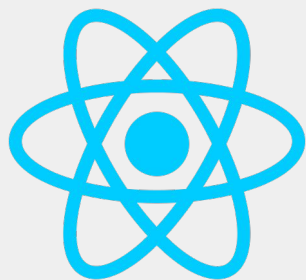


NEXT.JS



django

REST

framework

# Next js



- Reactのフレームワーク
- SSG (Static Site Generation)
  - > SEO (HTML事前生成、高速表示)
- フォルダ構造に基づくPage Navigation
- デプロイが簡単 (Vercel)
- ISR (Incremental Static Regeneration)
- SSG+ISR+CSR(Client-Side-Rendering)の融合
  - > SEO対応かつリアルタイムデータ取得

# CRA (Create React App) -> SEO NG



Pre-rendering  
(事前にHTML生成)

-> コンテンツをクローラー  
にアピール可能

-> ブラウザの負荷低減 高速表示

<https://nextjs.org/learn/basics/data-fetching/pre-rendering>



Client-side-rendering  
(リクエスト毎にブラウザでJSが実行されてHTML生成)

# Next.js はDefaultで全ページをPre-render(HTML事前生成)

1: SSG



Document, Help

2: SSG + Pre-fetch



ブログ、商品一覧など

3: SSG + Client side fetching



Todo, Dashboard

4: SSG + Pre-fetch + Client side fetching



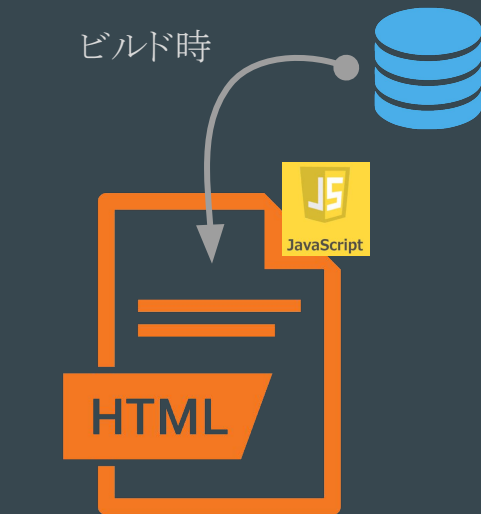
News site

\*SEOに有益なデータは、外部 DBにある(ブログの記事など)

\*Client side fetching -> ユーザーアクセス時に最新データ取得可

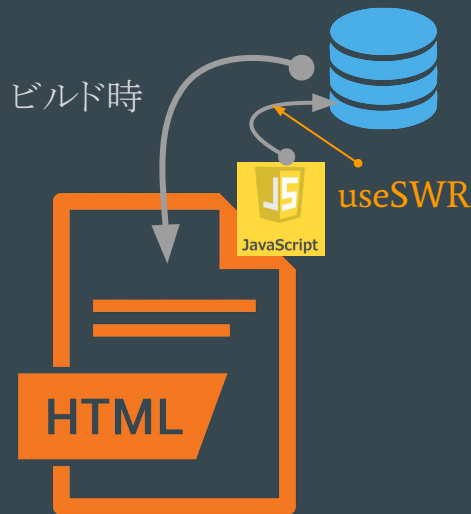
# Incremental Static Regeneration (ISR)

