2021 Winter B B CON

대학생 프로젝트의 상상과 현실..

아잉

발표자 와 내용

```
# 백준푸는방 닉네임
**아잉**
```

```
# 발표 내용
1. 또래 대학생과 함께한 웹 어플리케이션 개발기

2. 쓴 기술과 아키텍쳐 설명(간단쓰)

3. 협업을 위한 github 잘쓰기(대학생 대상)

4. 대학생 팀 프로젝트의 장단과 소감
```

프로젝트 시작부터 끝까지

1. 우리학교 학생 개발자끼리 모이는 모임 존재
2. 그 모임에서 팀을 꾸려서 프로젝트 진행함
3. 팀에 학교 디자인과 재학중인 디자이너도 있어서 디자인도 있음
4. 웹 서비스 개발 도전하면서 제출할 공모전도 찾아봄
5. 공개SW개발자대회에 출품해서 장려상 받음
6. 결론: 온몸비틀기해서 어떻게 상은 받음

서비스 배포 url.

https://upgle.hisfolio.com

소셜로그인(카카오, 구글) 지원하니 간단하게 회원가입하셔서 구경하세요.

대신 모바일 지원은 미흡하니 컴/태블릿으로 보세요.

https://github.com/Jandy-SeoulTech/Jandy_Web_Back
https://github.com/Jandy-SeoulTech/Jandy_Web_Front

github 주소.

개발 팀 구성

PM 겸 BackEnd 개발 : 나

디자이너 1명

백엔드 개발자 2명

프론트엔드 개발자 2명

- PM : Product Manager의 준말. 서비스 전체를 기획하고 조율하는 기획자의 역할이라고 생각해주세요.

왜 PM을 하겠다고 했을까

1. 초기 프로젝트 진행이 매우 우유부단하고 지지부진했음.
2. 그때 상황
    - 개발자 5명/디자이너 1명이서 다같이 기획짜기
    - 다들 의견안냄
    - 다같이 안친해서 분위기 썰렁탕
    - 다같이 온라인 회의 2시간동안 하는데 아무것도 결정못함
    - 개답답해서 내가 기획자할테니 따라오라고 하였음

어이 그 앞은 [HELL]이다.

결론만 말하자면 개발자는 개발에 집중하는게 좋다고 느꼈습니다.

기획, 디자인, 그외 잡다한 거를 개발자가 하는 것은 조또 쓸모없습니다.
특히 할 줄 아는것 없는 주니어가 저런데 눈 돌리면 개발을 못해요.

디자이너가 UX나 비즈니스적으로 의미있는 서비스를 할려고 그래서 나도 같이 고민하고 서비스 설계하느라 개발 참여도↓↓↓↓↓↓

기획 다 끝내고 개발 참여하려니 내가 하고 싶었던 설계나 기술같은건 참여못하고 디버깅만 했음.

프로젝트에 쓰인 기술

1. 웹 프론트엔드

- React, mui, emotion -
- react-router, redux, redux-saga
- axios, socket.io

HTML, CSS, JS에 대한 기본지식을 알고 있다고 생각하고 설명합니다.

웹 프론트엔드

1. 다들 아는 React

- 자바스크립트 UI 라이브러리입니다.
- 뷰와 상태에 대해 스스로 책임지는 캡슐화된 컴포넌트를 기반으로 뷰를 짜게 해줍니다.
- 상태(state)에 대해 뷰를 선언형으로 작성하기 때문에 개발이 쉬워짐

2. Mui

- 이쁜 UI를 백지부터 뚝딱 만들기는 힘듭니다.
- 그런 당신을 위해 이쁜 React 기반 UI 라이브러리, MUI가 있습니다.
- 웬만한 것들이 다 있기 때문에 MUI기반으로 UI를 짜면 제로부터 안해 됩니다.
- 최신 버전부터 CSS 커스터마이징도 지원함.

3. Emotion

- CSS in JS 라이브러리
- CSS in JS 란 무엇인가?
  + 기존의 css는 따로 파일을 만들어 링크를 걸거나 html안에 집어넣거나였음.
  + 네임스페이스가 겹치는 문제가 있기 때문에 다양한 css 전처리기가 등장함
  + 근데 이제 React를 쓰면 js안에다 html을 쓰네??
  + css도 js안으로 넣어서 관리해보자~
  + 장단이 있는데 그건 직접 찾아보세요. 나도 몰라용.

- MUI를 CSS 커스터마이징 하기 위해 선택했음
- 뷰에 관련된 모든 것을 모두 한 js 파일에서 작성하기 때문에 좀 더 편했습니다.

React + MUI + emotion 예시

```jsx
import { css } from '@emotion/react';
import { TextareaAutosize, Typography } from '@material-ui/core';
import { useState } from 'react';

const TextArea = ({ onChange, maxLength, inputRef, ...props }) => {
  const [lengthError, setLengthError] = useState();

  const handleCahnge = (e) => {
    if (e.target.value.length > maxLength) {
      setLengthError(true);
      return;
    }
    setLengthError(false);
    if (onChange) onChange(e);
  };

  return (
    <>
      <TextareaAutosize
        placeholder={maxLength && `${maxLength}자 이내로 작성해주세요.`}
        onChange={maxLength ? handleCahnge : onChange}
        ref={inputRef}
        {...props}
        css={[defaultStyle(lengthError)]}
      />
      {lengthError && (
        <Typography sx={{ color: 'red', fontSize: '0.75rem' }}>
          길이 제한을 초과했습니다.({maxLength}자 이내)
        </Typography>
      )}
    </>
  );
};

const defaultStyle = (lengthError) => css`
  font-size: 0.875rem;
  font-family: 'Barlow', 'Noto Sans KR';
  font-weight: 500;
  padding: 0.9375rem 0.75rem;
  resize: none;
  outline: ${lengthError && '1px solid red'};
  &:focus-visible {
    outline: ${lengthError ? '2px solid red !important;' : '2px solid black !important;'};
  }
`;
```

4. Redux

- React에서는 컴포넌트 사이의 데이터를 주고받으면서 컴포넌트 간의 상호작용을 합니다.
- 데이터 흐름은 위에서 아래로 흐르게 되어있습니다.
- 즉 아래 컴포넌트는 위 컴포넌트로 데이터를 못 보내고, 받기만 가능합니다.
- 컴포넌트의 트리 구조가 데이터의 폿포라고 생각하면 데이터는 아래로만 진행합니다.

