FJTR

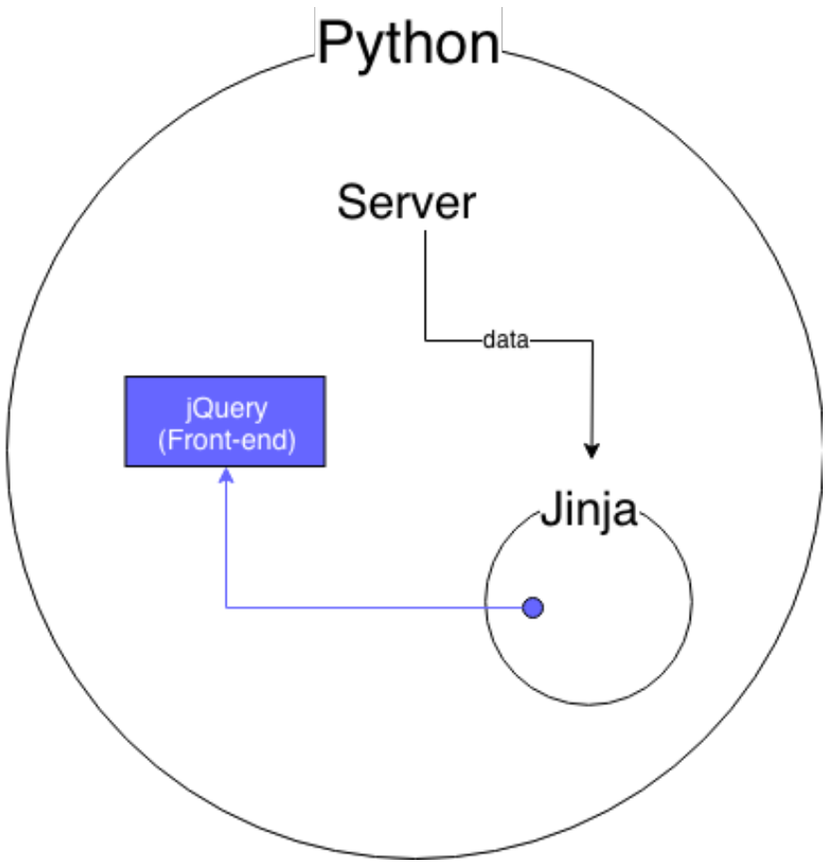
From jQuery To React

FJTR

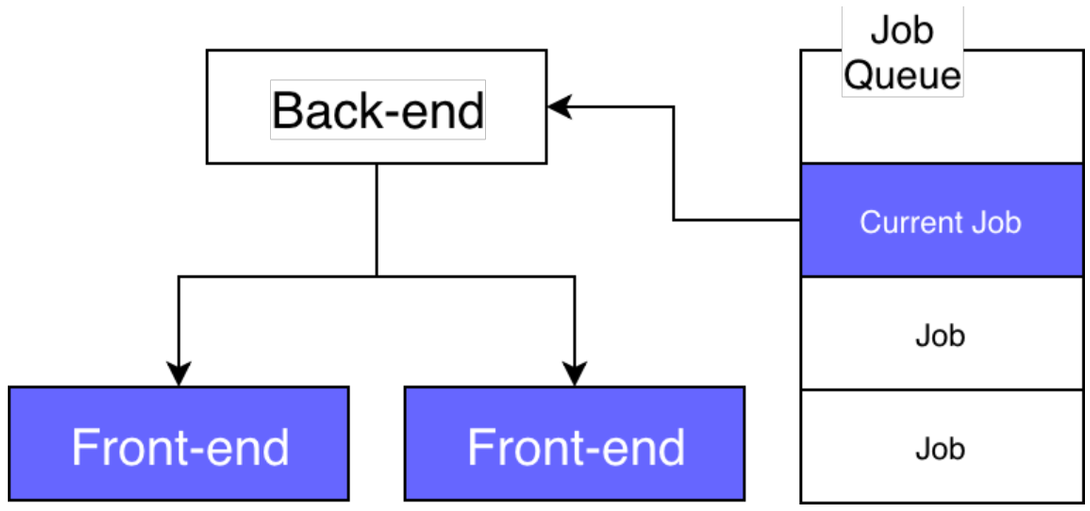
쏟아지는 업무를 거스르는 리팩토링 노하우

원티드 초기 구조

2015년 10월 당시



프론트엔드 구조



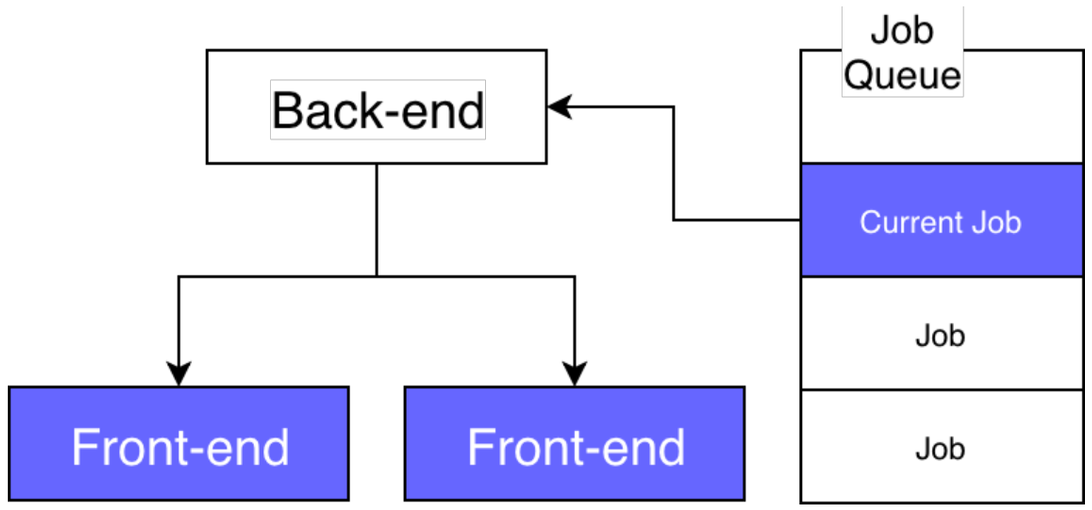
업무 진행 방식

원티드 초기 구조

2015년 10월 당시

이슈 하나당 한명 이상의
서버 개발자가 필요

비동기 통신을 사용하지 않는
정적 웹사이트를 지향



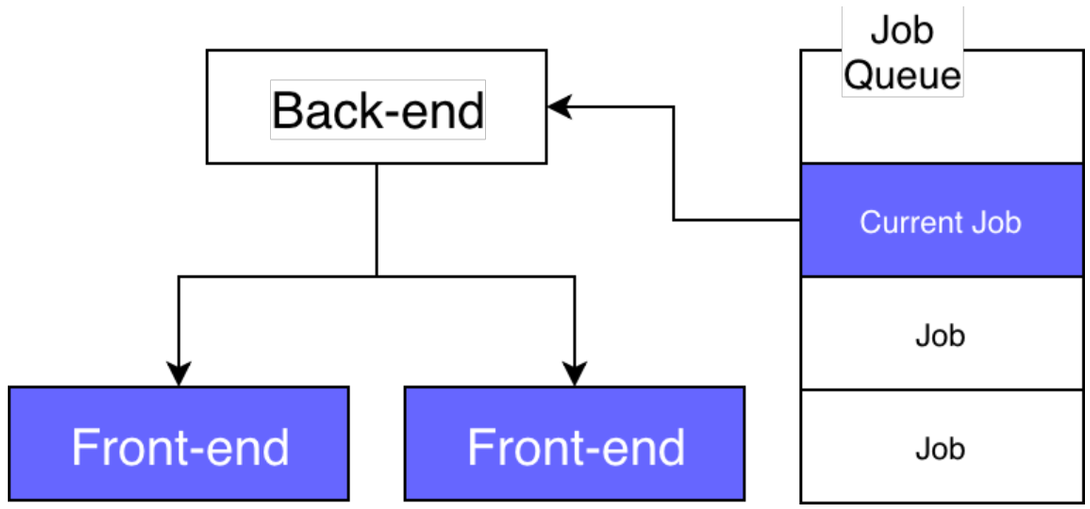
업무 진행 방식

원티드 초기 구조

2015년 10월 당시

이슈 하나당 한 명 이상의
서버 개발자가 필요

