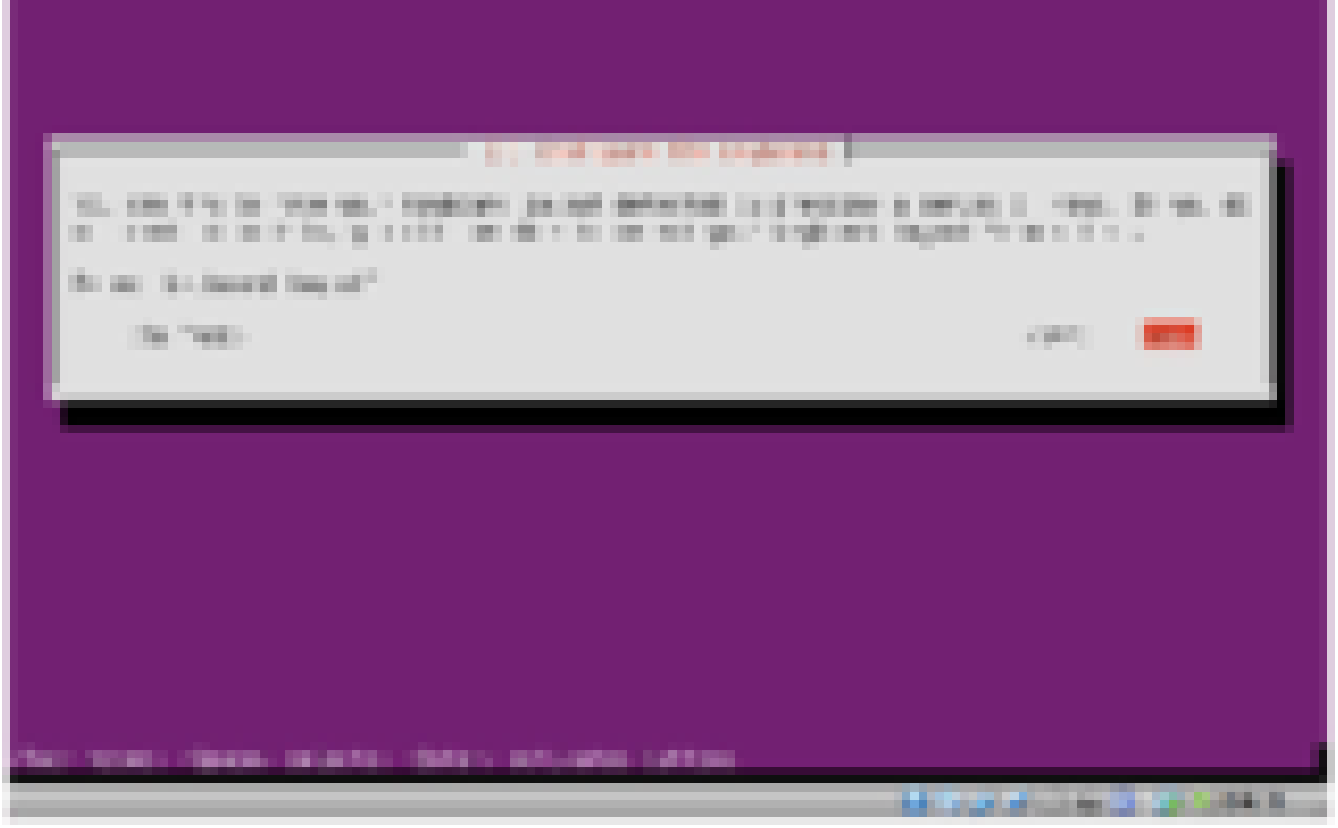
그림 3. [illegible] 에 [illegible] 을 [illegible] 합니다.



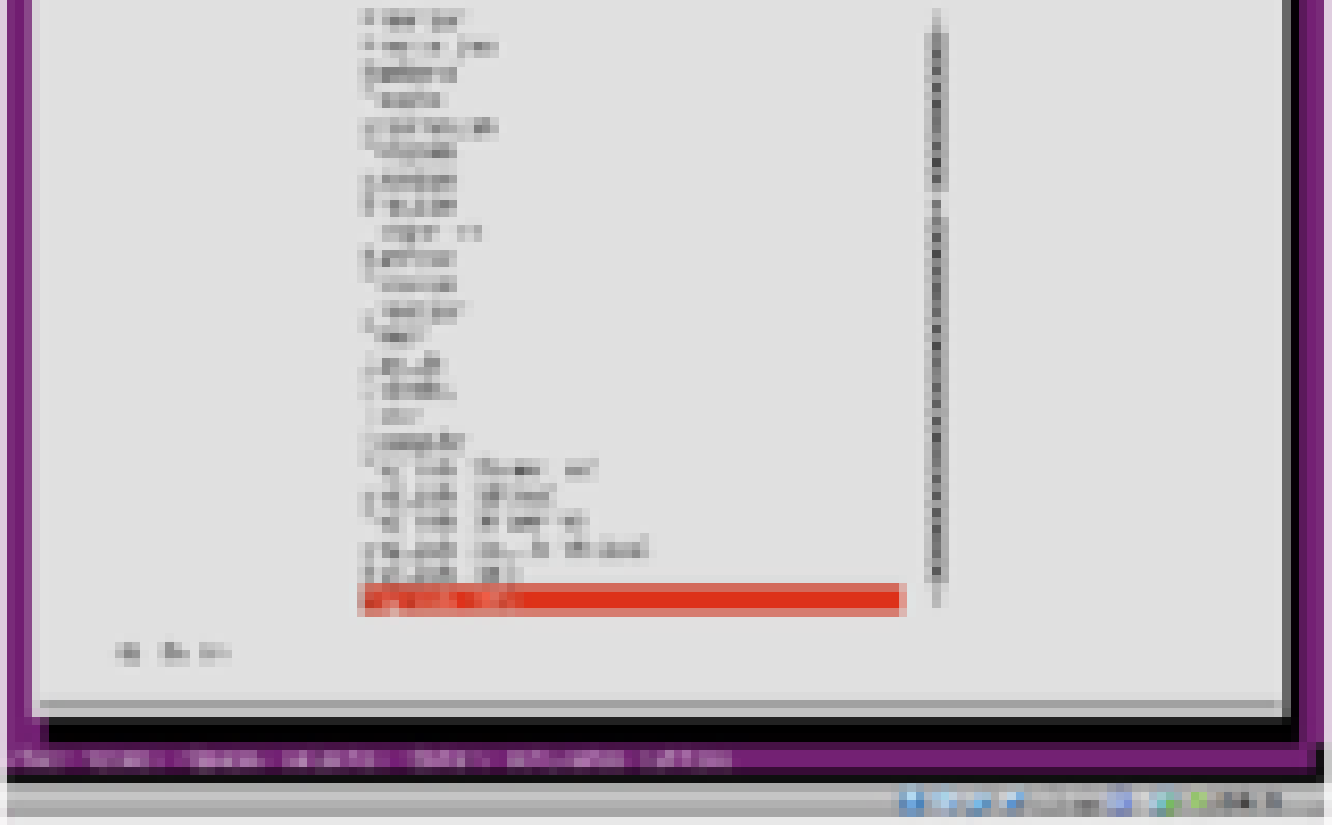
16. 프개일 설정 화면에서는 기본값인 Wicked Shades + ex_US.UTF-8 이 맞으면 엔터를 눌러줍니다.



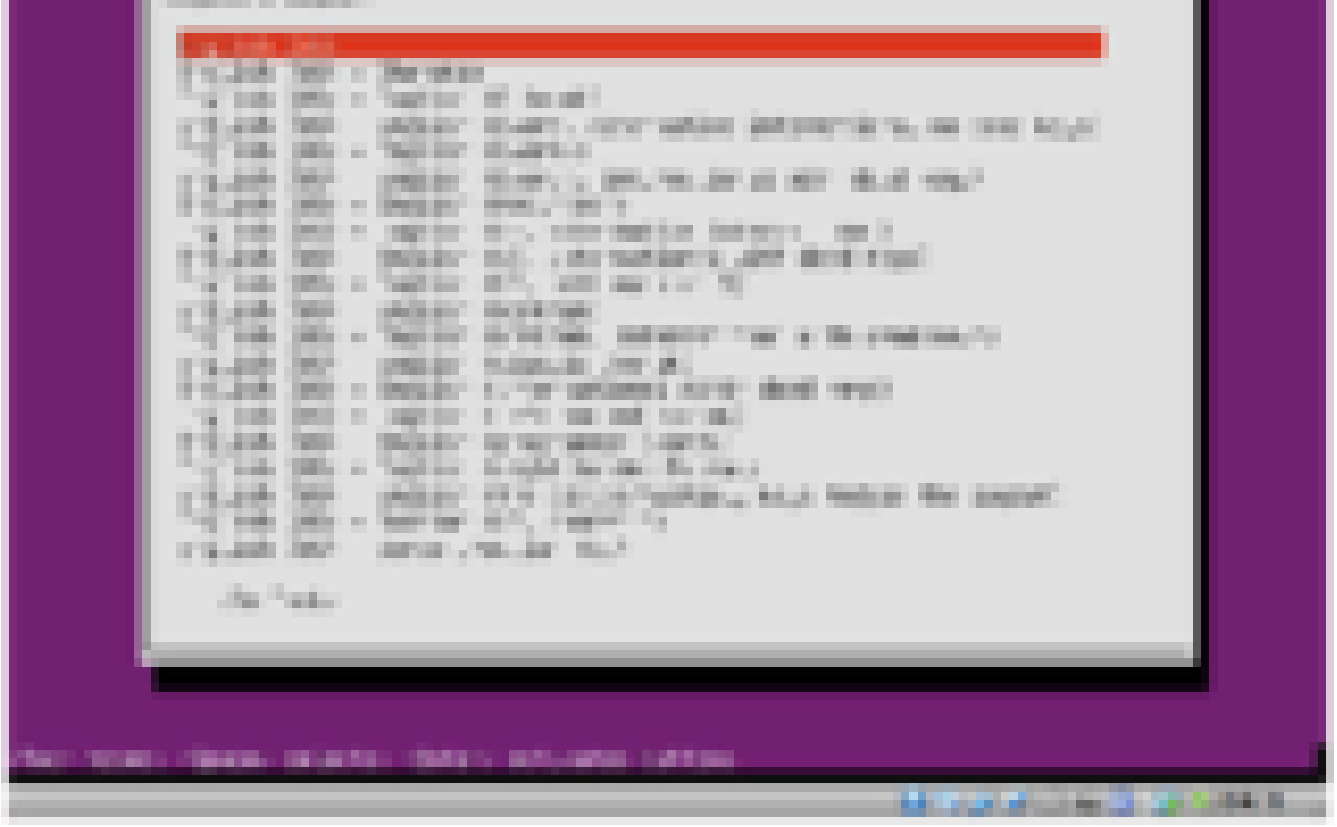
19. 키보드 입력 화면에서는 기본값인 <Ok>를 선택한다.



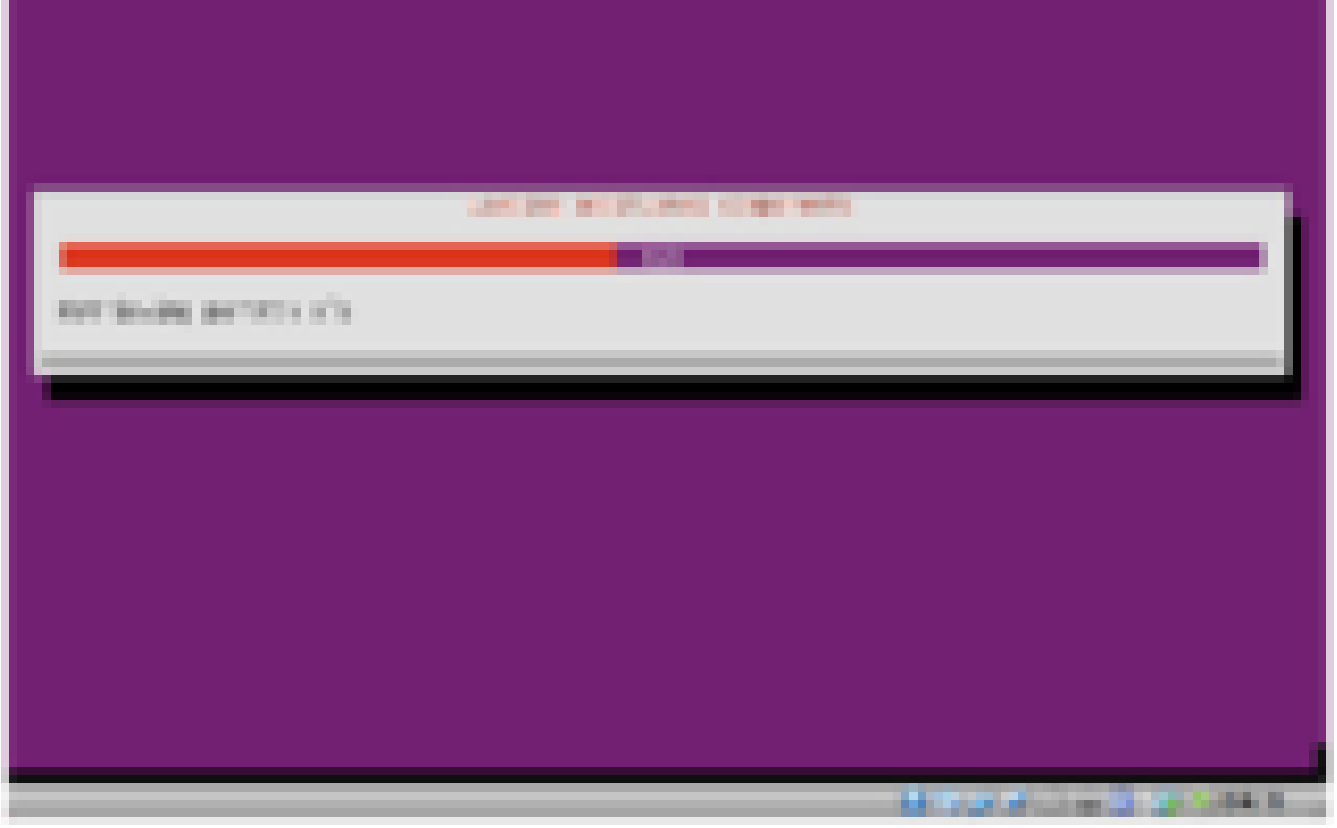
26. 키보드 레이아웃 선택 화면(Country of origin이라고도 기본값인 English (US) 를 선택합니다.



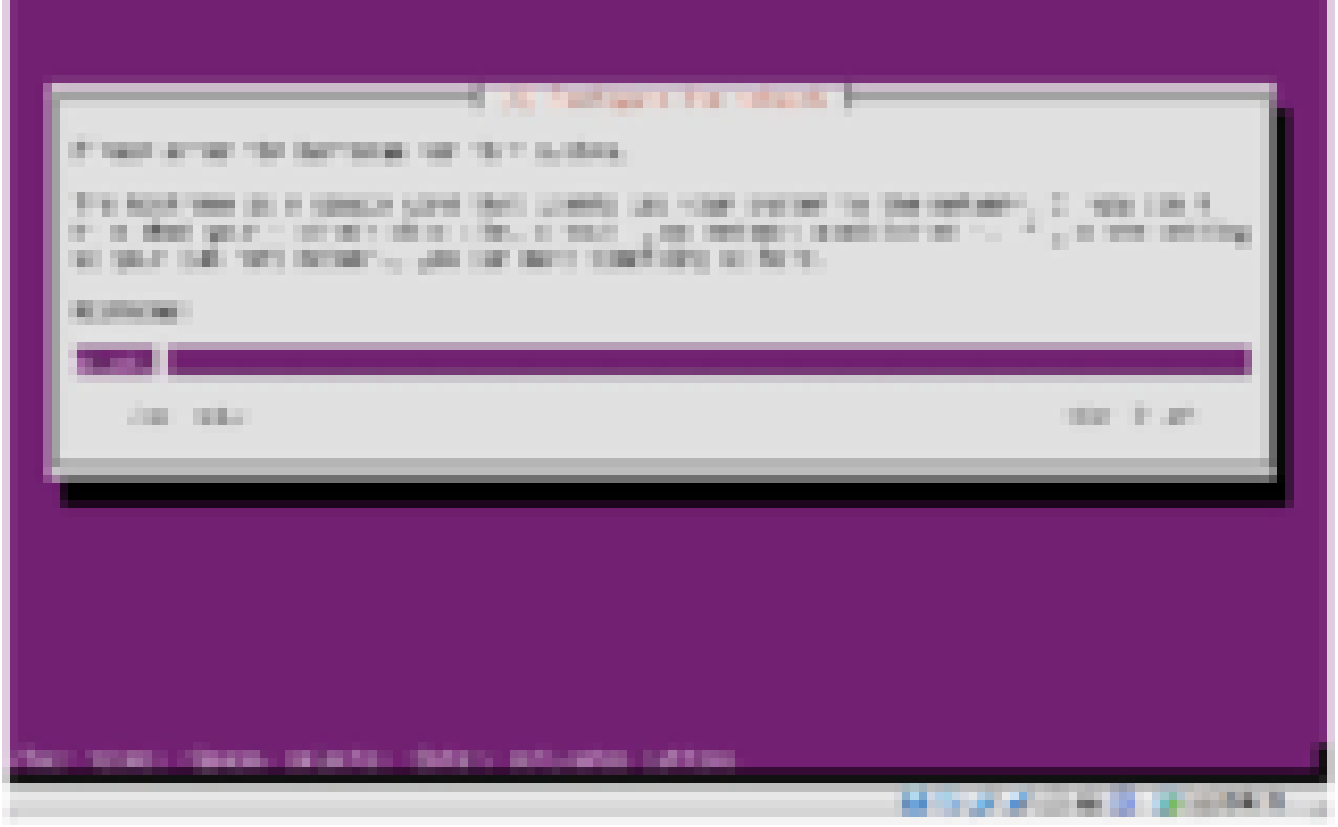
21. 키보드 레이아웃의 기본값인 English (US) 를 선택합니다.