- 그럼 컴포넌트의 트리구조 상 멀리 떨어져있는 두 컴포넌트가 동일한 데이터를 사용해야 한다면?
- 이땐 두 컴포넌트의 최소 공통 조상 컴포넌트에 필요한 데이터를 넣어야합니다.
- 두 컴포넌트에 데이터를 전달해주기 위해 중간에 끼인 컴포넌트들은 데이터를 계속 전달
- 자기가 안쓰는 데이터를 전달해야하는 중간자 컴포넌트: 자기가 할일이 아닌걸 함
- 이외에도 여러 안좋은 점 때문에 전역 데이터 저장소의 필요성이 조금은 있습니다.

- 그런 당신을 위한 Redux
- 하지만 redux를 쓴다는 것은 데이터가 거치는 단계가 더 늘어나고, 코드도 길어집니다.
- 쓸지 안쓸지 잘 고민해보아야 합니다.

5. 그외 잡것

+ redux-saga : redux와 비동기 통신을 같이 할때 쓰는 건데, 코드 존나게 쳐야댐. 이거보다 좋은 대안이 있을것입니다.(react query라던가 graphql이라던가)

+ react-router : 웹 서버로서의 react를 활용할때, 경로(라우트)에 따라 다른 페이지를 보여주어야 겠죠? 접속 경로와 보여줄 컴포넌트를 매칭시켜주는 라이브러리.

+ axios : 자바스크립트 http 요청 라이브러리. 사용하기 쉬운 인터페이스로 http 요청을 보내게 해줌. 근데 fetch api나 딴 거 써도 됩니다.

+ socket.io : 채팅 구현을 위해 쓴 웹 소켓 라이브러리. 더이상의 자세한 설명은 생략.

프로젝트에 쓰인 기술

2. 웹 백엔드

- Express
- passport, prisma, mysql
- socket.io

HTTP 통신과 서버, DB에 대해 이해가 있다고 가정합니다.

1. Express

- Node.js를 위한 빠르고 개방적인 간결한 웹 프레임워크 - expressjs.com
- 서버를 쉽게 구성하게 해주는 웹 프레임워크입니다.
- 가벼운 프레임워크라서 DB, 인증, 뷰, 테스트, 기타 등등에 정해진 게 없음.
- Express는 http 통신과 라우팅, http 본문 처리 등등을 담당함.

- 자유롭다는 말은 다른 말로 체계가 없다는 말입니다.
- 이것저것 실험하고 자유롭게 해보기엔 좋지만, 체계가 잡힌 프레임워크를 쓰며 얻는 이점도 많으니 고려해보세요. (node 생태계엔 nestjs가 있음)

2. passport

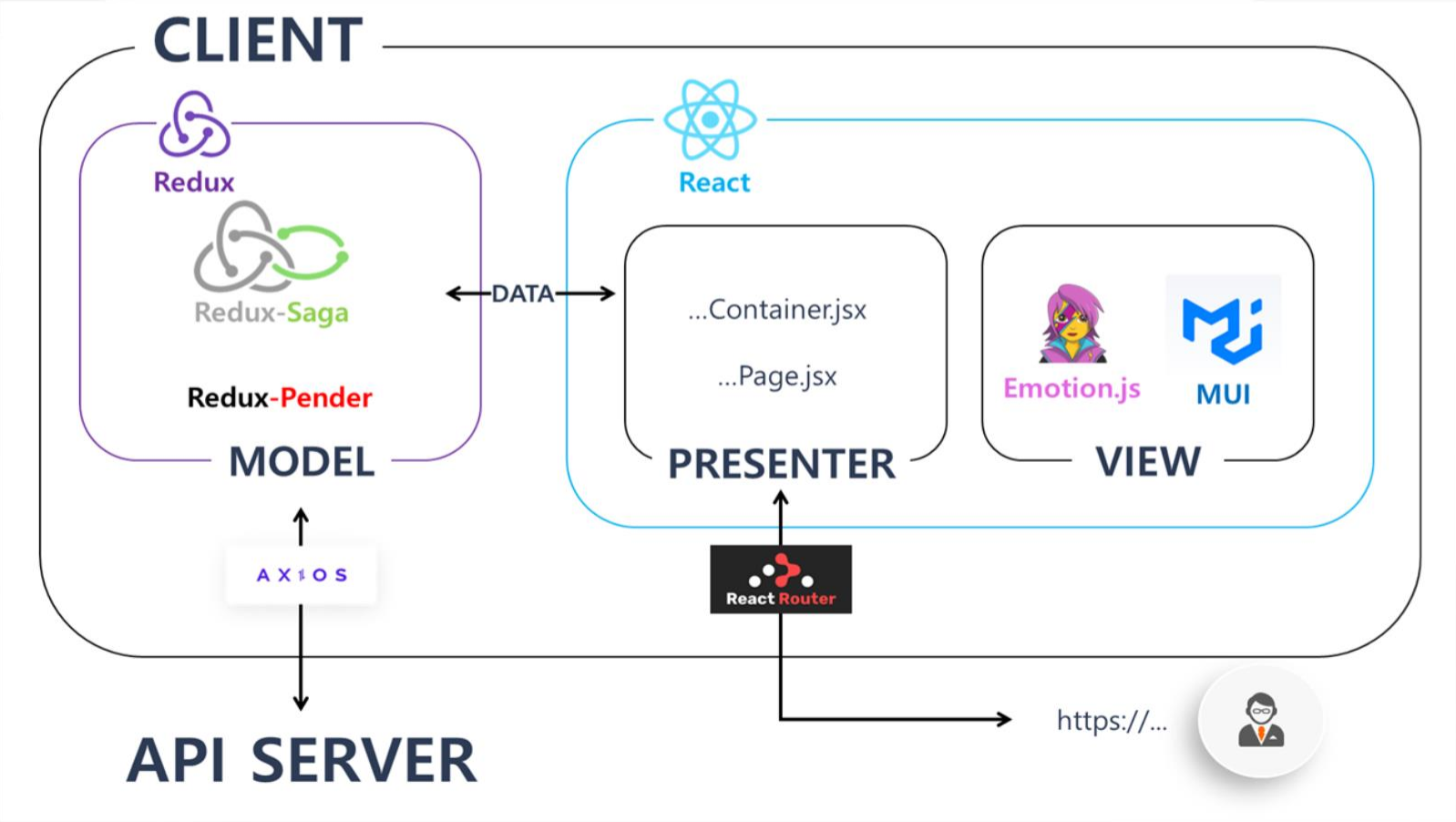
- 인증을 위한 라이브러리
- passport를 통하면 여러 다른 인증 로직을 쉽게 처리할 수 있지만, passport에 코드가 조금 종속될 수 있음.
- 인증이란?
  + 사용자들의 리소스에 대한 접근 권한을 설정하기 위한 체계
  + 사용자들은 인증 정보를 가지고 리소스를 요청하고, 서버는 인증 정보를 검증해서 리소스를 줄지 말지 결정함
  + 이때 웹 서버와 사용자에 대입해보면 사용자는 인증 정보를 가지고 웹 서버에 어떤 행동을 요청하고, 웹 서버는 인증 정보를 검증하여 그 행동을 하거나 하지않음