비동기 통신을 사용하지 않는
정적 웹사이트를 지향



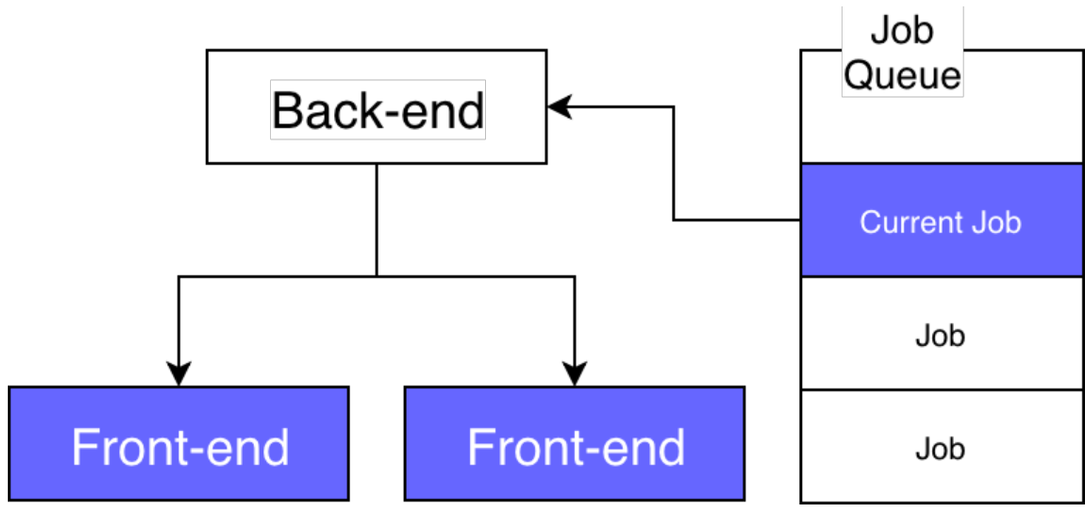
업무 진행 방식

원티드 초기 구조

2015년 10월 당시

이슈 하나당 한 명 이상의
서버 개발자가 필요

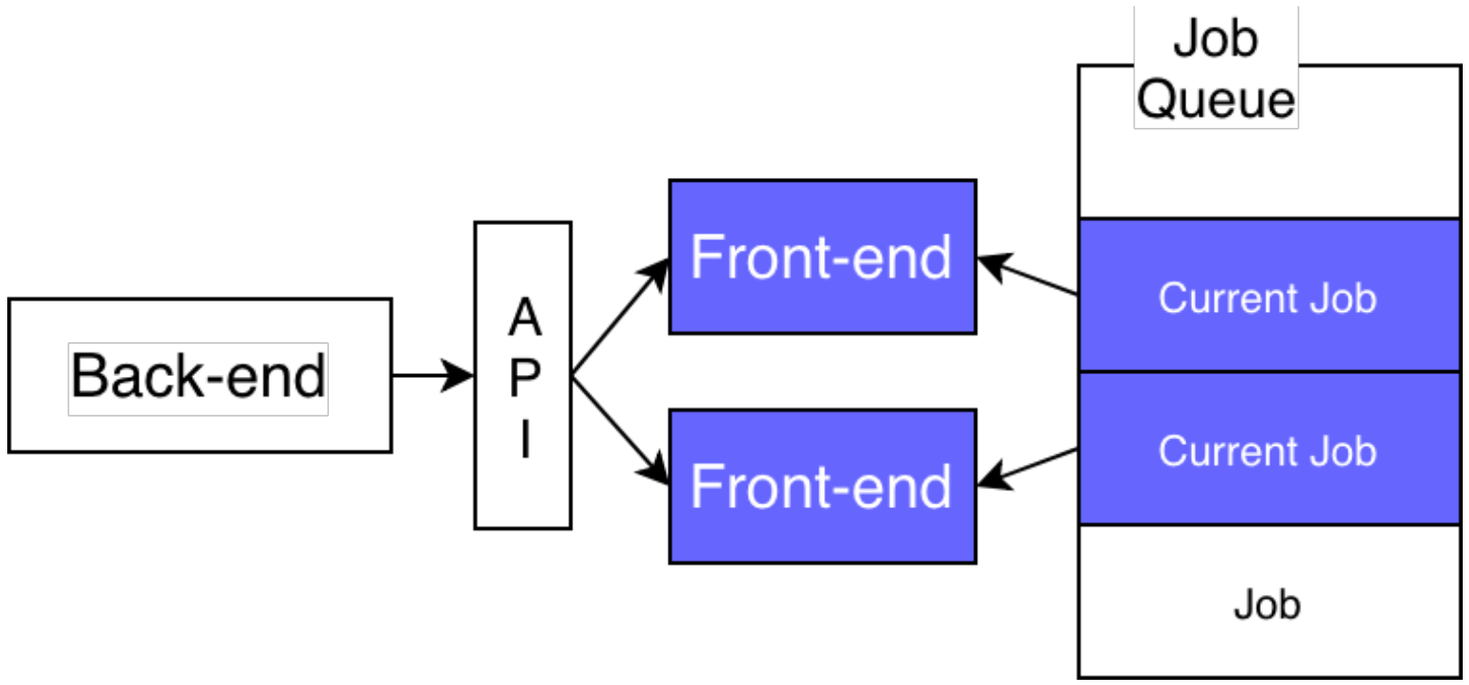
비동기 통신을 사용하지 않는
정적 웹사이트를 지향



업무 진행 방식

원티드 초기 구조

2015년 10월 당시

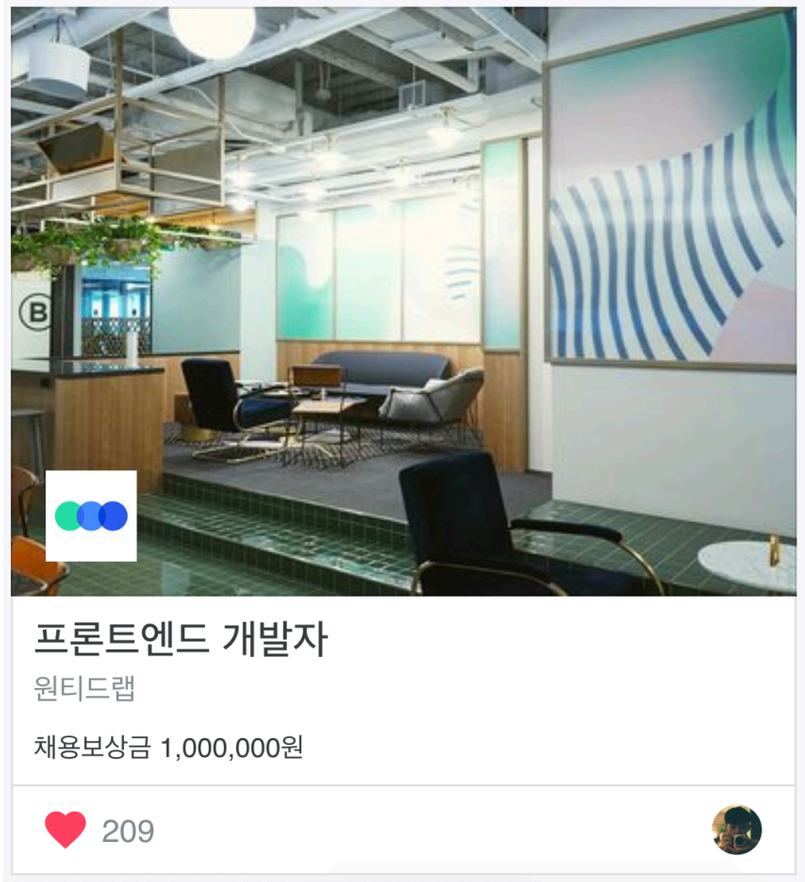


새로운 업무 진행 방식

프론트엔드 문제점

1. 모듈 재사용이 불가능한 구조
2. 심각한 UI 컴포넌트 파편화
3. 빌드 환경의 부재

프론트엔드 문제점

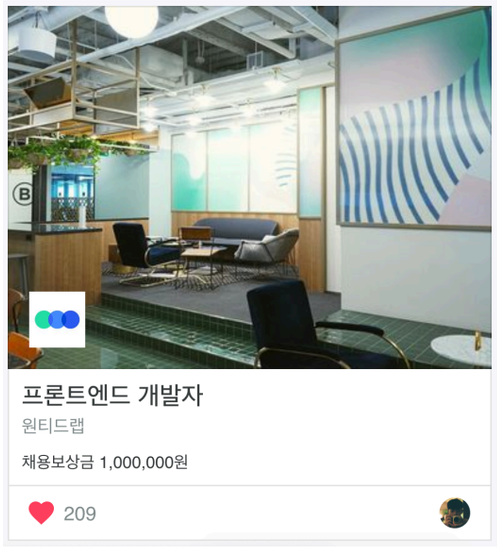


