OSTROVOK.RU
БРОНИРОВАНИЕ ОТЕЛЕЙ

Как подружить чистую архитектуру и RSC?

Вадим Царегородцев

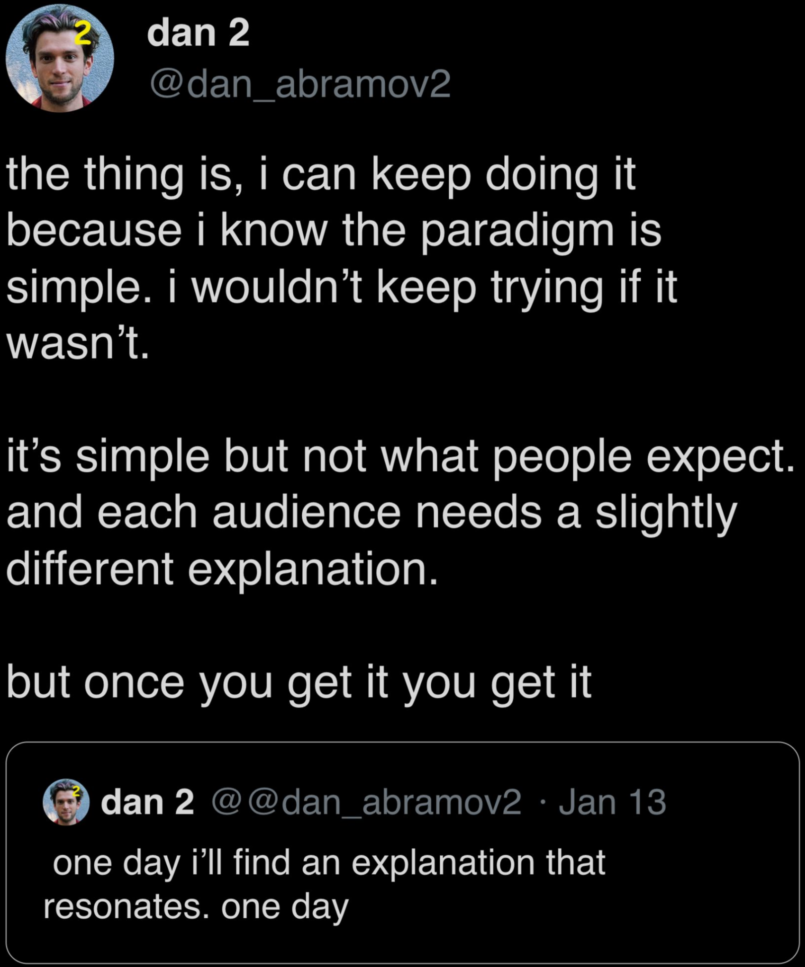
Frontend Team Lead

Спасибо, я лайкнул

t.me/thxilikeit

RSC

React Server Components


dan 2
@dan_abramov2

the thing is, i can keep doing it because i know the paradigm is simple. i wouldn’t keep trying if it wasn’t.

it’s simple but not what people expect. and each audience needs a slightly different explanation.

but once you get it you get it

dan 2 @@dan_abramov2 · Jan 13

one day i’ll find an explanation that resonates. one day

Это просто понять, но не так как все ожидают

Однажды я подберу слова

OSTROVOK.RU
БРОНИРОВАНИЕ ОТЕЛЕЙ

Я прочитал 100 статей про серверные компоненты.

Я что-нибудь понял???

➔ Что и как решают RSC?

- ➔ Что и как решают RSC?
- ➔ Ментальная модель RSC

- ➔ Что и как решают RSC?
- ➔ Ментальная модель RSC
- ➔ Фулстек компоненты

- ➔ Что и как решают RSC?
- ➔ Ментальная модель RSC
- ➔ Фулстек компоненты
- ➔ Что такое и как помогает чистая архитектура?

- ➔ Что и как решают RSC?
- ➔ Ментальная модель RSC
- ➔ Фулстек компоненты
- ➔ Что такое и как помогает чистая архитектура?
- ➔ Что стало лучше?