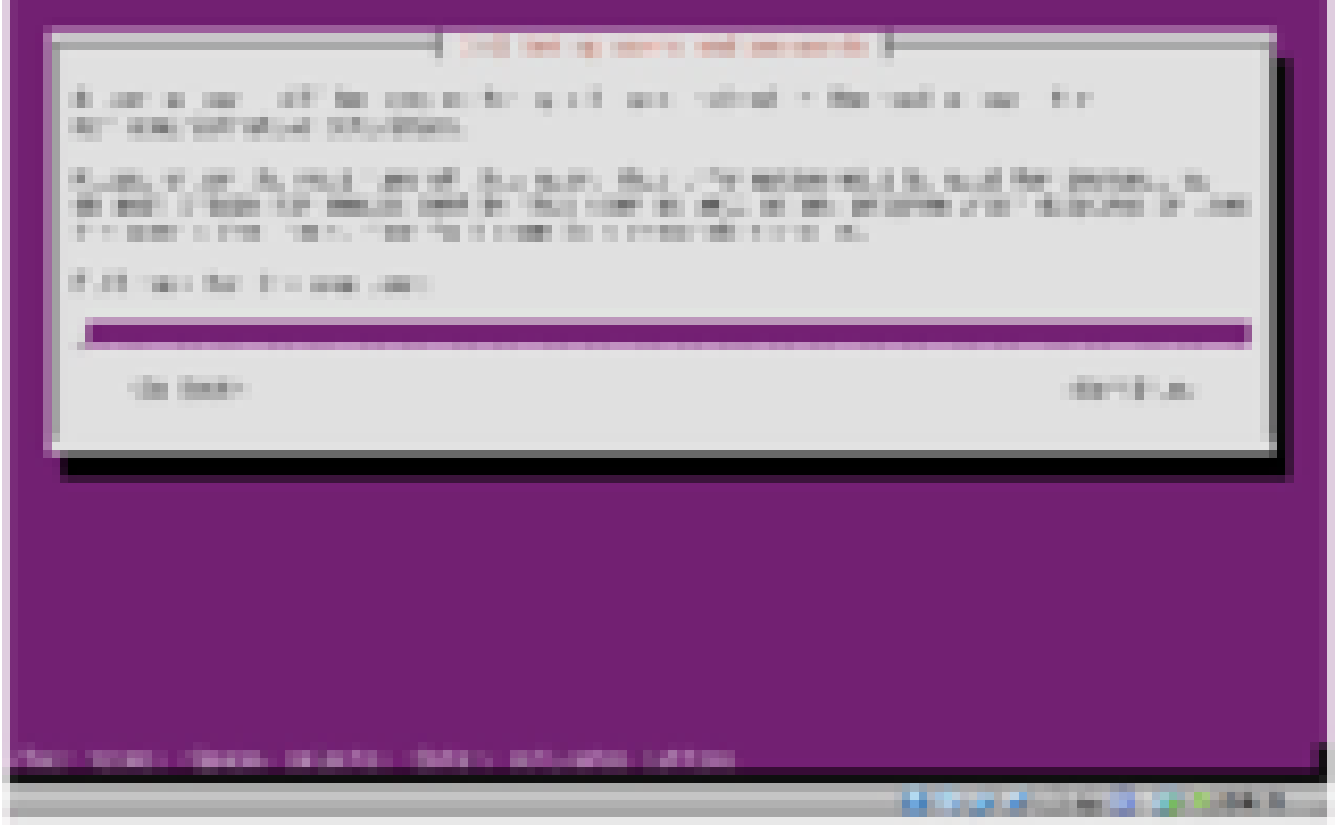
23. 입금상자 인쇄하기



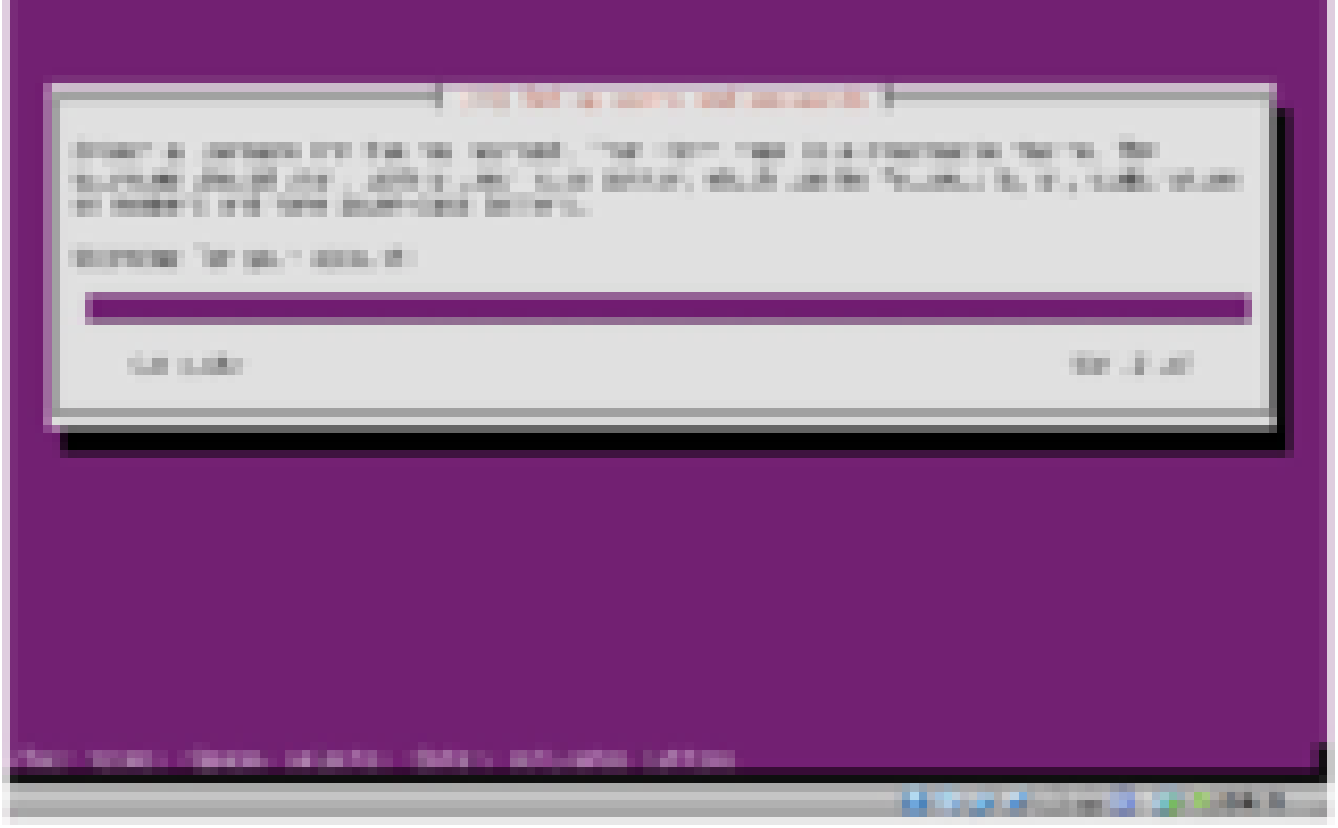
23. 각 상태 신호 사용한 호스트 사용률 정정하기.



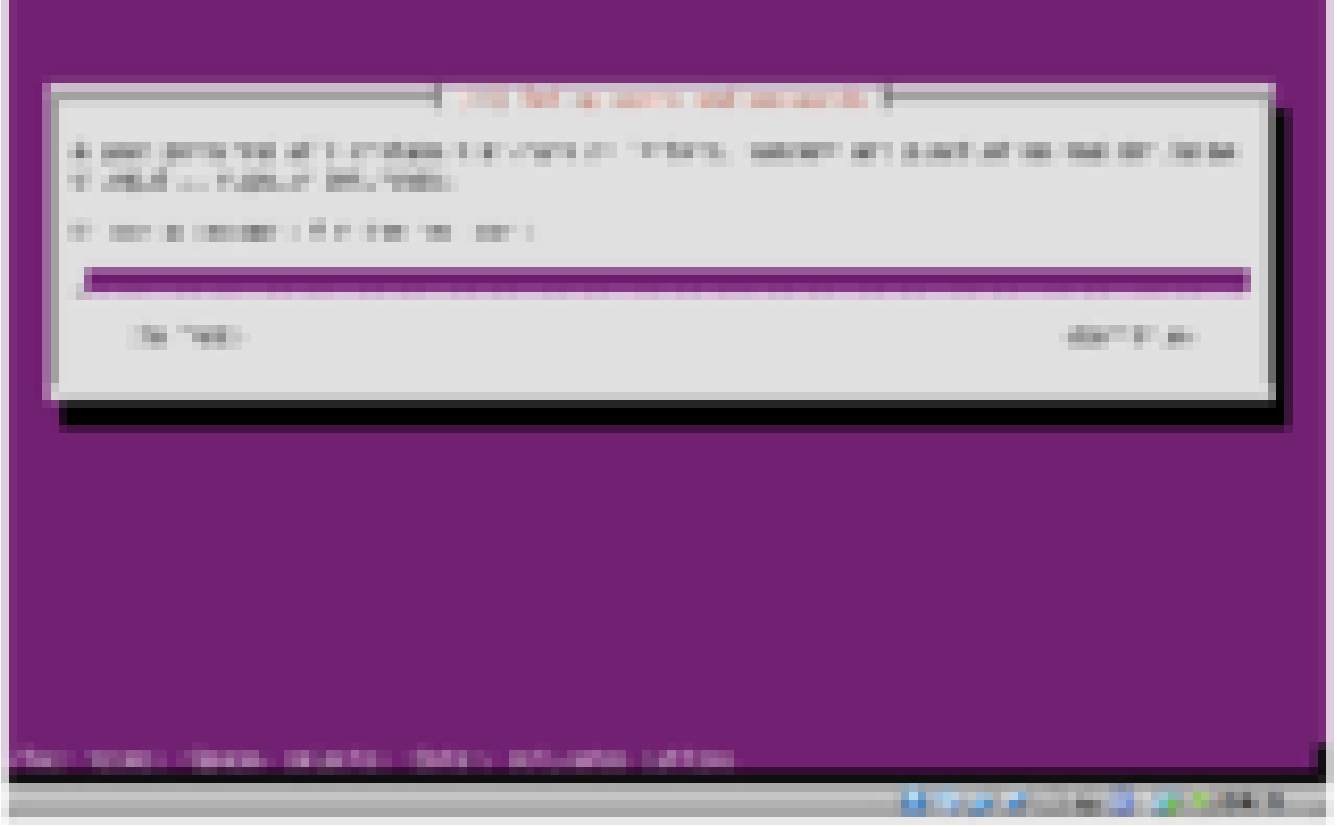
24. 유저 이름을 입력한다.



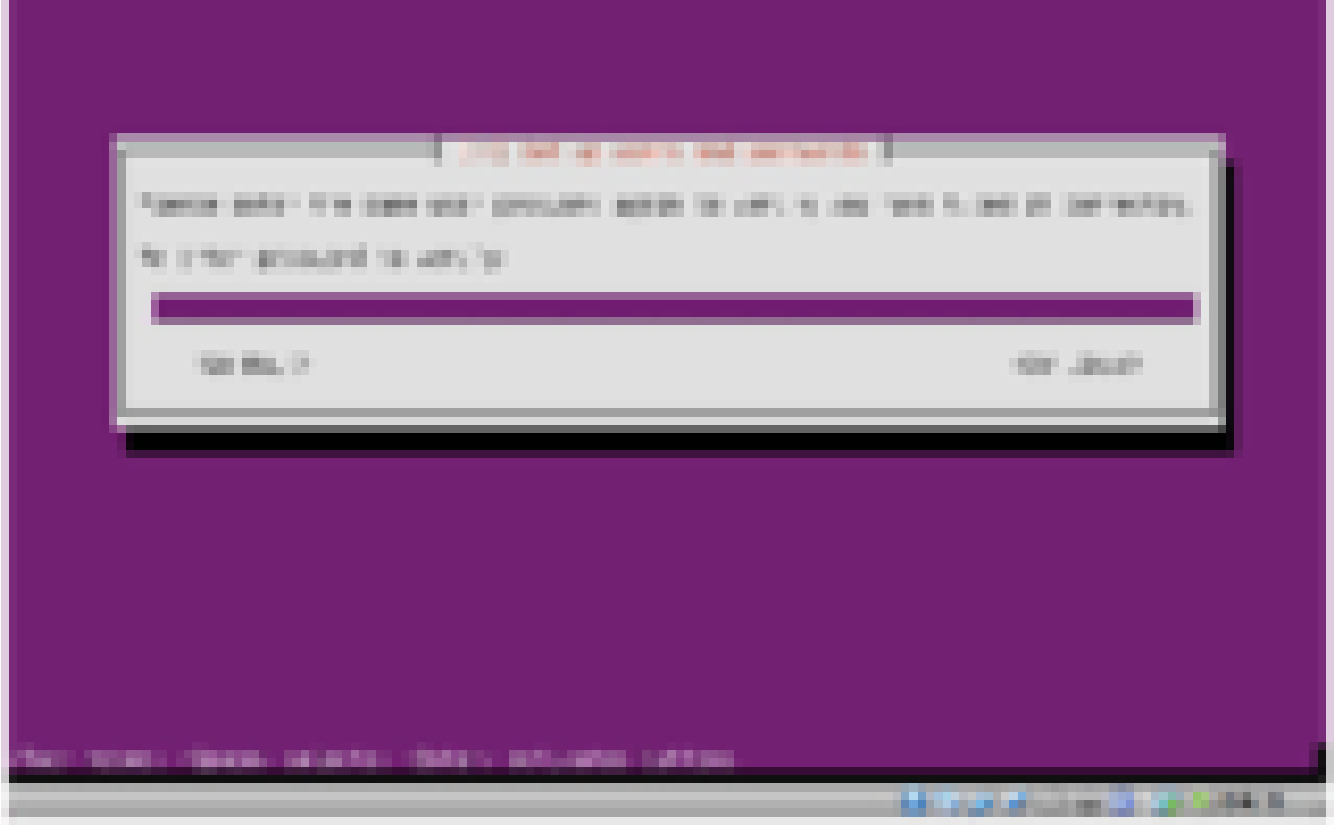
25. 계정 이름을 입력한다.



