Multi Step Form, 중앙 집권에서 자율 조직으로

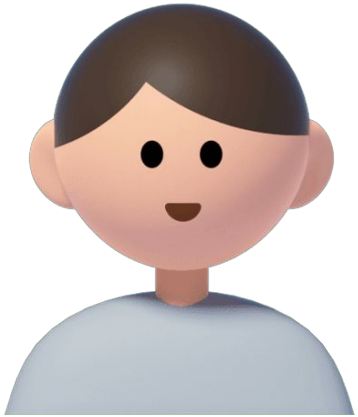
Multi Step Form 구현을 위한 Bottom-Up Approach

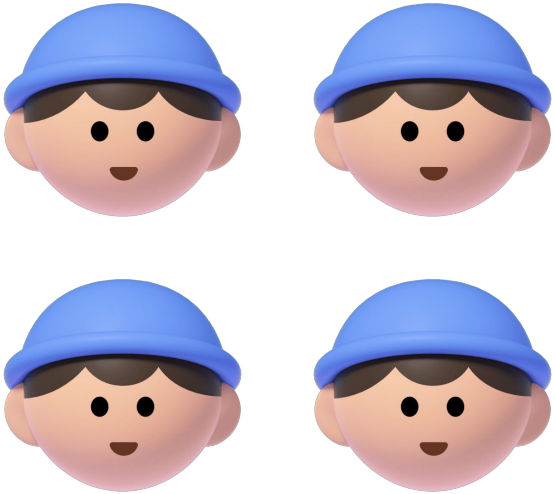
토스코어 Product Platform Team / 박종호

Frontend Diving Club 6th (2025.02.07)