DISCLAIMER

- ➔ Только личный опыт
- ➔ Не призываю к холиварам, а только беру лучшее из разных миров
- ➔ Полностью следовать правилам иногда опаснее их полного несоблюдения
- ➔ Исключительно развлекательно-образовательные цели

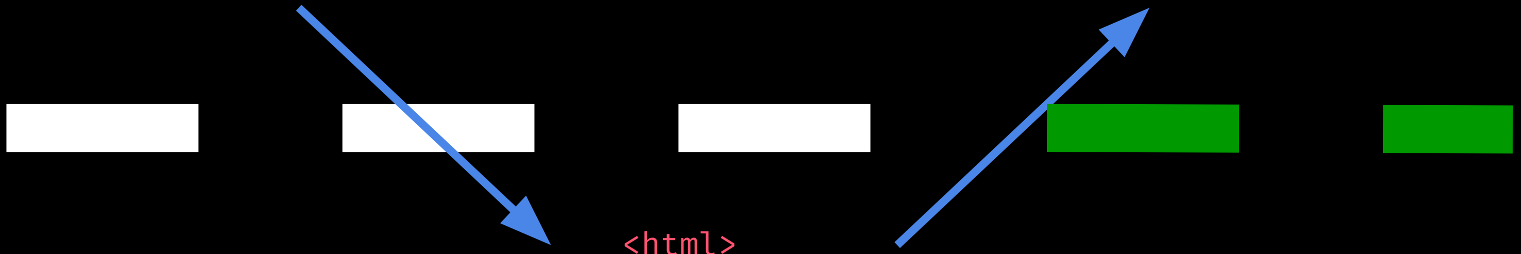
RSC

React **Server** Components

SSR

Server Side Rendering

Браузер



```
<html>
  <body>
    window.__STATE = {state}
    <script src="src/bundle.js" />
  </body>
</html>
```

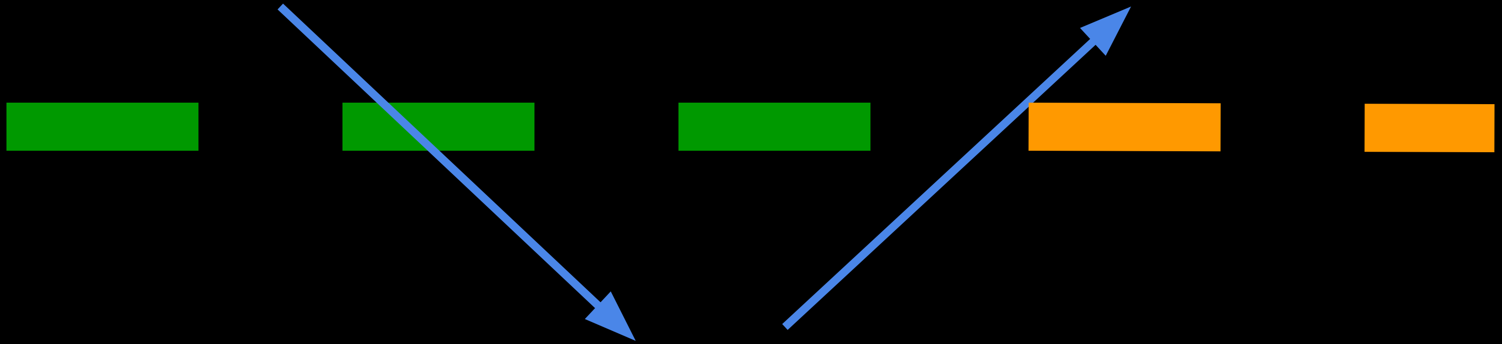
```
ReactDOM.renderToString(<App />)
```

