

와플스튜디오 23.5기

프론트엔드 세미나 7주차

강의자: 김연우

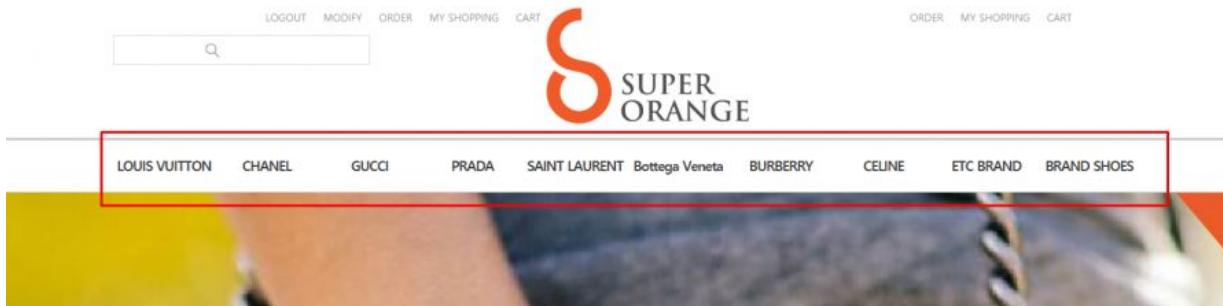
오늘의 목표

- ✓ CSS 쉽게 사용하기
- ✓ 어떻게 하면 코드를 잘 짤 수 있나요?

CSS 쉽게 사용하기

CSS는 어려운 언어입니다

여러분이 지금까지 plain css를 많이 다뤄 봐서 아시겠지만, 마크업 언어여서 더 어렵고 복잡한 면이 있습니다.



가운데 정렬이 틀어집니다

CSS는 어려운 언어입니다

여러분이 지금까지 plain css를 많이 다뤄 봐서 아시겠지만, 마크업 언어여서 더 어렵고 복잡한 면이 있습니다.



컨텐츠끼리 겹쳐집니다



Plain CSS의 Pain Point

1. 클래스 이름 충돌이 발생할 수 있다.
2. 복잡한 스타일링 시 구문이 매우 복잡해진다.
3. 재사용성이 떨어진다.

보완 방식의 등장

1. 클래스 이름 충돌이 발생할 수 있다 -> BEM, **CSS Module**
2. 복잡한 스타일링 시 구문이 매우 복잡해진다. -> **CSS 전처리기** (SASS 등)
3. 재사용성이 떨어진다. -> **CSS in JS** (Styled Components, Emotion, Vanilla Extract 등)

BEM

CSS클래스 명 충돌을 방지하기 위해 **클래스 이름을 짓는 규칙을 체계적으로 만들어보자**

Block, Element, Modifier에 따라 클래스명을 짓습니다. (e.g. card__title—danger)

하지만 **CSS Module**을 사용하면 웬만한 게 다 해결되니 모르셔도 특별히 상관 없습니다.

CSS Module

CSS의 적용 범위를 각 컴포넌트로 제한하자

```
// Button.module.css
.button {
  background-color: blue;
  color: white;
}

// Button.jsx
import styles from './Button.module.css';

export default function Button() {
  return <button className={styles.button}>Click
Me</button>;
}
```

SASS

CSS를 더 효율적으로 작성하도록 하는 문법

CSS에서 사용할 수 없던 다양한 기능들을 사용할 수 있음.

SASS

1. 변수 사용

```
$main-color: #3498db;  
  
.button {  
    background-color: $main-color;  
}
```

SASS

2. 중첩

```
.card {  
    border: 1px solid #ccc;  
  
    .title {  
        font-size: 18px;  
    }  
  
    .content {  
        font-size: 14px;  
    }  
}
```

SASS

3. Mixin - @include (재사용성 향상)

```
@mixin flex-center {  
  display: flex;  
  justify-content: center;  
  align-items: center;  
}  
  
.container {  
  @include flex-center;  
}
```

SASS

4. 조건문과 반복문

```
@for $i from 1 through 3 {  
  .margin-#$i {  
    margin: #{$i}rem;  
  }  
}
```

```
$size: large;  
  
.button {  
  padding: 8px 16px;  
  font-size: 14px;  
  
  @if $size == small {  
    font-size: 12px;  
    padding: 4px 8px;  
  } @else if $size == medium {  
    font-size: 14px;  
    padding: 8px 16px;  
  } @else if $size == large {  
    font-size: 18px;  
    padding: 12px 24px;  
  } @else {  
    font-size: 14px;  
    padding: 8px 16px;  
  }  
}
```

SASS

단, SASS는 동적 스타일링이 불가능합니다.

✓ 정적 스타일링: 코드를 작성하는 시점에 고정되는 스타일

```
.btn {  
  color: blue;  
}
```

✓ 동적 스타일링: 실행 중에 조건에 따라 바뀌는 스타일

```
$theme: light;  
  
@if $theme == 'dark' {      theme 값을 실행 중에 변경할 수 없습니다.  
  background-color: #111;  
}
```

SASS와 동적 스타일링

내용물을 바꾸지 말고, 클래스 명을 바꾸면 됩니다.

```
import { useState } from 'react';
import './main.scss';

export const Main = () => {
  const [isDarkMode, setIsDarkMode] = useState(false);

  const toggleTheme = () => {
    setIsDarkMode((prevMode) => !prevMode);
  };

  return (
    state에 따라 클래스 명을 변경합니다.
    <div className={`app ${isDarkMode ? 'dark-mode' : 'light-mode'}`}>
      <button onClick={toggleTheme}>Toggle Dark Mode</button>
    </div>
  );
}
```

```
$light-bg: #ffffff;
$dark-bg: #121212;
$light-text: #000000;
$dark-text: #ffffff;

.app {
  &.dark-mode {
    background-color: $dark-bg;
    color: $dark-text;
  }

  &.light-mode {
    background-color: $light-bg;
    color: $light-text;
  }
}
```

CSS in JS

스타일링이 적용된 컴포넌트를 만들자

props를 사용한 동적 스타일링이 용이합니다.

현재 styled component는 개발 중단되었으니

만약 CSS in JS를 사용하고 싶다면 Vanilla Extract, Emotion 등을 사용하는 것이 좋습니다.
(사실 다 비슷하게 생겼습니다)

```
import { useState } from 'react';
import styled from '@emotion/styled';

type ButtonProps = {
  darkMode: boolean;
};

type ContainerProps = {
  theme: {
    background: string;
    color: string;
  };
};

const lightTheme = {
  background: '#ffffff',
  color: '#000000',
};

const darkTheme = {
  background: '#121212',
  color: '#ffffff',
};

const Container = styled.div<ContainerProps>`  
  background-color: ${({ theme }) =>  
    theme.background};  
  color: ${({ theme }) => theme.color};  
`;  
  
const Button = styled.button<ButtonProps>`  
  background-color: ${({ darkMode }) => (darkMode ?  
    '#333' : '#ddd')};  
  color: ${({ darkMode }) => (darkMode ? '#fff' :  
    '#333')};  
  &:hover {  
    background-color: ${({ darkMode }) => (darkMode ?  
      '#444' : '#ccc')};  
  }  
`;
```

```
export const App = () => {
  const [darkMode, setDarkMode] = useState(false);

  // 다크모드 토글 함수
  const toggleTheme = () => {
    setDarkMode(!darkMode);
  };

  return (
    <Container theme={darkMode ? darkTheme : lightTheme}>
      <div>
        <h1>{darkMode ? 'Dark Mode' : 'Light Mode'}</h1>
        <Button
          darkMode={darkMode}
          onClick={toggleTheme}
        >
          Toggle Dark Mode
        </Button>
      </div>
    </Container>
  );
}
```

Styled Component 사용 예제

CSS in JS

컴포넌트에 스타일이 붙어 있어 디버깅이 불편합니다.