원티드 대표 컴포넌트

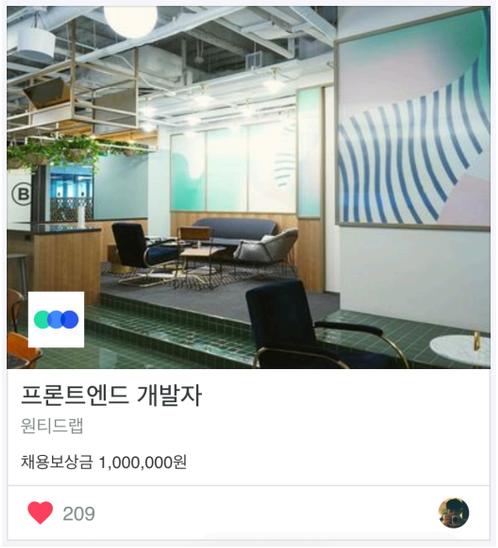
잡카드

프론트엔드 문제점

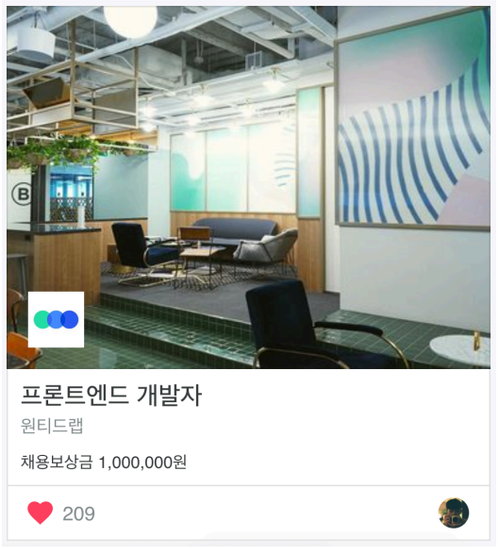
wd.html



wdlist.html



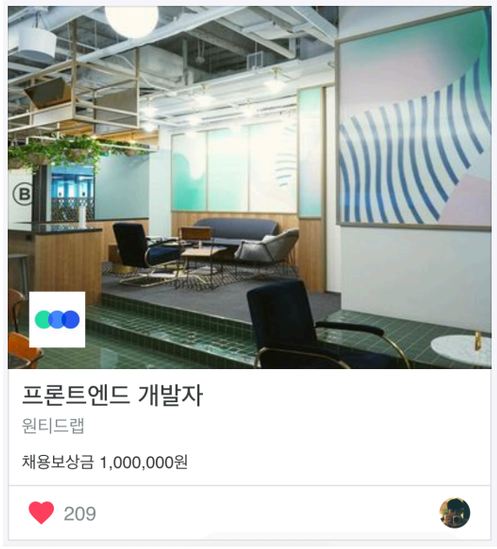
profile.html



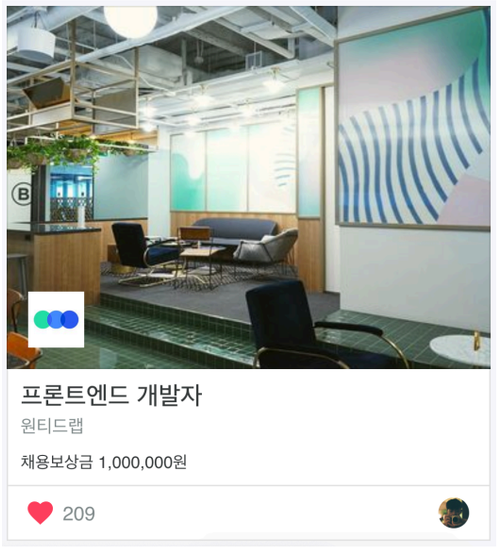
하나의 컴포넌트가 각각의 HTML 파일에 따로 정의되어 있음

프론트엔드 문제점

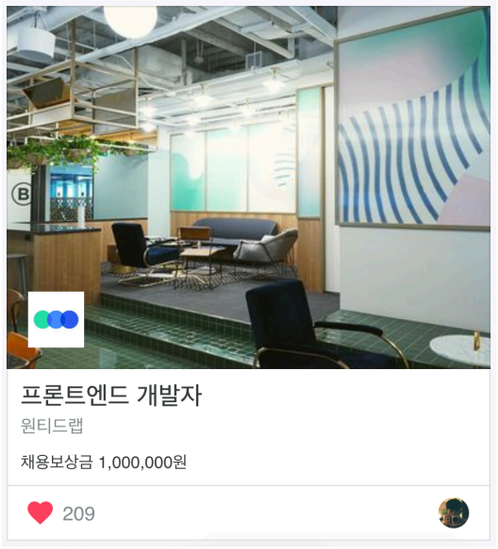
wd.html



wdlist.html



profile.html



모듈 재사용을 코드를 CTRL+C CTRL+V하면서 해결하고 있었다...

