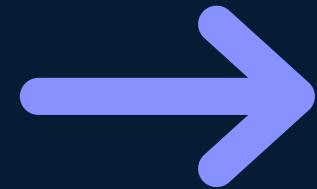


# THE FUTURE OF SERVER-SIDE RENDERING NEXT.JS 14 SERVER ACTIONS



**Abdoelaziz gamal**

@ag\_coding 



# INTRODUCTION

Next.js Server Actions هي فانكشنز بتتنفذ على السيرفر وجود الفانكشنز الخاصة دي اللي بتشتغل بس على السيرفر بيدي developers إمكانية نقل مهام زي الـ data fetching والـ mutations للسيرفر، وده يحميهم من الثغرات والمشاكل الأمنية اللي بتحصل لما بتجيip أو تعديل الداتا من الـ client عشان نفهم كوييس Server Actions، لازم نعرف المشاكل اللي بتحلها، وإننا كنا بنعمل إيه قبلها، والتطور اللي حصل ليها. هنتناقش في كل الجوانب دي خلال البوست ده، خلينا نبدأ من الأول!

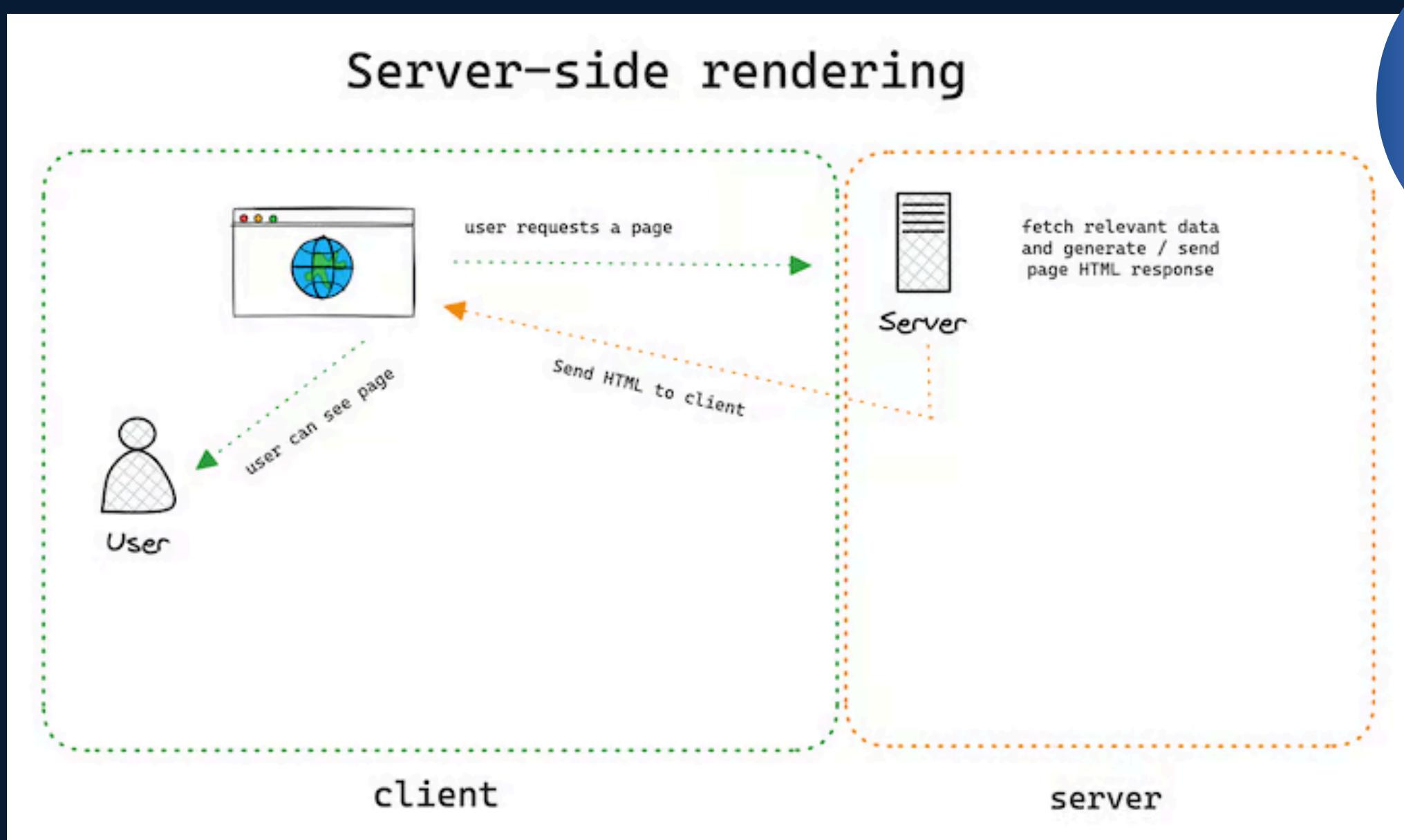


Abdoelaziz gamal  
@ag\_coding



# THE HISTORY OF SERVER ACTIONS

المرحلة بدأت مع الـ (SSR) في server-side rendering، حيث كانت كل操作ات تتم على السيرفر، عشان بتسهلك الـ CPU بشكل كبير، وتحصل كلها على السيرفر عشان المتصفح يصل له صفحة خفيفة وجاهزة للعرض للمستخدمين.