Сервер

Браузер

Запрашивает src/bundle.js

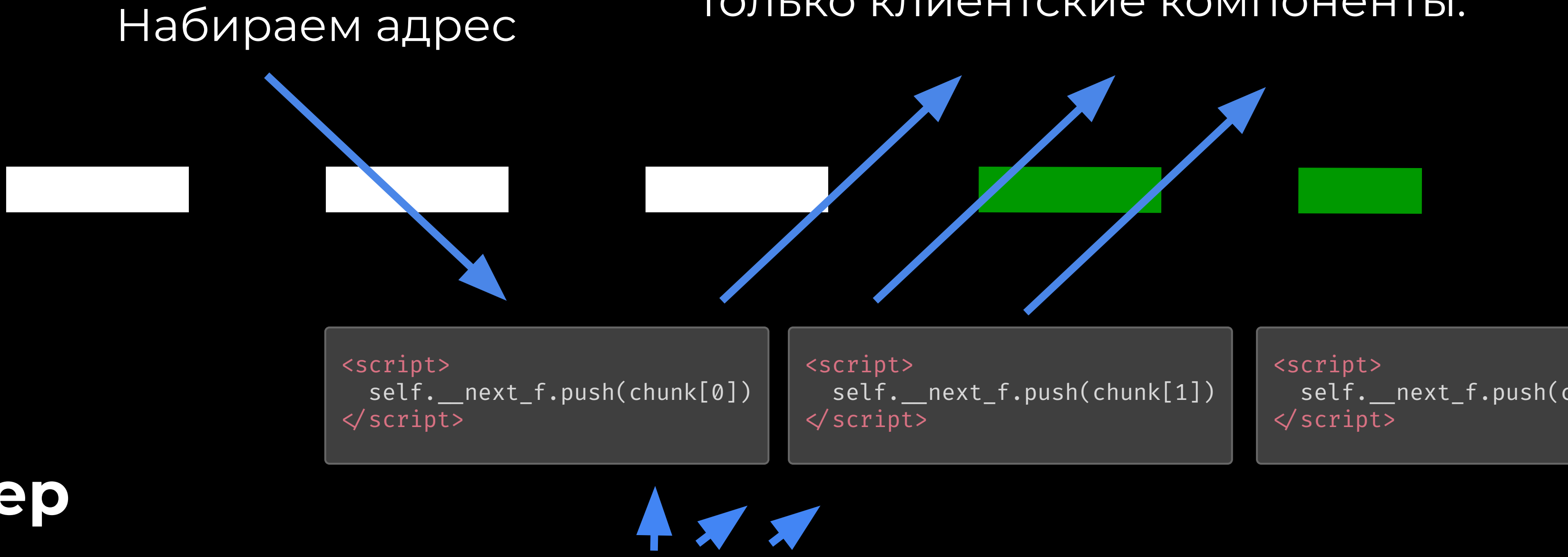
```
ReactDOM.hydrate(<App />, document.body)
```



Отдает код приложения
и фреймворка

Сервер

Браузер


Набираем адрес
Парсинг RSC Payload и вставка контента без гидратации. Гидрируем только клиентские компоненты.

```
<script>
  self.__next_f.push(chunk[0])
</script>
```

```
<script>
  self.__next_f.push(chunk[1])
</script>
```

```
<script>
  self.__next_f.push(c
</script>
```

Сервер

```
ReactDOM.renderToPipeableStream(<App />)
```

```
["$","div",null,{"children":
["Hello World",
["$","$L1",null,{}]]}]
```

```
"use server"

export default async function () {
  const appState = await getAppState();
   ...
}
```

“use server” значит только на сервере?

```
"use server"

export default async function () {
  const appState = await getAppState();

  return (
    <div>
      <span>Hello World</span>
    </div>
  )
}
```

```
React.createElement(
  "div",
  null,
  React.createElement(
    "span",
    null,
    "Hello World"));
```