프론트엔드 문제점

재사용 가능한 모듈이 너무나도 절실하다!

왜 React인가?

1. JSX 형태로 구현된 Virtual DOM이 있다
2. 기존 Jinja 템플릿과 혼합해서 사용할 수 있다
3. 컴포넌트를 재사용하기 쉬운 구조를 갖고 있다

왜 React인가?

```
const ReactDOM = require('react-dom/server');

ReactDOM.renderToString(<div>hello, world</div>);
```

왜 React인가?

```
const ReactDOM = require('react-dom/server');
ReactDOM.renderToString(<div>hello, world</div>);
```

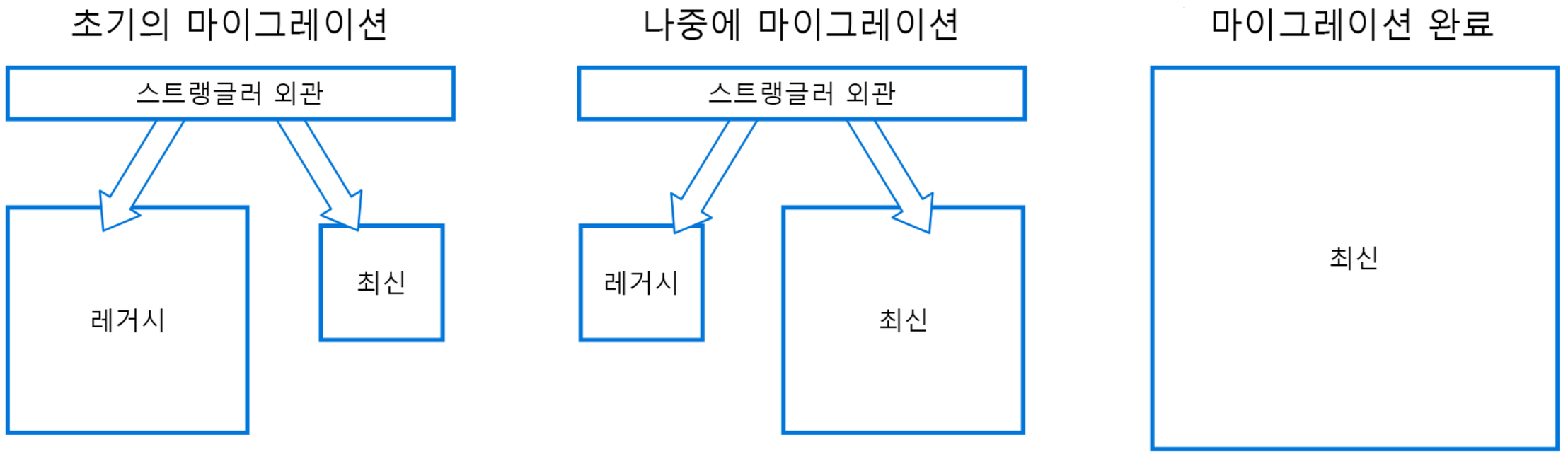
이론상 서버 렌더링이 가능

왜 React인가?