**Abdoelaziz gamal**

@ag\_coding



# THE HISTORY OF SERVER ACTIONS

لكن ده كان معناه إن مع كل مرة المستخدم يتنقل فيها، كان محتاجين نعمل الـ `round trip` ده تاني: نبعث `request` للسيرفر نعمل `generate` للصفحة، وبعددين نبعتها للـ `client`. المستخدم كان لازم يستنى العملية دي تخلص كل مرة قبل ما يشوف الصفحة ويتفاعل معها – وده كان حاجة مش بنبهها وهنا جات الموجة الجديدة من الابتكار الـ `client-side rendering` (CSR) بدل ما نبعث طلب للسيرفر كل مرة المستخدم يتنقل فيها، ليه ما نخليش الـ `client` هو اللي يتولى عملية الـ `navigation` وده معناه إن أول مرة السيرفر يرد فيها، هييعرف الكود اللي بيعمل client للـ `rendering`



Abdoelaziz gamal

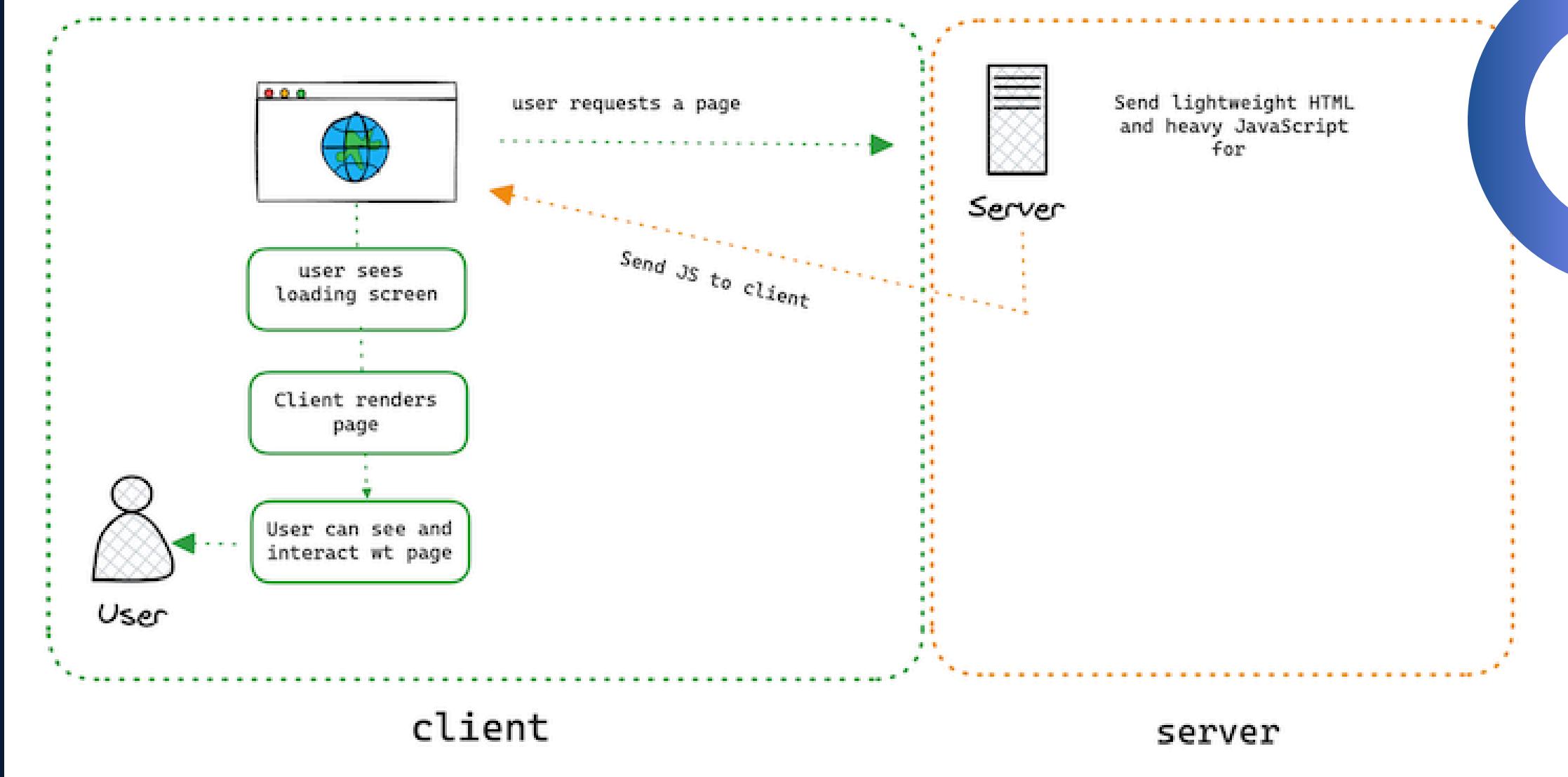
@ag\_coding



# THE HISTORY OF SERVER ACTIONS

وقد يمكن لل client إله يتولى عملية عرض المفحات كل ما المستخدم يتنقل في الموقع

## Client-side rendering



**Abdoelaziz gamal**

@ag\_coding



# THE HISTORY OF SERVER ACTIONS

مع الـ CSR، حلينا مشكلة الـ round trips وحققنا تنقلات أسرع بين الصفحات، لكن في نفس الوقت، وقعنا في مشكلة جديدة لما السيرفر بيبيعت JavaScript للمتصفح، محركات البحث مش بتقدر تعمل indexing للموقع بشكل صح، لأن الـ HTML الفعلي بتاع الموقع مش بيكون كامل لسه. المتصفح لازم الأول ينزل ويشغل الجافاسكريبت من السيرفر وبعدين يعمل "hydrate" للصفحة عشان يكمل بناء الـ markup الخاص بالموقع هنا ظهر مفهوم الـ static site generation (SSG) كان الابتكار عشان نحقق سرعة أكبر في تحميل الصفحات مع وجود markup جاهز لتحسين الـ SEO، الفكرة هي الجمع بين أحسن مميزات الـ CSR والـ SSR عشان نطلع بأفضل حل من الناحيتين. فكرنا، ليه ما نعملش build pre-generate لكل صفحات الموقع على السيرفر وقت الـ static site generator أو build script بتعالج الكود والمحتوى عشان تعمل ملفات static HTML لكل صفحة على الموقع