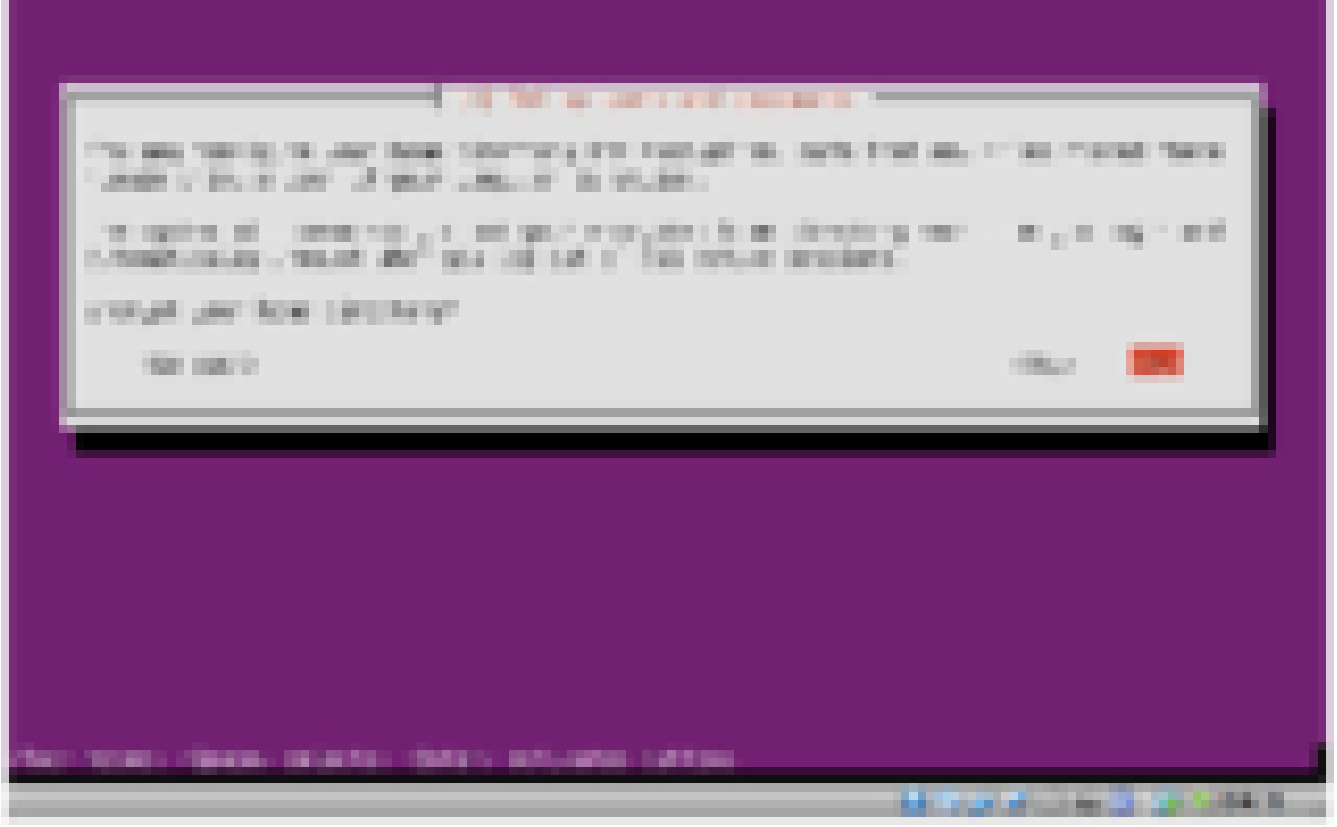
24. 암호를 입력합니다.



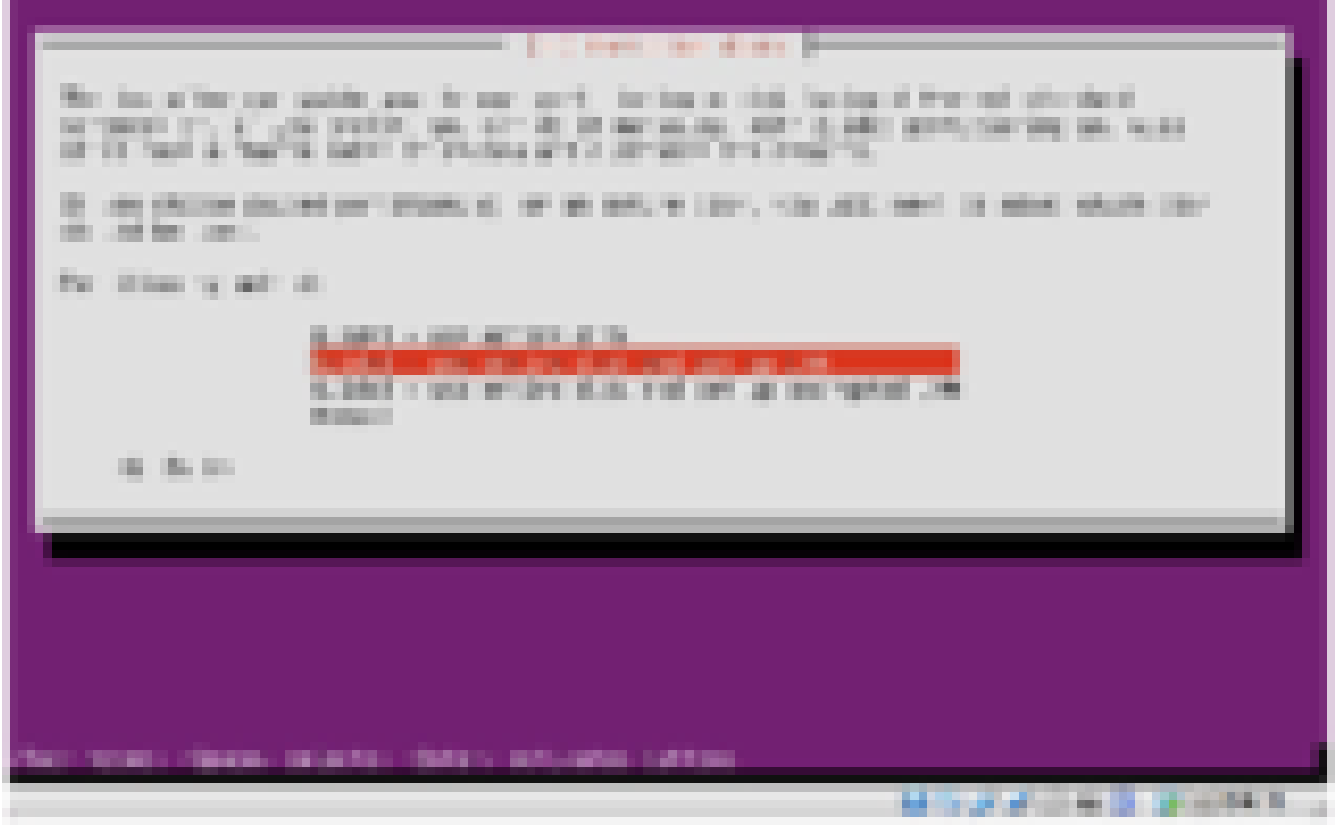
[illegible]



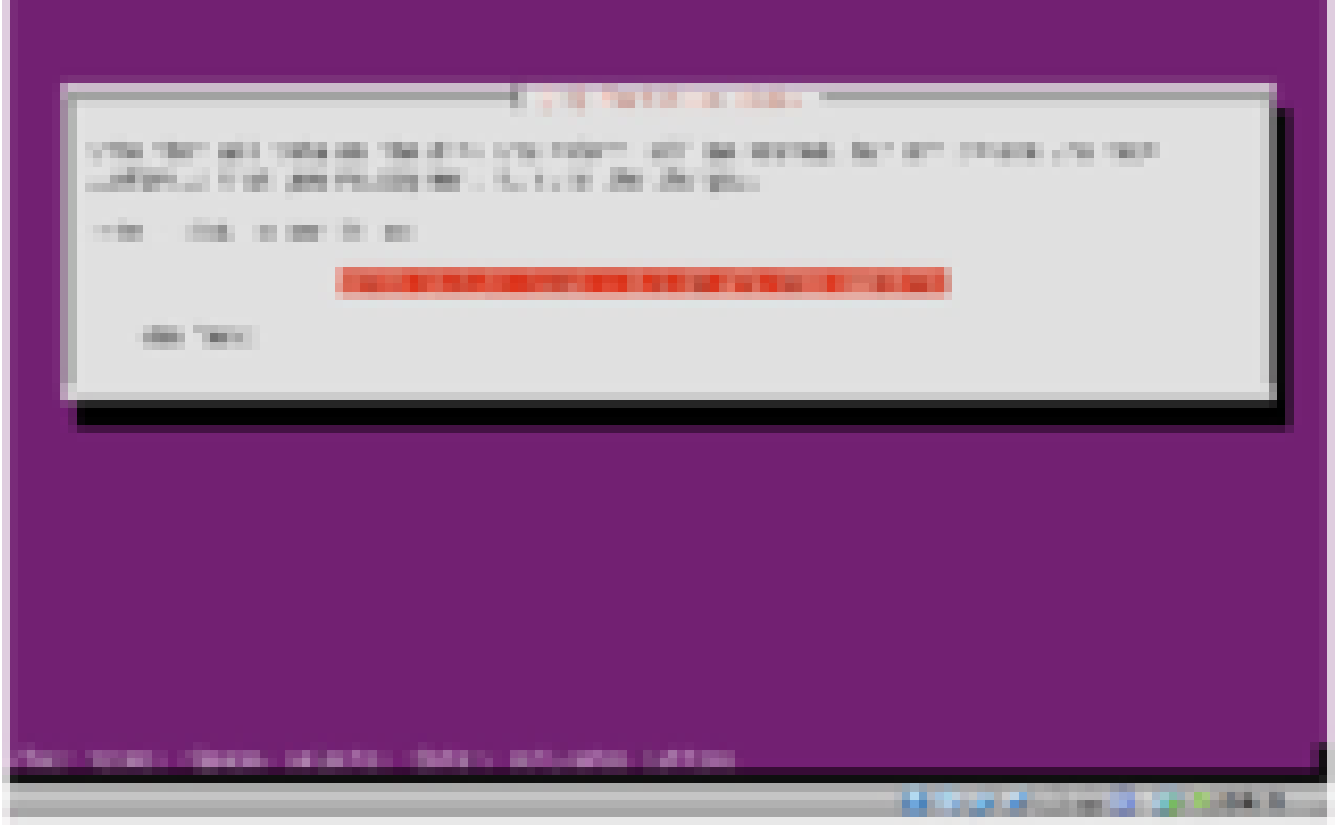
27. 솔 다이얼로그 입력창에 대부분 기본값인 <Ok> 를 선택합니다.



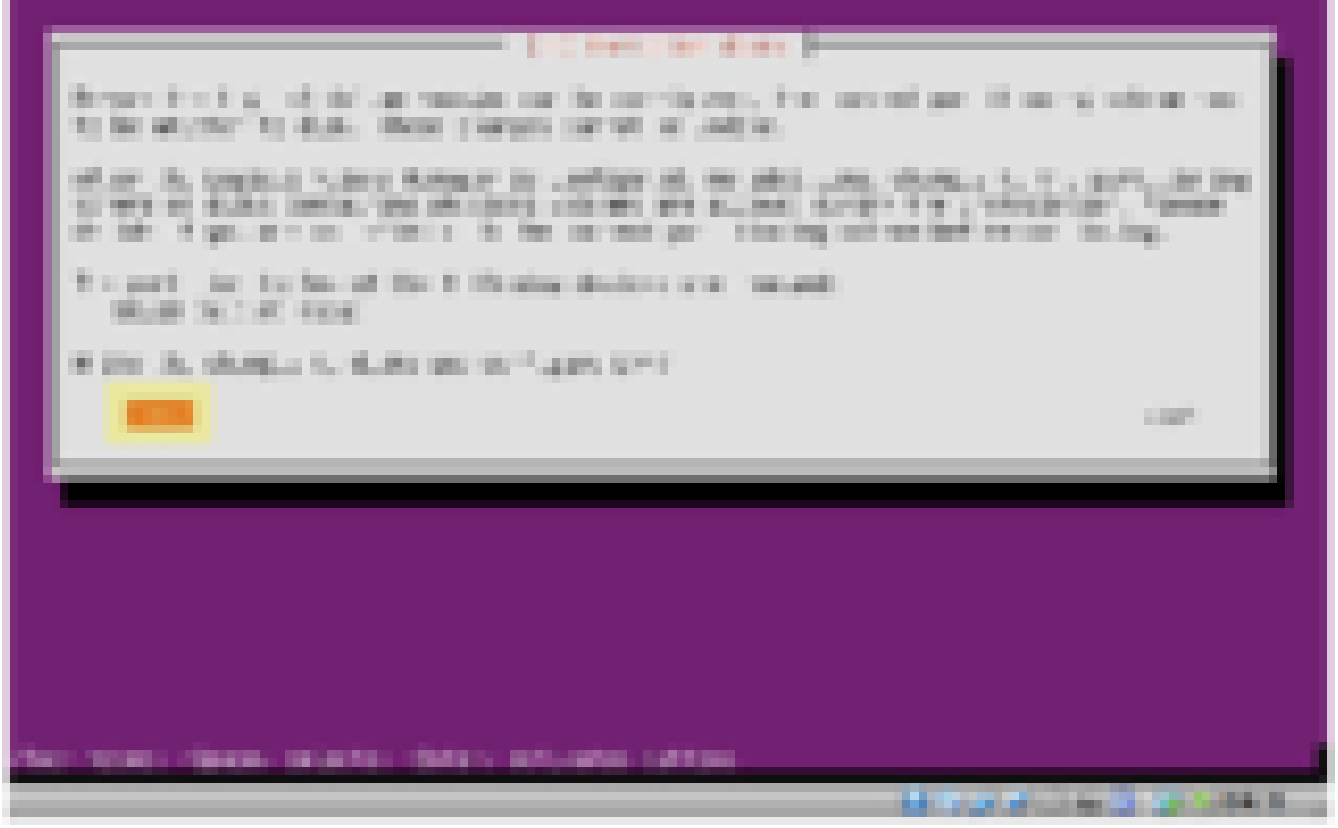
[28] 하드디스크를 파티션 형태로 기본값인 (sda sda1 sda2 sda3 sda5 sda6 sda7 sda8 sda9 sda10 중 선택합니다).



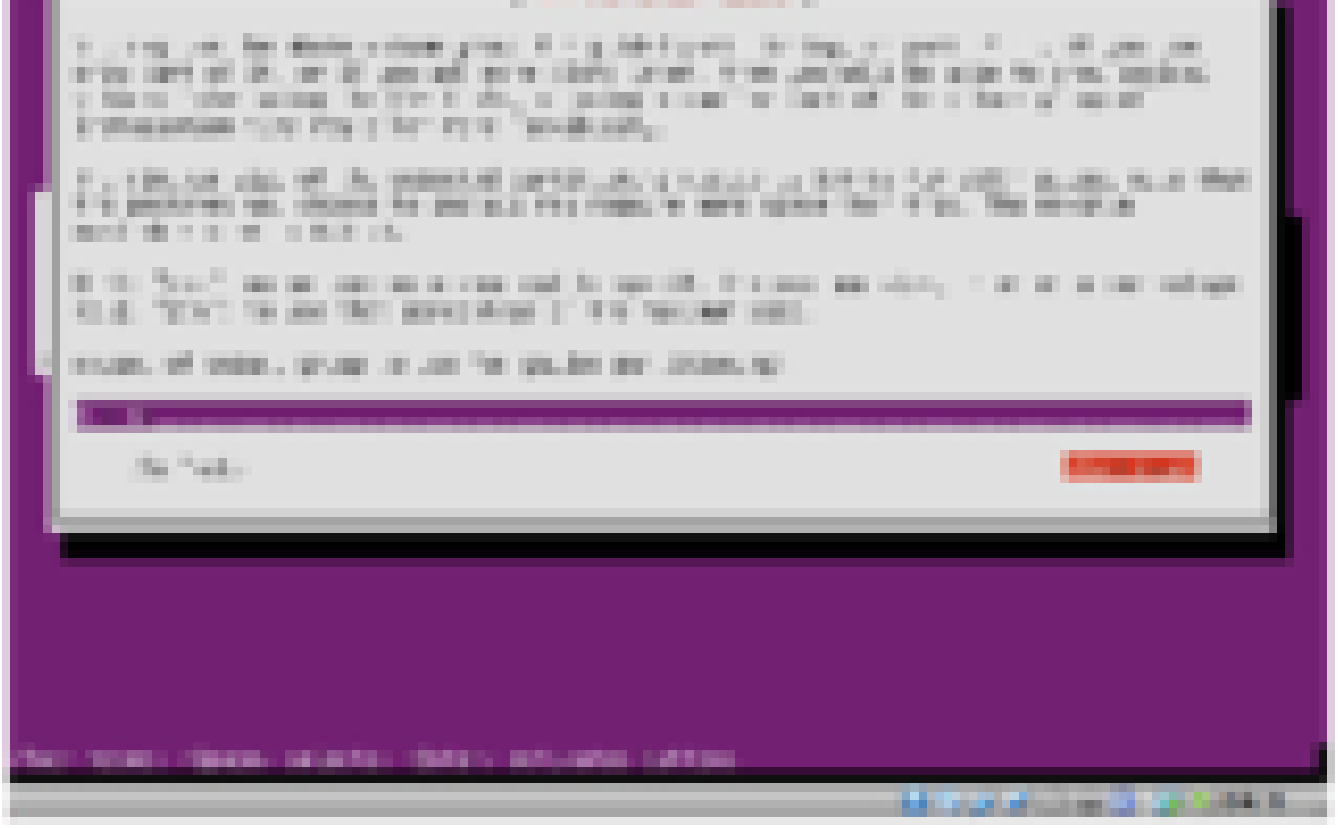
29. 파티션할 디스크를 선택하고 기본값을 선택한다.