네이밍이 힘듭니다.

(사실 요즘은 GPT가 잘 짜주긴 해서, 그렇게 크리티컬한 문제는 아니라고 생각합니다)

개인적으로 느끼는 단점) 파일이 엄청나게 길어집니다 !

Utility First

자주 사용하는 CSS 스타일링 속성을 각각 클래스명으로 분리하자

여러 클래스 명의 조합으로 스타일 속성을 보여줍니다.

개개인의 컴포넌트 네이밍으로 스타일 속성 판단 (CSS in JS)

-> 클래스명만 보면 어떤 스타일인지 바로 이해할 수 있음. (Utility First - Tailwind)

```
import { useState } from 'react';

const App = () => {
  const [darkMode, setDarkMode] = useState(false);

  const toggleTheme = () => {
    setDarkMode(!darkMode);
  };

  return (
    <div className={darkMode ? 'bg-gray-900 text-white' : 'bg-white text-black'}>
      <div className="text-center">
        <button className={darkMode ? 'bg-gray-700 text-white hover:bg-gray-600' : 'bg-gray-200 text-gray-800
        hover:bg-gray-300'}
          onClick={toggleTheme}
        >
          Toggle Dark Mode
        </button>
      </div>
    </div>
  );
};


```

어떤 것을 사용하면 좋을까요?

어떤 프로젝트냐에 따라 다릅니다.

각 방식의 장단점을 파악하여 사용해보세요
(아니면 채용 공고를 보면 요즘 많이 사용하는 것들을 파악해도 좋아요)

간단한 프로젝트 or 빠르게 만들어야 하고 디자인이 자주 바뀔 때: tailwind

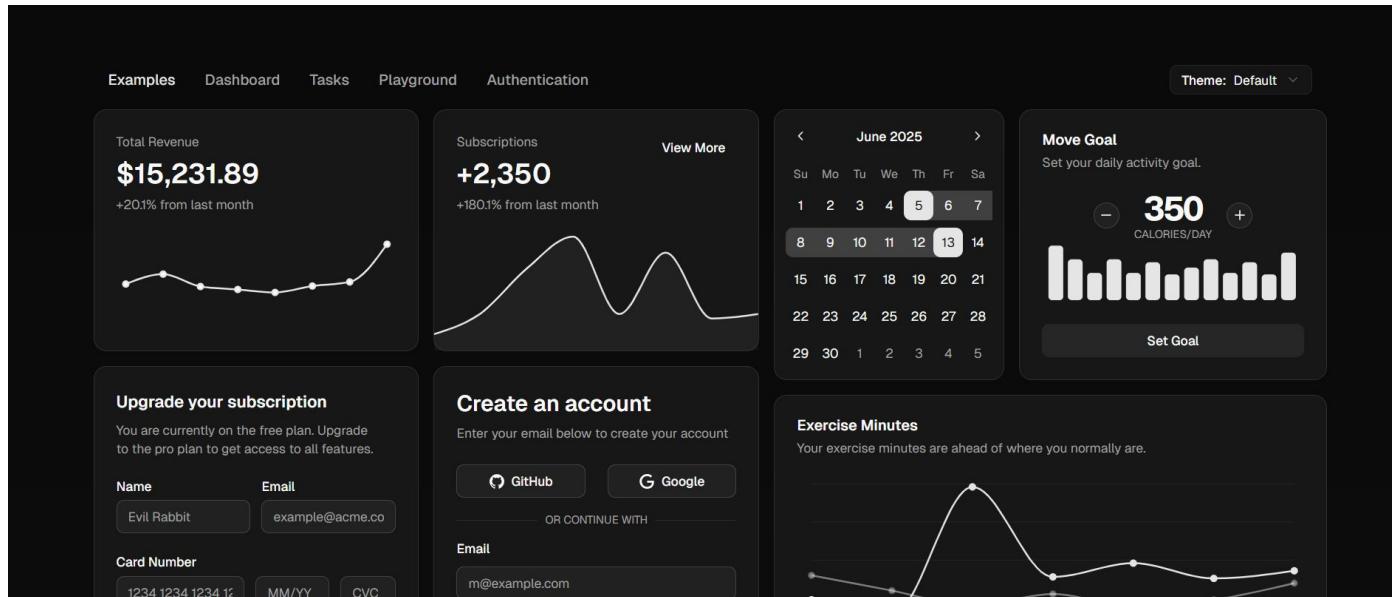
복잡한 동적 스타일링이 많은 경우 & 타입 안정성이 필요할 때: CSS in JS

정적 스타일링이 복잡할 때 & CSS in JS를 못 쓰는 환경: SASS + CSS module

Shadcn/ui의 등장 - 0부터 디자인하지 않습니다

자주 사용하는 요소들에 대해서는 미리 기능이 만들어진 컴포넌트를 사용합니다.

대신, 기능과 스타일링이 분리되어 있어 보다 편리하게 디자인을 커스터마이징할 수 있도록 한다



어떻게 하면 코드를 잘 짤 수 있나요?

잘 만든 코드는 무엇일까요?

1. 잘 돌아가야 합니다.
2. 유지보수가 편리해야 합니다.

이번 챕터에서는 “유지보수”에 집중합니다

기능을 빠르게 만드는 것은 누구나 할 수 있습니다.

하지만 유지보수는 나 혼자 만드는 프로젝트에서는 적당히 타협하고 넘어가는 경우가 많습니다.

따라서 본 챕터에서는 개인 과제에서 놓치기 쉬운 “유지보수성”에 대해 배웁니다.

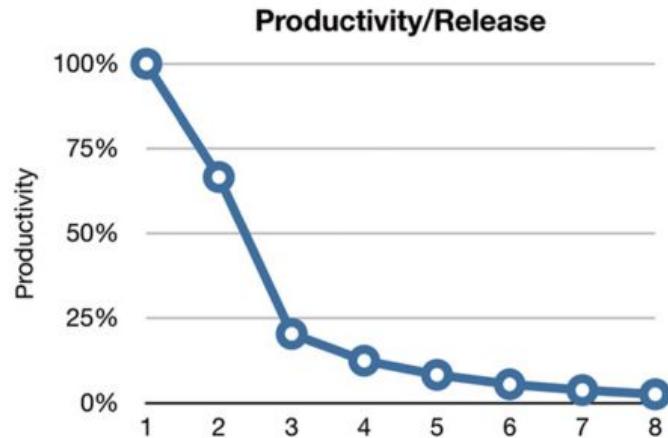
유지보수는 왜 필요한가요?