3. DB

- Mysql(관계형 데이터베이스)를 DB로 씁니다.
- ORM(객체-관계 매퍼)로 prisma orm을 씁니다.
- ORM이란?
  + Object(객체)와 Relation(관계)는 서로 1대1로 정확히 들어맞지 않습니다.
  + 예시를 들자면 객체는 일반적으로 다른 객체를 포함할수 있고 별다른 처리 없이 바로 사용할수 있습니다.
  + 관계는 다른 관계를 외래 키 형태로 포함하고, 해당 키로 다시 검색하여야 관계를 가져올 수 있습니다.
  + 이렇게 객체와 관계가 완벽히 들어맞지 않는 것을 객체-관계 부정합이라고 합니다.
  + 객체-관계 부정합을 교정해주고 어떤 관계를 그에 대응하는 객체로 변환해주는 소프트웨어가 객체-관계 Mapper입니다.
  + 객체-관계 부정합에 대해선 알아서 공부해보세요. 저도 몰라용ㅎ
- prisma는 사용이 어려운 라이브러리는 아니었는데, 다른 ORM을 써보지 않아서 다른 ORM과 비교해 어떤 점이 좋은지 판단이 안서네요.

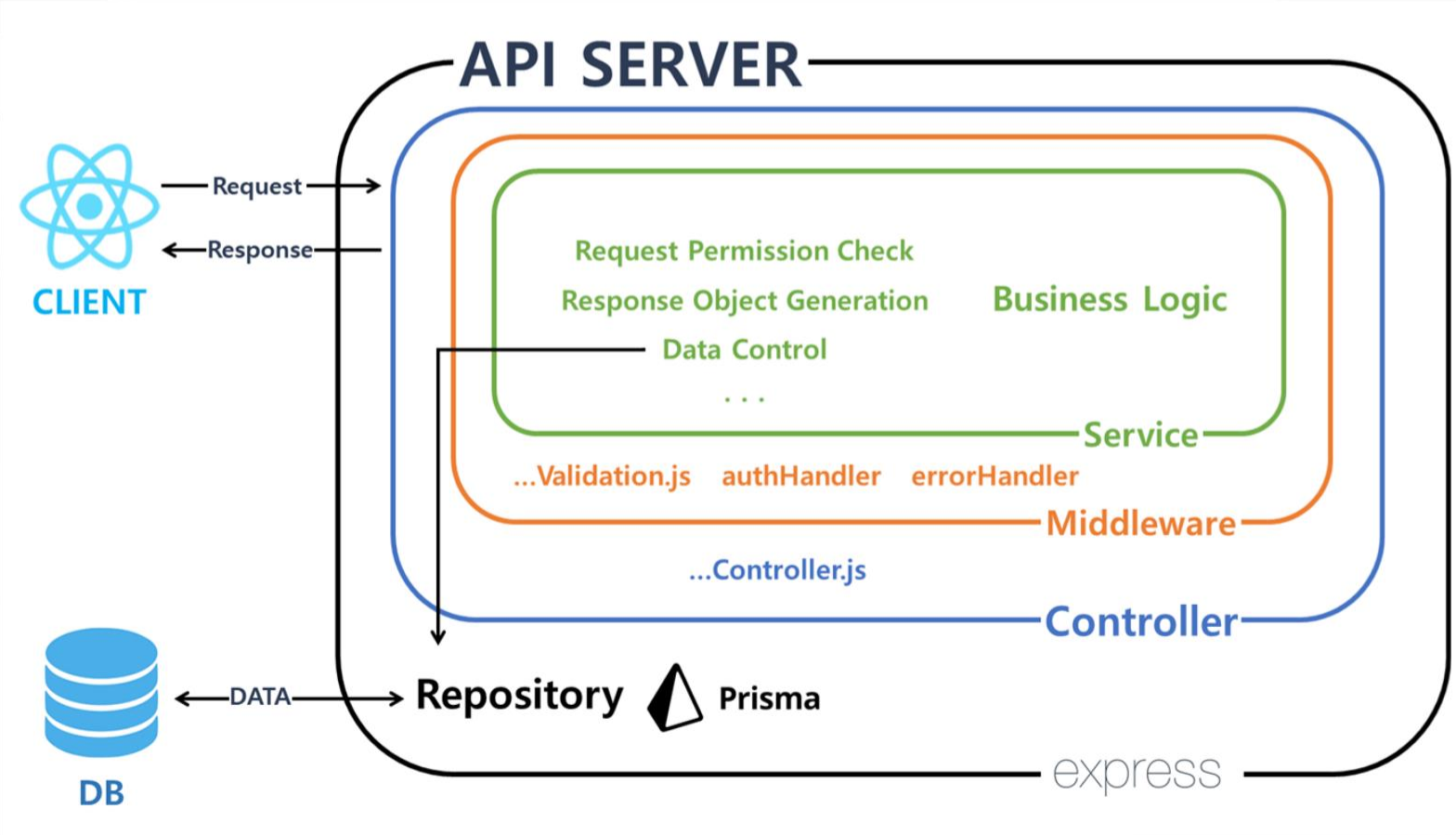
프로젝트에 쓰인 설계

1. 웹 프론트엔드



프로젝트에 쓰인 설계

2. 웹 백엔드



협업을 위한 github

github는 협업을 위해 꽤 많은 기능을 지원합니다.

특히 대학생이라면 팀 프로젝트에 잡다한 툴 쓰지 말고,
github로 모조리 해결해보세요.

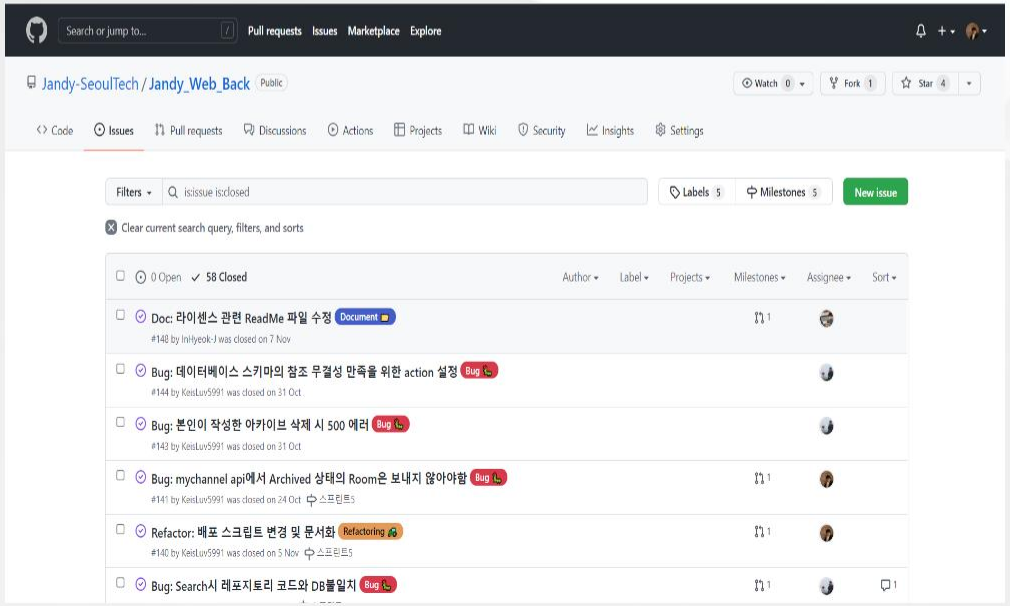
잔디도 심고 나중에 오픈소스 기여하는 데에도 쓰일 수 있습니다.

기본적인 github 사용법은 안다고 가정합니다.

협업을 위한 github 1.