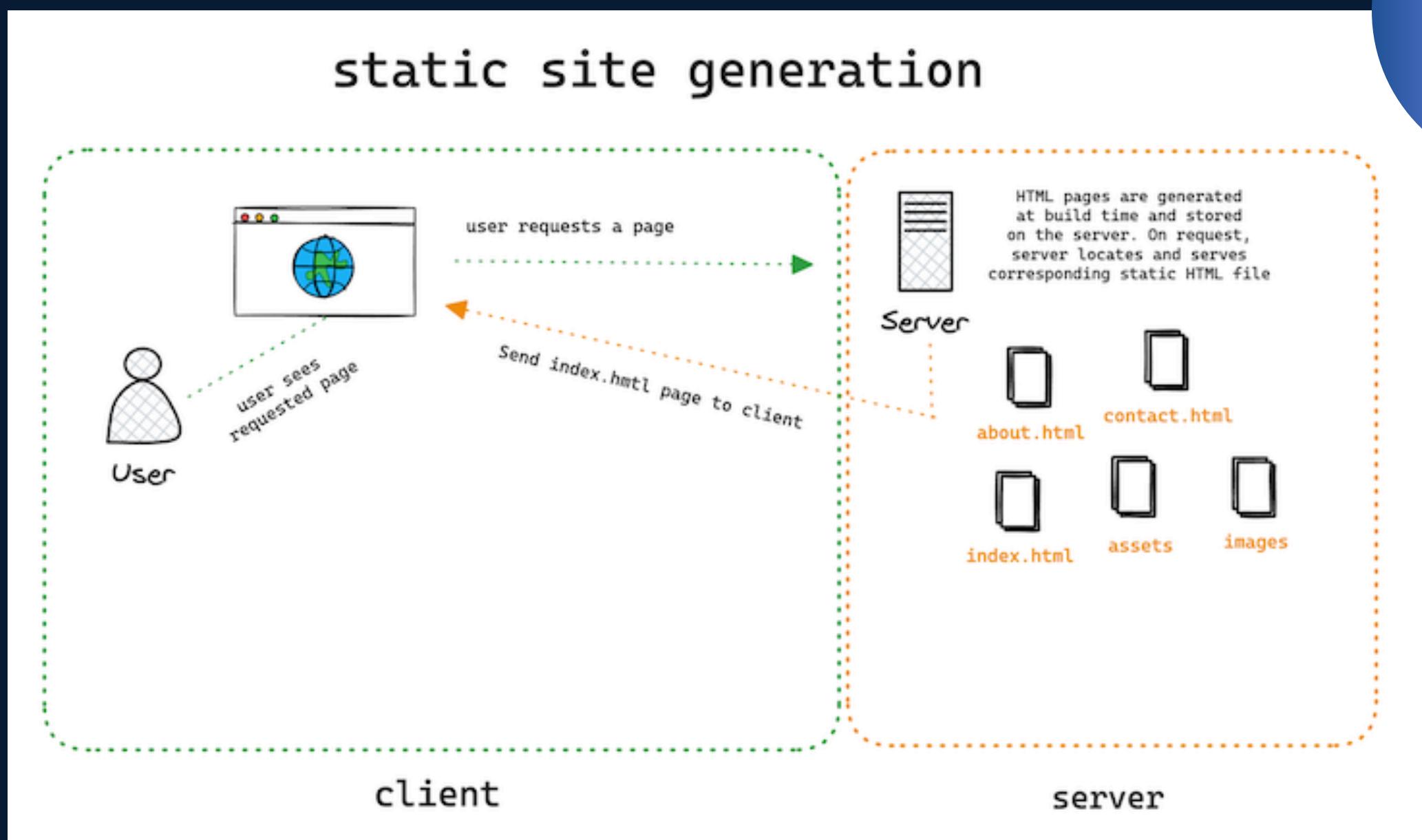
Abdoelaziz gamal

@ag\_coding



# THE HISTORY OF SERVER ACTIONS

عملية الـ build ممكن كمان تتضمن fetching بيانات من مصادر مختلفة زي الـ databases أو APIs أو HTML ثابتة. لكن كان فيه قلق بخصوص إزايا التعامل هيكون مع المحتوى динاميكي أو البيانات اللي بتتحدد في الوقت الحقيقي عدم وجود حل واضح للحالات دي خلانا نفكّر إن الـ SSG مش هيكون مناسب لكل أنواع الواقع، وده اللي رجعنا تاني لفكرة الـ SSR، ولكن المرة دي مع React Server Actions



**Abdoelaziz gamal**

@ag\_coding



# REACT SERVER COMPONENTS AND ACTIONS

مع الرحلة اللي مررنا بيها من الـ SSR لـ CSR، وكل الحلول اللي في النص، بقى واضح إن مفيش حل واحد يناسب كل الحالات. وهنالا ظهرت اللي بتدبي للمطوريين القدرة على React Server Components (RSCs) فصل المهام (separate concerns) دلوقتي، مع components Server Actions، تقدر تقول مثلاً: "الـ XYZ لازم تنفذ بس على السيرفر، بينما الـ ABC تنفذ على الـ client". الوظيفة دي بقت متاحة من خلال Next.js Server Actions، وده المفهوم اللي بنبي عليه Actions. فانكشنز مخصصة عشان تنفذ إما على السيرفر أو على الـ client، وده مفيد بشكل خاص في حاجات زي الـ data fetching والmutations. الجمع بين RSCs و Server Actions خلانا نفكربشكل أفضل في موضوع الـ data fetching قبل كده، كنا بنجيب الداتا بطرق كتير مختلفة.



Abdoelaziz gamal

@ag\_coding



# REACT SERVER COMPONENTS AND ACTIONS

على سبيل المثال، كنت ممكن تستخدم ال useEffect Hook عشان تجيب الداتا وتدير ال loading states على مستوى الصفحات ب GetServerSideProps و GetStaticProps ولما كنت تحتاج تعمل اتصال بقاعدة البيانات باستخدام credentials حساسة، كنت بتعمل API route و تكتب الفانكشنز بتاعتك هناك دلوقتي، مع ال RSCs و Server Actions، الأمور بقت أسهل بكثير كل component ممكن تجيب الداتا الخاصة بيها، وال mutations كمان ممكن تحصل جوه ال component نفسها من غير ما تحتاج تعمل API خارجية routes  
خلينا نشوف Server Actions في التطبيق العملي



Abdoelaziz gamal

@ag\_coding



# GETTING STARTED WITH NEXT.JS SERVER ACTIONS

هنبني مشروع بسيط لتطبيق to-do عشان نوضح إزاي كنا بنعمل  
الحالات زمان وإزاي بنعملها دلوقتي مع Server Actions  
المشروع ده هيقبل مدخلات المستخدم ويعمل تحديث لقاعدة بيانات  
MongoDB