유지보수가 잘 되지 않는다면

1. 신기능이 늘어날 때마다 고쳐야 하는 코드가 늘어납니다.

(e.g. B 기능을 추가했더니 기존에 있던 A 기능이 작동하지 않는다.)

2. 개발자가 2배로 늘어나도 생산성이 2배로 늘어나지는 않습니다.



유지보수는 왜 필요한가요?

software는 hardware와 다르게 변경에 쉽게 대응할 수 있어야 합니다.

구현을 위해 변경하기 쉬운 시스템을 포기한다면 소프트웨어 개발자로서의 의무를 지키지 않은 것과 같습니다.

(조금 더 와닿는 이유로) 유지보수는 왜 필요한가요?

혼자 과제할 때에도 유지보수의 필요성을 느끼기 어렵습니다.

방학에 토이프로젝트를 해도 유지보수의 필요성을 느끼기 어렵습니다.

(뇌피셜) 단, 이 코드를 다른 사람에게 넘겨야 하거나 누군가의 코드를 넘겨받게 된다면 유지 보수의 필요성을 뼈저리게 느끼게 됩니다.

욕 먹지 않는 코드는 어떻게 만들 수 있나요?

1. 먼저 잘못 만들어서 크게 고통을 받아봅니다.
2. 고민하고 개선해야겠다고 결심합니다. (절대 그냥 덮어두지 않습니다)
3. 고수의 코드, 책, 블로그 등으로 인사이트를 얻어 개선해봅니다.

조금 더 구체적인 방법

개발 세계에서 통용되는 몇가지 원칙들이 있습니다.

모든 규칙이 진리는 아니며, 다수의 사람들이 좋다고 여기는 팁에 가까우니 나의 상황에 맞게 적용하면 됩니다.

키워드: 네이밍, DRY, SRP, 폴더 구조, 리팩토링

네이밍

이름짓기는 쉬워보이지만 매우 힘들기도 합니다.



**PROGRAMMERS WHILE
MAKING A SOFTWARE**

**PROGRAMMERS WHILE
NAMING A SOFTWARE**

좋은 네이밍

1. 너무 일반적인 이름을 사용하지 않습니다.

```
const func = (a: number) => {
  return a > 1;
};
```

좋은 네이밍

2. 거짓말을 하지 않습니다.

```
const validateIsUnder10 = (age: number) => {
  return age > 1;
};
```

좋은 네이밍

2. 거짓말을 하지 않습니다.

조금 더 일반적인 상황: 이전에는 사실이었지만 외부의 변경으로 현재는 거짓말이 되는 것

```
// 과거: 사용자가 직접 업로드한 경우만 true
type User = {
  name: string;
  profilePictureUrl?: string;
};

const hasProfilePicture = (user) => {
  return user.profilePictureUrl !== undefined;
};
```

```
// 현재: 기본 이미지도 자동으로 들어감
const user = {
  name: 'Bob',
  profilePictureUrl: '/images/default-avatar.png',
  // 업로드한 건 아님
};
```

좋은 네이밍

3. 결과가 아닌 의도를 담아 작성합니다.

```
// Bad: 결과만을 드러냄. 내부 로직이 변경되면 거짓말이 됨.  
const validateIsOver1 = (age: number) => {  
    return age > 1;  
};
```

```
// Good: 해당 함수를 만들게 된 의도를 드러냄.  
const validateAge = (age: number) => {  
    return age > 1;  
};
```

JS/TS 생태계에서의 네이밍 규칙

1. 함수는 동사로 시작합니다.

```
const valid = (age: number) => {}      // ✗  
const validate = (age: number) => {} // ✓
```

2. 함수가 아닌 값은 명사로 시작합니다.

```
const getAge = 10 // ✗  
const age = 10;   // ✓
```

JS/TS 생태계에서의 네이밍 규칙

3. 부정적인 단어보다는 긍정적인 단어를 사용합니다.

```
const isNotValid = validate(age).result === 'invalid'; // ✗  
const isValid = validate(age).result === 'valid'; // ✓
```

4. 변수는 camelCase, 상수는 UPPER_CASE, 컴포넌트/type/interface는 PascalCase를 사용합니다.

```
const isValid = validate(age).result === 'valid'; // ✓  
const MAP_2048_SIZE = 4; // ✓  
type Todo = { id: 1, text: 'hello' }; // ✓
```

JS/TS 생태계에서의 네이밍 규칙

5. 변수명에서는 줄임말보다는 풀네임을 사용합니다. (재사용하지 않는 경우에는 상관 없습니다)

```
const onClickBtn = () => {};      // ✗  
const onClickButton = () => {}; // ✓  
const studentAges = students.map((s) => s.age);           // ✓ 배열 콜백에서는 시원하게 줄여도 됨  
const studentAges = students.map((student) => student.age); // ✓ 당연히 안 줄여도 됨
```

6. 헝가리안 표기법 사용은 지양합니다.

```
type TStudent = { name: string };    // ✗  
type StudentType = { name: string }; // ✗  
type Student = { name: string };     // ✓
```

DRY: Don't Repeat Yourself

모든 지식은 시스템 내에서 단 한번만, 애매하지 않고, 권위있게 표현해야 한다.

=> 불필요한 반복은 지양합니다.

DRY: Don't Repeat Yourself

```
const fetchTodos = async () => {
  const response = await
fetch(`https://jsonplaceholder.typicode.com/todos`);
  const data = await response.json() as
TodoResponse[];
  return data;
};

const fetchTodo = async (id: number) => {
  const response = await
fetch(`https://jsonplaceholder.typicode.com/todos/${
id}`);
  const data = await response.json() as
TodoResponse;
  return data;
};
```

DRY: Don't Repeat Yourself

```
const fetchTodos = async () => {
  const response = await
fetch(`https://jsonplaceholder.typicode.com/todos`);
  const data = await response.json() as
TodoResponse[];
  return data;
};

const fetchTodo = async (id: number) => {
  const response = await
fetch(`https://jsonplaceholder.typicode.com/todos/${id}`);
  const data = await response.json() as
TodoResponse;
  return data;
};
```

```
const baseUrl =
`https://jsonplaceholder.typicode.com`;

const fetchTodos = async () => {
  const response = await fetch(`${baseUrl}/todos`);
  const data = await response.json() as
TodoResponse[];
  return data;
};

const fetchTodo = async (id: number) => {
  const response = await
fetch(`${baseUrl}/todos/${id}`);
  const data = await response.json() as
TodoResponse;
  return data;
};
```