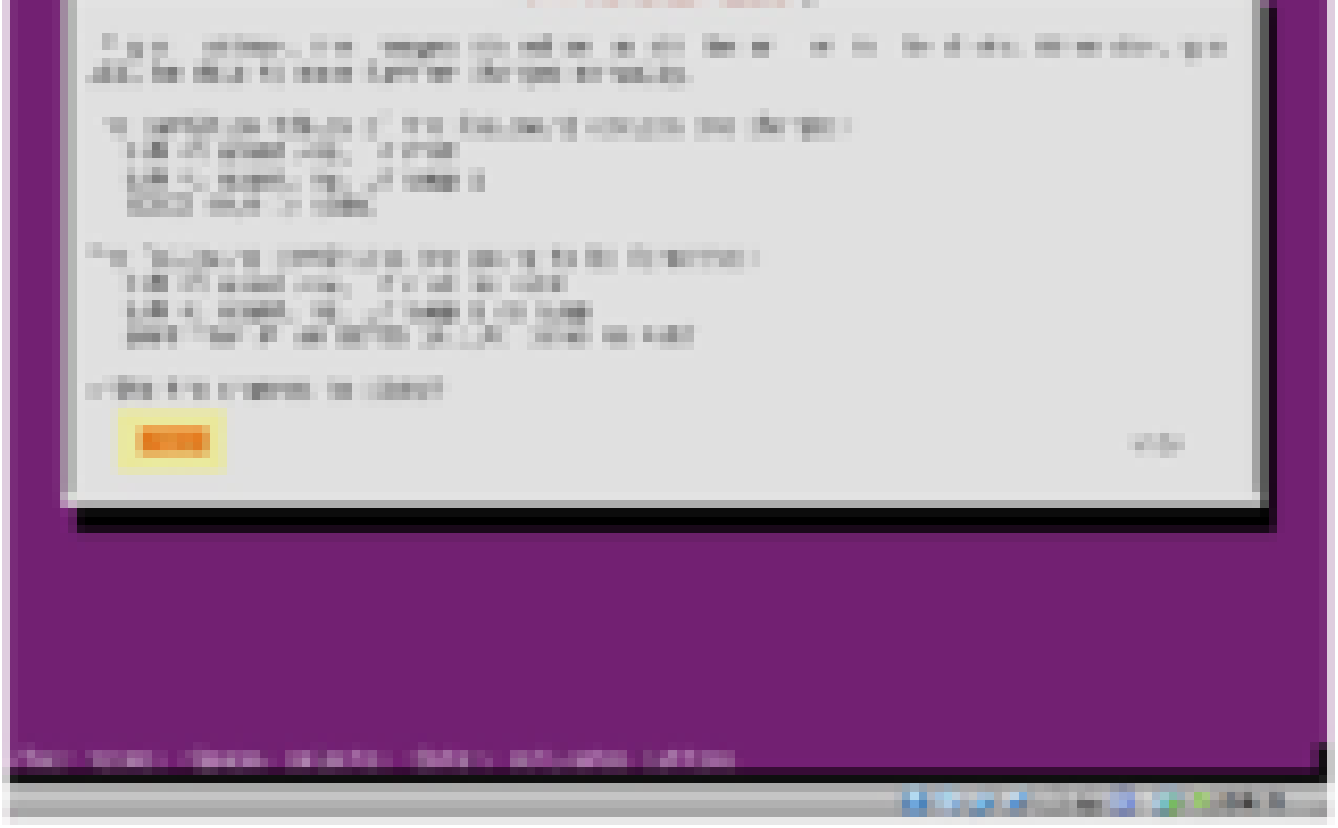
[ok] 변경 내용을 저장 하겠습니까? <Yes> 를 선택합니다.



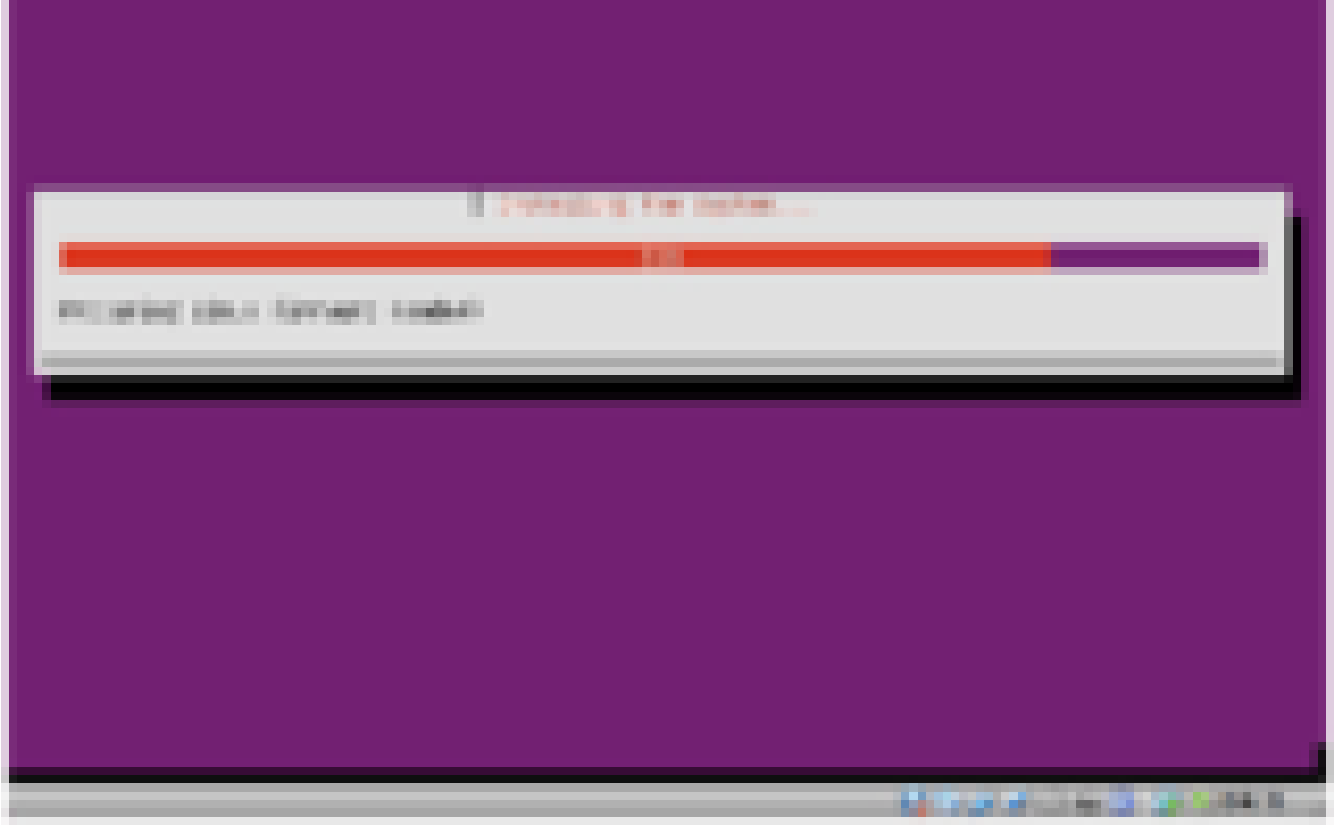
31. 디스크를 활용도 기본값으로 선택하고 계속합니다.



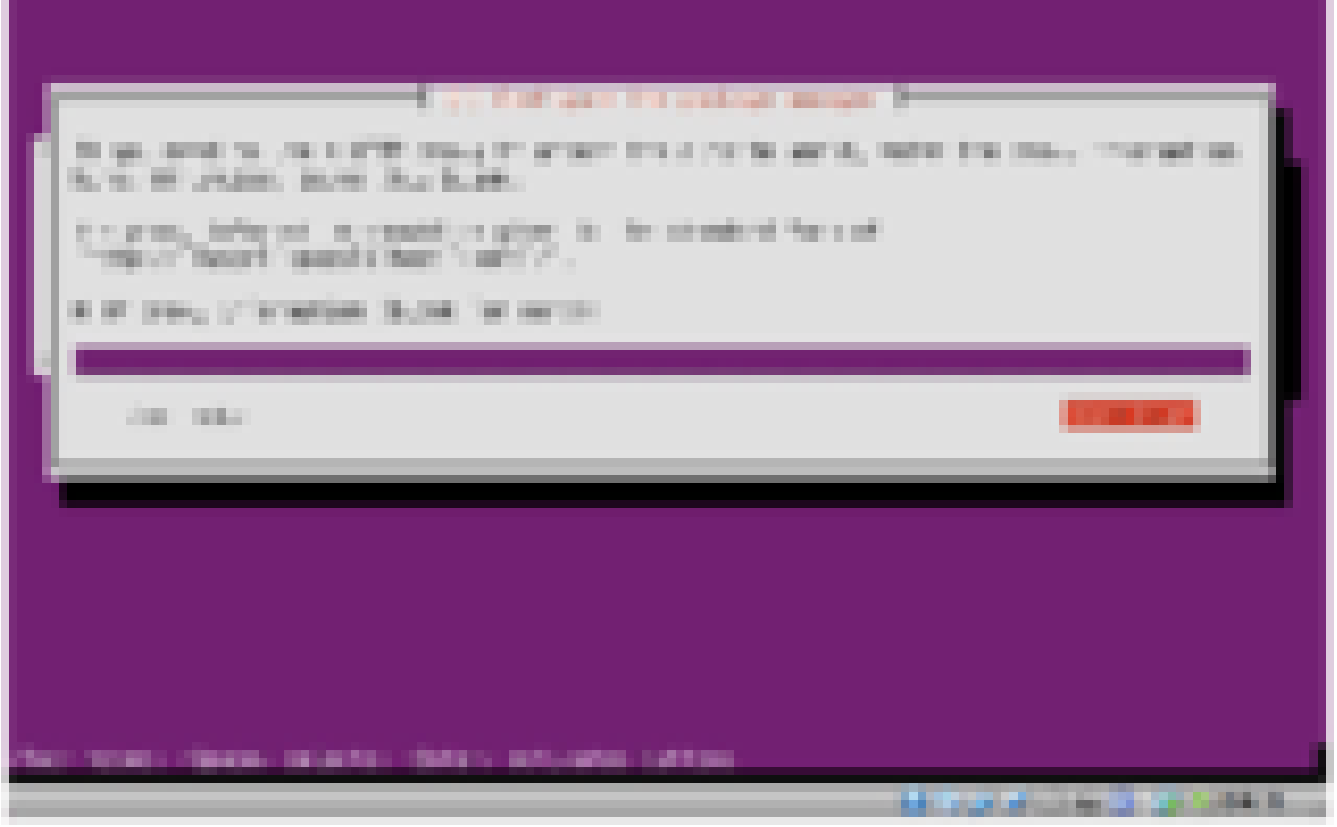
33. 변경 내용을 저장하는 화면에서 <Yes> 를 선택한다.



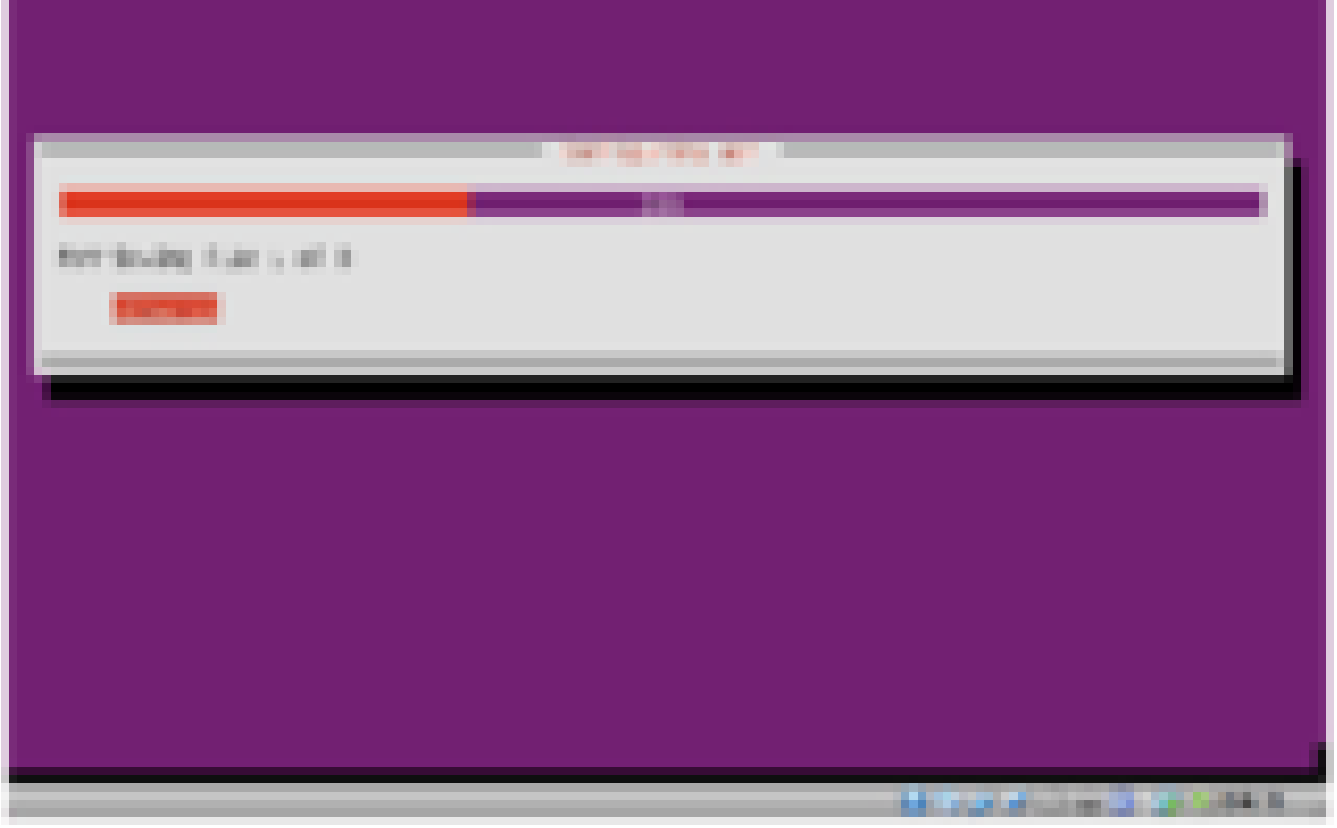
88. 시스템 설치 재부



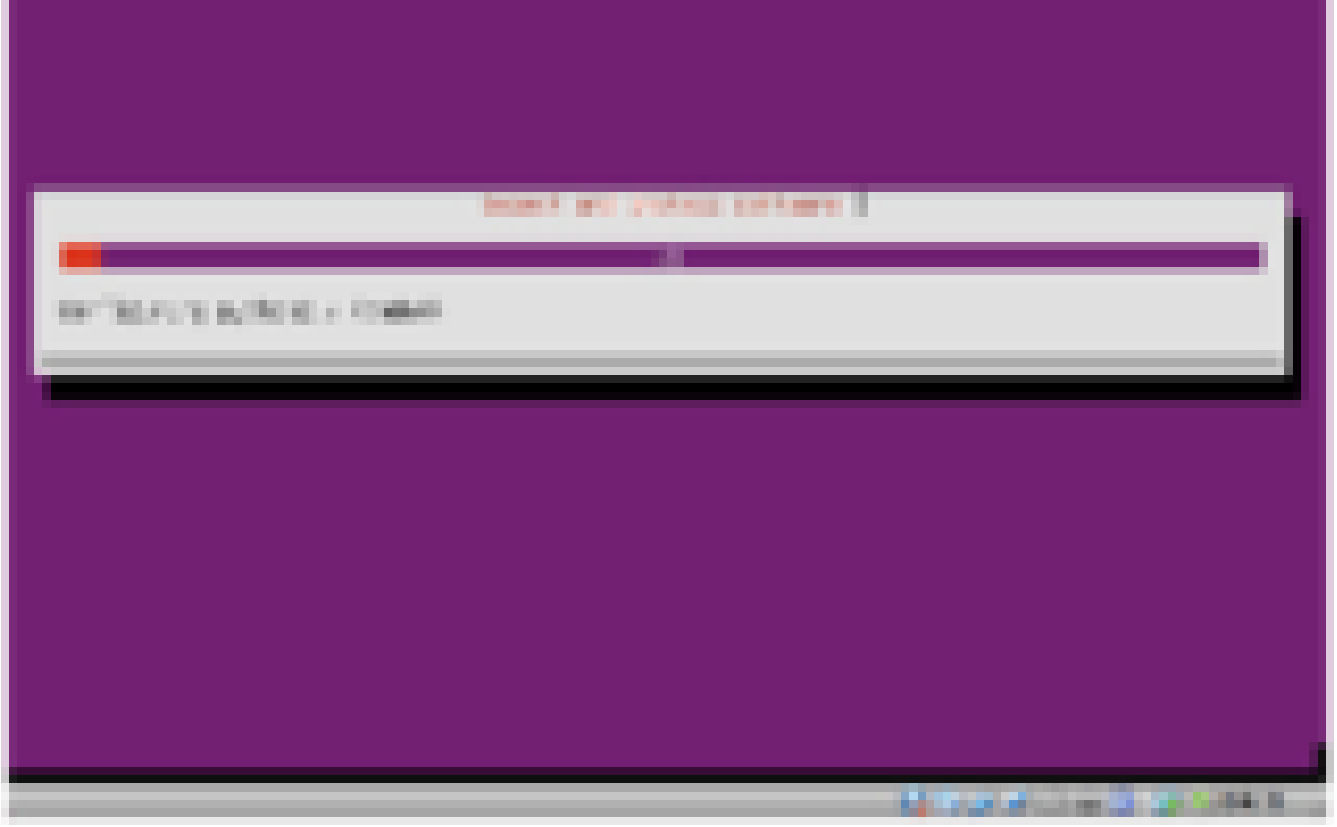
34. Fsck 명령을 사용하여 실행하고 재시작하도 합니다.