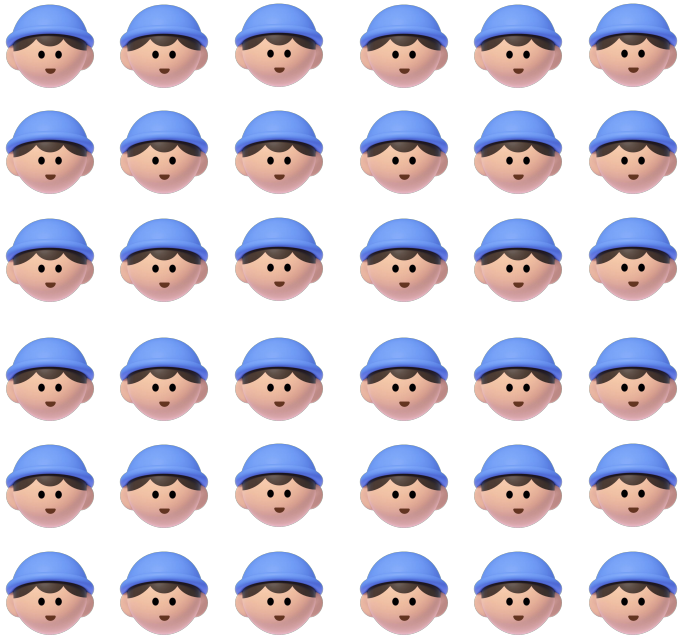
Metaphor is a programmer's weapon

조직 구조 이야기

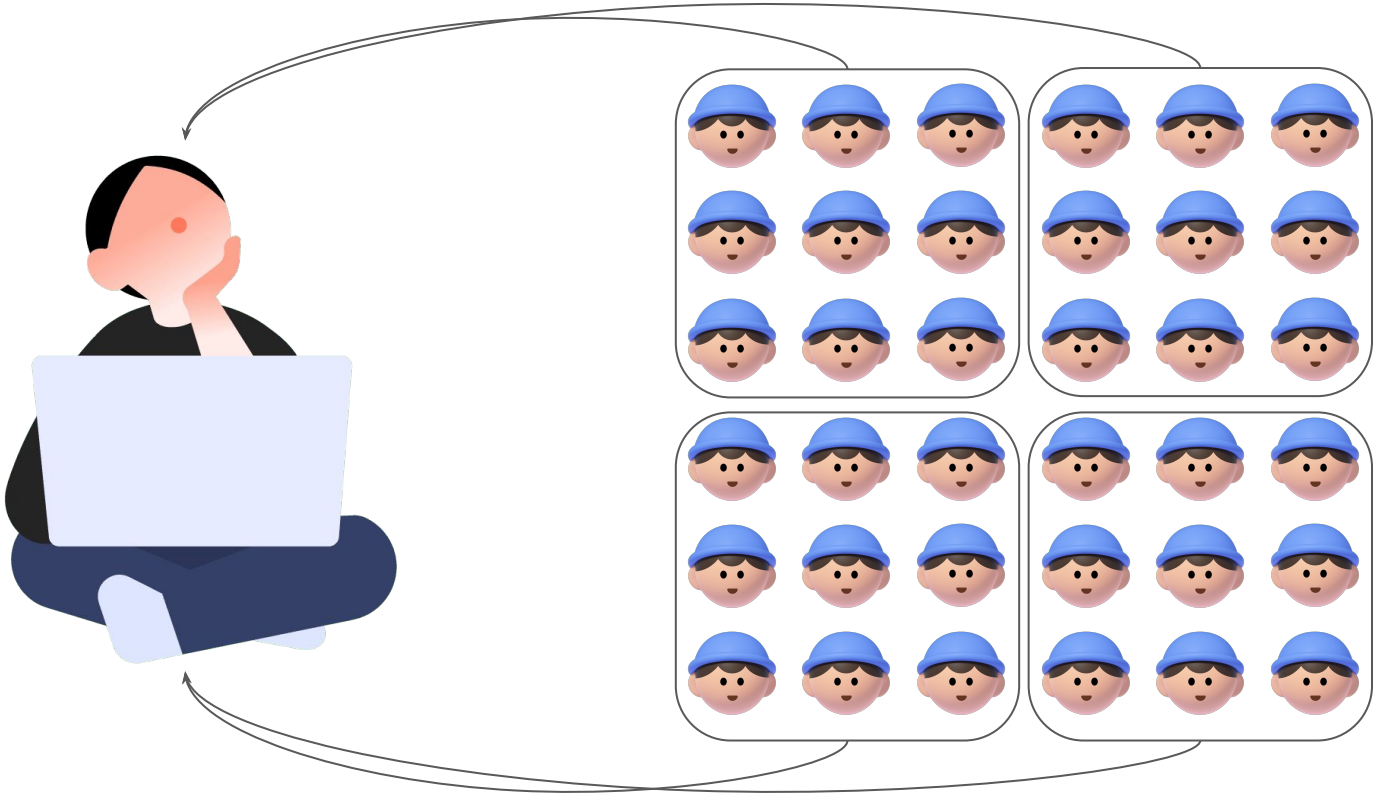




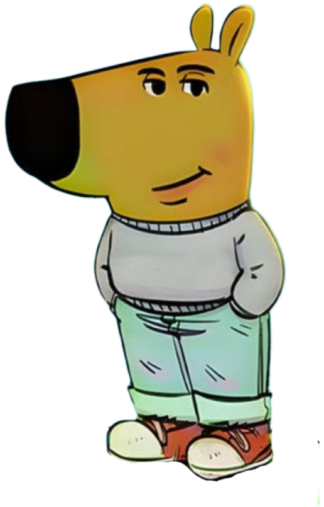
조직 구조 이야기

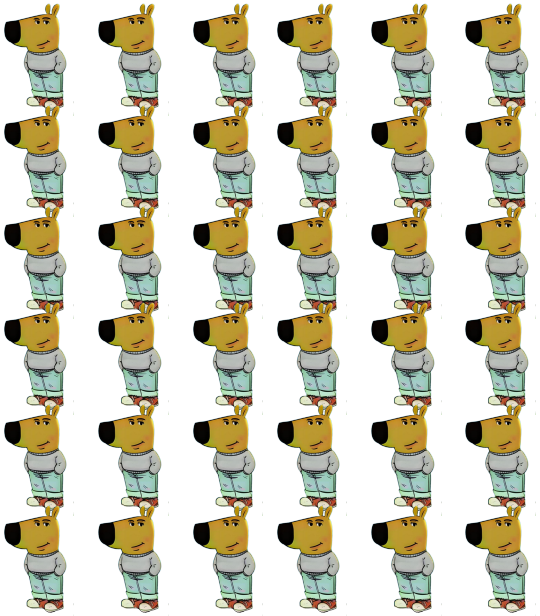


조직 구조 이야기



조직 구조 이야기





Multi-Step Form?