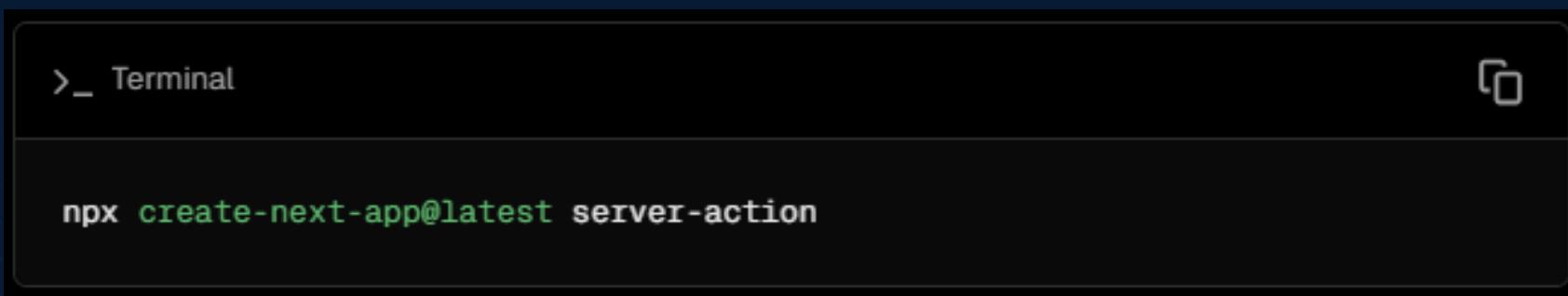
## Prerequisites

- حساب قاعدة بيانات Supabase لتخزين الـ to-dos (تقدّر تستخدم أي قاعدة بيانات تانية حسب اختيارك)
- حساب قاعدة بيانات MongoDB ≥v18 •

## Setting up your Next.js project

أول حاجة، هنشئ تطبيق Next.js 14 جديد عن طريق تشغيل الأمر

التالي



```
>_ Terminal
npx create-next-app@latest server-action
```



**Abdoelaziz gamal**

@ag\_coding 



# GETTING STARTED WITH NEXT.JS SERVER ACTIONS

اقبل كل الـ prompts الافتراضية بالضغط على Enter لحد ما يخلص  
لما المشروع يتعمّل، روح على الـ directory بـتاع المشروع وشغل  
الـ development server

```
>_ Terminal
cd server-action && yarn dev
```

دلوقي في الـ development server يكون شغال  
وأخيراً هن الـ install الـ JavaScript package  
كقواعد البيانات بـتاعتك  
هتسخدم Supabase

```
>_ Terminal
npm install @supabase/supabase-js
```



**Abdoelaziz gamal**

@ag\_coding



# BUILDING THE FORM ELEMENT

هنا هننشئ فورم للتعامل مع user inputs — في الحالة دي، الـ app/page.tsx عشان نعمل كده، ضيف الكود ده في ملف to-dos



```
// app/page.tsx
export default function TodoList() {
  return (
    <>
      <h2>Server Actions Demo</h2>
      <div>
        <form
          action="#"
          method="POST"
        >
          <div>
            <label htmlFor="todo">Todo</label>
            <div>
              <input
                id="todo"
                name="text"
                type="text"
                placeholder="What needs to be done?"
                required
              />
            </div>
          </div>
          <div>
            <button type="submit"> Add Todo</button>
          </div>
        </form>
      </div>
    </>
  );
}
```