1. JSX 형태로 구현된 Virtual DOM이 있다.
2. 기존 Jinja 템플릿과 혼합해서 사용할 수 있다.
3. 컴포넌트를 재사용하기 쉬운 구조를 갖고 있다.

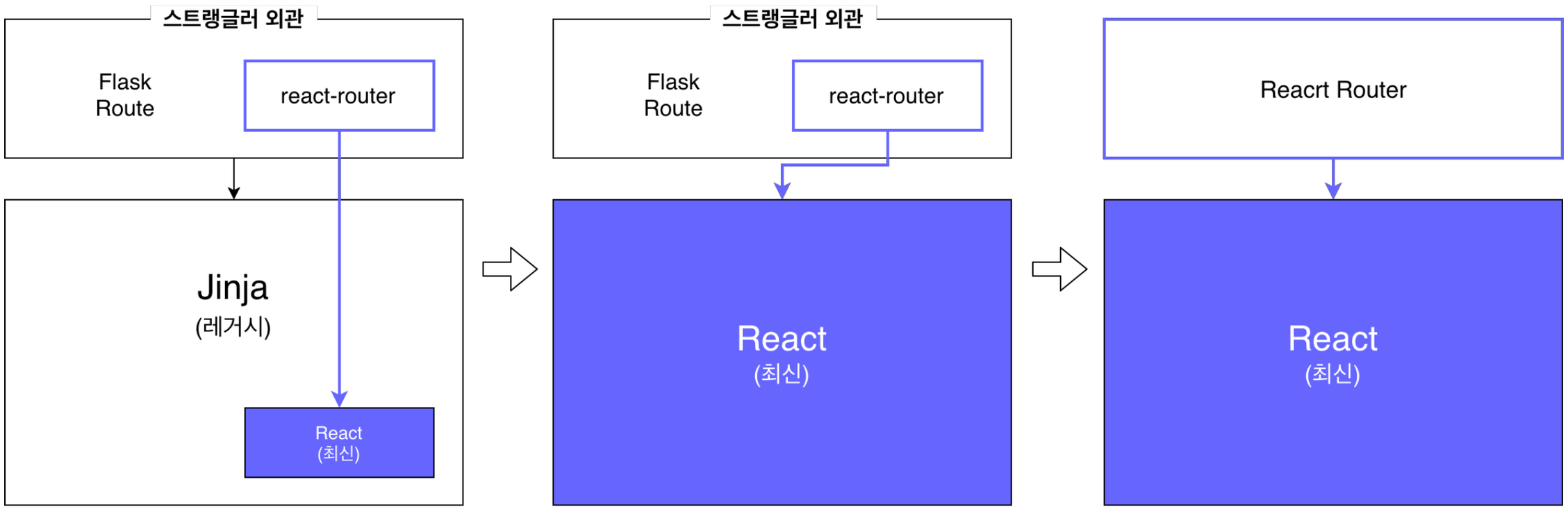
FJTR

스트랭글러 패턴을 이용한 리팩토링



FJTR

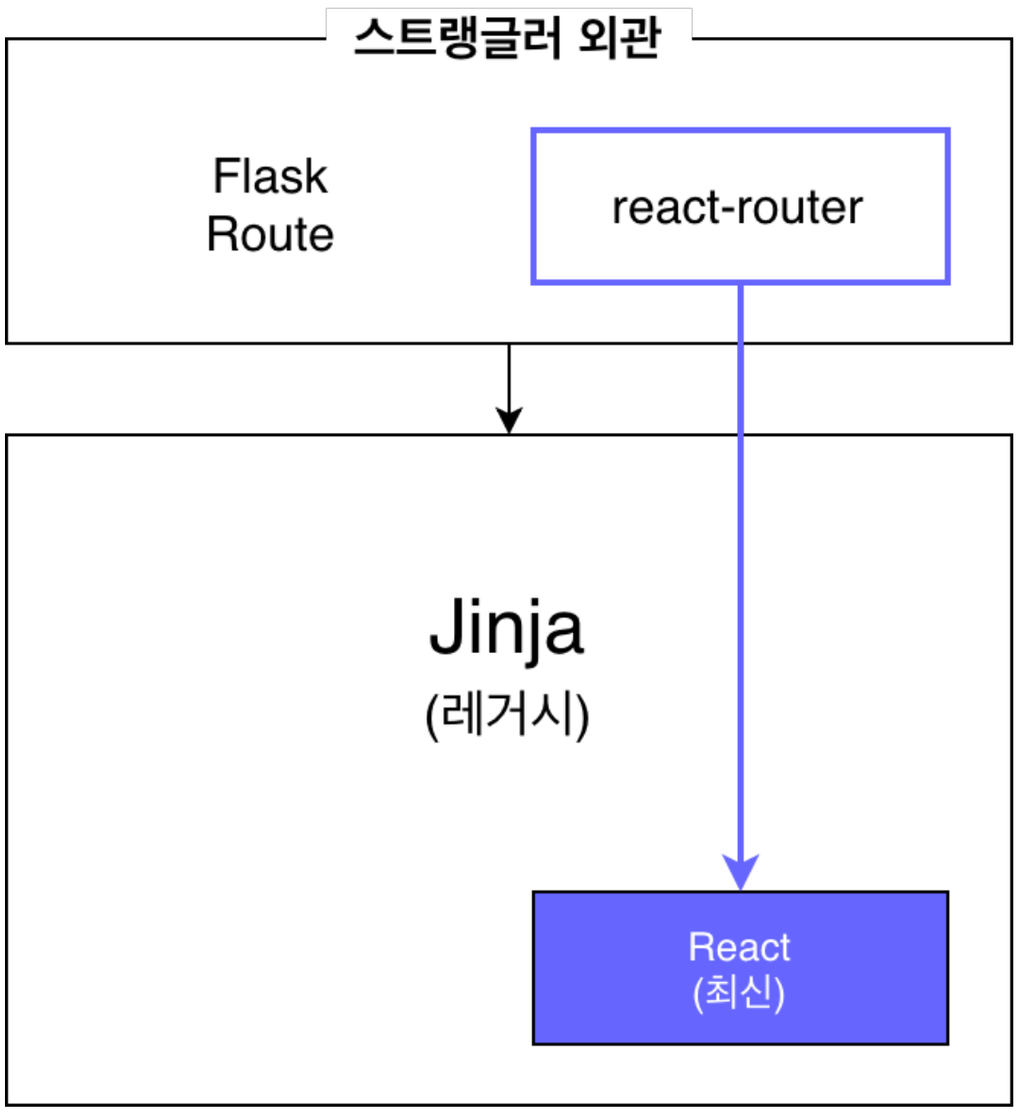
스트랭글러 패턴을 이용한 리팩토링



FJTR 리팩토링의 진행과정

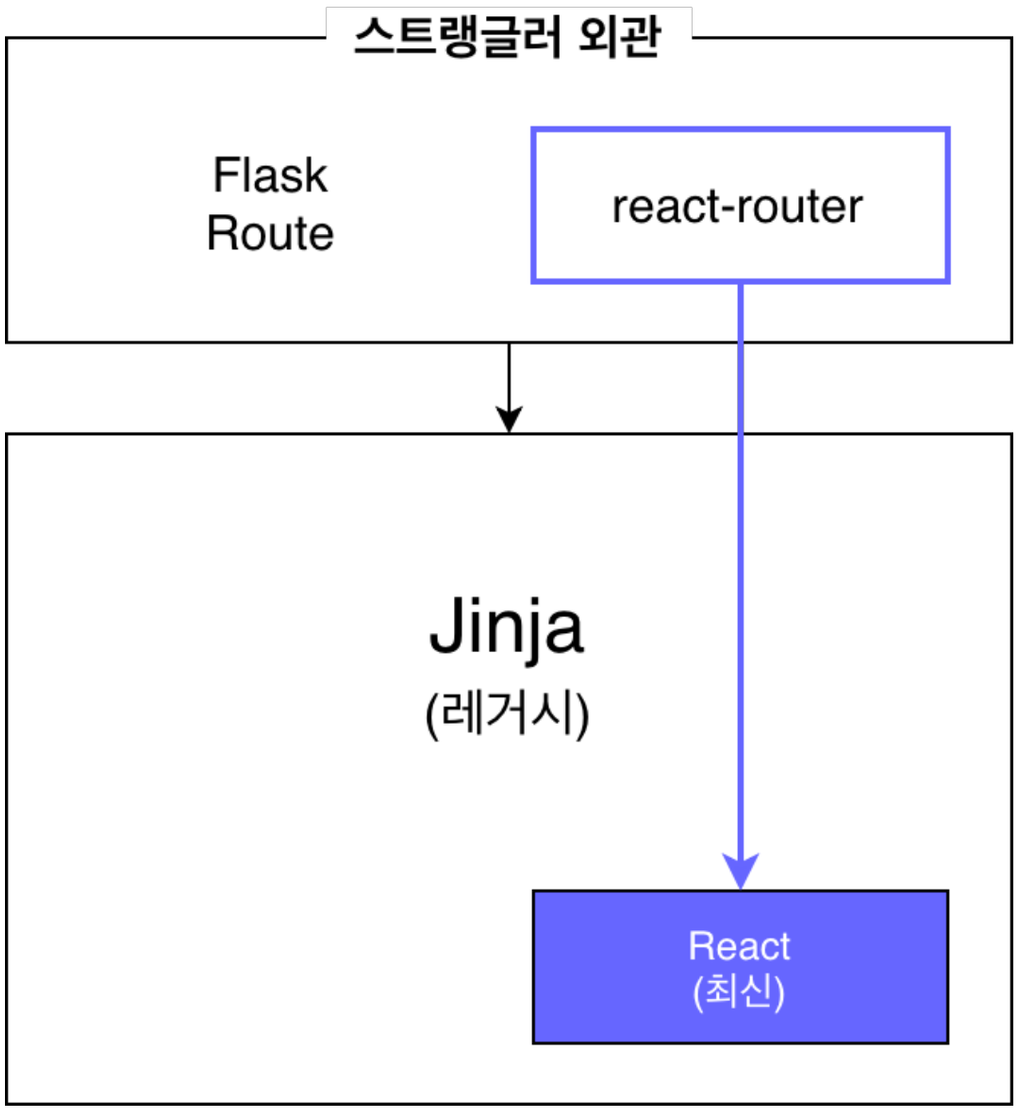
FJTR

스트랭글러 패턴을 이용한 리팩토링



FJTR

스트랭글러 패턴을 이용한 리팩토링



스트랭글러 패턴은 외관을
안정적으로 구현하는 것이 중요

FJTR

스트랭글러 패턴을 이용한 리팩토링

1. **서비스 초기다보니 비교적 구조가 단순함**
2. **jQuery 코드를 유지보수하는 것보다 React로 다시 만드는 것이 훨씬 편리함**

FJTR

리팩토링 이후 성과

1. **대만, 일본, 싱가폴, 홍콩 진출**

2. **많은 기능들을 높은 생산성으로 구현**
   대시보드, 어드민, 매칭모드, 이력서 작성 도구, 추천사, 연봉정보, 이벤트 등

FJTR

리팩토링 이후 발생했던 이슈들

1. 기술부채
2. 큰 번들파일과 속도저하
3. SPA 구조로 인한 SEO의 어려움
4. 서버 개발자 사이의 회색영역

FJTR

리팩토링 이후 발생했던 이슈들

All-in-One 해결책은

서버 사이드 렌더링

FJTR

원티드의 기술부채

둘중 무엇이 좋은 코드일까?

```
const isMe = (name) => {
  const myName = 'riverleo';

  return name === myName;
};
```

A

```
function isMe (name) {
  var myName = ‘riverleo’;

  if (myName === name) {
    return true;
  }

  return false;
}
```

B

FJTR

원티드의 기술부채

기술 부채란 코드를 이해할 수 없거나
받아들일 수 없는 상태

FJTR

원티드의 기술부채

기술 부채란 코드를 이해할 수 없거나 받아들일 수 없는 상태

결과적으로 개인의 능력이나 라이브러리 선택과는 다른 이슈

FJTR

원티드의 기술부채

코드를 다수가 이해할 수 있는 방식으로
바꾸면 부채를 해소할 수 있다

FJTR

원티드의 기술부채

1. 문서화
2. 테스트
3. 함수형 프로그래밍

FJTR

원티드의 기술부채

1. 문서화

2. 테스트

3. 함수형 프로그래밍

이야기가 길어지는 이유로 여기까지만...