Multi Step Form

Multi Step Form

```
const [payload, setPayload] = useState<FormPayload>({
  email: 'jane.doe@johndoehub.com',
  password: 'janedoe',
  name: 'Jane Doe',
  age: 14,
  gender: 'male',
});
```

payload 선언

Multi Step Form

```tsx
const [payload, setPayload] = useState<FormPayload>({
  email: 'jane.doe@johndoehub.com',
  password: 'janedoe',
  name: 'Jane Doe',
  age: 14,
  gender: 'male',
});
```

payload 선언

```tsx
{match(currentStep)
  .with(Step.UserName, () => {
    const [firstName, lastName] = payload.name.split(' ');

    return (
      <UserNameStep
        value={{ firstName, lastName }}
        onChange={(value) => {
          setPayload((prev) => ({ ...prev, name: `${value.firstName} ${value.lastName}` }));
        }}
      />
    );
  })
  .with(Step.Account, () => (
    <AccountStep
      value={payload}
      onChange={(value) => {
        setPayload((prev) => ({ ...prev, ...value }));
      }}
    />
  ))
  .with(Step.AdditionalInfo, () => (
    <AdditionalInfoStep
      value={payload}
      onChange={(value) => {
        setPayload((prev) => ({ ...prev, ...value }));
      }}
    />
  ))
  .exhaustive()}
```

Step 컴포넌트가 변경하는
값을 payload 에 업데이트

Multi Step Form

```tsx
const [payload, setPayload] = useState<FormPayload>({
  email: 'jane.doe@johndoehub.com',
  password: 'janedoe',
  name: 'Jane Doe',
  age: 14,
  gender: 'male',
});
```

payload 선언

```tsx
{match(currentStep)
  .with(Step.UserName, () => {
    const [firstName, lastName] = payload.name.split(' ');

    return (
      <UserNameStep
        value={{ firstName, lastName }}
        onChange={(value) => {
          setPayload((prev) => ({ ...prev, name: `${value.firstName} ${value.lastName}` }));
        }}
      />
    );
  })
  .with(Step.Account, () => (
    <AccountStep
      value={payload}
      onChange={(value) => {
        setPayload((prev) => ({ ...prev, ...value }));
      }}
    />
  ))
  .with(Step.AdditionalInfo, () => (
    <AdditionalInfoStep
      value={payload}
      onChange={(value) => {
        setPayload((prev) => ({ ...prev, ...value }));
      }}
    />
  ))
  .exhaustive()}
```

Step 컴포넌트가 변경하는
값을 payload 에 업데이트

```tsx
{!isFirstStep && (
  <button
    onClick={() => {
      match(currentStep)
        .with(Step.Account, () => setCurrentStep(Step.UserName))
        .with(Step.AdditionalInfo, () => setCurrentStep(Step.Account))
        .exhaustive();
    }}
  >
    이전
  </button>
)}

{!isLastStep && (
  <button
    disabled={!isStepValid}
    onClick={() => {
      match(currentStep)
        .with(Step.UserName, () => setCurrentStep(Step.Account))
        .with(Step.Account, () => setCurrentStep(Step.AdditionalInfo))
        .exhaustive();
    }}
  >
    다음
  </button>
)}

<button type="submit" disabled={!isFormValid} style={{ display: 'block' }}>
  제출
</button>
```