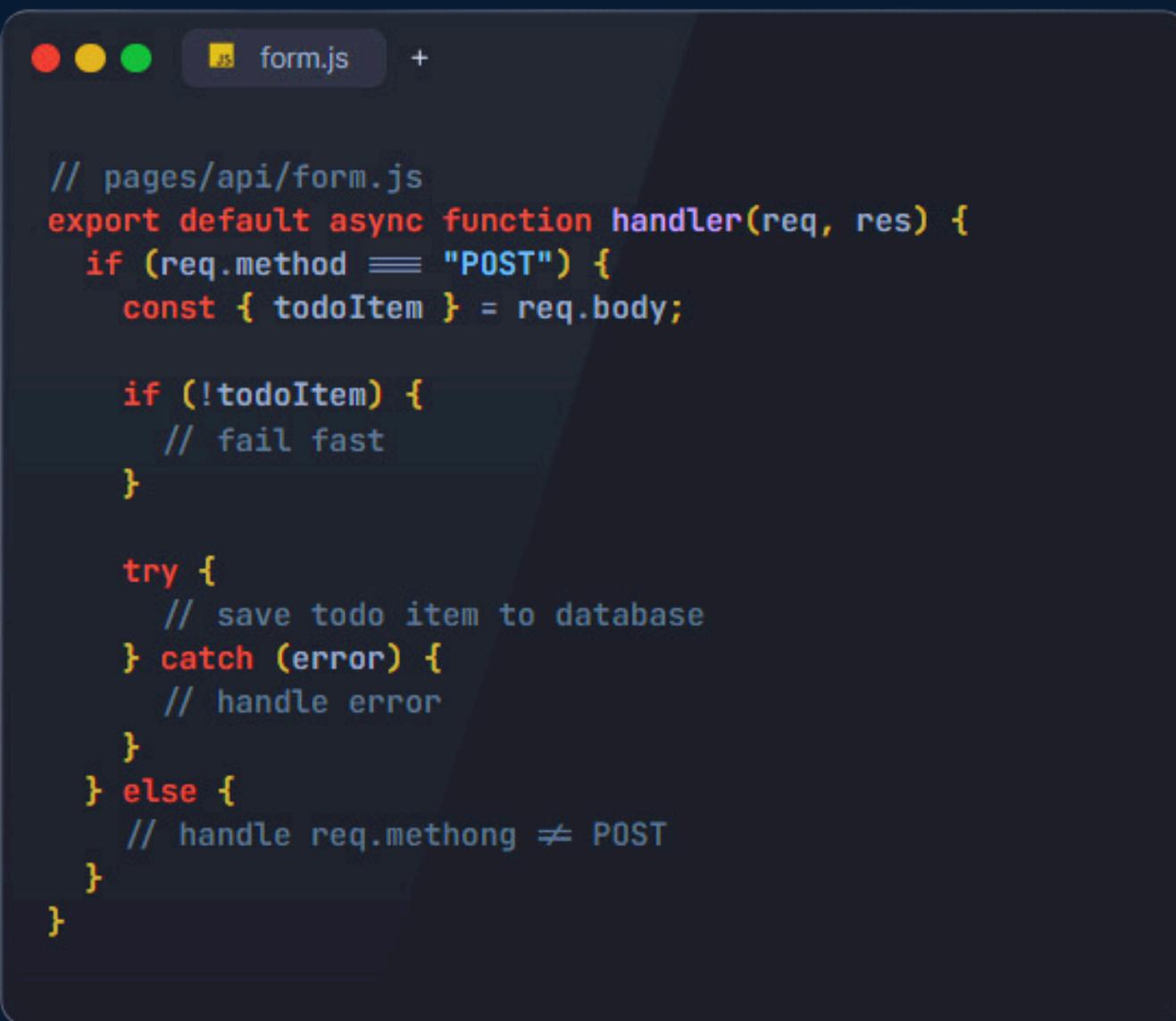
Abdoelaziz gamal

@ag\_coding



## HOW WE DID THINGS BEFORE SERVER ACTIONS

زمان لما كنا بنحب نتعامل مع عمليات التعديل على البيانات في Next.js، كنا بنعمل API route زي `pages/api/form.js`. في الملف ده، بنكتب اللي بتسقبيل البيانات اللي المستخدم بيدخلها في الفورم ويتخزنها في قاعدة البيانات الملف كان غالباً بيقى بالشكل ده



```
// pages/api/form.js
export default async function handler(req, res) {
  if (req.method === "POST") {
    const { todoItem } = req.body;

    if (!todoItem) {
      // fail fast
    }

    try {
      // save todo item to database
    } catch (error) {
      // handle error
    }
  } else {
    // handle req.method != POST
  }
}
```



Abdoelaziz gamal

@ag\_coding 



## HOW WE DID THINGS BEFORE SERVER ACTIONS

بعد كده كنا بنحتاج نضبط الكلاينت عشان يبعث بيانات الفورم (العنصر بتاع الـ API route لما المستخدم يضيف العنصر والطريقة دى كنا بنسخدمها قبل ظهور Server Actions

```
// pages/index.tsx
import { useState } from "react";

export default function Home() {
  const [todoItem, setTodoItem] = useState("");

  const handleSubmit = async (event: React.FormEvent<HTMLFormElement>) => {
    event.preventDefault();
    try {
      // Send a POST request to the API route with the todo item
      const response = await fetch("/api/form", {
        method: "POST",
        headers: {
          "Content-Type": "application/json"
        },
        body: JSON.stringify({ todoItem })
      });

      if (response.ok) {
        console.log("Todo item added successfully");
      } else {
        console.error("Failed to add todo item");
      }
    } catch (error) {
      console.error("Error:", error);
    }
  };

  return (
    <div>
      <h1>Todo Application</h1>
      <form onSubmit={handleSubmit}>
        <label>
          Todo Item:
          <input
            type="text"
            value={todoItem}
            onChange={(e) => setTodoItem(e.target.value)}
          />
        </label>
        <button type="submit">Add Todo</button>
      </form>
    </div>
  );
}
```



## HOW WE DID THINGS BEFORE SERVER ACTIONS

ده كان هيبيعت بيانات الفورم لـ API route api/form، اللي  
بدوره هيبيعت البيانات دي لـ database لكن بشكل عملي، كنا  
محتاجين كمان نحدث الـ UI عشان نعرض العنصر الجديد اللي تم  
إضافته

عشان الـ process دي تكتمل لما الـ form submission يحصل،  
الـ steps المطلوب

1. المستخدم يـ submit الـ form

2. بيانات الفورم تتبع لـ API route api/form

3. الـ API route تستقبل بيانات الفورم وتحفظها في قاعدة  
البيانات

4. الـ UI تتحدد عشان تجيب البيانات الجديدة من قاعدة البيانات  
ده كان شكل الـ handled form submissions قبل كده، لكن مع  
ظهور Server Actions، العملية بقت أسهل بكثير



Abdoelaziz gamal

@ag\_coding



## HOW WE DO THINGS NOW WITH SERVER ACTIONS

زي ما قلت قبل كده، functions Server Actions بتسمح لنا نعمل تشتغل بس على السيرفر، يعني نقدر نتعامل مع التعديلات في جوه الكمبونتس بتاعتنا، عشان نبعث بيانات الفورم لـ database باستخدام Server Actions، هندّث ملف بالشكل التالي بص لسلайд الجاية app/page.tsx



**Abdoelaziz gamal**

@ag\_coding



```
// app/page.tsx
import { createClient } from '@supabase/supabase-js';
export default function TodoList() {

  const addTodo = async (formData: FormData) => {
    'use server';
    const supabaseUrl = 'YOUR_SUPABASE_URL';
    const supabaseKey = process.env.SUPABASE_KEY;
    const supabase = createClient( supabaseUrl, supabaseKey);
    const todoItem = formData.get('todo');
    if (!todoItem) {
      return;
    }
    // Save todo item to database
    const { data, error } = await supabase.from('todos').insert({
      todo: todoItem,
    });
  };

  return (
    <>
      <h2>Server Actions Demo</h2>
      <div>
        <form action={addTodo} method="POST">
          <div>
            <label htmlFor="todo">Todo</label>
            <div>
              <input id="todo" name="text" type="text"
                placeholder="What needs to be done?"
                required
              />
            </div>
          </div>
          <div>
            <button type="submit"> Add Todo</button>
          </div>
        </form>
      </div>
    </>
  );
}
```



Abdoelaziz gamal

@ag\_coding



## HOW WE DO THINGS NOW WITH SERVER ACTIONS

وكده، بنكون حداًثاً الـ action بتاعة الفورم عشان تتصل بـ Server Action دي هي addTodo function بتستقبل prop عشان تسهل الوصول لقييم الفورم الـ directive 'use server' بتحدد إن الفانكشن دي هتشتغل بس على السيرفر. وبكده، نقدر نعمل تعديلات على قاعدة البيانات جوه الفانكشن دي زي ما عملنا في المرحلة دي، لو المستخدم ضغط على زرار "AddTodo" هتلaci إن العنصر اللي ظافه في الفورم اضاف لـ database بشكل طبيعي لكن ممكن تلاحظ إن الـ UI مش بيتحدى عشان نحل المشكلة دي، هنجيب بيانات الـ to-dos ونظهرها على الصفحة. أول حاجة، هنجيب بيانات الـ to-do من الـ database عن طريق تحديث ملف app/page.tsx بالشكل التالي