FSTS

NextJS 기반의 서버 사이드 렌더링

FSTS

NextJS 기반의 서버 사이드 렌더링

5개국에 서비스 중인 상태에서
리팩토링 해야 하는 부담감

FSTS

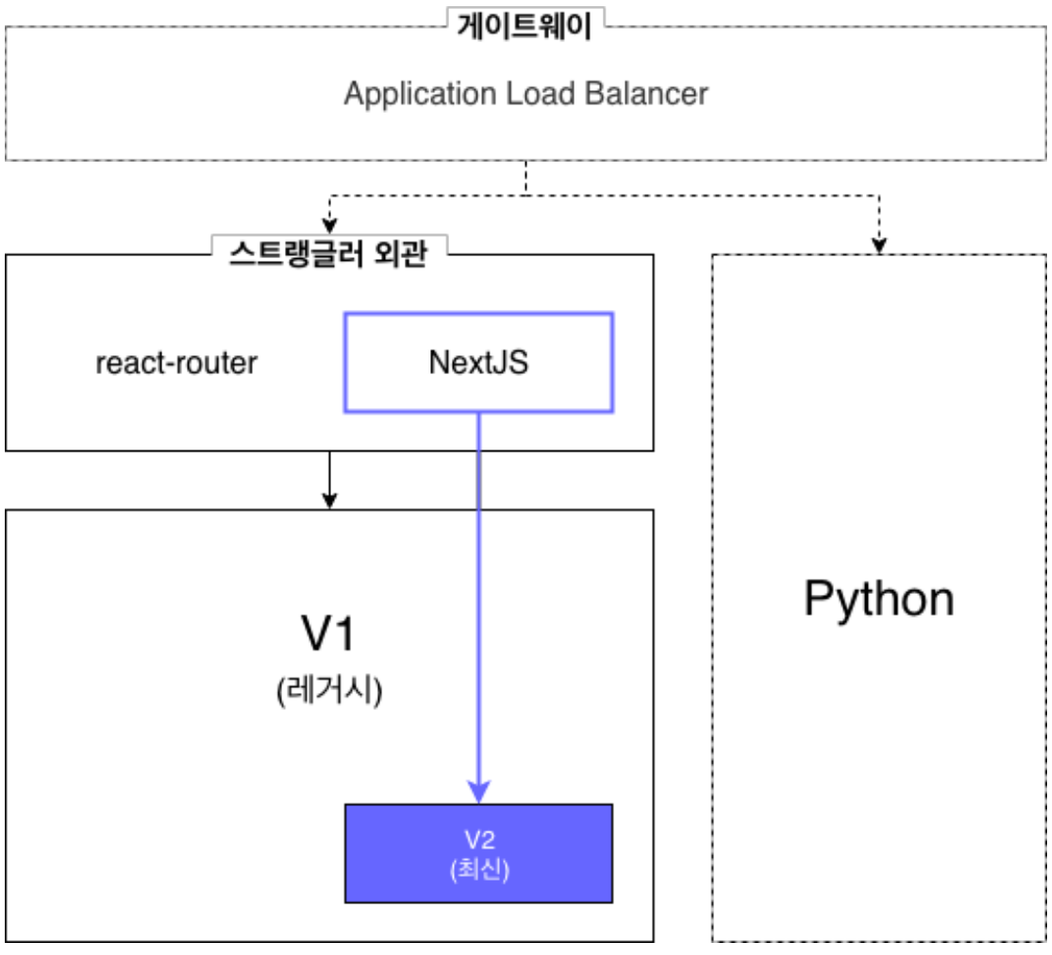
NextJS 기반의 서버 사이드 렌더링

이번에도 역시

스트랭글러 패턴

FSTS

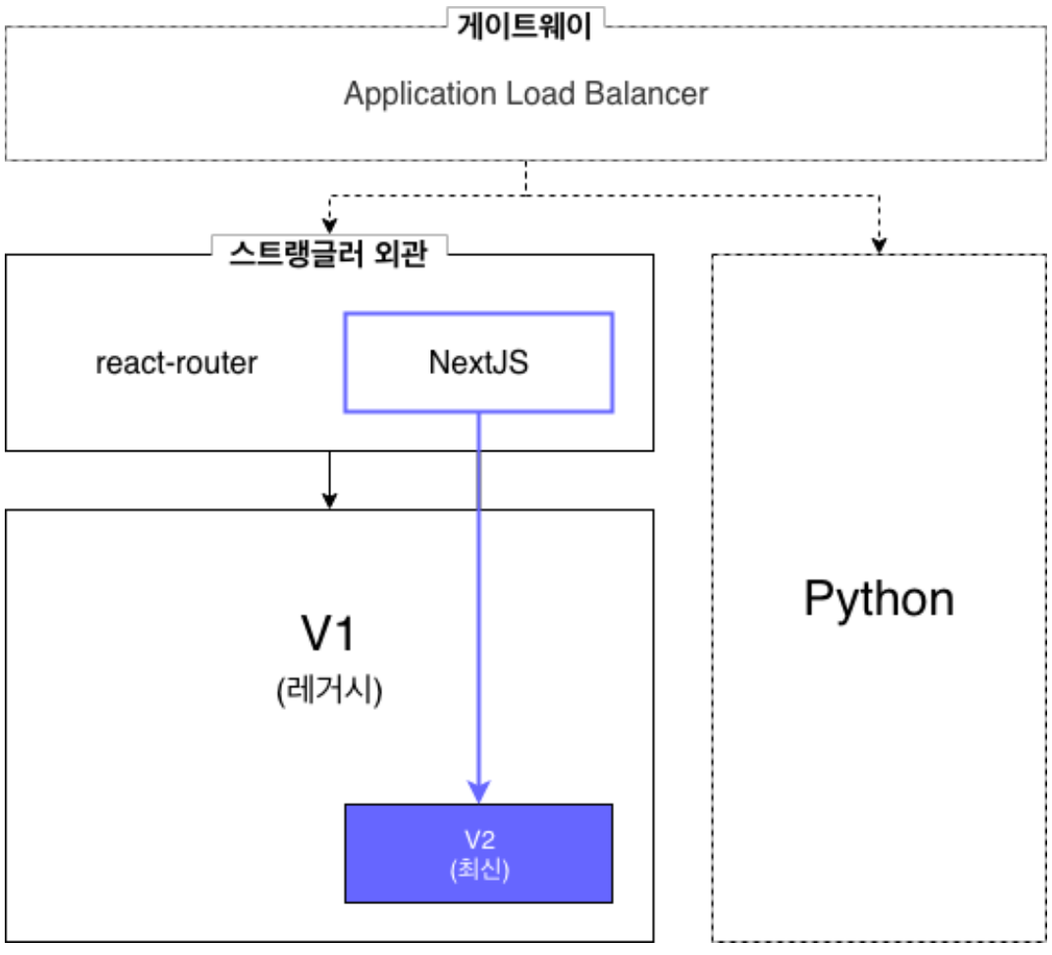
NextJS 기반의 서버 사이드 렌더링



NextJS 기반의 스트랭글러 외관

FSTS

NextJS 기반의 서버 사이드 렌더링

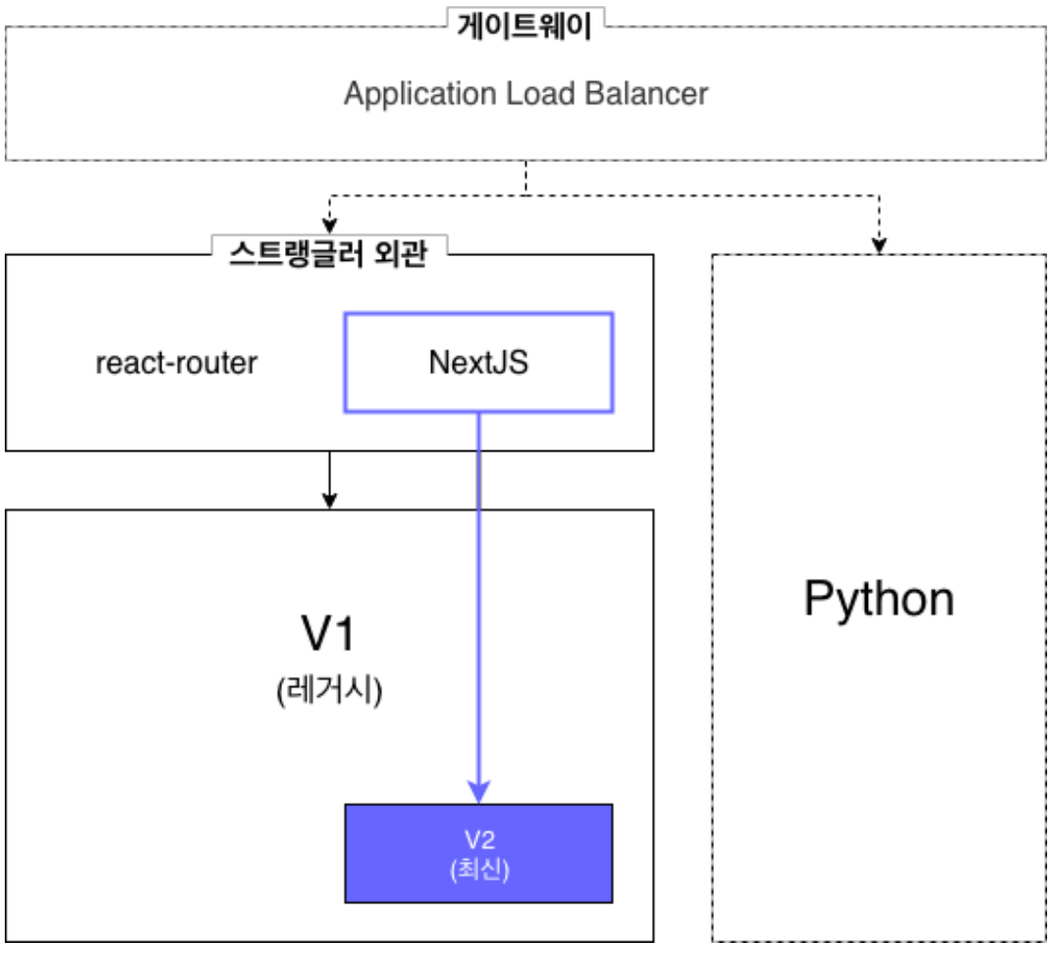