2-1: SSG + Pre-fetch + ISR



ブログ、商品一覧など

4-1: SSG + Pre-fetch + Client side fetching + ISR



News site

A  
B

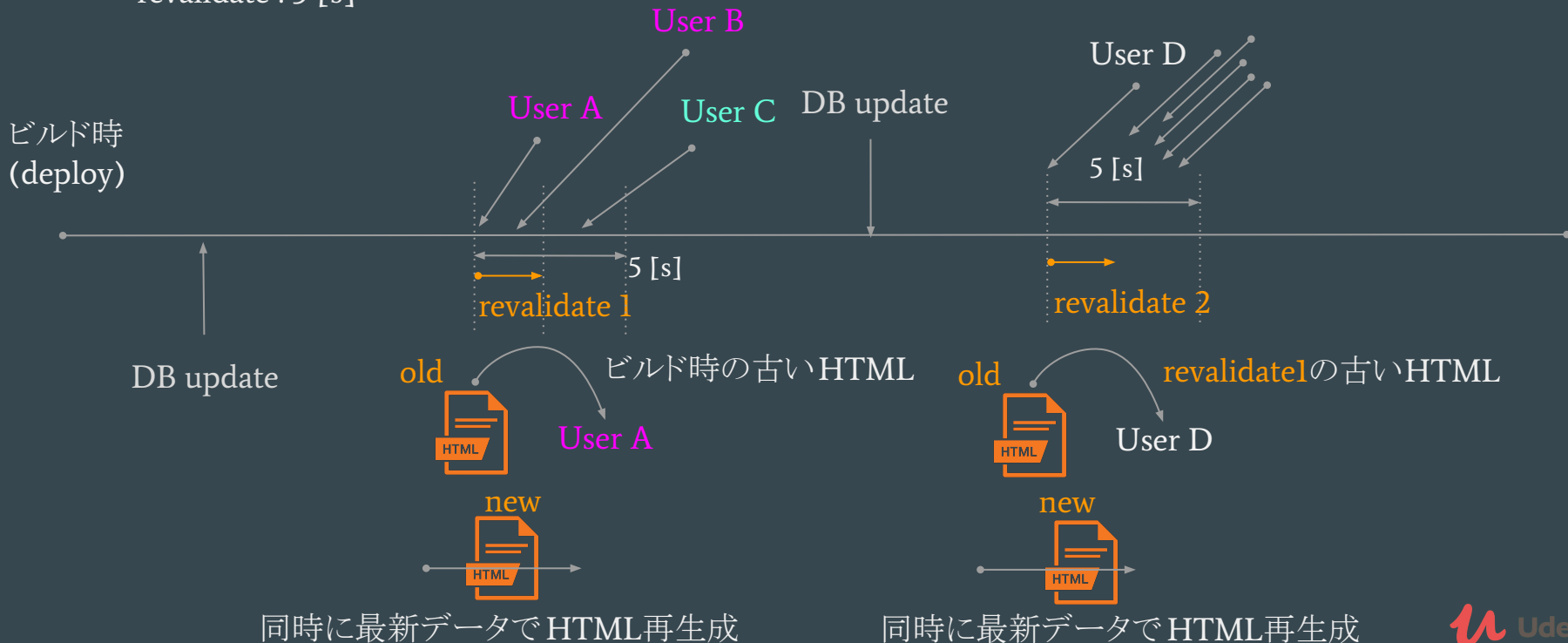
C  
D  
E  
F  
G

“データベースの最新データで静的 HTMLを作り直す技術”

# Incremental Static Regeneration (Stale while revalidation)

“HTML再生成は5秒間で一回だけ”

revalidate : 5 [s]



# Tailwind CSSの利点

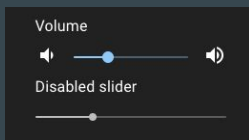
- classのユーティリティ集
- CSS class名の命名で悩む必要がない
- xxx.module.cssが不要
- 可読性が上がる / 記述量が減る
- 自由度が高すぎるCSSから解放される

<https://nerdcave.com/tailwind-cheat-sheet>



Low

High



## Margin

Utilities for controlling an element's margin

Class

Properties

<code>m-0</code>	<code>margin: 0px;</code>
<code>m-0.5</code>	<code>margin: 0.125rem;</code>
<code>m-1</code>	<code>margin: 0.25rem;</code>
<code>m-1.5</code>	<code>margin: 0.375rem;</code>
<code>m-2</code>	<code>margin: 0.5rem;</code>
<code>m-2.5</code>	<code>margin: 0.625rem;</code>

margin

docs

Controls margin (and negative margin) in 0.25rem increments.

AZ

sm: md: lg: xl: 2xl:

<code>.my-0</code>	<code>margin-top: 0px;</code> <code>margin-bottom: 0px;</code>	
<code>.mx-0</code>	<code>margin-left: 0px;</code> <code>margin-right: 0px;</code>	
<code>.my-1</code>	<code>margin-top: 0.25rem;</code> <code>margin-bottom: 0.25rem;</code>	4px
<code>.mx-1</code>	<code>margin-left: 0.25rem;</code> <code>margin-right: 0.25rem;</code>	4px
<code>.my-2</code>	<code>margin-top: 0.5rem;</code> <code>margin-bottom: 0.5rem;</code>	8px
<code>.mx-2</code>	<code>margin-left: 0.5rem;</code> <code>margin-right: 0.5rem;</code>	8px
<code>.my-3</code>	<code>margin-top: 0.75rem;</code> <code>margin-bottom: 0.75rem;</code>	12px
<code>.mx-3</code>	<code>margin-left: 0.75rem;</code> <code>margin-right: 0.75rem;</code>	12px
<code>.my-4</code>	<code>margin-top: 1rem;</code> <code>margin-bottom: 1rem;</code>	16px