Abdoelaziz gamal

@ag\_coding





```
// app/page.tsx
import { createClient } from '@supabase/supabase-js';
export default function TodoList() {
+  const supabaseUrl = 'YOUR_SUPABASE_URL';
+  const supabaseKey = process.env.SUPABASE_KEY;
+  const supabase = createClient(supabaseUrl, supabaseKey);

+  const { data, error } = await supabase.from('todos').select('todo');

    const addTodo = async (formData: FormData) => {
      'use server';
      const supabaseUrl = 'YOUR_SUPABASE_URL';
      const supabaseKey = process.env.SUPABASE_KEY;
      const supabase = createClient(supabaseUrl, supabaseKey);
      // add todo to DB
    };

    return (
      <>
        <h2>Server Actions Demo</h2>
        <div>
          <form action={addTodo} method="POST">
            <div>
              <label htmlFor="todo">Todo</label>
              <div>
                <input id="todo" name="text" type="text"
                  placeholder="What needs to be done?"
                  required
                />
              </div>
            </div>
            <div>
              <button type="submit"> Add Todo</button>
            </div>
          </form>
+          <ul>
+            {data &&
+              data.map((todo: any) => (
+                <li key={todo._id}>
+                  <span>{todo.todo}</span>
+                </li>
+              )));
+          </ul>
        </div>
      </>
    );
}
```



Abdoelaziz gamal

@ag\_coding



## HOW WE DO THINGS NOW WITH SERVER ACTIONS

طيب بص، الفكرة هنا إنا بنعمل database initialize لـ `TodoList` مرتين في نفس الملف: مرة جوه الـ `TodoList()` علشان نجيب الـ `to-dos` على الـ `client`, والمرة الثانية جوه الـ `addTodos` اللي هي `Server`, والمفهوم اللي هو `database` بنبعث فيها الـ `to-dos` لـ `Action` السبب فيي ده إنا ما نقدرش نستخدم المتغيرات أو الـ `props` اللي متعرفة فيي الـ `client` جوه الـ `Server Action`. بمعنى أبسط، ما ينفعش تعمل `define` وتسخدم الـ `Server Actions` بشكل مباشر `Client Component` جوه

طب السؤال هنا، إزاي بقى نستخدم الـ `Server Actions` في `Components`? الحل ببساطة إنك ممكن تعمل `function Server Side` وتبعد البيانات من ملف لوحدته أو كـ `form` باستخدام الـ `Server Action` أو أي طريقة تانية زي مثلا `fetch request` وبعدها ترجع النتيجة أو تعمل أي حاجة عايزها بعد ما الـ `Server Action` يخلص شغلها



Abdoelaziz gamal

@ag\_coding



## HOW TO USE SERVER ACTIONS IN CLIENT COMPONENTS

طيب بص، الفكرة هنا إنا بنعمل database initialize لـ `TodoList` مرتين في نفس الملف: مرة جوه الـ `TodoList()` علشان نجيب الـ `to-dos` على الـ `client`, والمرة الثانية جوه الـ `addTodos` اللي هي `Server`, والمفهوم اللي هو `database` بـ `to-dos` لـ `Action` السبب فيي ده إنا ما نقدرش نستخدم المتغيرات أو الـ `props` اللي متعرفة فيي الـ `client` جوه الـ `Server Action`. بمعنى أبسط، ما ينفعش تعمل `define` وتسخدم الـ `Server Actions` بشكل مباشر `Client Component` جوه

طب السؤال هنا، إزاي بقى نستخدم الـ `Server Actions` في `Client Components`? الحل ببساطة إنك ممكن تعمل `function Server Side` وتبعد البيانات من ملف لوحدته أو كـ `form` باستخدام الـ `Server Action` أو أي طريقة تانية زي مثلا `fetch request` وبعدها ترجع النتيجة أو تعمل أي حاجة عايزها بعد ما الـ `Server Action` يخلص شغلها



Abdoelaziz gamal

@ag\_coding



# HOW TO USE SERVER ACTIONS IN CLIENT COMPONENTS

الموضوع ده معكן يبقى شويه tricky, بس ده الحل الأساسي لما تيجي تعامل مع Client Components و Server Actions جوه. علشان نستخدم الـ addTodos Server Action بـشكل طح في الـ Client Component، محتاجين نفصلوا في ملف تاني، وبعدين نعمل لها import في الـ Client Component. دي طريقة كويسة علشان نطبق مبدأ Separation of Concerns، وتأكد إن كل حاجة الـ Action محتاجها تبقى متعرفة جوه الـ Action نفسه، وما يحصلش أي تداخل مع الكود اللي شغال على الـ client-side. علشان نحل الموضوع ده، هننقل الـ Server Action دي لملف تاني جوه الـ actions directory، ونعرفها هناك. يعني هتعمل ملف جديد اسمه app/src/actions/addTodo.ts بالشكل ده