스트랭글러 패턴은 외관을
안정적으로 구현하는 것이 중요

NextJS 기반의 스트랭글러 외관

FSTS

NextJS 기반의 서버 사이드 렌더링



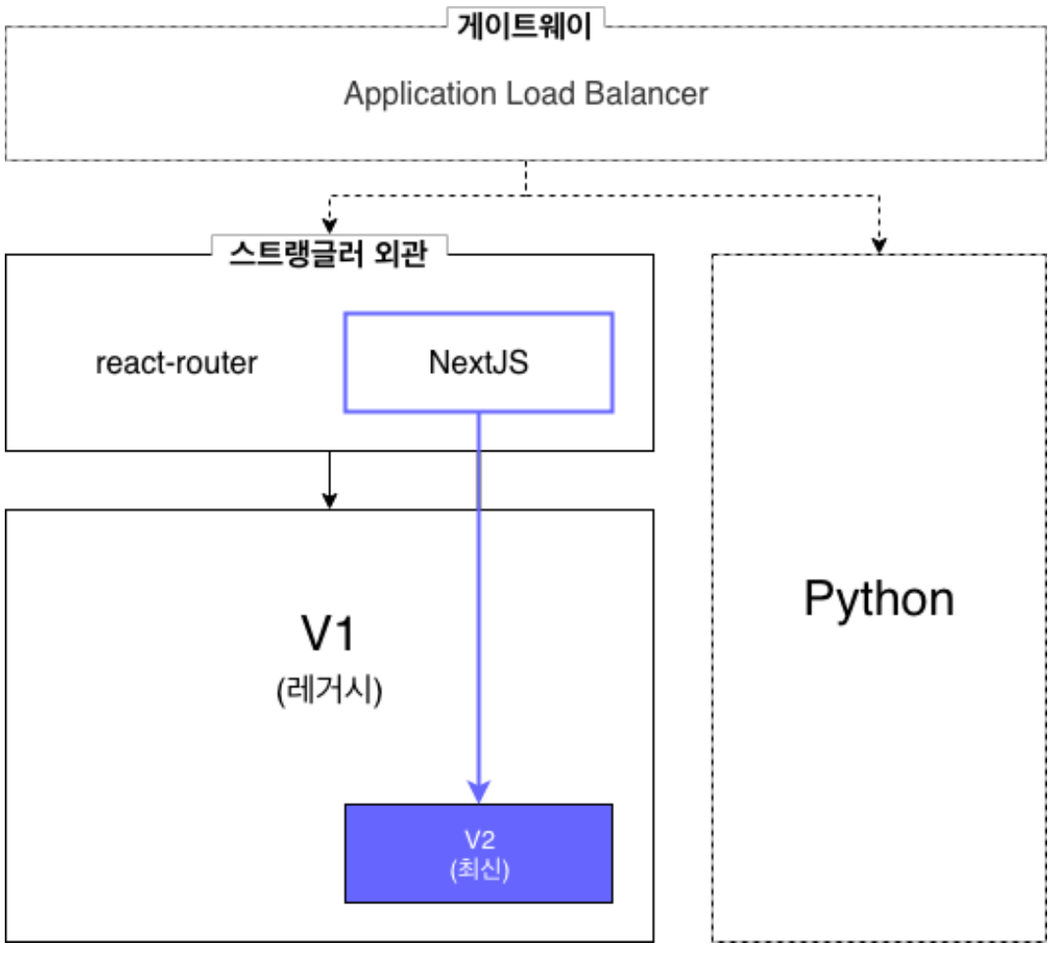
스트랭글러 패턴은 외관을
안정적으로 구현하는 것이 중요

스트랭글러 외관을 구현하는데만
2달 넘게 테스트와 수정 작업을 반복

NextJS 기반의 스트랭글러 외관

FSTS

NextJS 기반의 서버 사이드 렌더링



내부적인 우려도 많았지만
원티드 고인물의 저력으로 강행돌파

NextJS 기반의 스트랭글러 외관

왜 NextJS인가?