DRY: Don't Repeat Yourself

```
const load = () => {
  try {
    return
    JSON.parse(localStorage.getItem('2048Data') as
BoardData);
  } catch (e) {
    return null;
  }
};

const save = (data: BoardData) => {
  localStorage.setItem('2048Data',
JSON.stringify(data));
};
```

DRY: Don't Repeat Yourself

```
const load = () => {
  try {
    return
    JSON.parse(localStorage.getItem('2048Data') as
BoardData);
  } catch (e) {
    return null;
  }
};

const save = (data: BoardData) => {
  localStorage.setItem('2048Data',
JSON.stringify(data));
};
```

```
const LOCALSTORAGE_KEY = { '2048Data': '2048Data' };
```

```
const load = () => {
  try {
    return JSON.parse(
      localStorage.getItem(LOCALSTORAGE_KEY['2048Data']) as
BoardData
    );
  } catch (e) {
    return null;
  }
};

const save = (data: BoardData) => {
  localStorage.setItem(LOCALSTORAGE_KEY['2048Data'],
JSON.stringify(data));
};
```

```
const save = (data: BoardData) => {
  localStorage.setItem(LOCALSTORAGE_KEY['2048Data'],
JSON.stringify(data));
};
```

DRY: Don't Repeat Yourself

```
const EmailInput = () => {
  const [gmailValue, setGmailValue] = useState('');

  const onSubmit = () => {
    registerEmail(gmailValue + '@gmail.com');
  };

  return (
    <div>
      <input
        type="text"
        value={gmailValue}
        onChange={(e) =>
          setGmailValue(e.target.value)}
        />
      <span>@gmail.com</span>
    </div>
  );
};
```

DRY: Don't Repeat Yourself

```
const EmailInput = () => {
  const [gmailValue, setGmailValue] = useState('');

  const onSubmit = () => {
    registerEmail(gmailValue + '@gmail.com');
  };

  return (
    <div>
      <input
        type="text"
        value={gmailValue}
        onChange={(e) =>
          setGmailValue(e.target.value)}
      />
      <span>@gmail.com</span>
    </div>
  );
};
```

```
const EMAIL_POSTFIX = '@gmail.com';

const EmailInput = () => {
  const [gmailValue, setGmailValue] = useState('');

  const onSubmit = () => {
    registerEmail(gmailValue + EMAIL_POSTFIX);
  };

  return (
    <div>
      <input
        type="text"
        value={gmailValue}
        onChange={(e) =>
          setGmailValue(e.target.value)}
      />
      <span>{EMAIL_POSTFIX}</span>
    </div>
  );
};
```

DRY: Don't Repeat Yourself

```
const validateAge = (age: number) => {
  if (age < 1) return false;
  return true;
};

const validateGrade = (grade: number) => {
  if (grade < 1) return false;
  return true;
};
```

DRY: Don't Repeat Yourself

```
const validateAge = (age: number) => {
  if (age < 1) return false;
  return true;
};

const validateGrade = (grade: number) => {
  if (grade < 1) return false;
  return true;
};
```

똑같이 생겼다고 항상 제거해야 하는 것은 아닙니다.
동일한 시점에 동일한 이유로 동일한 변경이 일어난다면 제거가
필요합니다.

SRP: Single Responsibility Principle

모듈이 커질수록 가독성이 떨어지게 됩니다.

따라서 유지보수가 쉽게 이루어지려면 읽어야 하는 코드의 양이 너무 많지 않아야 합니다.

=> 즉, 코드를 잘 쪼개줘야 합니다.

SRP: Single Responsibility Principle

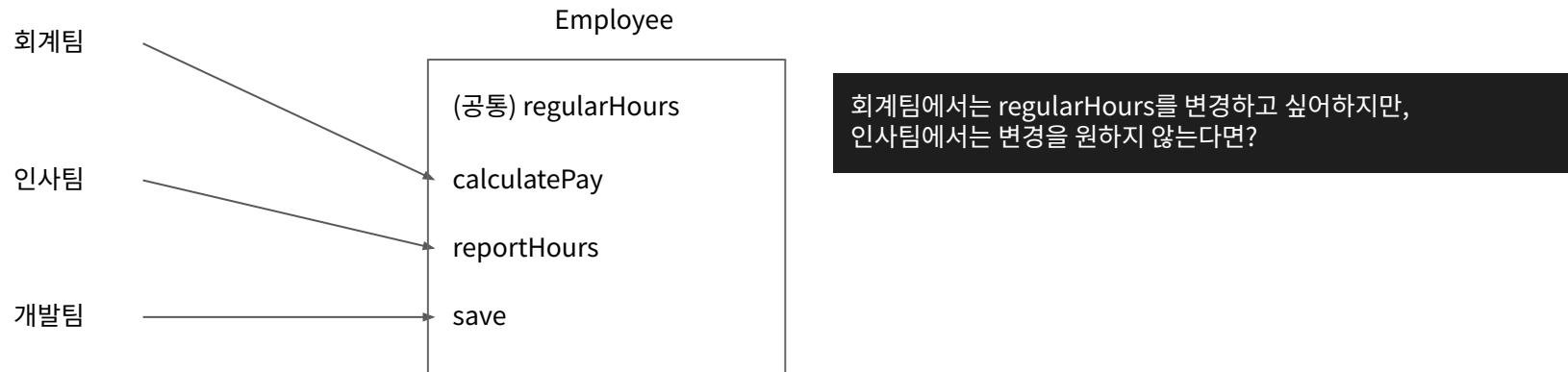
하나의 모듈은 하나의 액터에 의해서만 변경되어야 한다.

액터: 서로 다른 관심사를 가진 이해관계자 (e.g. 디자이너, 기획자, API 개발자 등등)

SRP: Single Responsibility Principle

만약 하나의 코드를 여러 액터들이 변경할 수 있다면 하나의 변경이 여러 팀에 영향을 미칠 수 있습니다.

서로 다른 액터들이 영향을 미친다면 “중복이 되더라도” 분리시켜야 합니다.