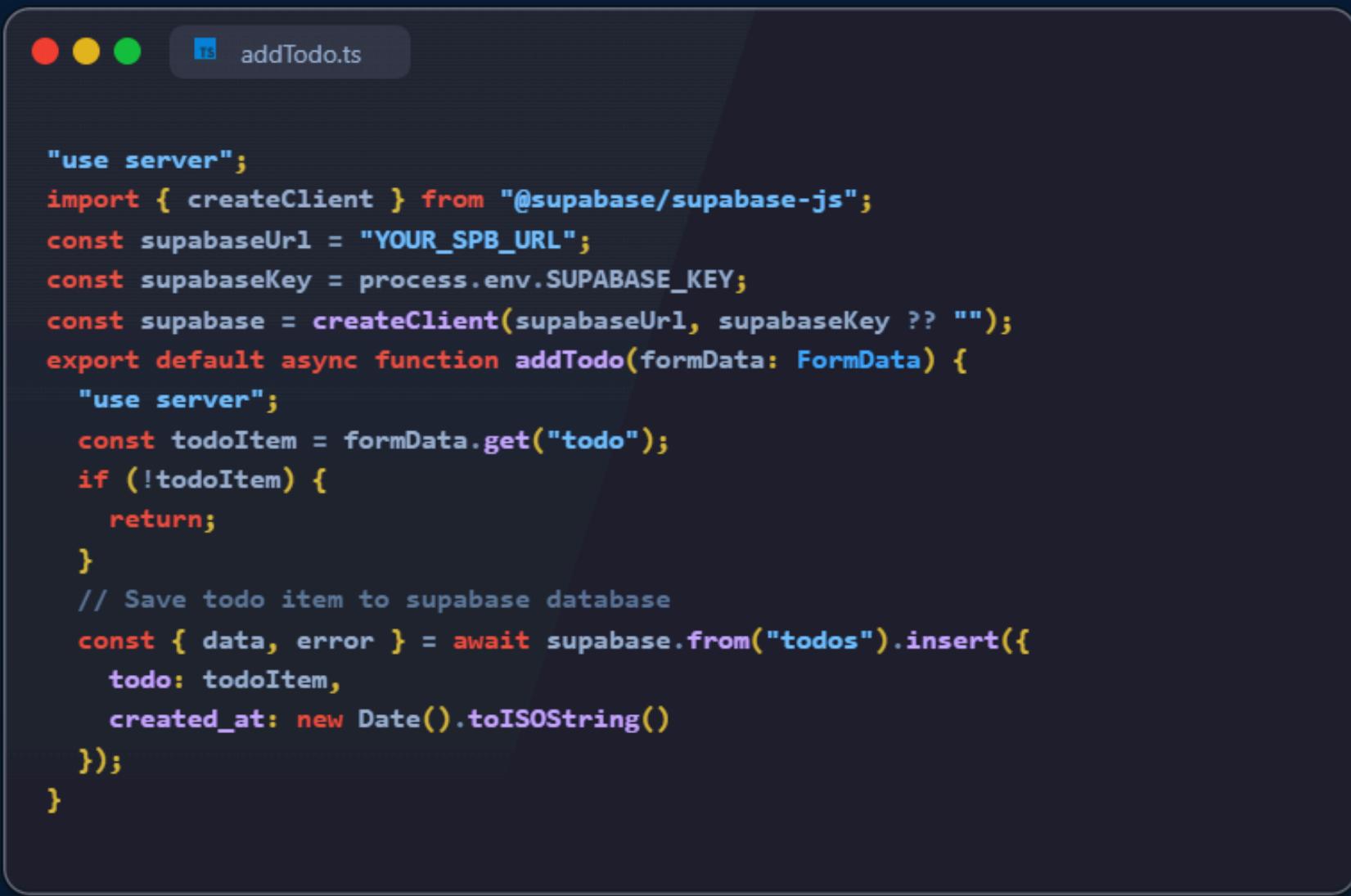
# Abdoelaziz gamal

@ag\_coding 



# HOW TO USE SERVER ACTIONS IN CLIENT COMPONENTS

بسهيل عليك التشغيل ويخلطي الـ Server Actions معزولة عن الـ Client Components، وبدة هيكون مفيد لو عايز تعدل أو تعيد استخدام الـ Action في أكثر من مكان.



```
"use server";
import { createClient } from "@supabase/supabase-js";
const supabaseUrl = "YOUR_SPB_URL";
const supabaseKey = process.env.SUPABASE_KEY;
const supabase = createClient(supabaseUrl, supabaseKey ?? "");
export default async function addTodo(formData: FormData) {
  "use server";
  const todoItem = formData.get("todo");
  if (!todoItem) {
    return;
  }
  // Save todo item to supabase database
  const { data, error } = await supabase.from("todos").insert({
    todo: todoItem,
    created_at: new Date().toISOString()
  });
}
```



Abdoelaziz gamal

@ag\_coding 



# HOW TO USE SERVER ACTIONS IN CLIENT COMPONENTS

هنا، احنا فطلا ال Client addTodo Server Action من ال directory و حطيناه في Component ممكن ببساطة نعمله import جوه ال التالي

```
import addTodo from '@/actions/addTodo';
```

الميزة في ال pattern ده إنه يسمح لنا نحدد أكثر من Action في نفس الفايل ونعملهم import في الأماكن اللي محتاجينهم فيها، من غير ما نخلطهم مع ال client-side logic



**Abdoelaziz gamal**

@ag\_coding



## UI UPDATES

بخصوص الـ UI updates، لحد دلوقتي قدرنا نعمل POST للـ to-do database في الـ Server Action باستخدام الـ `Server Action`، وكمان بنجيب الـ to-do من الـ database ونعرضها للمستخدمين لكن لو المستخدم ضاف to-do جديد، الـ UI مش هيعمل update - حتى لو عملنا reload للصفحة السبب في كده هو طريقة الكاشينج اللي aggressive requests بتكون Next.js عشان نحل المشكلة دي، محتاجين نعمل revalidate للمسار اللي إحنا عليه لما نعمل submit للـ form عشان نكسر الكاش ونجيب آخر data بعد الـ `form submission`. بالتالي هنحدث الـ Server Action بالشكل دا