버튼 관련
유효성 로직 추가

문제점

```
const [firstName, lastName] = payload.name.split(' ');

return (
  <UserNameStep
    value={{ firstName, lastName }}
    onChange={(value) => {
      setPayload((prev) => ({ ...prev, name: `${value.firstName} ${value.lastName}` }));
    }}
  />
);
```

Payload 와 Step 의 인터페이스가 다른 경우,

이를 변환하는 로직을

Form 컴포넌트에서 작성해줘야 함

문제점

```tsx
const [firstName, lastName] = payload.name.split(' ');

return (
  <UserNameStep
    value={{ firstName, lastName }}
    onChange={(value) => {
      setPayload((prev) => ({ ...prev, name: `${value.firstName} ${value.lastName}` }));
    }}
  />
);
```

Payload 와 Step 의 인터페이스가 다른 경우,

이를 변환하는 로직을

Form 컴포넌트에서 작성해줘야 함

```tsx
const isUserNameStepValid = !(payload.name === '');
const isEmailStepValid = !(payload.email === '' || payload.password === '');
const isAdditionalInfoStepValid = !(isNaN(payload.age ?? NaN) || payload.gender == null);
const isFormValid = isUserNameStepValid && isEmailStepValid && isAdditionalInfoStepValid;

const [currentStep, setCurrentStep] = useState<Step>(Step.UserName);
const isFirstStep = currentStep === Step.UserName;
const isLastStep = currentStep === Step.AdditionalInfo;
const isStepValid =
  currentStep === Step.UserName
    ? isUserNameStepValid
    : currentStep === Step.Account
    ? isEmailStepValid
    : currentStep === Step.AdditionalInfo
    ? isAdditionalInfoStepValid
    : true;
```

버튼 유효성 로직을

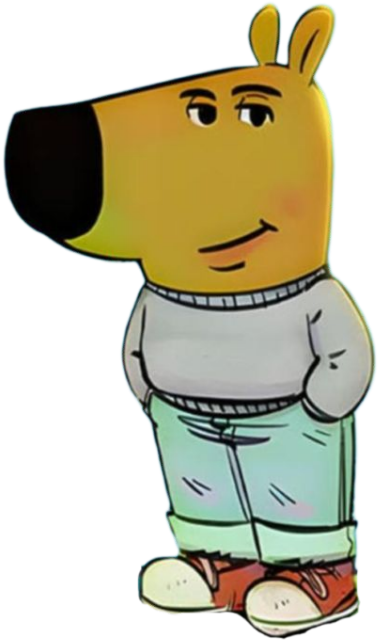
Form 컴포넌트에서 작성해줘야 함

***Field, Step* 이 더 많아진다면?**

***Form* 에 비즈니스 로직이 더 추가된다면?**

Bottom Up Approach

- 왜 혼자서 모든걸 다 하려고 그래
- 각 팀의 성과가 조직의 성과고
- 각 팀의 성공이 조직의 성공이잖아
- 너는 팀이 필요한 예산과 목표만 관리해



Bottom Up Approach

- 폼 Payload = 모든 Step Payload 를 조합한 값
- 폼이 유효하다 = 모든 Step 이 유효하다

Bottom Up Approach - Step

- StepValue
  - Step 이 가져야 할 값
- serialize
  - Step 의 값이 Payload 에 어떻게 담겨야 하는가?
- validate
  - Step 의 값이 유효한가?

```
type StepValue = Record<string, unknown>;

type Serialize = (value: StepValue) => Record<string, unknown>;

type Validate = (value: StepValue) => boolean;
```

Bottom Up Approach - Step

성을 입력해주세요.

이름을 입력해주세요.