# Project 1 HPサイト

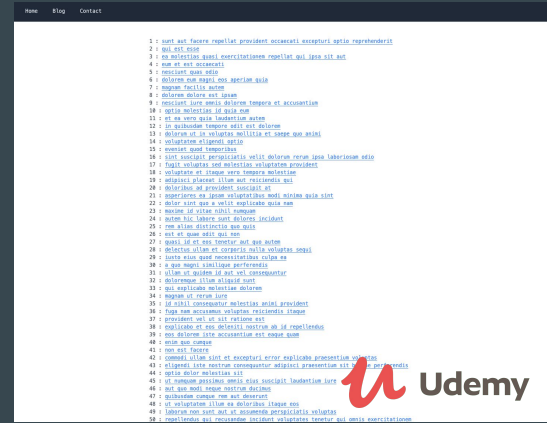
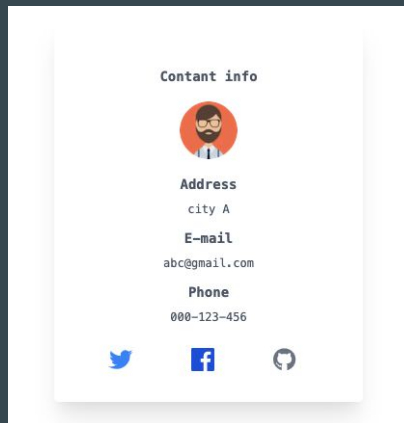
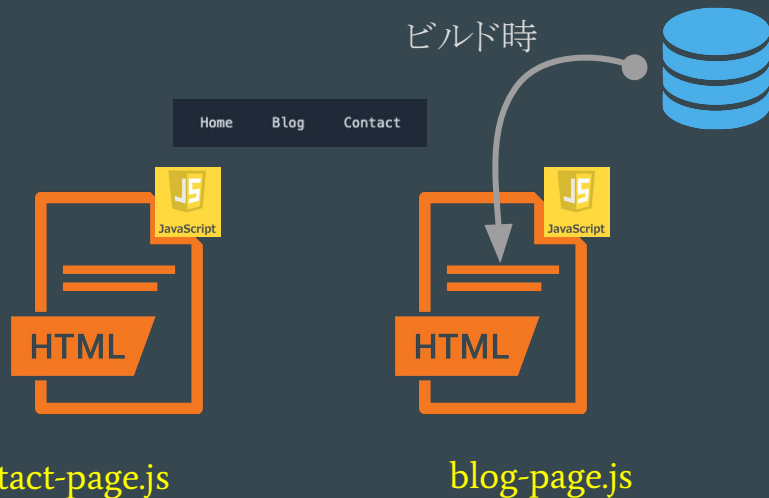


- Next.js 環境設定
- Tailwind CSS
- Page Navigation (Client Site Navigation)
- Layout component
- Image component
- SSG (Static Site Generation)
- getStaticProps
- getStaticPaths
- Dynamic routes
- Push to GitHub
- Deploy to Vercel

```
1 : sunt aut facere repellat
2 : qui est esse
3 : ea molestias quasi exerci
4 : eum et est occaecati
5 : nesciunt quas odio
```