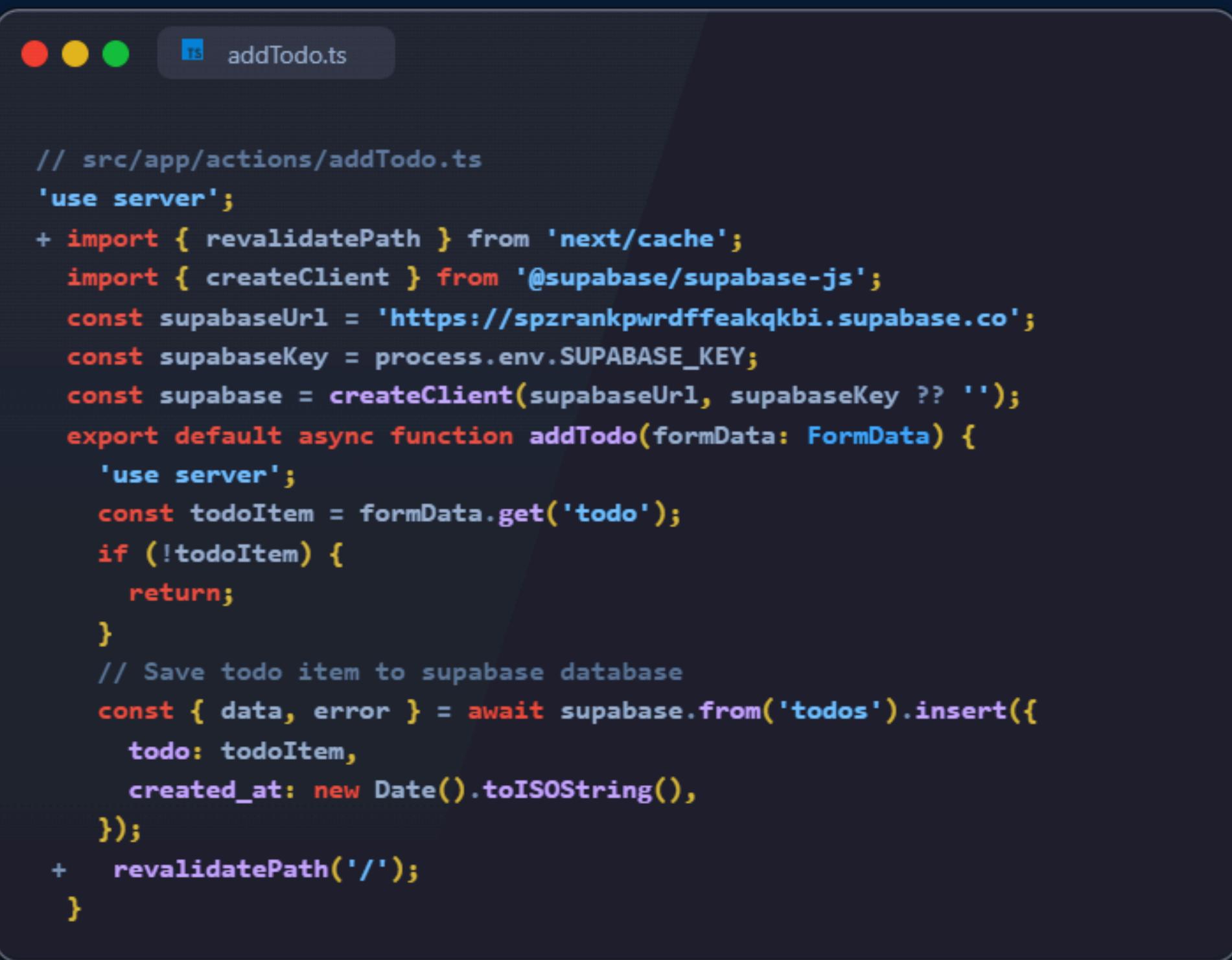
Abdoelaziz gamal

@ag\_coding



# UI UPDATES

دلوقي بعده ما ضفنا حته الـ revalidation أي to-do جديد هتخلي الـ UI يتعدث ويظهر العنصر الجديد اللي في الـ database



The screenshot shows a code editor window with a dark theme. At the top, there are three circular status indicators (red, yellow, green) followed by the file name "addTodo.ts". The code itself is written in TypeScript and is as follows:

```
// src/app/actions/addTodo.ts
'use server';
+ import { revalidatePath } from 'next/cache';
import { createClient } from '@supabase/supabase-js';
const supabaseUrl = 'https://spzrankpwrdfakeqkbi.supabase.co';
const supabaseKey = process.env.SUPABASE_KEY;
const supabase = createClient(supabaseUrl, supabaseKey ?? '');
export default async function addTodo(formData: FormData) {
  'use server';
  const todoItem = formData.get('todo');
  if (!todoItem) {
    return;
  }
  // Save todo item to supabase database
  const { data, error } = await supabase.from('todos').insert({
    todo: todoItem,
    created_at: new Date().toISOString(),
  });
  +   revalidatePath('/');
}
```



Abdoelaziz gamal

@ag\_coding 



## UI UPDATES

بكله نكون خلصنا الـ demo بـ `to-do` باستخدام

Next.js في `Actions`

تقدير تطور الـ demo دا أكثر عن طريق استخراج الـ `to-do` render `component` في `functionality` لوحدها وبعد كده تعمل `component` في `homepage`, على حسب الـ

.React بـ `component-based architecture`

كمان ممكن تعمل `optimistic update` لـ API عشان تحسن سرعة

الاستجابة وتدبّي تجربة مستخدم أحسن باستخدام الـ

`useOptimistic Hook`

وأخيرًا, لو عايز تروح لآخر, تقدير تستخدم الـ `React useTransition`

عشان تظهر `loading states` على زرار الـ `Add Todo` لما API

المستخدم يضيف `to-do` جديد

لسه عندى اللي أديوك أصبر "))))"



Abdoelaziz gamal

@ag\_coding



# THE REVALIDATEPATH FUNCTION

قبل ما نبص على شوية تحديات معنكن تقابلنا مع `Server Actions` خلينا ناخذ لحظة نفهم أكثر في الـ `revalidatePath function` واستخداماتها المحمولة الـ `revalidatePath function` بتسخدم عشان تحدث محتوى قديم على الصفحة بشكل ديناميكي من غير ما تعمل `refresh` للصفحة كلها، ده بيضمن إن كاش الصفحة يتم تفريغه وأي بيانات أو تغييرات جديدة تظهر بعض استخدامات الـ `revalidatePath function` تشمل • `Real-time data updates` • `Sports and weather forecast` : زي `Dynamic` platforms لتحديث المحتوى الجديد بشكل دوري • `موقع التجارة الإلكترونية`: معنكن تحدث أسعار المنتجات والمخزون بسوجة أكثر باستخدام الـ `revalidatePath function`



**Abdoelaziz gamal**

@ag\_coding





**FOLLOW ME TO GET  
MORE INFORMATION  
AND TIPS LIKE THIS.**

تحبوا المقالة الجاية تكون عن إيه ؟ سيب كومنت باللي محتاجه



**Abdoelaziz gamal**

@ag\_coding