1. 풍부한 예제
2. next/head
3. HMR 기본 탑재
4. Code Splitting과 Dynamic Import 기본 탑재

왜 NextJS인가?

풍부한 예제

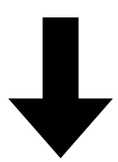
무려 170+개가 넘는 예제를 이용해서
원하는 구성으로 사용할 수 있다

왜 NextJS인가?

next/head

```
import Head from 'next/head';

export default () => (
  <div>
    <Head>
      <title>A</title>
      <meta name="viewport" content=“initial-scale=1.0" />
    </Head>
    <Head>
      <title>B</title>
      <meta name="viewport" content=“initial-scale=1.2" />
    </Head>
    <p>Hello world!</p>
  </div>
);
```



```
<title>B</title>
<meta name="viewport" content="initial-scale=1.2" />
```

왜 NextJS인가?

- **zero configuration**
- **공식 플러그인 지원**
  @zeit/next-sass, @zeit/typescript 등

이 외에도 무수히 많은 기능들을 제공

물론 **NextJS**만 사용한다고

모두 해결되는게 아닙니다

캐싱없는 서버 렌더링은

속도가 오히려 느릴 수도 있어요

캐싱

웹서비스 초기화에 필요한
수많은 데이터를 캐싱없이 로드한다면 ...

Status	Type	Initiator	Size	Time
200	docum...	Other	43.5 KB	3.45 s

3.45s

NextJS만 사용한 경우

캐싱

Status	Type	Initiator	Size	Time
200	docum…	Other	43.5 KB	25 ms

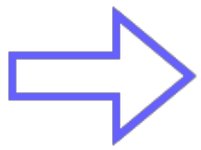
25ms

캐싱 적용 후

캐싱

프로덕션 환경에서도

Status	Type	Initiator	Size	Time
200	docum…	(index)	2.7 KB	541 ms



Initiator	Size	Time
Other	11.8 KB	56 ms

541ms -> 56ms

최대 속도는 **26ms**

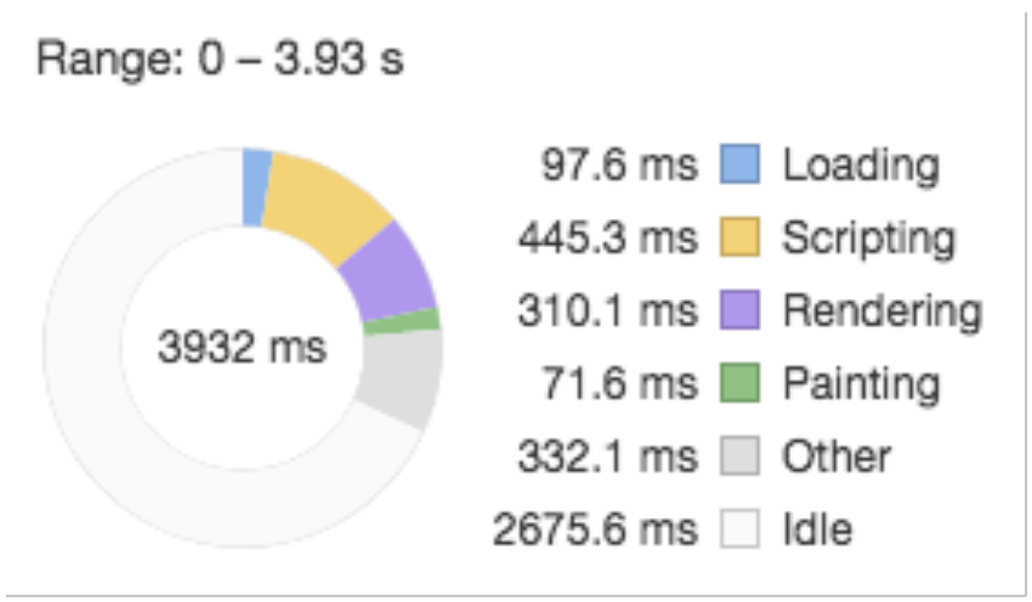
캐싱

그렇다면 서버렌더링으로

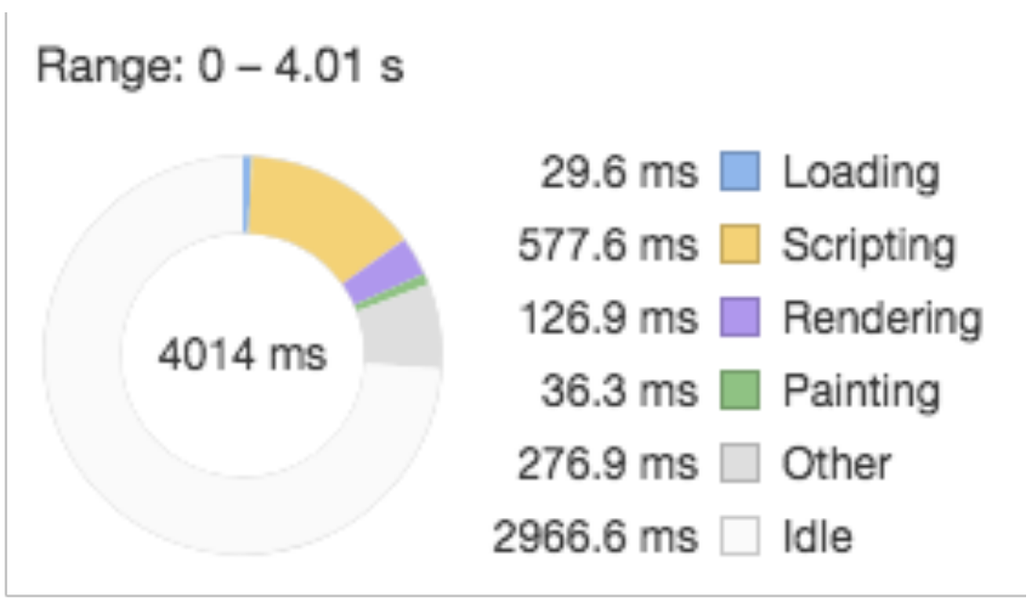
속도를 어디까지 올릴 수 있을까?

캐싱

기대 속도



네이버



카카오 페이지

카카오 페이지도 NextJS를 사용

이야기를 마치며

이야기를 마치며

React+NextJS란
지난 10년 간 자바스크립트
진영의 수많은 노력과 결실

이야기를 마치며

자유롭게 서버와 클라이언트를 오가는

"범용 자바스크립트 어플리케이션"

이야기를 마치며

더 자세한 내용이 궁금하다면

https://rkdehddnr.com/

강 동 욱

이야기를 마치며

함께 하고 싶은 분이 계시다면

https://wntd.co/srH8u5

감사합니다

In Memory of BACKBONE.JS

2010-2017