issue 기반 작업 관리

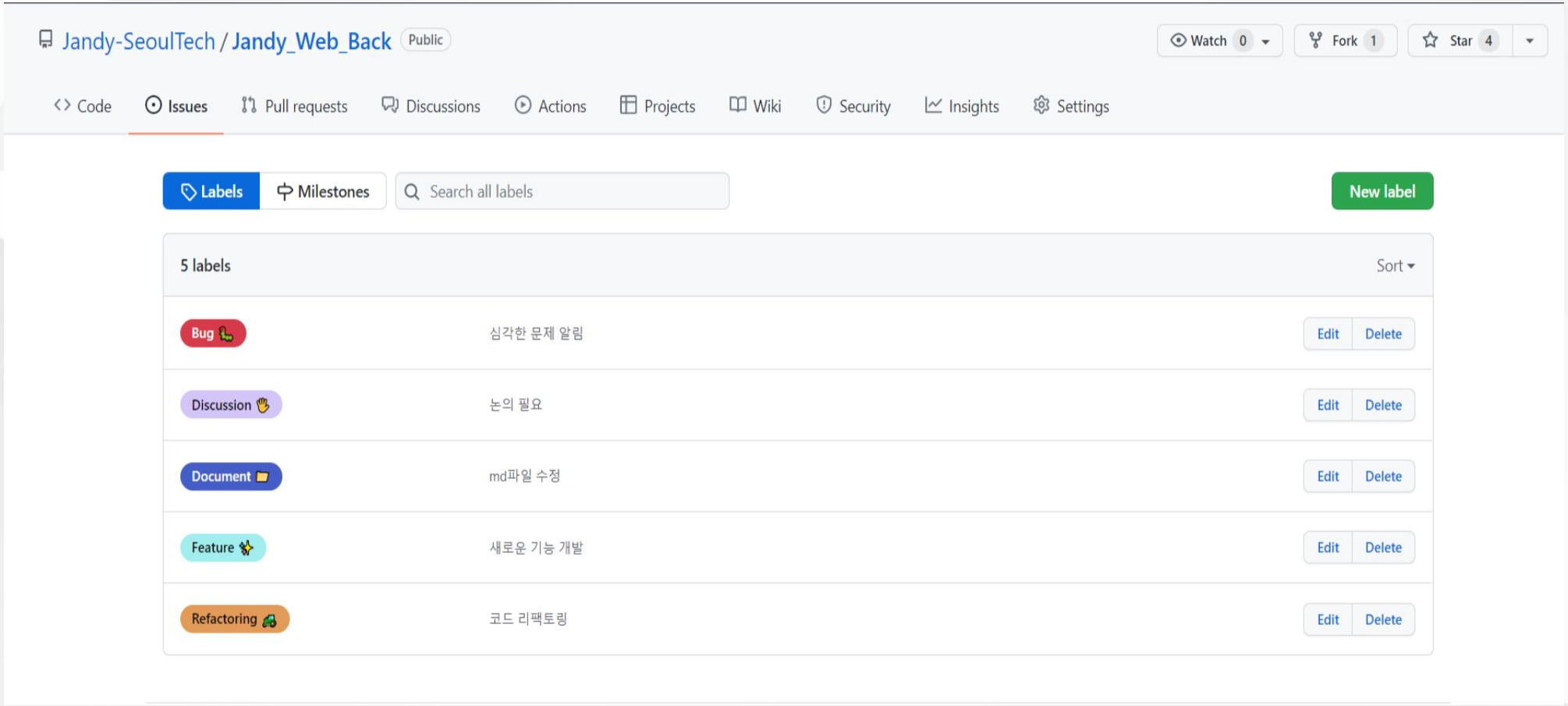
- 모든 작업, 논의, 리뷰, 등등...할 일과 얘기할 주제는 모두 이슈로 만듭니다.
- github repository의 issue 탭에서 새 이슈를 만들면 되겠죠?



Issue 분류하기

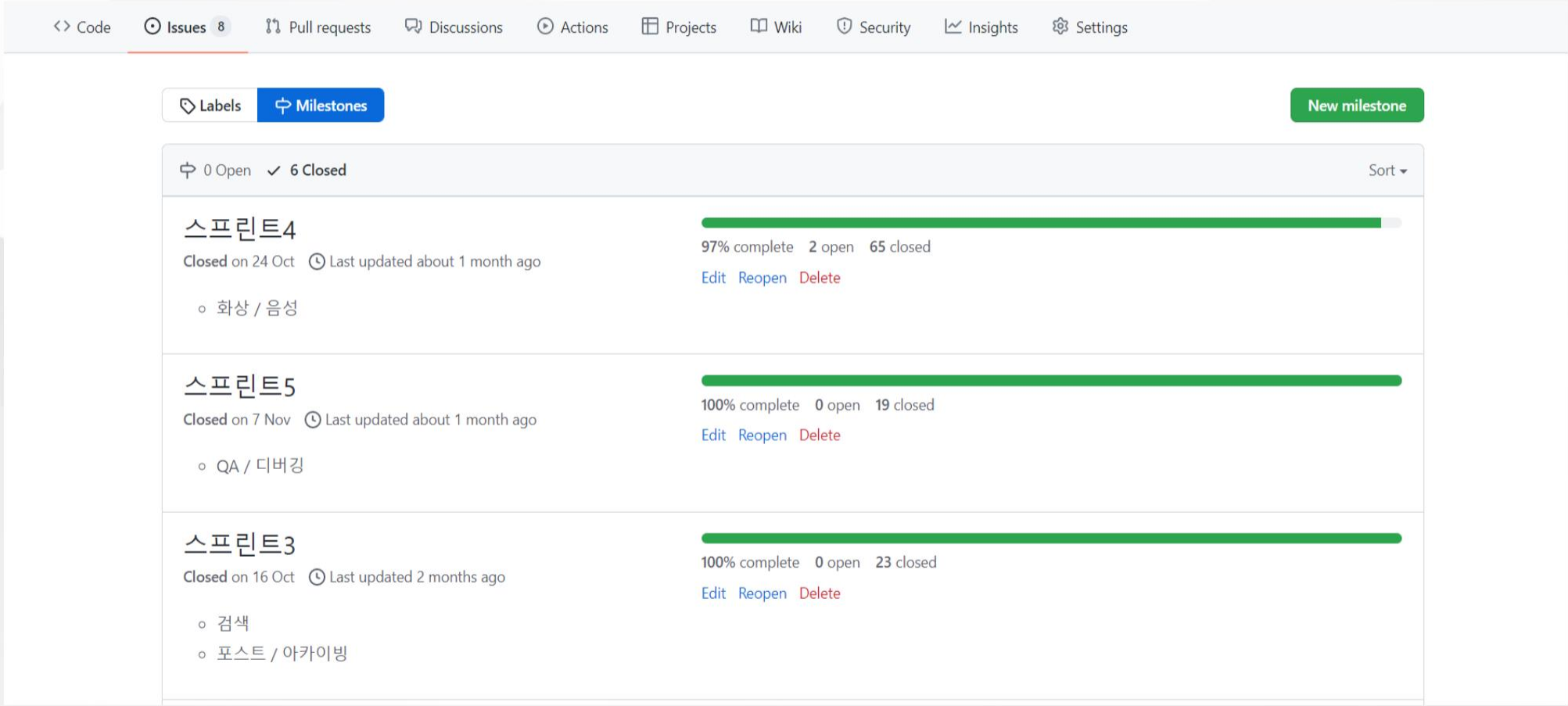
```
- 이슈에는 Assignees, Labels, Projects, Milestone, Linked PR이 있습니다.
- Assignees: 이 이슈를 처리하도록 할당받은 사람
- Labels   : 이 이슈를 분류하는 적절한 라벨
- Projects : 이 이슈가 속해있는 github project
- Milestone: 이 이슈가 처리될 일정에 대한 목표
- Linked PR: 이 이슈가 언급된 Pull Request
```