SRP: Single Responsibility Principle

```
type User = { id: number; name: string; joinedAt: string };

export const UserProfile = ({ userId }: { userId: number }) => {
  const [user, setUser] = useState<User | null>(null);
  useEffect(() => {
    fetch(`/api/users/${userId}`)
      .then((res) => res.json())
      .then(setUser);
  }, [userId]);
  const formatDate = (dateStr: string) =>
    new Date(dateStr).toLocaleDateString('en-US');
  if (user === undefined) return <p>Loading...</p>;
  return (
    <div>
      <h1>{user.name}</h1>
      <p>Joined: {formatDate(user.joinedAt)}</p>
    </div>
  );
};
```

SRP: Single Responsibility Principle

```
type User = { id: number; name: string; joinedAt: string };

export const UserProfile = ({ userId }: { userId: number }) => {
  const [user, setUser] = useState<User | null>(null);
  useEffect(() => {
    fetch(`api/users/${userId}`)
      .then((res) => res.json())
      .then(setUser);
  }, [userId]);
  const formatDate = (dateStr: string) =>
    new Date(dateStr).toLocaleDateString('en-US');
  if (user === undefined) return <p>Loading...</p>;
  return (
    <div>
      <h1>{user.name}</h1>
      <p>Joined: {formatDate(user.joinedAt)}</p>
    </div>
  );
};
```

API 개발자가 변경 가능

포매팅 함수: 기획자가 변경 가능

화면: 디자이너가 변경 가능

SRP: Single Responsibility Principle

```
type Todo = { id: number; title: string; done: boolean };

export const TodoList = ({ todos }: { todos: Todo[] }) => {
  const countDone = () => todos.filter((todo) => todo.done).length;

  return (
    <div>
      <p>
        Done: {countDone()} / {todos.length}
      </p>
      <ul>
        {todos.map((todo) => (
          <li key={todo.id}>{todo.title}</li>
        )));
      </ul>
    </div>
  );
};
```

SRP: Single Responsibility Principle

```
type Todo = { id: number; title: string; done: boolean };

export const TodoList = ({ todos }: { todos: Todo[] }) => {
  const countDone = () => todos.filter((todo) => todo.done).length;
  return (
    <div>
      <p>
        Done: {countDone()} / {todos.length}
      </p>
      <ul>
        {todos.map((todo) => (
          <li key={todo.id}>{todo.title}</li>
        )));
      </ul>
    </div>
  );
};
```

비즈니스 로직 -> 기획자가 변경 가능

SRP: Single Responsibility Principle

유의사항: 분리가 많이 일어날수록 하나의 기능을 수정할 때 많은 파일을 수정해야 할 수 있습니다.

따라서 상황에 맞는 분리 규칙을 정하는 것이 좋습니다.

풀더구조

많은 분들이 풀더 구조를 고민하셨을텐데, 사실 풀더구조보다는 결국 이전의 모듈 설계에 따른 연장선입니다.

따라서 풀더 구조 설계보다는 모듈을 잘 쪼개는 게 더 중요합니다.

풀더에는 그냥 쪼개놓은 모듈을 넣어주기만 하면 되니까요.

풀더구조

- 관련있는 것일수록 가까이 두면 좋다
- 관련이 없을수록 멀리 두면 좋다

참고로 프론트엔드 생태계에서는 [FSD](#)라는 방식이 유명한 편입니다.

하지만 팀마다 풀더구조를 취하는 양식이 매우 다르므로 서로 합의하여 자유롭게 결정하시면 됩니다.

리팩토링

기존 코드의 **동작방식을 변경하지 않으면서도**, 코드를 유지보수하기 쉽게 조정하는 것

시간이 생기면 리팩토링한다는 생각을 버려주세요

리팩토링이 필요하다면 발견한 즉시 처리하는 게 제일 빠릅니다.

라고 말하는 저도 리팩토링을 많이 미루는 편입니다 ㅜㅜ 여러분은 이렇게 하지 마세요

과제 공지

과제 1: 스누인턴 3 - 마이페이지 만들기

이번에는 마이페이지 창을 생성해봅니다.

관심 공고 보기, 내 정보 보기, 내 정보 생성/수정 기능을 구현합니다.

과제 1: 스누인턴 3 - 마이페이지 만들기

네비게이션 바에 기존에는 “000님”이 보였으나,
이 부분을 “마이페이지” 버튼으로 변경해주세요.

해당 버튼을 클릭하면 마이페이지로 이동할 수 있도록 해주세요.

마이페이지

관심공고 내 정보

 000 회사	Frontend 개발자 <small>마감</small>
 000 회사	App 개발자 <small>상시모집</small>
 000 회사	Frontend 개발자 <small>D-37</small>

과제 1: 스누인턴 3 - 마이페이지 만들기

마이페이지에 도착했을 때 관심 공고와 내 정보 탭을 볼 수 있도록 해주세요.

기본은 관심공고이며, 내가 북마크한 회사를 볼 수 있도록 해주세요.

마이페이지

관심공고 내 정보

 000 회사	Frontend 개발자 마김
 000 회사	App 개발자 상시모집
 000 회사	Frontend 개발자 D-37

과제 1: 스누인턴 3 - 마이페이지 만들기

[GET /api/post/bookmarks](#)를 요청하여

내가 북마크한 공고 정보를 볼 수 있도록 해주세요.

이때 회사명, 공고제목, 마감여부를 볼 수 있어야 합니다.

마감이면 빨간색 글씨, 상시모집이거나 D-day가 남아있으면 파란색 글씨로 보여주세요.

마이페이지

관심공고 내 정보

	000 회사	Frontend 개발자
		마감

	000 회사	App 개발자
		상시모집

	000 회사	Frontend 개발자
		D-37

과제 1: 스누인턴 3 - 마이페이지 만들기

API 상세

GET api/post/bookmarks

header에 JWT를 담아서 보내기

Request

```
{  
    "posts": [  
        {  
            "id": "string",  
            ...  
            "companyName": "string", // 회사명  
            ...  
            "employmentEndDate": "2025-11-07T02:44:18.901Z", // 채용 마감일  
            "positionTitle": "string", // 공고명  
            ...  
        }  
    ]  
}
```

Response

과제 1: 스누인턴 3 - 마이페이지 만들기

내 정보 탭에 들어가면 나의 프로필 정보를 확인할 수 있어야 합니다.

이때 프로필이 없다면 “내 프로필 생성” 버튼을 보여주세요.

“내 프로필 생성”, “지금 바로 프로필 작성하기” 버튼을 클릭하면
프로필 생성 페이지로 이동하도록 해주세요.

마이페이지

관심공고

내 정보

내 프로필 생성

아직 프로필이 등록되지 않았어요!

기업에 소개할 나의 정보를 작성해서 나를 소개해보세요.

지금 바로 프로필 작성하기

과제 1: 스누인턴 3 - 마이페이지 만들기

내가 프로필을 작성했는지는 [GET api/applicant/me](#)로 확인할 수 있습니다.

만약 해당 API를 쓰았을 때 상세 에러코드가 “APPLICANT_002”인 경우,
아직 프로필이 등록되지 않은 상태입니다.

마이페이지

관심공고 내 정보

내 프로필 생성

아직 프로필이 등록되지 않았어요!

기업에 소개할 나의 정보를 작성해서 나를 소개해보세요.

지금 바로 프로필 작성하기

과제 1: 스누인턴 3 - 마이페이지 만들기

API 상세

GET api/applicant/me (프로필이 없을 경우)

header에 JWT를 담아서 보내기

Request

```
{  
    "timestamp": "...",  
    "message": "...",  
    "code": "APPLICANT_002",  
    "details": {  
        ...  
    }  
}
```

Response

과제 1: 스누인턴 3 - 마이페이지 만들기

프로필 생성 페이지에서는 나의 프로필 정보를 만들 수 있습니다.

학번, 학과, 이력서를 필수로 작성할 수 있어야 합니다.

프로필 생성

필수 작성 항목

아래 항목은 필수로 작성해주세요.