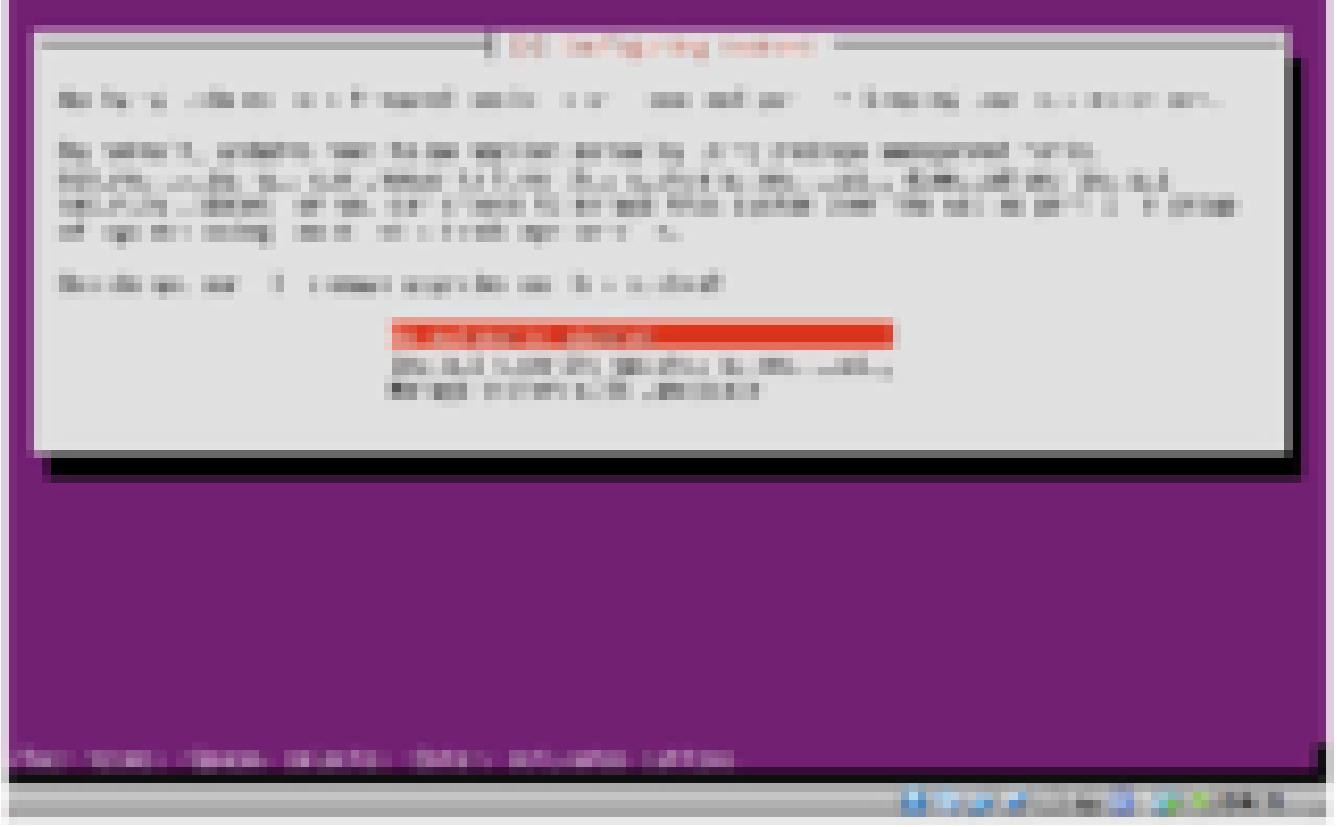
35. 설치 진행 대장 (1)



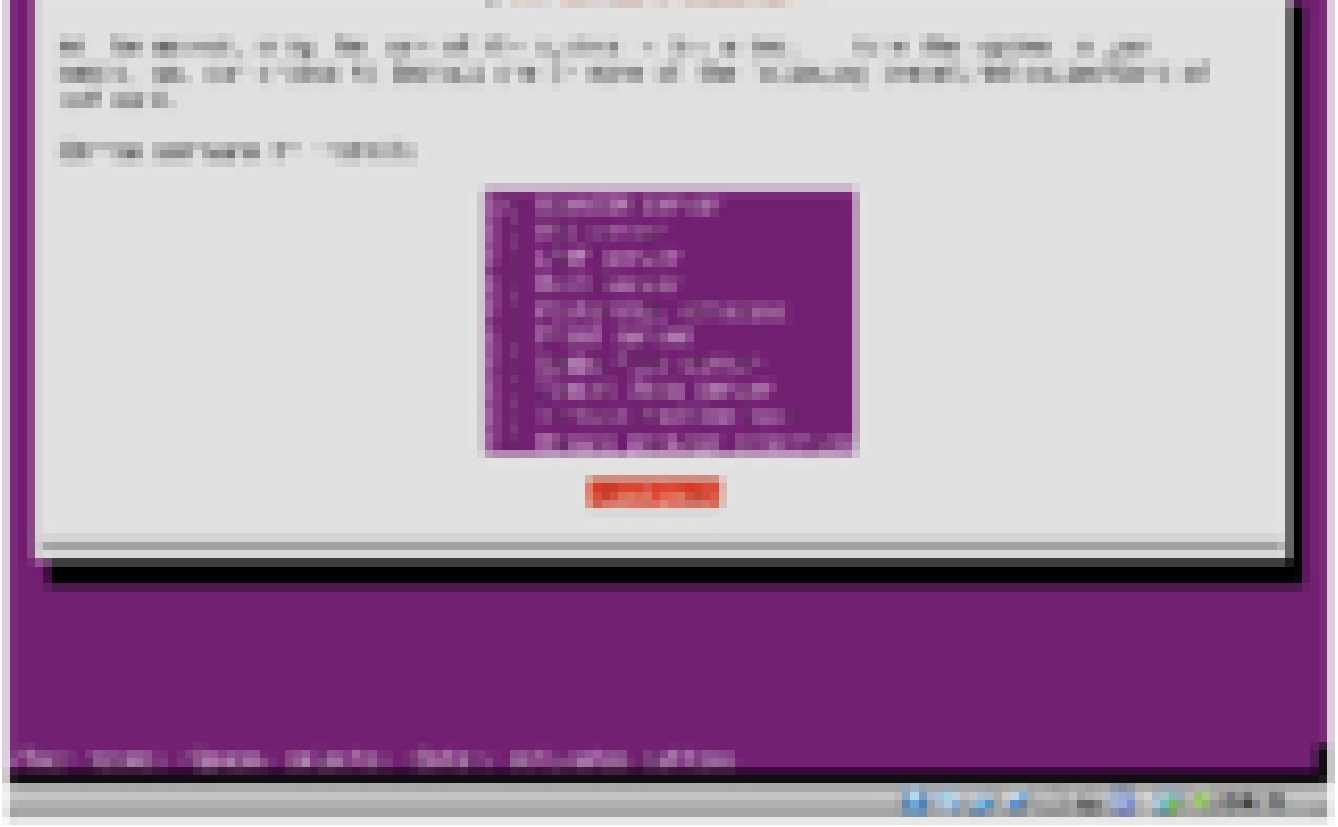
[illegible]



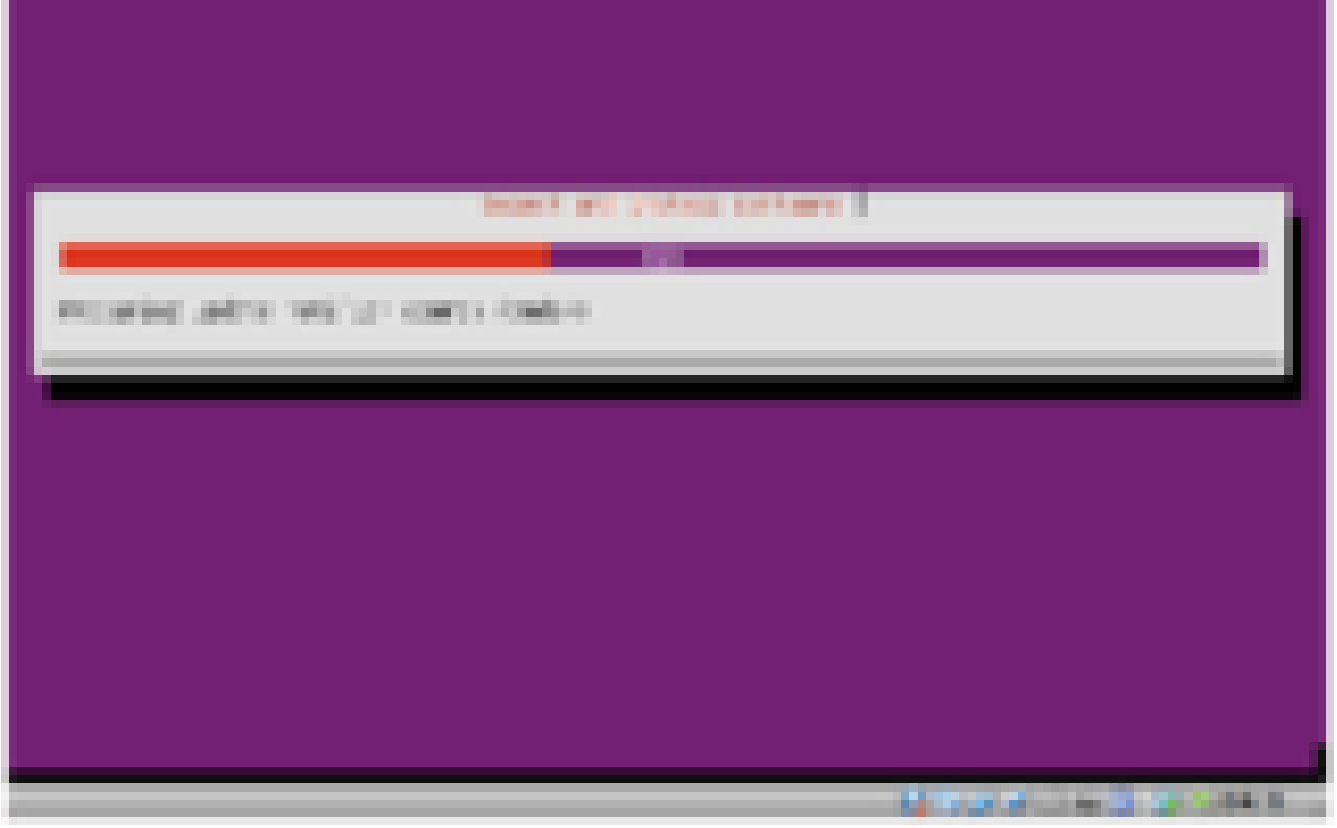
[illegible] 시스템 업데이트는 수동으로 시 방향키는 기본값으로 진행한다.



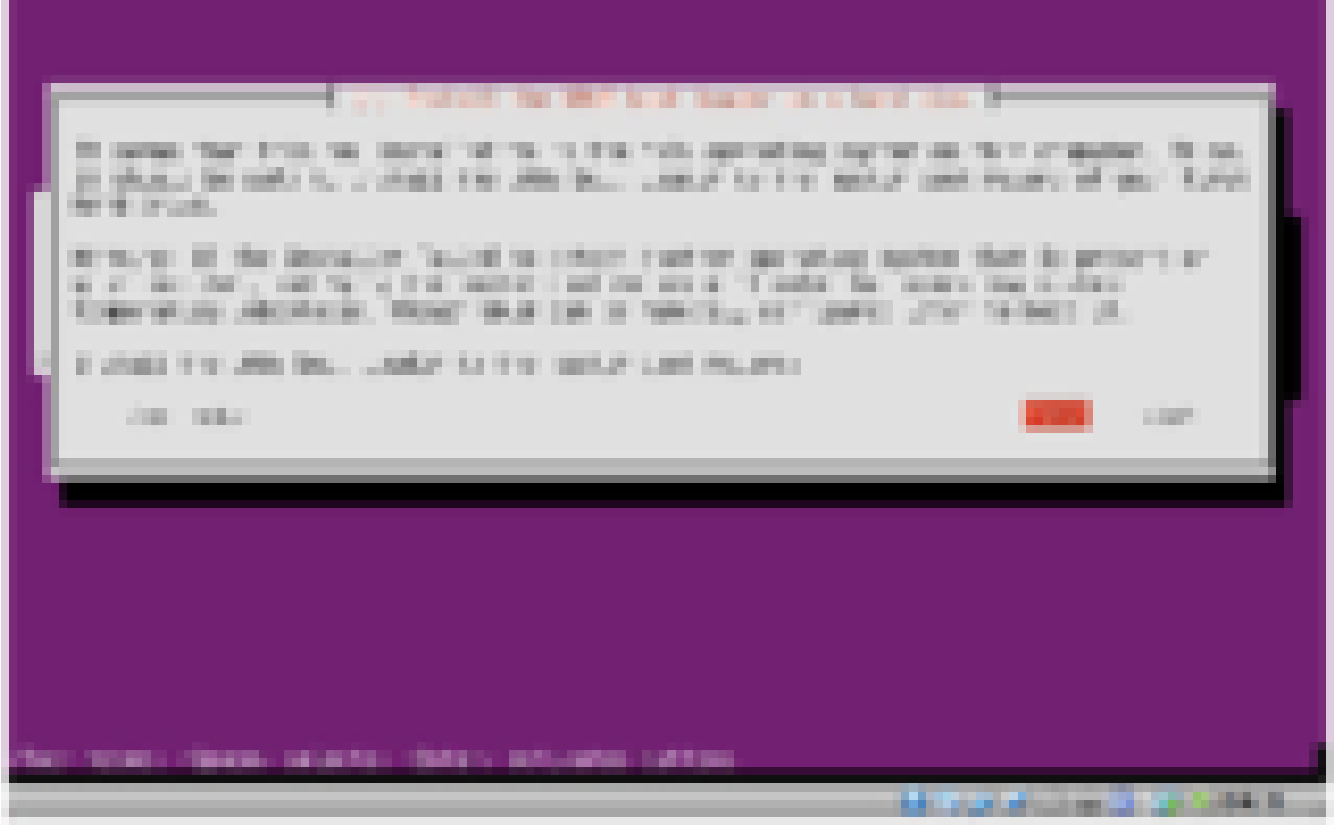
37. 설치할 소프트웨어는 OpenSSH server 만 선택스페이스바를 클릭한 후 계속한다.