Label 수정하기



- 라벨은 수정할 수 있으니 프로젝트에 맞게 수정해서 쓰세요.

마일스톤 설정하기



- 기간을 잡고 무슨 일을 할건지 적으세요.

이슈 템플릿 만들기

- 라벨과 마일스톤을 정했으면, 이슈를 작성합니다.
- 하지만 이슈의 내용을 백지부터 써내려가려면, 조금 막막하죠.
- 협업하는 사람들과 이슈 형식도 통일하는 게 좋을 겁니다.

그런 당신을 위한 ISSUE TEMPLATE.

이슈 템플릿 깃헙에서 만들기

1. Repository Insights 탭으로 갑니다.
2. Community 항목을 봅니다.
3. CheckList 중에 Issue templates옆 Add 클릭
4. 템플릿을 원하는 만큼 추가합니다.
5. 왠만하면 위에서 설정한 라벨마다 템플릿 하나씩 만들고, 해당 템플릿은 해당 라벨을 자동으로 부여받게끔 설정하면 좋겠죠?
6. 제목과 본문 내용 컨벤션을 정해서 미리 작성해놓은 템플릿을 만들면 됩니다.

이슈 템플릿 로컬에서 만들기

1. 위 방법대로 하면 default branch에 커밋을 하게 됩니다.
2. 그 후 default branch를 보면 root에 .github 폴더가 생기고 그 안에 ISSUE_TEMPLATE 폴더가 있고 그안에 만들어놓은 이슈 템플릿 파일들이 있습니다.
3. 이걸 로컬에서 하면 됩니다.
4. root에 .github 폴더를 만들고, 그 안에서 ISSUE_TEMPLATE 폴더를 만듭니다.

이슈 템플릿 로컬에서 만들기

5. Markdown 파일을 만들어서 안쪽에 아래와 같이 작성해줍니다.

```
---
name: Feature request
about: 기능 구현 예정을 작성해주세요.
title: 'Feat: '
labels: Feature
assignees: ''

---
(contents)
```

6. (contents) 부분엔 본문 템플릿을 자유롭게 쓰면 됩니다.
7. github에 푸시하고 default branch에 머지하면 적용됨.

github Project 만들기

- 이제 이슈를 만들었으니, 이슈를 한데 모아서 이슈마다 진행상황을 한눈에 보기 편하게 해주는 Project를 만듭니다.
- 레포지토리 Project 탭에서, beta 붙은거 말고 일반 프로젝트를 클릭하고 새 프로젝트를 만듭니다.
- 이름, 설명은 알아서 입력하고, 프로젝트 템플릿을 선택해야 합니다.
- Automated kanban with reviews 추천.
- 이슈가 생성될 때 자동으로 Todo에 추가되기 때문에 간편하고, 코드리뷰를 위한 단계도 존재합니다.

Project로 issue 관리하기

- project를 automated kanban으로 만들고, 해당 프로젝트가 링크된 레포에서 새 이슈를 만들면 이슈가 자동으로 Todo에 추가됩니다.
- 팀은 정기적으로 Todo에 쌓인 이슈를 확인하고 팀원에게 Assignee를 설정하여 분배하며, 보드에서 각 이슈당 진행 척도를 확인하면 됩니다.
- 이슈가 큰 작업 단위인 경우 이슈의 본문에서 아래와 같이 써서 할일을 체크박스로 렌더링할 수 있습니다.

```
- [ ] 할일1
- [ ] 할일2
```

PR 템플릿 만들기

- 이슈를 만들어서 이슈를 해결했으면, PR을 올려야겠죠?
- 이때 PR 내용을 쓰는것도 고민입니다.
- 따라서 PR 템플릿을 활용해 봅시다.

로컬에서 PR 템플릿 만들기

1. Issue template 만들때처럼, root에 .github 폴더로 들어갑니다.
2. PULL_REQUEST_TEMPLATE.md 파일을 만듭니다.
3. PR 본문 템플릿을 마크다운으로 써줍니다.

```
# PR 제목입니다.

1. 오늘 한 일
2. 내일 할 일
3. 등등쓰
```

4. 푸시하고 default branch에 머지하면 적용됨.

PR 과 이슈의 관계

0. 이슈 기반 관리에서 PR은 보통 어떤 이슈를 해결하는 형태이므로, 모든 PR은 연결된 이슈가 있어야 할 겁니다.
1. 보통 ISSUE(하나) 대 PR(하나) 관계로 PR을 작성합니다.
2. 이슈가 너무 큰 작업단위면, 이슈를 쪼개서 새로운 이슈를 만드는게 좋아요.
3. 이슈가 너무 자잘한게 많으면, 비슷한 이슈끼리 묶어서 모두 해결하는 PR 하나를 올리는 것도 괜찮습니다.
4. 팀 사정에 맞게 알아서 전략짜서 적용해보세요.