학번 *

25

학번

학과 *

컴공

추가

이력서 (CV) *



PDF 파일만 업로드 가능해요.

저장

뒤로가기

과제 1: 스누인턴 3 - 마이페이지 만들기

제한 조건

- 모든 필수 작성 항목이 정상적으로 작성되지 않았다면, “저장” 버튼을 눌러도 서버에 요청을 보내지 않아야 합니다.
- 학번은 두 자리수의 정수만 입력 가능합니다.
- 학과는 주전공 필수, 복부전공은 최대 6개까지 작성 가능합니다.
- 이력서는 pdf 파일이어야 하며, 최대 5MB까지 등록 가능합니다.

프로필 생성

필수 작성 항목

아래 항목은 필수로 작성해주세요.

학번 *

25

학번

학과 *

컴공

추가

이력서 (CV) *



PDF 파일만 업로드 가능해요.

저장

뒤로가기

과제 1: 스누인턴 3 - 마이페이지 만들기

위 조건을 만족하지 않는 경우에는 오른쪽과 같은 문구가 나타납니다.

필수 작성 항목

아래 항목은 필수로 작성해주세요.

학번 *

25

학번

두 자리 수 숫자로 작성해주세요. (e.g. 25)

학과 *

주전공 학과명을 입력해주세요. (예시: 컴퓨터공학부, 경)

추가

주전공은 필수 작성이며, 다전공은 총 6개 이하로 중복되지 않게 입력해주세요.

이력서 (CV) *



PDF 파일만 업로드 가능해요.

5MB 이하의 PDF 파일을 올려주세요.

과제 1: 스누인턴 3 - 마이페이지 만들기

제한 조건 상세 (학번)

- 두자리수 정수로 작성하도록 해주세요.
 - 입력받은 학번은 20xx 또는 19xx 형태로 변환해주세요.
- e.g. 19 입력 -> 2019로 서버에 전달
e.g. 09 입력 -> 2009로 서버에 전달 (9만 입력은 허용 X)
e.g. 85 입력 -> 1985로 서버에 전달

프로필 생성

필수 작성 항목

아래 항목은 필수로 작성해주세요.

학번 *

25

학번

학과 *

컴공

추가

이력서 (CV) *



PDF 파일만 업로드 가능해요.

저장

뒤로가기

과제 1: 스누인턴 3 - 마이페이지 만들기

제한 조건 상세 (학과)

하나는 무조건 작성, 복부전공은 최대 6개까지 작성하도록 해주세요.
학과를 중복하여 작성하지 않도록 점검해주세요.

- 추가 버튼을 누르면 하위에 작성 가능한 input이 추가적으로 생성됩니다.
- 첫번째 칸: 주전공명을 작성합니다. 삭제 버튼은 보이지 않습니다.
- 두번째 칸: 복부전공명을 작성합니다. 삭제버튼이 보입니다.

프로필 생성

필수 작성 항목

아래 항목은 필수로 작성해주세요.

학번 *

25

학번

학과 *

컴공

학번

추가

학과 *

주전공 학과명을 입력해주세요. (예시: 컴퓨터공학부, 경제학부 등)

삭제

이력서 (CV) *



PDF 파일만 업로드 가능해요.

저장

다전공 학과명을 입력해주세요. (예시: 컴퓨터공학부, 경제학부 등)

삭제

다전공 학과명을 입력해주세요. (예시: 컴퓨터공학부, 경제학부 등)

추가

주전공은 필수 작성이며, 다전공은 총 6개 이하로 중복되지 않게 입력해주세요.

뒤로가기

과제 1: 스누인턴 3 - 마이페이지 만들기

프로필 생성

제한 조건 상세 (학과)

서버에 요청을 보낼 때에는 ',' 로 모든 학과를 묶어서 보내주세요.

이때 주전공은 항상 앞에 있어야 합니다.

e.g. 조경지역시스템공학부 (주전공), 전기정보공학부 (복부전공), 정보문화학 (부전공)

-> “조경지역시스템공학부,전기정보공학부,정보문화학”으로 묶어서 보내기

필수 작성 항목

아래 항목은 필수로 작성해주세요.

학번 *

25

학번

학과 *

컴공

학번

학과 *

주전공 학과명을 입력해주세요. (예시: 컴퓨터공학부, 경제학부 등)

추가

다전공 학과명을 입력해주세요. (예시: 컴퓨터공학부, 경제학부 등)

삭제

이력서 (CV) *



PDF 파일만 업로드 가능해요.

추가

저장

주전공은 필수 작성이며, 다전공은 총 6개 이하로 중복되지 않게 입력해주세요.

뒤로가기

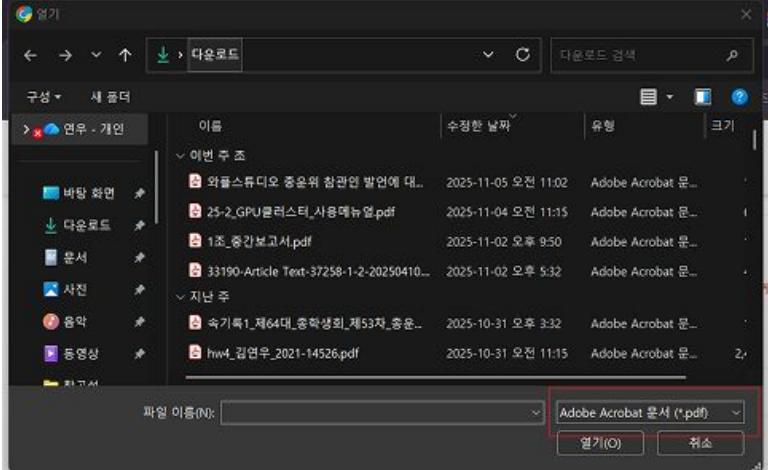
과제 1: 스누인턴 3 - 마이페이지 만들기

제한 조건 상세 (이력서)

이력서에서 “PDF 파일만 업로드 가능해요” 버튼을 누르면 PDF 파일을 업로드할 수 있도록 합니다.

파일 업로드 시 “.PDF” 확장자로 제한되는지 확인해주세요. (오른쪽 위 캡쳐본 참조)

- 업로드한 이후에는 파일명을 확인할 수 있어야 하고,
- “삭제” 버튼을 누르면 업로드한 파일을 삭제할 수 있어야 합니다.



이력서 (CV) *

hw4_김연우_2021-14526.pdf

삭제

저장

뒤로가기

이력서 (CV) *

↑ PDF 파일만 업로드 가능해요.

저장

과제 1: 스누인턴 3 - 마이페이지 만들기

제한 조건 상세 (이력서)

서버에 요청을 보낼 때에는 아래의 형식에 맞추어 보내주세요.

"static/private/CV/{랜덤문자열 10개}_20YYMMDD/{파일 이름}.pdf"

참고) 현재 이력서 파일은 어디에도 저장되지 않습니다.