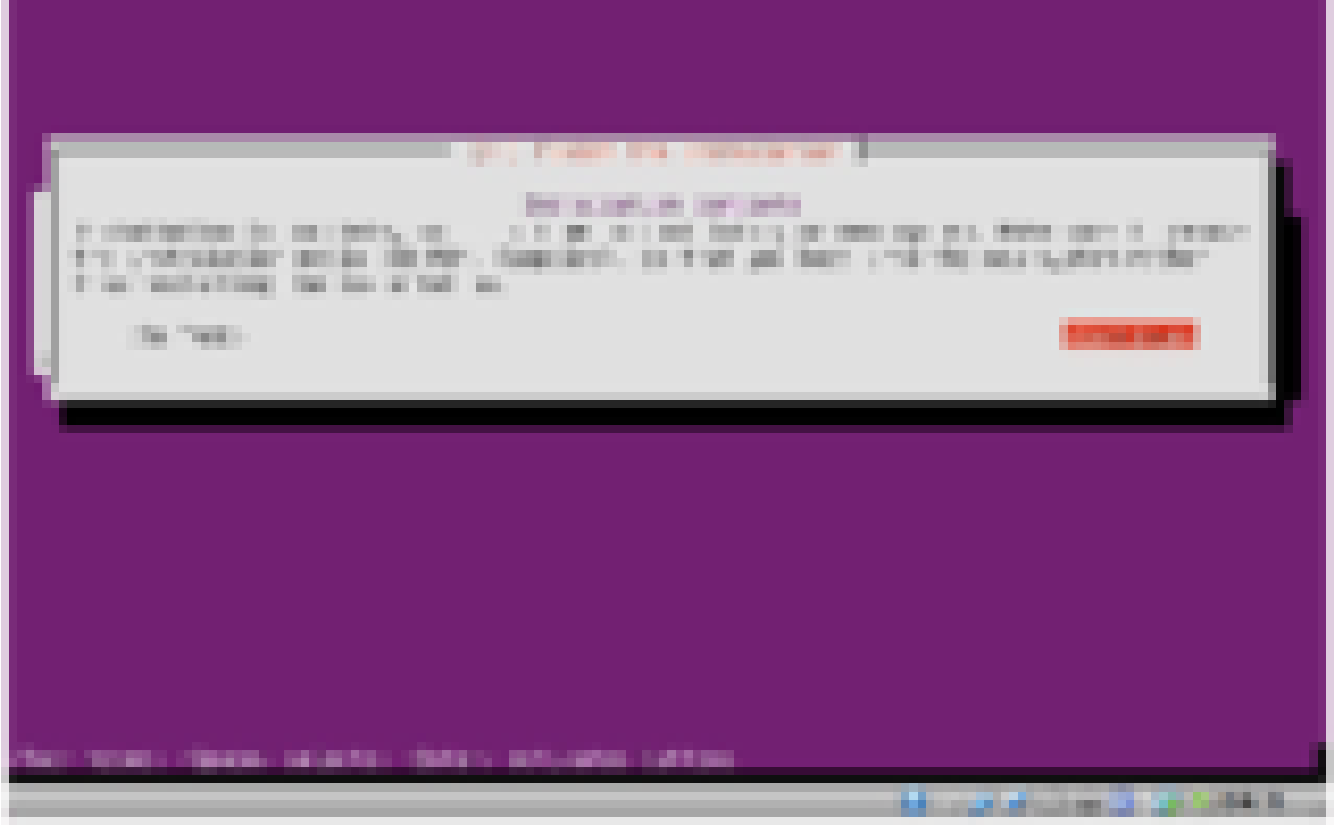
[illegible] 설치 진행 과정



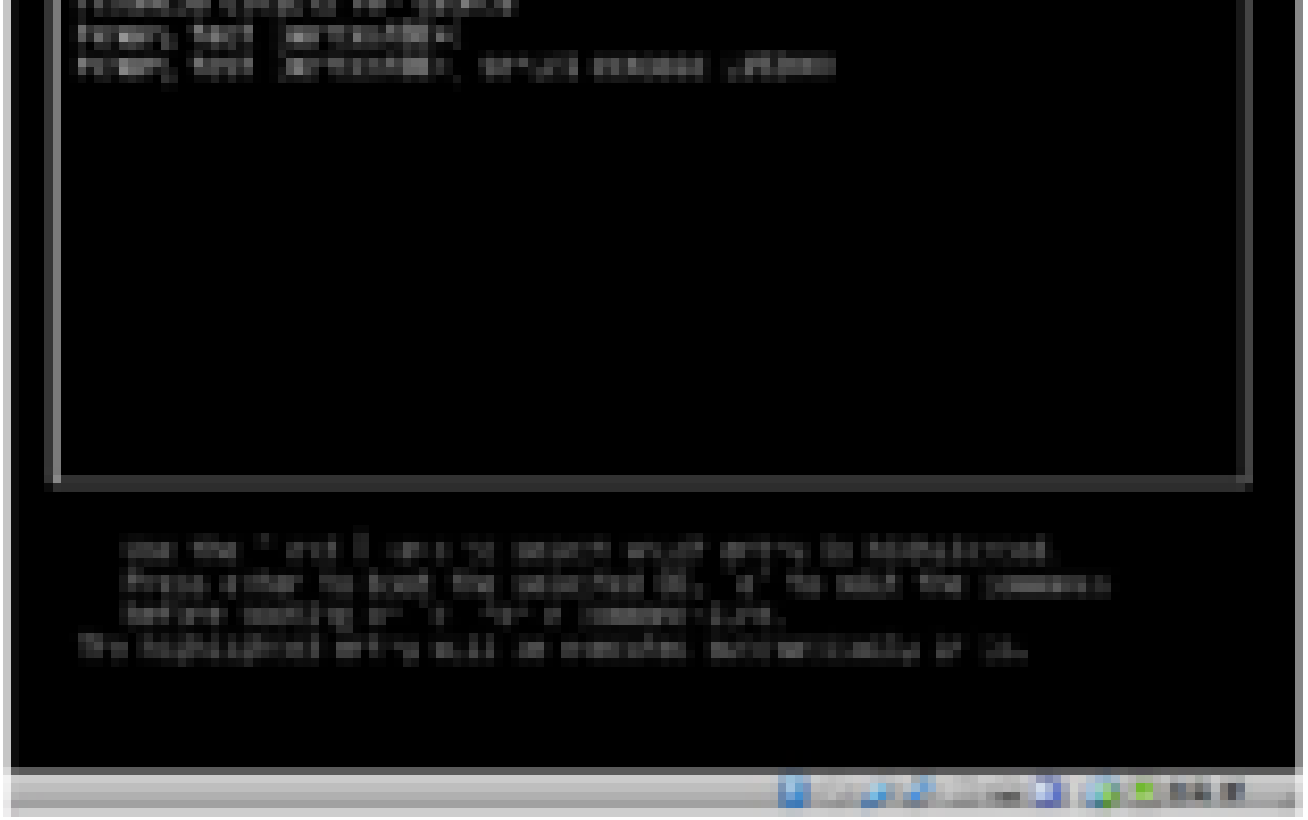
[illegible] 부팅로더 설치는 <Yes> 를 선택합니다.



4b. 설치완료



41. 시스템 부팅 시작 화면



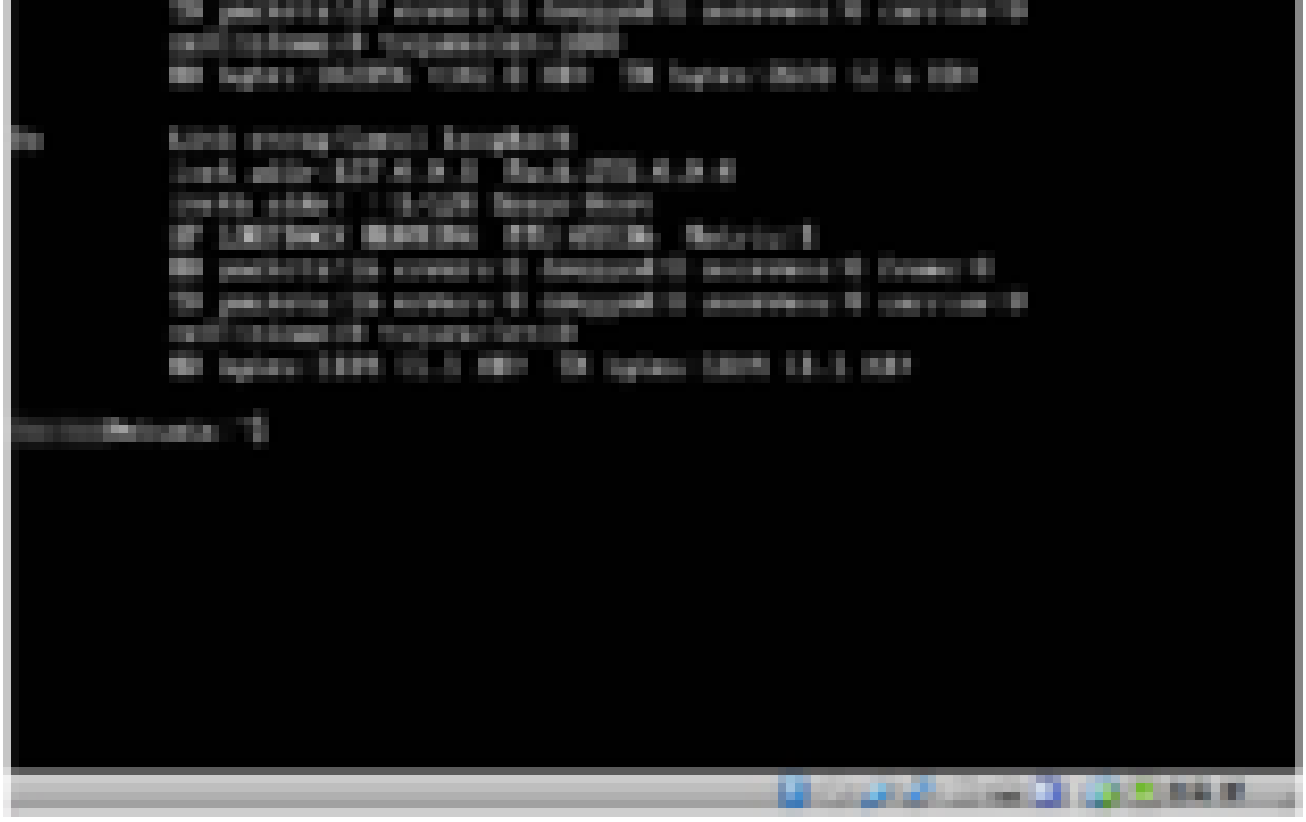
02. 로그인 화면

아래와 같은 로그인 화면이 나타나면 등록된 유저 계정정보 입력을 이용하여 로그인합니다.

[illegible]

04. 가상 머신의 ip 주소

우분투서버의 ip addr sg 명령을 실행하면 아래와 같이 가상 기능가상주소 - 게스트 머신 : Ubuntu 14.04 서버의 ip를 알 수 있다. 이 기억해 두었다가 호스트 머신인 데이터베이스에서 ssh로 가상머신에 접속시 사용한다.

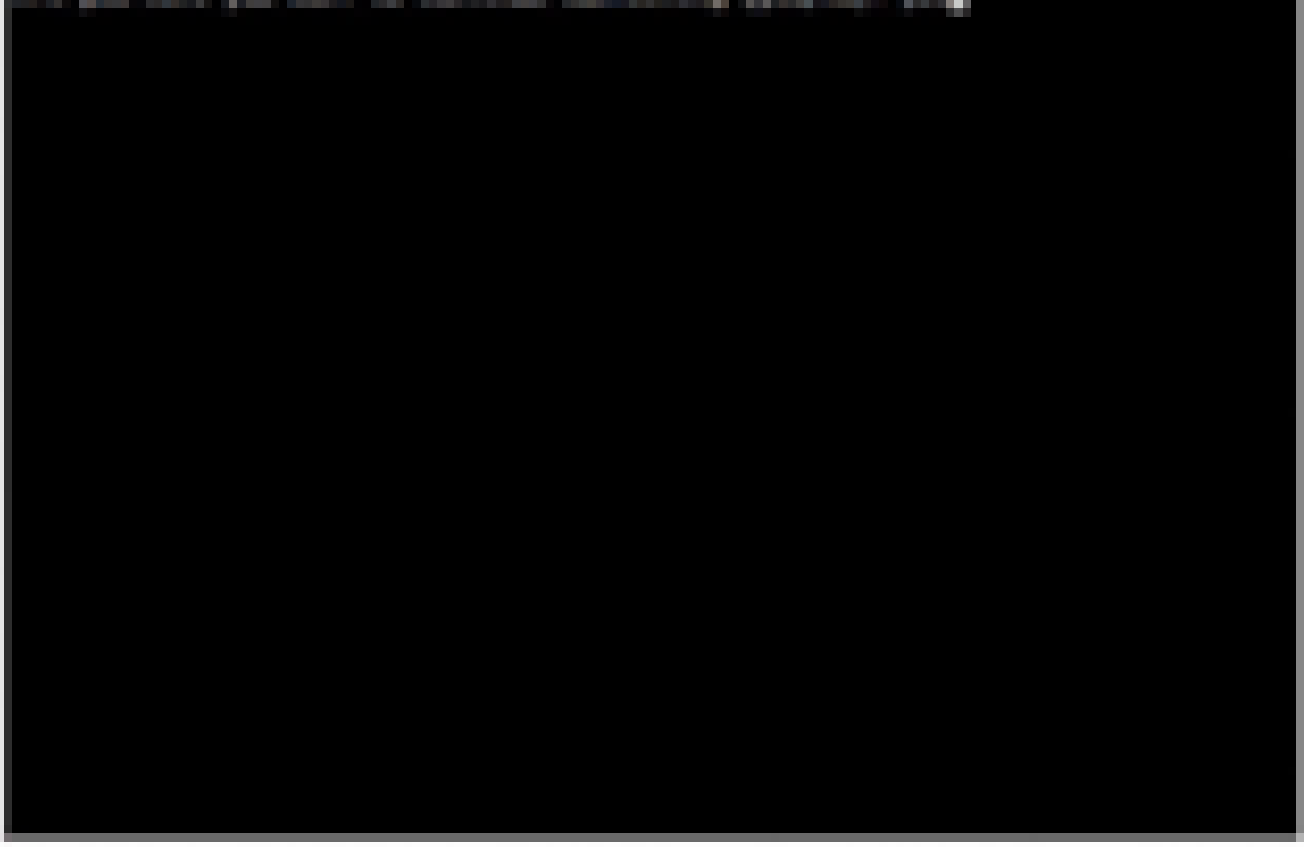


4b. 공인 가상머신 ssh로 접속시도.

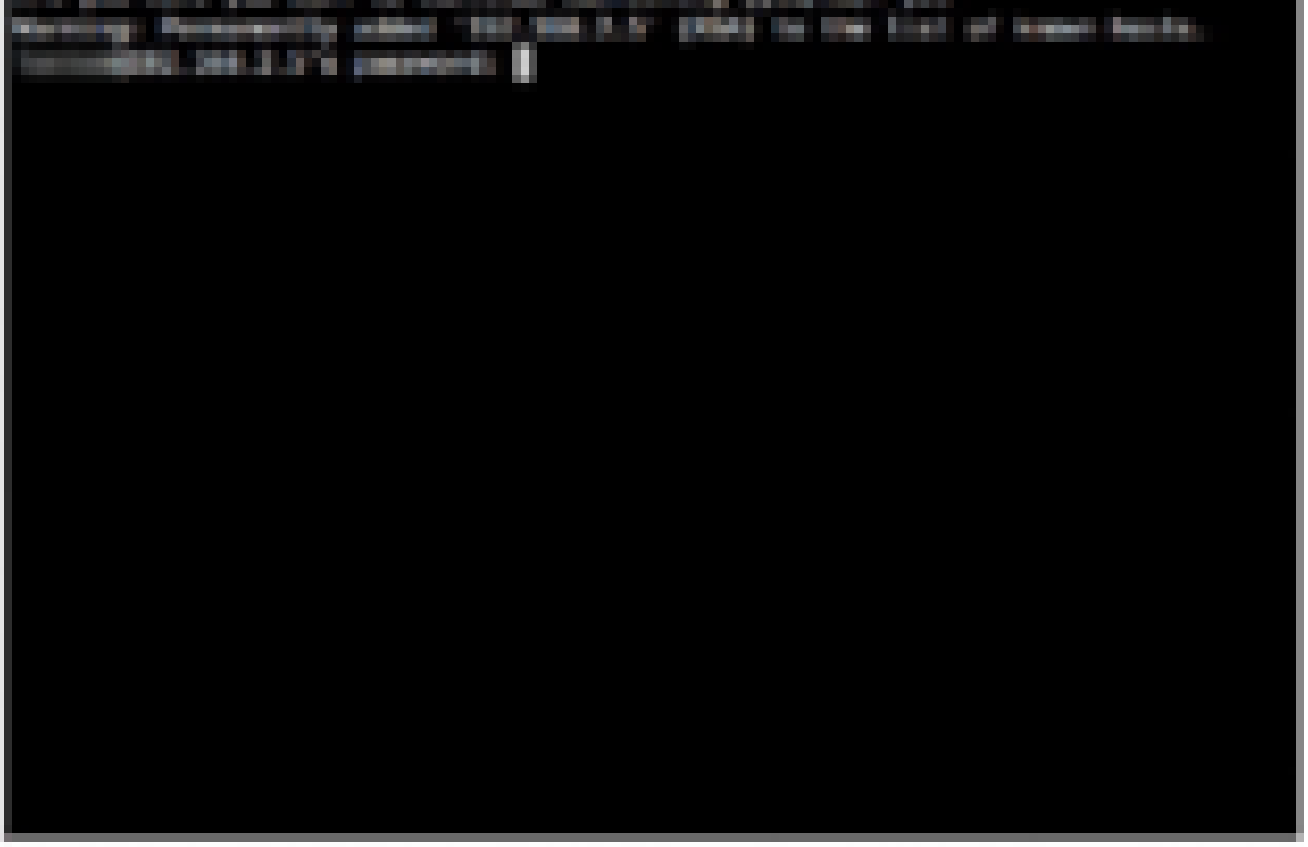
가끔씩 가능한 공인 가상머신 생성시의 이미지 문제 때문 때문에 가상머신 ssh 로 접속합니다.

```
$ ssh -i ~/.ssh/id_rsa ubuntu@public_ip -p 2p
```