PR과 이슈 연결하기

1. 깃허브에서 직접 연결

- PR을 작성하여 올리고, 자신이 작성한 PR 페이지로 들어가서 오른쪽을 보면 Reviewers, Assignees, Labels...이렇게 있습니다.
- Linked issues라는 것이 있는데, 클릭해서 링크할 이슈를 선택하세요.

2. PR 본문에서 언급

- PR을 작성할때 본문(description)에서 closing keyword와 함께 #이슈번호 를 작성하면 자동으로 링크됩니다. 다만 default branch로의 PR만 자동 링크가 적용되니 주의!

3. 커밋메시지에서 언급 : closing keyword를 커밋msg에 쓰면 해당 커밋을 포함하는 PR이 이슈와 링크됩니다.

PR 분류하기

- PR도 이슈처럼 Assignees, label, project, milestone이 있습니다.
- PR은 일반적으로 issue와 연결되있는 만큼, 위의 것들은 연결된 이슈의 것들을 따라가면 됩니다.
- PR의 특이점은 Reviewer가 있다는 건데, 코드리뷰를 해달라고 요청할 사용자들입니다.
- 협업 중이고 내가 백엔드인데 다른 백엔드 팀원이 있다면, 그 팀원에게 자신의 코드를 리뷰 요청하여 서로 검증하는 게 어떨까요?

협업을 위한 github 2.

코드 리뷰

1. github에서 코드 리뷰 기능을 쓰면 간편하게 가능합니다.
2. 코드 리뷰는 대체로 PR review 단계에서 이뤄질텐데, 더 펀쿨섹하게 하는 법은 나도 모릅니다.
3. 기본적인 스킬을 알려드릴테니 즐겁게 리뷰해봅시다.

PR reviewer 설정하기

- PR을 올렸으면, 내 코드를 같이 씹뜯맛즐해줄 사람을 리뷰어로 선택합니다.
- 오른쪽 Reviewer를 클릭하면 이 레포지토리에 권한이 있는 사용자를 보여주니 선택하시면 됩니다.
- 선정가능한 리뷰어 숫자는 private 레포는 구독구매 안하면 1명까지, public 레포는 여러명 가능하니까 참고하세요.

본격적으로 코드 봐보기

- 내가 만약 리뷰요청을 받았다면, 해당 PR에 들어가 File changed를 보면 됩니다.
- 없어진 코드는 빨간색, 추가된 코드는 초록색으로 표시되는데, 변경점을 유의해서 봐보세요.
- 톱니클릭해서 view 설정 변경도 가능합니다.
- 파일 필터링, PR에 여러 커밋이 들어있는 경우 커밋별로 보기, 등등 기능을 잘 활용해보세요.

뭔가 이상한 것을 찾았다

- 뭔가 찾았습니까? 그럼 바로 리뷰 드가자~~
- 코드에 커서를 올리면 파란색 + 박스가 뜹니다.
- 클릭하면 바로 그 라인밑에다가 코멘트를 달 수 있습니다.
- 근데 나중에 PR Conversation tab에서 볼때는 코드 달랑 한줄에 코멘트 달려있으면 맥락파악이 힘들어요.
- 왼쪽에 라인넘버를 클릭하고 드래그하면 여러줄이 선택되고, 선택된 여러줄의 맨 밑 줄에서 코멘트를 달면 나중에 볼때 여러줄이 모두 보이기 때문에 이 기능을 활용해보셈.

리뷰 마무리하기

- 코드를 싹 훑고 코멘트를 다 달았다면 리뷰 마무리를 해야할 차례입니다.
- 오른쪽 위에 초록색 review changes 버튼을 눌러서, 마무리 코멘트를 작성합니다.
- 세가지 선택지가 있습니다.
- Comment : 그냥 댓글만 달고 끝납니다. 별다른 의미 없음
- Approve : 이 PR을 머지하는 것을 허가한다는 뜻입니다.
- Request changes : 고칠 것이 있으니 머지하지 말란 뜻입니다.

리뷰 대응하기

- 리뷰로 달아준 소중한 댓글들에 반응을 보여주어야겠죠?
- 질문엔 답을, 코드 수정 요청엔 수정하여 새로 커밋을 올립니다.
- 해결된 코멘트에는 코멘트창 하단에 resolve conversation을 눌러 해결되었다고 표시해주세요.
- 코드를 수정하여 다시 커밋한 경우 코드가 바뀌었기 때문에 리뷰요청을 다시 합니다.
- 다시 리뷰 프로세스를 거쳐서 Approve를 모두 받으면 그때 머지하면 됩니다.

PR에서 말고 그냥 리뷰하기

1. 자.. 당신은 어떤 버그를 고치려고 씨름 중입니다.
2. 엇!! 당신은 버그의 원인이 어떤 코드 때문인 것을 발견합니다.
3. 이 코드를 누가 누가 썼을까?? 당신은 정말 궁금해졌습니다.
4. 이때 활용할 수 있는 것이 git blame입니다.
5. github 레포 code 탭에서 당신이 보고싶은 파일로 들어가 문제의 라인 넘버로 이동합니다.
6. 라인넘버를 클릭하면 왼쪽에 ... 버튼이 뜹니다.
7. 클릭해서 view git blame을 볼까요?
8. 와! 누가 언제 어떤 커밋을 했는지 한눈에 보이네요.

git blame 활용하기

- 당신이 작성하지 않은 코드에 대해 정보가 필요할 때 활용하면 좋습니다.
- 코드의 각 라인이 어떤 커밋에서 수정되었는지 한눈에 볼 수 있음.
- 이 파일이 어떤 커밋에서 바뀌기 전에 어땠는지 보고싶으면 라인넘버 왼쪽에 [[[[] 처럼 생긴 버튼 눌러서 볼 수 있음.
- 특히 커밋메시지에 이슈나 PR링크가 있으면 해당 커밋과 연결된 이슈와 PR을 바로 확인할 수 있기 때문에 유용합니다.
- `커밋메시지: 이것은 메시지입니다 (#12)` 이런식으로 있으면 #12에 하이퍼링크가 생김.