필수 작성 요건이기 때문에 string 값을 양식에 맞게 넣어줄 뿐, 정상적으로 파일이

저장되지는 않았으니 참고해주세요. (불러오기 등이 불가능)

프로필 생성

필수 작성 항목

아래 항목은 필수로 작성해주세요.

학번 *

25

학번

학과 *

컴공

추가

이력서 (CV) *



PDF 파일만 업로드 가능해요.

이력서 (CV) *

hw4_김연우_2021-14526.pdf

삭제

저장

뒤로가기

과제 1: 스누인턴 3 - 마이페이지 만들기

모든 값이 정상적인 상태에서 “저장” 버튼을 누르면
PUT api/applicant/me로 요청을 보내주세요.

프로필 생성

필수 작성 항목

아래 항목은 필수로 작성해주세요.

학번 *

25

학번

학과 *

컴공

추가

이력서 (CV) *



PDF 파일만 업로드 가능해요.

저장

뒤로가기

과제 1: 스누인턴 3 - 마이페이지 만들기

API 상세

PUT api/applicant/me (프로필 생성 및 수정)

header에 JWT를 담아서 보내기

```
{  
    "enrollYear": 2021, // 19xx, 20xx 형태로 변환해서 전달  
    "department": "조경지역시스템공학부, 전기정보공학부", // ','로 모든 학과를  
    묶어서 전달. 이때 주전공은 맨 앞에  
    "cvKey": "static/private/CV/97272fdf02_20251107/1조_중간보고서.pdf",  
    // 반드시 양식 맞춰서 작성  
}
```

Request

Response

과제 1: 스누인턴 3 - 마이페이지 만들기

프로필을 생성한 이후에는 작성한 프로필을 마이페이지에서 볼 수 있도록 해주세요.

이름, 이메일, 학번, 학과가 모두 보여야 합니다.

이력서는 보이지 않도록 해주세요. (어차피 보여줄 수 있는 방법이 없습니다)

또한 “내 프로필 생성” 버튼 대신 “내 프로필 수정” 버튼이 보이도록 해주세요.

“내 프로필 수정” 버튼을 누르면 수정 페이지로 이동합니다.

마이페이지

관심공고

내 정보

내 프로필 수정

김연우

ywk0524@snu.ac.kr

조경지역시스템공학부 · 컴퓨터공학부(복수전공) 21학번

과제 1: 스누인턴 3 - 마이페이지 만들기

API 상세

GET api/applicant/me (프로필이 있을 경우)

header에 JWT를 담아서 보내기

```
{  
    "id": "string",  
    "name": "string", // 내 이름  
    ...  
    "email": "string", // 등록 시 사용한 이메일  
    "enrollYear": 0, // 학번, 2021 등의 형태로 내려옴.  
    "department": "string", // 학과, `,'로 뮤여서 내려옴.  
    ...  
    "cvKey": "string", // 이력서 링크, 사용하지 않음.  
    ...  
}
```

Request

Response

과제 1: 스누인턴 3 - 마이페이지 만들기

프로필 수정 페이지 UI는 생성과 동일합니다.

단, 내가 이미 작성했던 내용들이 미리 들어가 있어야 합니다.
(이력서는 제외)

학번, 학과는 내가 사전에 작성했던 내용에 맞게 들어가도록 설정해주세요.

HINT: 수정 페이지에서 GET
api/applicant/me를 써주면 사전에
들어가야 하는 데이터를 얻을 수 있습니다.

HINT: ','로 묶어서 들어오는 응답을 모두
풀어서 state 안에 넣어줘야 합니다.

필수 작성 항목

아래 항목은 필수로 작성해주세요.

학번 *

25

학번

학과 *

컴공

조경

삭제

추가

이력서 (CV) *

인턴하샤_이력서.pdf

삭제

과제 1: 스누인턴 3 - 마이페이지 만들기

API 상세

PUT api/applicant/me (프로필 생성 및 수정)

header에 JWT를 담아서 보내기

```
{  
    "enrollYear": 2021, // 19xx, 20xx 형태로 변환해서 전달  
    "department": "조경지역시스템공학부, 전기정보공학부", // ','로 모든 학과를  
    묶어서 전달. 이때 주전공은 맨 앞에  
    "cvKey": "static/private/CV/97272fdf02_20251107/1조_중간보고서.pdf",  
    // 반드시 양식 맞춰서 작성  
}
```

Request

Response

과제 1: 스누인턴 3 - 마이페이지 만들기

선택 스펙) 선택 작성 폼 만들기

선택 작성 폼을 생성합니다.

선택스펙 1: 희망직무, 기술스택, 한줄소개 부분 생성하기 + 마이페이지에서 보이게 하기

선택스펙 2: 자기소개, 기타소개 링크 부분 생성하기 + 마이페이지에서 보이게 하기

둘 다 성공하면 + 1일 추가 지급

E.G. (희망직무, 기술스택, 한줄소개) 또는 (자기소개, 기타소개 링크) 중 한 파트만 구현 → Grace day 1일 지급

E.G. 희망직무 ~ 기타소개 링크 모두 구현 → Grace day 3일 지급

과제 1: 스누인턴 3 - 마이페이지 만들기

선택 스펙) 선택 작성 품 만들기

희망직무: “추가” 버튼 클릭 시 input 추가, “삭제” 버튼 클릭 시 input 삭제

기술스택: 엔터를 누르면 태그 추가, 태그에서 “x” 버튼 누르면 태그 삭제

한 줄 소개, 자기소개: 단순 텍스트 작성

기타 소개 링크: 링크 제목+링크가 하나로 묶여서 input처럼 관리됨.

선택 작성 항목

아래 항목은 필수로 작성하지 않아도 괜찮지만, 작성해 주시면 채용 담당자가 지원자의 강점을 이해하는 데 더욱 도움이 되어요.

희망 직무

희망 직무를 입력해주세요. (예시: 웹 프론트)

추가
삭제

기술 스택

사용할 수 있는 상세 기술 스택을 입력해주세요.(최대 10개)

기술 스택은 엔터로 구분되어 한 개당 최대 30자까지 입력할 수 있어요.

한 줄 소개

한 줄 소개를 입력해주세요.

나를 소개하는 한마디를 입력해주세요.

자기소개

자신에 대한 상세 소개를 작성해주세요.

[예시 작성 문장]

- 전공 및 지원 분야에 대한 관심
- 참여한 프로젝트등의 관련 경험
- 성격적 강점
- 팀 협업 경험

0/5000

기타 소개 링크

링크 제목을 작성해주세요. (e.g. 깃허브)

<https://>

추가
X

깃허브, 링크드인, 개인 홈페이지 등 자신을 소개할 수 있는 기타 링크를 첨부해주세요.

서장

뒤로가기

선택 작성 항목

아래 항목은 필수로 작성하지 않아도 괜찮지만, 작성해 주시면 채용 담당자가 지원자의 강점을 이해하는 데 더욱 도움이 되어요.