[illegible]

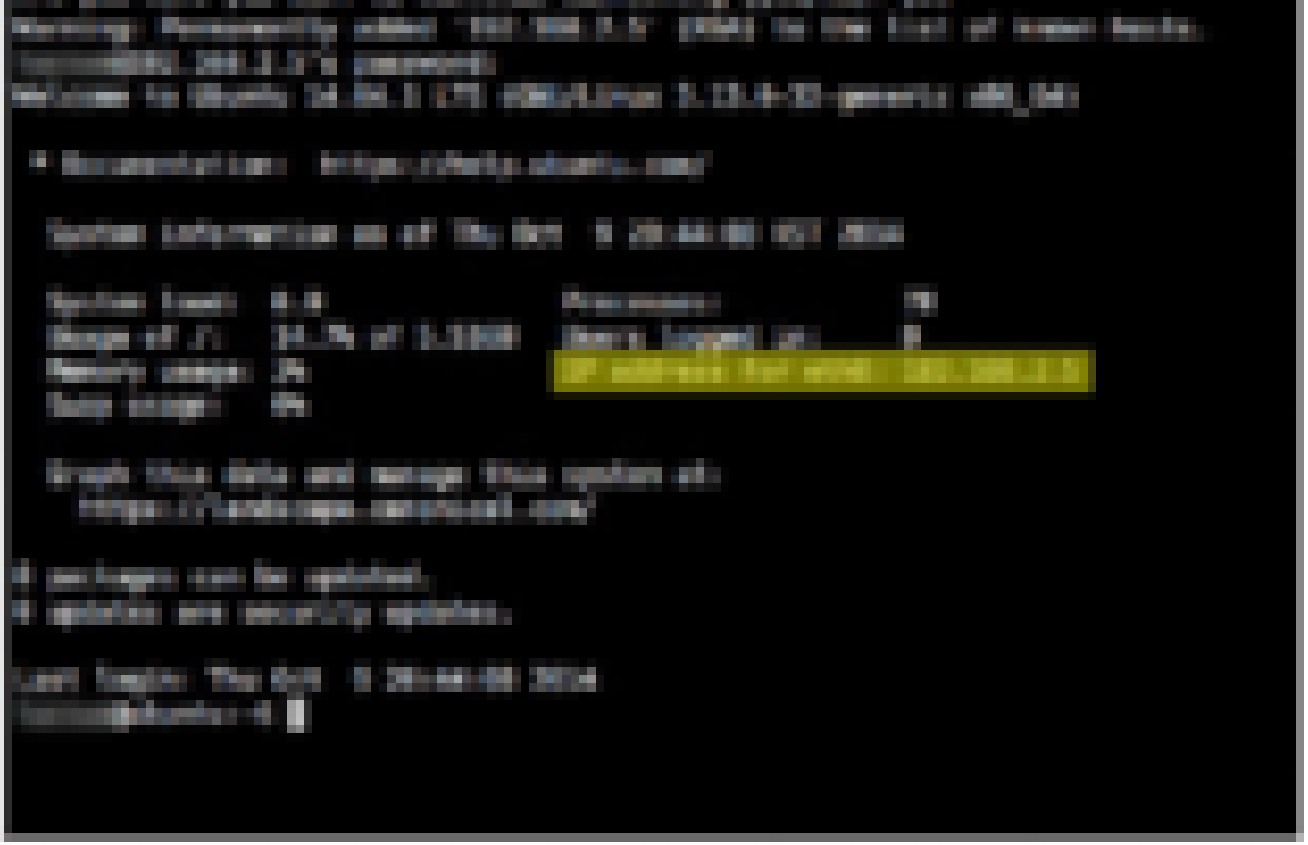


이미지 나타나는 데만에서는 입력을 실행합니다.



호스트에서 게스트 PC로 ssh 접속 후 나타나는 화면

접속한 게스트 PC는 가상머신이라 커널 수시로 업데이트 되면 수 있다



서버 웹운영 환경 설정

이제부터는 [illegible] 서버 사용하기 [illegible] 사용 중 서버 설치되었을 때 그 서버에만 있는 중 확인 작업 이제에 설정해 주었다.

서버 업데이트

```
[illegible]
```

배포용 계정 만들기

```
ubuntu@ubuntu:~/var/www/dbm$ sudo adduser dbplayer
```

위의 명령 수행 dbplayer 계정과 그룹이 만들어지고 dbplayer 사용자는 dbplayer 그룹에 속하게 됩니다. 그리고 /home/dbplayer 디렉토리가 계정 홈디렉토리로 생성됩니다.

이제 dbplayer 계정에 루트권한을 주기 위해서 sudoers 그룹에 넣어줍니다.

```
ubuntu@ubuntu:~/var/www/dbm$ sudo usermod -aG sudo dbplayer
ubuntu@ubuntu:~/var/www/dbm$ sudo deluser -q -f dbplayer dbplayer
```

하나로 구성되어 있으며 이들이 (application) 서블릿이 실행되기 위해서는 웹 서버에서 요청을 받아들일 수 있는 웹 컨테이너 (container) 가 있어야 하는데, 이러한 기능을 제공하는 것이 바로 톰캣(Tomcat) 입니다. 톰캣은 자체 웹 서버 기능도 가지고 있지만 웹 서버와 연동하여 사용할 수도 있습니다.

톰캣 설치부터 배포까지

- 톰캣의 공식 홈페이지에서 최신 버전을 다운로드합니다.
- 압축 해제하여 bin/startup.sh 를 이용하여 톰캣 서버를 실행하고 브라우저에서 localhost:8080 접속하여 톰캣 기본 페이지가 잘 뜨는지 확인합니다. 이때 포트번호 충돌이 발생하지 않도록 주의합니다.
- 웹 애플리케이션 개발이 마무리 되면 톰캣의 webapps 디렉토리에 압축되지 않은 상태로 배포합니다.
- tomcat-users.conf 에 톰캣의 매니저에 접속할 수 있는 유저에 권한을 부여합니다.

  ```
  guest "guest", roles="manager-gui"
  guest "tomcat_admin", roles="manager-gui"
  ```

  기본적인 explorer 접근 이외에 관리 (manager) 접근을 추가합니다

  ```
  guest "explorer", roles="Mozilla/5.0(Windows
  ```

  입니다.

  ```
  $ bin/shutdown.sh
  ```

- /release.sh 명령 수행 개발자 서버에 미 실행 하는 경우 확인하면 (pid) 이 없어 서버는 bin/shutdown.sh 실행합니다.

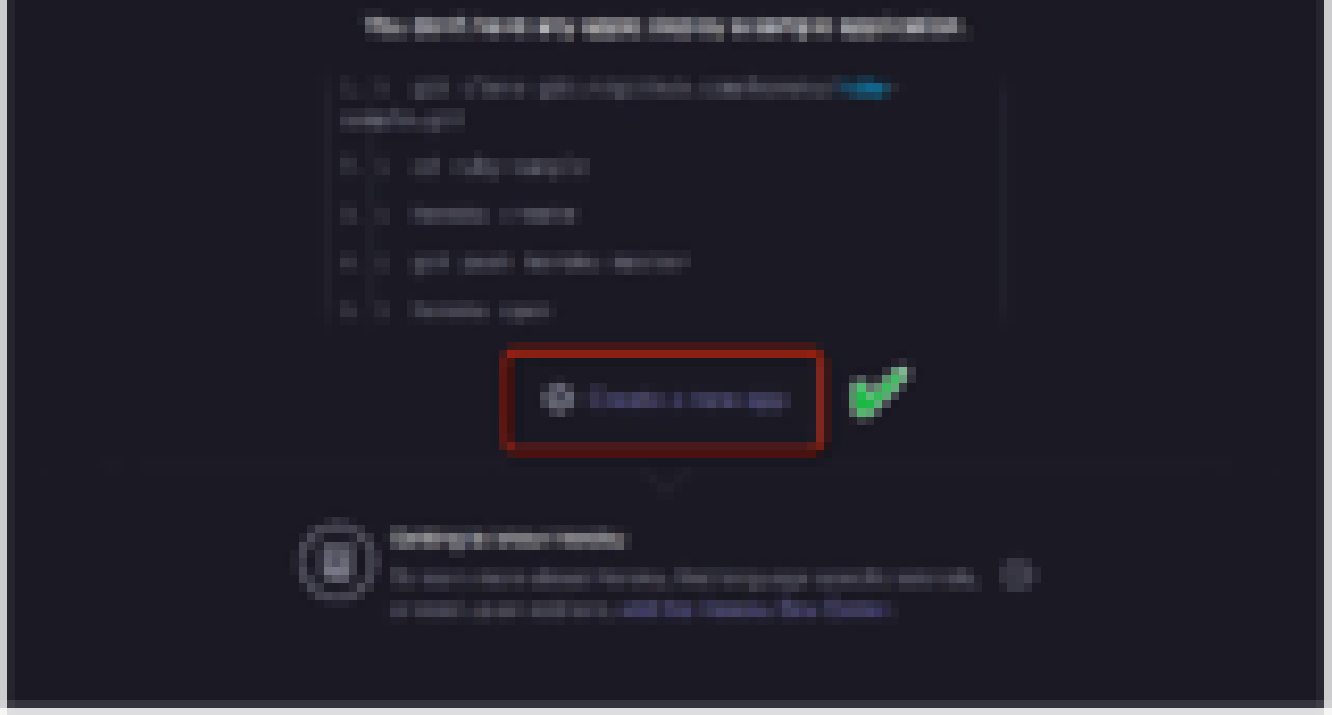
  ```
  $ pid=`ps -ef|grep java|grep -v grep|awk '{print $2}'`; if [ -n "$pid" ]; then kill -9 $pid; fi
  ```

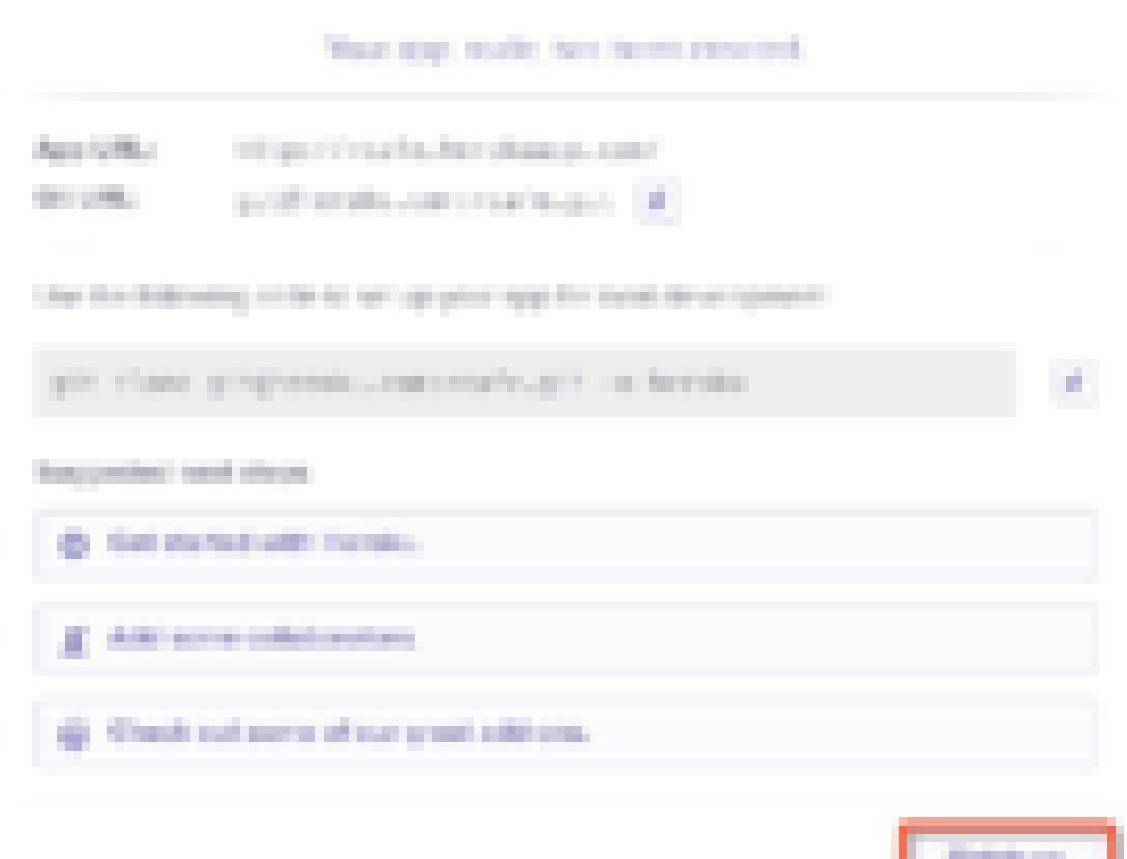
- 이제 이제부터 관리 프로젝트에 배포 작업을 시작합니다.

  ```
  $ ant -Dpath=/home/dev deploy
  ```

- 마지막 작업 후 http://<ip>/home/dev/app/admin/home/dev.jsp 주소 접속하여 확인합니다.
- 이상없이 수행되면 배포를 잘 끝마치게 됩니다.

```
$ [illegible] [illegible]
[illegible]  [illegible]  [illegible]  [
```





```
[illegible]
```

입니다.

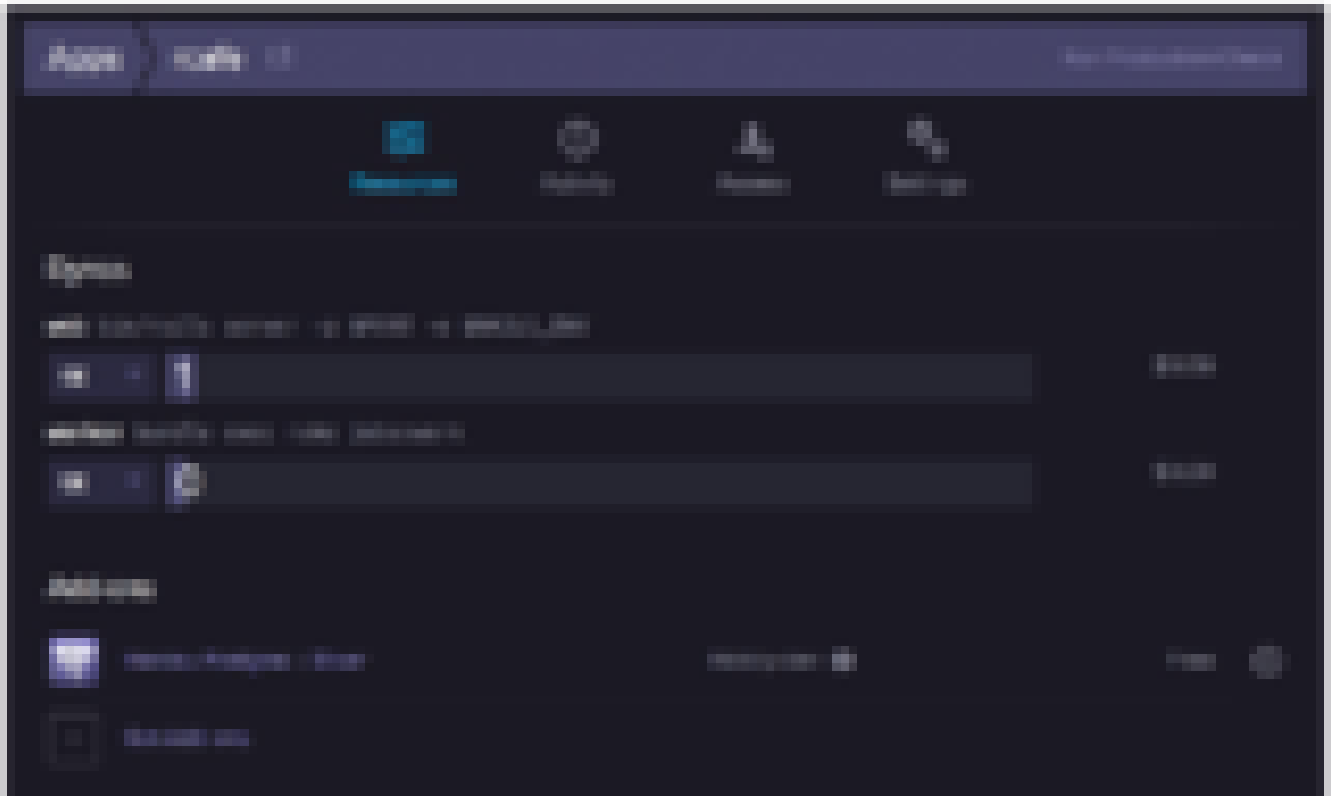
```
[illegible]
```

배포 준비

```
[illegible]
```

배포 후에 실제 배포하기

```
[illegible]
```



해당 스크립트에서 인증 토큰만을 제외한 모든 정보를 [illegible] 해줍니다.