다음

제출

```
const value = { firstName: '', lastName: '' };

const serialize = ({ firstName, lastName }) => ({ name: `${firstName} ${lastName}` });

const validate = ({ firstName, lastName }) => firstName.length > 0 && lastName.length > 0;
```

Bottom Up Approach - Step

- Jotai 를 사용
- **valueAtom**
  - step payload 가 담긴 atom
- **serializeAtom**
  - valueAtom 에 기반해 payload 에 담길 형태로 변환한 Atom
- **validateAtom**
  - valueAtom 에 기반해 해당 값이 유효한지 판별하는 Atom

```ts
export default step<UserName, FormPayload>(UserNameStep, {
  stepId: Step.UserName,
  valueAtom: ({ initialValue }) => {
    const [firstName = '', lastName = ''] = (initialValue?.name ?? '').split(' ');

    return atom<UserName>({ firstName, lastName });
  },
  serializeAtom: ({ valueAtom }) =>
    atom((get) => {
      const { firstName, lastName } = get(valueAtom);

      return { name: `${firstName} ${lastName}` };
    }),
  validateAtom: ({ valueAtom }) =>
    atom((get) => {
      const { firstName, lastName } = get(valueAtom);

      return firstName.length > 0 && lastName.length > 0;
    }),
});
```

Bottom Up Approach - Form

valueAtoms

- 모든 step 의 valueAtom, serializeAtom, validateAtom 을 정의

```
const valueAtoms = useMemo(() => {
  const valueAtoms: Record<string, StepAtom> = {};

  Object.entries(steps).forEach(([stepId, step]) => {
    const valueAtom = step.valueAtom({ initialValue });
    const serializeAtom = step.serializeAtom({ valueAtom });
    const validateAtom = step.validateAtom?.({ valueAtom }) ?? atom(true);

    valueAtoms[stepId] = { valueAtom, serializeAtom, validateAtom };
  });

  return valueAtoms;
}, [initialValue, steps]);
```

Bottom Up Approach - Form

payloadAtom

- valueAtoms 의 값 중 serializeAtom 만 모아서 payload 를 정의

```
const payloadAtom = useMemo(() => {
  return atom<Payload>((get) => {
    const data = Object.values(valueAtoms).reduce((acc, step) => {
      return { ...acc, ...get(step.serializeAtom) };
    }, {});

    return data;
  });
}, [valueAtoms]);
```

Bottom Up Approach - Form

validateAtom

- valueAtoms 의 값 중 validateAtom 만 모아서 폼 유효성을 정의

```
const validateAtom = useMemo(() => {
  return atom<boolean>((get) => {
    const data = Object.values(valueAtoms).every((step) => {
      return get(step.validateAtom ?? atom(true));
    });

    return data;
  });
}, [valueAtoms]);
```

Bottom Up Approach - Form

```tsx
{match(currentStep)
  .with(Step.UserName, (stepId) => (
    <StepComponent<UserName> stepId={stepId}>
      {({ value, onChange }) => <UserNameStep value={value} onChange={onChange} />}
    </StepComponent>
  ))
  .with(Step.Account, (stepId) => (
    <StepComponent<Account> stepId={stepId}>
      {({ value, onChange }) => <AccountStep value={value} onChange={onChange} />}
    </StepComponent>
  ))
  .with(Step.AdditionalInfo, (stepId) => (
    <StepComponent<AdditionalInfo> stepId={stepId}>
      {({ value, onChange }) => <AdditionalInfoStep value={value} onChange={onChange} />}
    </StepComponent>
  ))
  .exhaustive()}
```

```tsx
const payload = useFormPayload();
const isFormValid = useFormValidation();

const [currentStep, setCurrentStep] = useState<Step>(Step.UserName);
const isFirstStep = currentStep === Step.UserName;
const isLastStep = currentStep === Step.AdditionalInfo;
const isStepValid = useFormStepValidation(currentStep);
```

Step 은 자신의 값(valueAtom)만 변경
Payload 는 업데이트 하지 않음

폼의 유효성= 이전에 만든 validateAtom 으로 판단
스텝의 유효성= 각 스텝의 validateAtom 으로 판단

마무리

There is no silver bullet.

마무리

- 참고 자료
  - https://codesandbox.io/p/sandbox/dazzling-leftpad-4wzs3y
  - https://codesandbox.io/p/devbox/bottom-up-approach-multi-step-form-xf7kz2
    - (codesandbox 계정이 있다면)
- 장표
  - https://speakerdeck.com/pumpkiinbell/multi-step-form-decentralized-autonomous-organization