희망 직무

희망 직무를 입력해주세요. (예시: 웹 프론트)

웹 프론트
삭제

웹 프론트

희망 직무를 입력해주세요. (예시: 웹 프론트)

추가
삭제

기술 스택

기술 스택은 엔터로 구분되어 한 개당 최대 30자까지 입력할 수 있어요.

기술 스택을 선택해주세요.

기술 스택은 엔터로 구분되어 한 개당 최대 30자까지 입력할 수 있어요.

기준 대고 중복되지 않는 100자 이내의 직무명을 작성해주세요.

한 줄 소개

한 줄 소개는 한마디를 입력해주세요.

나를 소개하는 한마디를 입력해주세요.

110/100

자기소개

한 줄 소개는 100자 이내로 작성해주세요.

0/5000

기타 소개 링크

안녕하세요

https://

5544/5000

상세 소개는 5000자 이내로 작성해주세요.

110/100

기타 소개 링크

깃허브, 링크드인, 개인 홈페이지 등 자신을 소개할 수 있는 기타 링크를 첨부해주세요.

중복되지 않는 유익한 링크와 100자 이내의 설명글을 입력해주세요.

외부 링크 링크는 최대 5개까지 입력 가능하며 링크는 https://로 시작해야 합니다.

과제 1: 스누인턴 3 - 마이페이지 만들기

희망직무 input

- 입력 안해도 무방
- “추가” 버튼을 누르면 최하단에 input 새로 생성
- “삭제” 버튼을 누르면 해당 직무 삭제됨.
- 중복된 값 입력 불가, 100자 이내로 작성해야 함.
- 서버에 전달할 때에는 string[] 값으로 전달

선택 작성 항목

아래 항목은 필수로 작성하지 않아도 괜찮지만, 작성해 주시면 채용 담당자가 지원자의 강점을 이해하는 데 더욱 도움이 되어요.

희망 직무

희망 직무를 입력해주세요. (예시: 웹 프론트)

삭제

추가

선택 작성 항목

아래 항목은 필수로 작성하지 않아도 괜찮지만, 작성해 주시면 채용 담당자가 지원자의 강점을 이해하는 데 더욱 도움이 되어요.

희망 직무

웹 프론트

삭제

웹 프론트

삭제

추가

중복되지 않는 100자 이내의 직무명을 작성해주세요.

과제 1: 스누인턴 3 - 마이페이지 만들기

기술스택 input

- 입력 후 엔터키를 입력하면 input 상단에 태그가 보임.
- 태그는 최대 30자
- 만약 input에 중복되거나 30자 이상의 글자가 입력되면 하단에 에러메세지 보이도록 하기
- 이때 엔터를 눌러도 태그로 추가되지 않음.
- 태그 옆 “x” 버튼을 누르면 해당 태그 삭제
- 서버에 전달할 때에는 string[]으로 전달

기술 스택

사용할 수 있는 상세 기술 스택을 입력해주세요.(최대 10)

기술 스택은 엔터로 구분되며 한 개당 최대 30자까지 입력할 수 있어요.

기술 스택

안녕

반가워요

반가워요

기술 스택은 엔터로 구분되며 한 개당 최대 30자까지 입력할 수 있어요.

기존 태그와 중복되지 않는 30자 이하의 기술 스택을 작성해주세요.

과제 1: 스누인턴 3 - 마이페이지 만들기

한줄소개, 자기소개 input

- 글자수 제한 넘는지 확인 필요
- “자기소개” input에서는 줄바꿈이 반영되어야 함.

한 줄 소개

한 줄 소개를 입력해주세요.

나를 소개하는 한마디를 입력해주세요.

나를 소개하는 한마디를 입력해주세요.

110/100

한 줄 소개는 100자 이내로 작성해주세요.

자기소개

자신에 대한 상세 소개를 작성해주세요.

[예시 작성 문항]

- 전공 및 지원 분야에 대한 관심
- 참여한 프로젝트등의 관련 경험
- 성격적 강점
- 팀 협업 경험

0/5000

5544/5000

상세 소개는 5000자 이내로 작성해주세요.

과제 1: 스누인턴 3 - 마이페이지 만들기

기타 소개 링크 input

- “추가” 버튼을 누르면 링크제목 input과 링크 input이 묶여서 나타나야 함.
- “x” 버튼을 누르면 해당 링크 삭제
- 링크제목이나 링크가 중복되지 않아야 함.
- 링크는 https://~로 시작해야 함.
- 서버에 전달할 때에는 {description: string; link: string}[] 형태로 전달

기타 소개 링크

링크 제목을 작성해주세요. (e.g. 깃허브)

https://

추가

깃허브, 링크드인, 개인 홈페이지 등 자신을 소개할 수 있는 기타 링크를 첨부해주세요.

저장

뒤로가기

기타 소개 링크

안녕하세요

https

추가

깃허브, 링크드인, 개인 홈페이지 등 자신을 소개할 수 있는 기타 링크를 첨부해주세요.

중복되지 않는 유효한 링크와 100자 이내의 설명글을 입력해주세요.

외부 소개 링크는 최대 5개까지 입력 가능하며 링크는 https로 시작해야 합니다.

과제 1: 스누인턴 3 - 마이페이지 만들기

API 상세

PUT api/applicant/me (프로필 생성 및 수정)

header에 JWT를 담아서 보내기

```
{  
    "enrollYear": 9999, // 학번, 20xx 또는 19xx 형태  
    "department": "string", // 학과, 모든 학과 정보를 ','로 뮤어서 전달  
    "positions": [  
        "string"  
    ], // 희망 직무, 직군 정보를 list에 담아서 전달  
    "slogan": "string", // 한 줄 소개  
    "explanation": "string", // 자기소개  
    "stacks": [  
        "string"  
    ], // 기술 스택, 각각의 태그를 list에 담아서 전달  
    "cvKey": "string", // 이력서 링크  
    "links": [  
        {  
            "description": "string",  
            "link": "string"  
        }  
    ] // 기타 외부 링크, 링크 설명과 링크 주소를 객체로 뮤은 뒤 list에 담아서  
    // 전달  
}
```

Request

Response

과제 1: 스누인턴 3 - 마이페이지 만들기

화면은 [피그마](#)에 나와 있으니 참고해주세요.

[인턴하샤 페이](#)지에서도 위 과제의 구현된 버전을 확인하실 수 있습니다.

코드 구현이 너무 어렵다면 [인턴하샤 웹 클라이언트 깃허브](#)를 참고해보세요.

과제 1: 스누인턴 3 - 마이페이지 만들기

11/21 (금) 오후 8시 전까지 스누인턴 3 구현 내용을 Frontend-잡담 방에 올려주세요.
인원수가 3명인 조도 필수스펙만 구현하셔도 무방합니다.

과제 제출 시 선택 스펙으로 어떤 것을 구현하셨는지 같이 적어주세요.

과제 2: 조원들과 모각작하기

11/21 (금) 오후 8시 전까지 조원들과 함께 모각작 인증샷을 찍어 “Frontend-잡담”에 업로드해주세요.