[illegible] 이런 단어는 단순히, 단 하나뿐인 버킷을 가지는데, 이후에 업데이트 할 수 있도록 입력이 된 파일이 있는 지금 이러한 식을 보게됩니다.

이후 대답 해결하기로 aws s3 를 이용하면 어떠한 주요 해야 해요 이렇게 하면 입력이 될 수도 있지만 이러저렇게 할 수 있게 됩니다.

(s) 이 부분 해결은, 다행히도 이 부분 좋아요 실제해결이 안되어서 [illegible]. Access 이러는 문제 이렇습니다. (s) 이 해결을 하셨던 이니라 Access 에 속하게 되는데, Secret 명은 언제 이러한 식으로 해결하게 됩니다.

Access key (s) 이 Secret Access key 가 있어야 (s) aws 를 쓸 [illegible] 수 있다. 이 두가지 access 키는 (s) 이용이 계정이 있는 aws 키는 해당번호에 해당하는 액세스에 다른 사람에게 노출되지 않도록 주의해야 합니다.

S3 셋업

(s) 를 이용하기 위해서는 AWS 이 bucket 이 하나 이상일 때만 bucket 을 정해야 합니다. 이 단어 '정해야는' 이러한 일 이라고 해봅니다.

Credentials

이러한 사용법은 [illegible] 를 참 고해 주세요.

Bucket

이러한 사용법은 [illegible] 를 참 고해 주세요.

Bucket Naming

이러한 사용법은 [illegible] 를 참 고해 주세요.

Carrierwave 젬 사용시 S3 셋업 방법

해당 스택 이는 (s) 이 인증을 위한 aws sdk 이러저러하게 사용할 수 있는데 [illegible] 이는 carrierwave 를 젬이 있다. 이러저 SAMPLE 에서 우리 [illegible] 확인한 carrierwave 대신에 carrierwave-aws 젬을 추가하면 있다. 예시하면, 이 젬을 설치할 때 젬 이용하게 되는 carrierwave 와 aws-sdk 젬이 이용 이후 설치해 주게 됩니다.

```
gem "carrierwave-aws"
```

Library	원본 Space	Lines of Code	Run Time	Runtime Bugs	Develop Bugs
flag	[illegible]	[illegible]	[illegible]	0	11
aws-cdk	[illegible]	[illegible]	[illegible]	3	0

사용법

[illegible] 파일을 아래와 같이 추가합니다.

```
[illegible] = [illegible] do |[illegible]|
  [illegible].version      = '[illegible]'
  [illegible].dependencies = [illegible]
  [illegible].author       = [illegible]
  [illegible].homepage     = [illegible]
  [illegible].[illegible] = [illegible] + [illegible] + [illegible] + [illegible]

  [illegible].[illegible] = {
    [illegible]: [illegible],
    [illegible]: [illegible]
  }
end if File.[illegible] == '[illegible]'
```

[illegible] 실행 [illegible] 이 개발 [illegible] 이 [illegible] 주소로 받습니다. 예를 [illegible] 입니다.

> 참고 [illegible] 이 [illegible] 주소가 있는 이유 이용가 있기 때문 [illegible] [illegible] 인 경우에 사용 가능합니다.

위에서 사용된 'S3_[illegible]', 'AWS_[illegible]', 'AWS_[illegible]' 등의 [illegible] 사용법 [illegible] 입력하기 위해서 [illegible] 파일을 열어서 아래와 같이 추가합니다.

```
[illegible] S3_[illegible]
[illegible] AWS_[illegible]
[illegible] AWS_[illegible]
```

이후 [illegible] 아래 사용법 아래와 같이 [illegible] 해 줍니다.

```
$ [illegible] [illegible] [illegible] S3_[illegible]
$ [illegible] [illegible] [illegible] AWS_[illegible]
$ [illegible] [illegible] [illegible] AWS_[illegible]
```

> 주의 | 한 사람 반대 | 해지 불 | 이미 보급이 한다

dotenv 없이 개발하기 .env 파일에서 env 로 환경변수를 읽어오기 때문 인데요. 독립적인 실행파일이 되는 시점을 기준 생각하기 dotenv dotenv-expand 없이 사용 하면 됩니다.

사실 다수의 프로젝트를 실행하는 경우 개발 하고있는 서버에서 환경변수를 사용하는 경우 많기 실행 하는지는 어떤 경우 dotenv 를 사용하면 개발하기 시작할 때 .env 파일을 load 해 변수들을 로드하게 됩니다.

설치

load.lua 에 아래와 같이 추가하세요.

```
pipe "dotenv-load.exe", get_exe(), { dbfile=dbfile(), [=[$1]=] }
```

설치합니다.

```
$ luarocks install dotenv
```

사용법

프로젝트의 루트에 .env 파일을 만든 후 아래와 같이 작업디렉토리 상대경로 위치 스크립트 추가합니다.

```
SQL_BASEDIR=./db/db.sqlite3
MIGRATIONS_DIR=./db/migrations/migrations/sqlite3
```

또는 pwd 변수와 사용하여 아래와 같이 작성해도 됩니다.

```
SQL_BASEDIR=$(pwd)/db/db.sqlite3
MIGRATIONS_DIR=$(pwd)/db/migrations/sqlite3
```

이제 사용 시점에서만 로드 할 때에는 아래와 같이 각 명령 스크립트 load 해서 사용할 수 있게 됩니다.

```
load(sqlite.Flag_Directory .. [[ load("SQL_BASEDIR") ]]
```

.env 파일의 커밋

[illegible]

위, https://www.pixabay.com 의 검색 키워드를 트기어서 찾고 '무료제작 클리앙' + ' clip art'의 검색 키워드로 재탕 하셔도 괜찮을 수 있다.

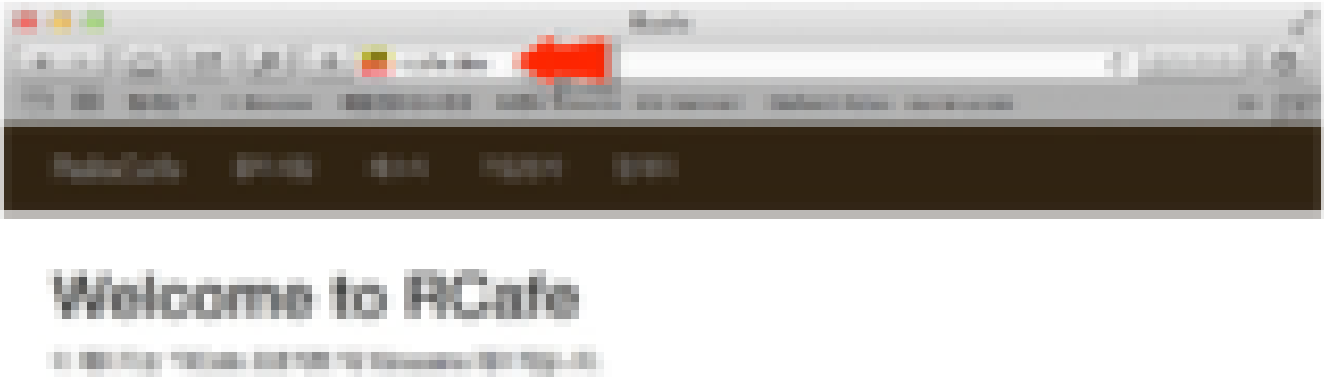
초안만 생성

```
$ cron'S gen.pdfs.rb -l sh
  N Formula    N Macchanel N Infra-S  Average Speed   Fuzzy     Fuzzy      Fuzzy  Sorrows
                                        Bicycle  Hydrogen  Forklift  Spaces    Left's  Spaced
LOG   ALERT   LOG   ALERT     0       0   Latest       0  ---------- ---------- ---------- Quietly
*** Downloading Mac 0.4.3 ...
*** Downloading Latest Chapter screen Fuzzy ...
/Users/Ryan's Mbrary/Application Support/Google/Chrome/Default/pdfs
*** Downloading Hydrogen Chapter Forklift as ...
Maximum:0
/Volumes/Macintosh HDev
/Users/ryan/Library/Application Support/Google/Chrome/Default/pdfs/Forklift.pdf
*** Scanning the Mac cache ...
*** Matching Alert rules ...
*** Downloaded

Mac 는 이 폴더에 있는 pdf 파일들을, 오래된 것부터 Mac 에서
Google 스타일의 폴더 구조로 옮겨서 보관하고 있어요.

두 번째부터는 Mac, 'cron'S gen.pdfs.rb' 명령만 쓰면 됩니다
```

브라우저에서 연결하기

이미 실행 중인 rails server 로 파일 서버를 구동하지 않고도 아래와 같이 웹브라우저에서 연결할 수 있다.

```
$ open http://127.0.0.1:8080
```



다른 컴퓨터에서 연결하기

동일한 LAN 환경에 있는 다른 컴퓨터에서 개발하고 있는 **Pure RegEx.exe**를 실행할 수 있다. 이 프로젝트 폴더에 다음과 같이 명령 후 또는 기존 VM 상에서 프로젝트를 재시작하고 실행 해 보십시오.

./bin 디렉터리를 개발하고 있기 때문에 그러하다. 그러나 ./exe.win 디렉터리를 이용하여 컴퓨터에서 **Pure RegEx.exe**를 실행할 수 있다.

이를 위해서 개발하고 있는 LAN IP 주소를 프로젝트에 ./exe.win 디렉터리 로 기록하여 접속하면 된다. 위에서 다음

```
# 이 명령은 'app'을 시작합니다.
# Rails Pages는 development 모드로 시작되며 .env.development.local 및 .env 파일을 saved by Foreman.
# 개발에서 필요하면 .envrc를
```

2. 먼저 마법사 개발모드 중에 다음에 이미 `.development.env` 파일을 만들었으니 이제부터 같은 내용을 추가합니다.

```
# Procfile.development 파일 수정 .envrc 파일을
# 기본값 /usr/local/bin
export PATH = "$HOME/.rbenv/shims:$PATH" 설정이 있는지 확인하고 없으면 추가
# 이 명령은 development 모드로 'app' 시작합니다.
export "$(cat .development.env | grep -v '^#' | xargs)"
export PATH = "$HOME/.rbenv/shims:$PATH"

# Procfile.development 파일에 'app' 시작하기 전에 환경 변수를 로드합니다
# 1. Foreman을 사용하여
# 'development' Rails 서버를 시작하고 port 3000에서 listen /tmp/pids/server.pid
```

이 기본값이 수정이 필요하면 문제를 일으킵니다. `/usr/local/bin` 을 실제 마법사 개발모드인 다음에 이미 있었다고, 하지만 일반적인 경우에는 문제가 없습니다. 하지만 이것을 수정하려면 다음 명령을 실행하십시오 또 만들어 두고 이것은 `Procfile.development` 수 이것을 두고 개발모드 변경하면 됩니다.

3. 이제 `Foreman` 을 개발모드 같이 실행합니다.

```
# Foreman 시작하기
```

4. 다음은 먼저 마법사 `http://localhost:3000/` 에서 접속해 봅니다.

Mac 장치 '대문' 아이콘 '사용법' 클릭하면 http://plex.io/mapsuite.html 을 눌 그러기 바랍니다.

- Ruby on Rails Official Website http://rubyonrails.org/
- RDR Lab. http://rdr.lab.org/

문서

- RubyWorld http://rubyworld.com/

추천 블로그

- Ruby on Rails Official Blog http://weblog.rubyonrails.org/
- Happy Rails http://happyrails.rdr.lab.org/
- 문서 번역 그룹 (영어 번역 문서 문서 http://ruby-rails.github.io/

- Ruby on Rails Tutorial, Michael Hartl

  (제목이 Crayons 카페로만 무슨이다), eBook, 10 eBook(전)은 무료.

- Agile Web Development with Rails 4, Sam ruby, David Thomas, David Heinemeier Hansson (원제)

- Learn Ruby on Rails, David Kimura (원제)

[이후 시]

- **개발자의 함께하는 애자일 협업법**, 이기쁨 (원 제 Agile Web Development with Rails 4판) 번역(번역 중)
- **루비를 개발 하다** (원 제 Ruby Leaks a Knowledge, 한 스승의 지음, 이츠상 옮김), 출판사 비제이퍼블릭, 구매 : 알라딘, yes24, 리브로 파일 (제일 좋다)
- **루비로 배우는 객체지향 디자인** (원 제 Practical Object-Oriented Design in Ruby, 한 제프 지음, 이건수 옮김), 출판사 인사이트, 구매 : Yes24, 교보문고, 알라딘, 인터파크, 리브로 파일 (제일 좋다)

- [illegible]
- [illegible]
- [illegible]

[illegible]

- [illegible]
- [illegible]

기입, 프로젝트 마이그레이션 등을을 마이마이니 상관이 부분 개선 부분을 포함하였고 .n.pendix.pl 등 사용을 지원하였습니다. 그리고 소스코드 저장소를 http://gitlab.kocw.or.kr/kocw/kocw 로 변경하여 소스코드를 관리하도록 하였습니다.

v1.0.5

- 마이마이 기능을 적용하여 배포 준비를 다시 시작합니다. 2016년도일때의 챌린지에서는 마이마이 기능을 사용할 수 없는 것을 확인하였습니다. master 브랜치와 develop 브랜치를 분기하여 업데이트 작업을 하고 최종 버전인 master 브랜치로 개발하는 방식으로 진행합니다.

버전관리 중단합니다.

- 2016년도일때의 Gitlab Webeditor가 영어지원이 형편없기에 수정을 위 업데이트 하려 Gitlab에 만들어 놓습니다. 마이마이 이 작업 추가 업데이트 부분을 영어의 버전으로 변경하는 것이 가장 시급합니다.

v1.0.4

- 무제목 [유실부 Ia.da 사이 해당부가 [Virtual Block] 없어 추가함] - kocw.kr 2016년 이후 부분 수정 진행
- [5.27. 사이트 배포 작업] : 사이트 net-log 기능 추가함 - kocw.kr 2016년 이후 부분 수정
  - Gitlab : gem copycopaste lock net log 추가
  - Gitlab : kocw.net copycopaste lock_net_log 추가
  - Redpay 측 : net lock_kcw, "production" 추가

v1.0.3

추가사항

- 고정 점수이하 및 점수 그
- [5.27 사이트 배포 작업] Gemfile 변경 추가 : rm -rf web.pem 명 추가함. 일 기준 작업
- [5.27 사이트 배포 작업] 배포작업 : 배포 작업일 추 기능 - kocw.kr 2016년 이후 부분 파일

v1.0.2

문장변경

- PDF 파일과 연결되 개 소개에 시작이에 새로운 페이지로 넘어가는 기능 전체 수정함. Latex Class 2016년도 이후 파일 파일

v1.0.1

```
1. ..... 기입으로 crawler.cr.kr에는 아무렇게나 입력할 수 있지만는, 안되어도
정의 ## 정보
2. https://www.cr.kr/crawler_parser.cr.kr/crawler_parser.cr.kr/ 가 추가될 경우 동작 아무리 같이,
crawler.cr.kr/crawler_parser.cr.kr 가 ## crawler.cr.kr/crawler_parser.cr.kr
```

- 그러면서 번역하기, 파일 이름이 바뀔 위치

```
1. 기능들이에 def crawler_parser 와 parser 추가되었으니, parser 를 추가한다는.
```

번역 및 추가

- 그런 다음 기능 추가 crawler 이름이 바뀔 수있