1: SSG

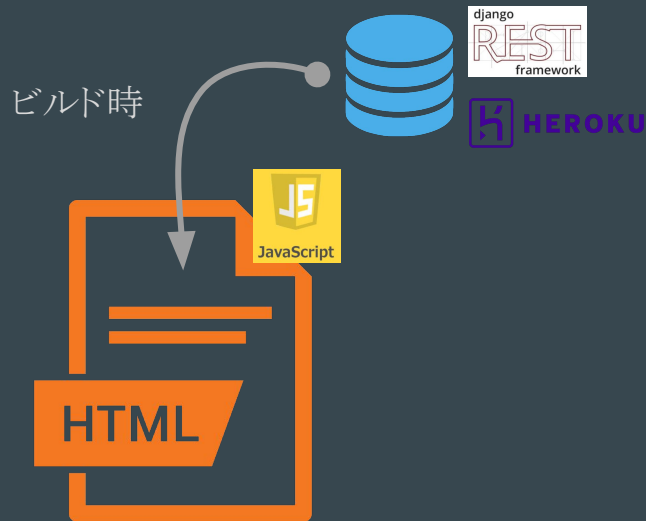
2: SSG + Pre-fetch



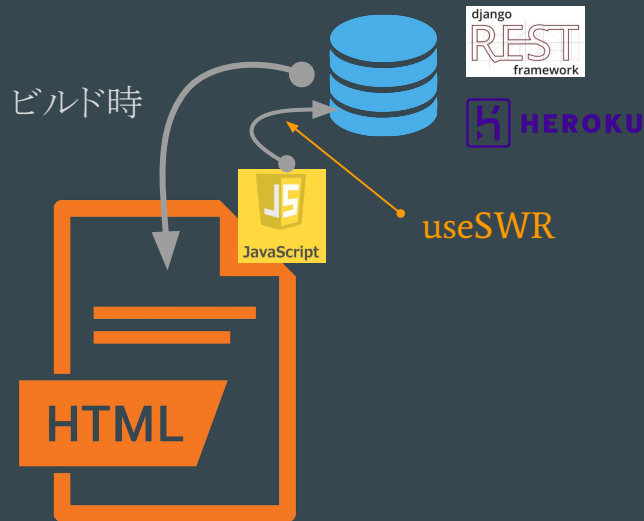


# Project 2 : Blog + Todos

2-1: SSG + Pre-fetch + **ISR**



4-1: SSG + Pre-fetch + Client side fetching + **ISR**



# Project 2 : Blog + Todos



HEROKU



index.js

## Login

change mode ?

Login with JWT

Eventual Consistency

SSG + ISR

blog-page.js

```
6 : Het
5 : New york
4 : New post
2 : When to Use Static Generation v.s. Server-side Ren
1 : Two Forms of Pre-rendering
```

<< Back to main page

main-page.js

Visit Blog by SSG + ISR

Visit Task by ISR + CSR

Strong Consistency

SSG + ISR +  
CSR(useSWR)

task-page.js

CREATE

```
18 : test
17 : new task !
16 : okay !
14 : dafa
13 : hello
11 : fdafadd345
9 : soso2
7 : Task X3
5 : Task E2
1 : Task A22
```



<< Back to main page

# Project 2 : REST API



REST API

api/register  
api/auth/jwt/create

api/list-post  
api/detail-post/id

api/list-task  
api/detail-task/id

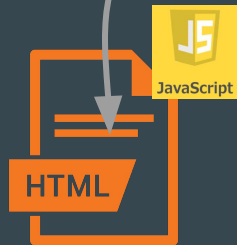
node-fetch  
ビルド時

node-fetch  
ビルド時

NEXT.js



Pre-fetch : ビルド時



api/tasks



fetch (JS)

POST  
PUT  
DELETE

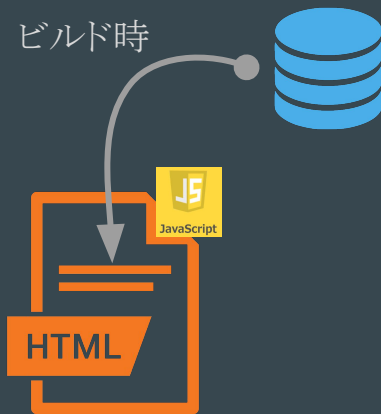


# getStaticProps

2: SSG + Pre-fetch

- 必ずServer sideで実行される
- pages内でのみ使用可能
- npm run dev -> リクエスト毎に実行される
- npm start -> ビルド時に実行される

ビルド時



blog-page.js

```
const Blog = ({ posts }) => {
  return (
    <Layout title="Blog">
      <ul className="m-10">
        {posts && posts.map((post) => <Post key={post.id} post={post} />)}
      </ul>
    </Layout>
  );
};

export default Blog;

export async function getStaticProps() {
  const posts = await getAllPostsData();
  return {
    props: { posts },
  };
}
```

# Dynamic routes

posts/[id]

✓  posts  
JS [id].js

posts/1



ブログ記事コンテンツ

posts/2



ブログ記事コンテンツ

...

posts/100



ブログ記事コンテンツ

# Dynamic routes (詳細ページのpre-renderingの流れ)

1. `getStaticPaths()` -> **id**の一覧を取得

2. `getStaticProps()`

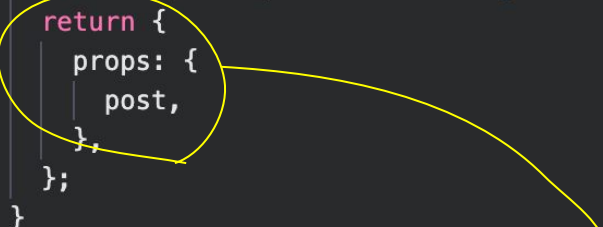
-> **各id**を使って個別データを取得

`https://jsonplace...../posts/id`

3. 取得したデータをpropsでReact Componentに渡してpre-rendering  
(HTML事前生成)

```
export async function getStaticPaths() {  
  const paths = await getAllPostIds();  
  
  return {  
    paths,  
    fallback: false,  
  };  
}
```

```
export async function getStaticProps({ params }) {  
  const { post: post } = await getPostData(params.id);  
  return {  
    props: {  
      post,  
    },  
  };  
}
```



```
export default function Post({ post }) {  
  if (!post) {  
    return <div>Loading...</div>;  
  }  
}
```

# Next.js pages

page A

React  
component

page B

React  
component

React  
component

page C

React  
component

React  
component

React  
component