코드 하이퍼링크 만들기

- 어떤 코드에 대해 issue나 PR, comment에서 보여주고 싶은데 복붙은 하수의 기술입니다.
- 원하는 파일의 원하는 커밋 버전의 코드 라인으로 가서 왼쪽 라인넘버를 클릭하세요.
- ... 버튼의 copy permalink를 클릭하면 그 코드에 대한 url이 클립보드에 복사됩니다.
- 여러줄도 문제없이 시작 라인넘버 클릭하고, shift 누른상태로 끝 라인넘버를 클릭하세요.
- 선택된 여러줄에 대해 copy permalink로 공유하면 됩니다.

커밋 히스토리 정리하기

- 코드볼 때 커밋 히스토리가 난장판이라면 커밋한 사람을 찾아가고 싶을 때가 있습니다.
- 모두의 정신건강을 위해서, 로컬에서 작업하고 github에 푸쉬하기 전 커밋 히스토리를 git squash로 정리해봐요.
- github에서 PR을 머지할 때도, 디폴트 옵션인 merge commit create 방식보다 Squash and merge 방식을 사용하면 PR에 포함된 여러 커밋을 squash 해줍니다.
- 특히 이 방식은 커밋메시지에 자동으로 PR 번호를 붙혀줘서 나중에 커밋히스토리 볼때 커밋메시지에서 PR로 바로 링크가 걸리기 때문에 좋음.

협업을 위한 다른 스킬들.

1. linter : 코딩 스타일을 일관되게 지켜주기 위해 필요합니다.
2. testing : 테스트 코드가 있다면 테스트가 자동화됩니다. 말이 필요?
3. CI / CD 프로세스 : 나도 잘 몰라요옹..ㅎㅎ 공부하는 중.

4. 무엇보다 중요 : 적극적&배려하며 소통하려는 자세.

팀원들과 의견교환 많이하고 건실한 토론해보세요.

상대에 대한 인격적 비난과 타당한 비판적 의견에 대한 방어적 자세는 소통을 가로막는 The Wall입니다.

대학생 팀 프로젝트는 좋을까?

-
-

내 생각엔 좋습니다.

누구랑, 무엇을, 어떻게 하는지에 따라서.

아래부터는 개인 의견입니다.

누구랑 하는가?

대학생이라면 학과 내에서 코딩 잘하거나/열심히한다고 소문난 사람들 기억해놨다가 이래저래 컨택해서 알고지내는 편이 좋습니다.

나중에 팀프로젝트 할때나, 평소 생활에서 서로 시너지를 주고받을 일이 많아요.

학과 이외에도 중앙동아리, 연합동아리 등등 다양하게 사람만나고 다니면서 고수와 성장잠재력이 높은 사람을 만나고 다녀보세요.

그렇게 만나고 다니면서 괜찮은 사람들은 더 친하게 지내면 됨.

그래서 ㄹㅇ 누구랑 함?

인맥 만든 고수들한테 세일즈해서 팀 빌딩하세요.

인생은 세일즈 잘해야 편한 부분도 있다고 생각함.

독고다이도 좋을지 몰라도 난 단체로서의 시너지가 크다고 생각함.

자신만의 코딩 전위대를 만들어서 황제가 되어보세요.

고수인데 인성터진 그런거 감수하진 말구요. 인성터진놈은 개발 잘하고 코딩 잘해도 같이 있으면 정신건강에 안좋음.

무엇을 하는가?

본인이 가고 싶은 진로가 있다면 그 진로에 알맞는 주제의 프로젝트가 좋겠죠?

얘를 들어
웹/앱 백엔드/프론트엔드 개발자? -> 웹/앱 어플리케이션

더 예시를 들기는 저는 잘 몰라서 들지 않겠습니다. 어쨋든 왠만하면 본인의 진로에 맞는 주제로 플젝을 해보세요.

그리고 플젝의 난이도는 적당하게 높아야 합니다.
너무 쉽거나/규모가 작거나 하면 경험하는게 없어요.
자신의 수준에서 조금 높은 수준을 목표로 플젝해야 얻는 것이 많음.

어떻게 하는가?

주도적으로 하세요.

팀원이 이래라 저래라 하는 것을 따르기만 하면 의미가 별로 안남아요. 플젝에서 얻어가고 싶은 부분을 정하고, 그 부분에 대해선 주도적으로 개발하고 참여해야 의미가 많습니다.

적당하게 하지 마세요.

적당적당히 할꺼면 왜 함? 열심히 한 만큼 자신이 공부하고 실력이 는다고 생각하고 열심히 하세요. 물론 팀이 좀 안좋다면 열심히 할 맘이 안들기도 할텐데, 그럼 나오세요. 억지로 하면서 적당히 하면 되는 것 하나도 없음.

같이 하면 즐겁다!

맘 맞는 사람들과, 관심있고 재미있는 주제로, 주도적으로 열심히 한다면 재밌게 할 수 있을 거에요.

누구는 공부/코딩/일이 어떻게 재밌냐고 물어볼 수 있겠지만 그들은 인생의 절반을 손해보는 겁니다.

본인이 씹부자가 아닌 이상 일/공부는 해야되는 거고, 이왕 하는 거 재밋게 하는 편이 좋잖아요?

사람은 사회적 동물이니 팀 프로젝트 성공기 한번쯤은 대학생활에서 얻고 가보시길 바랍니다.

끝~