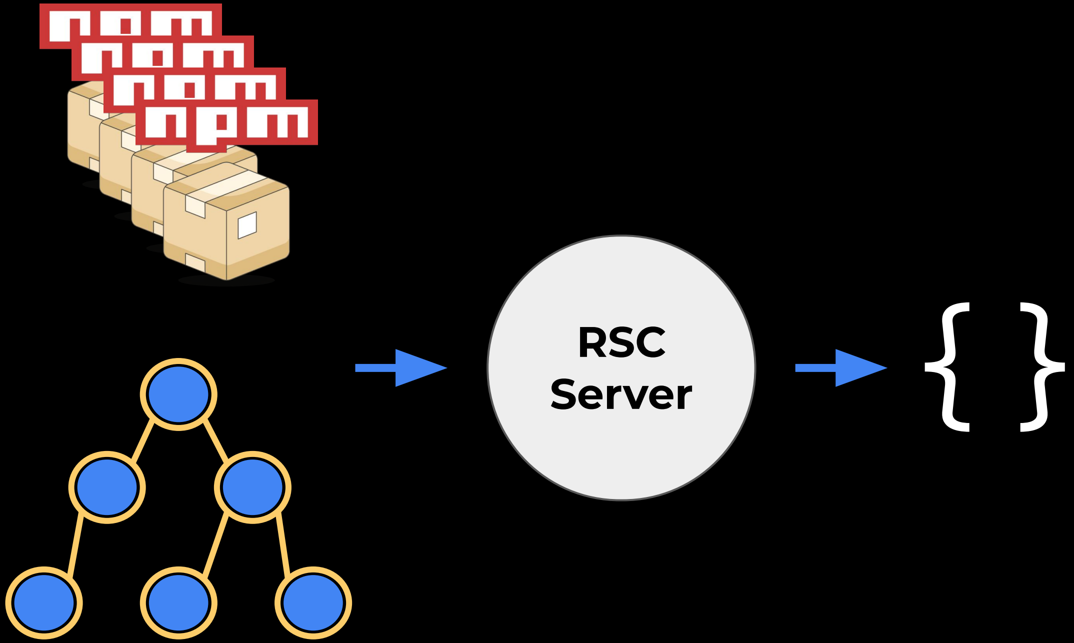
```
{
  type,
  props,
  children
}
```

type props children

```
["$","div",null,{"children":
["Hello World",
["$","$L1",null,{}]]}]
```

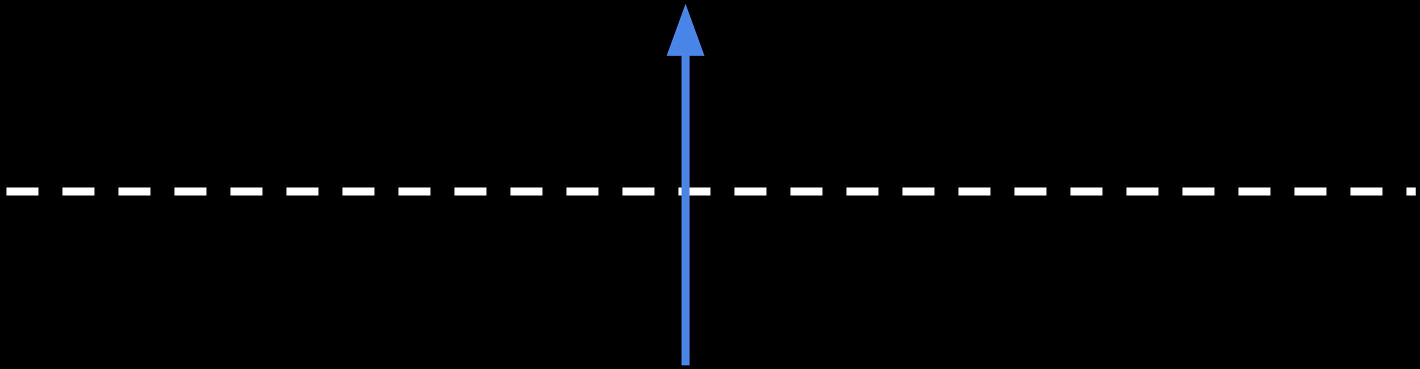
```
<script>
  self.__next_f.push(chunk[0])
</script>
```

В чем, собственно, проблема?


npm
npm
npm
npm
RSC Server
{ }

Браузер

Все зашито в представлении,
не нужно идти в настоящий
бэкенд или БД за данными.



Серверные компоненты
быстро собирают данные
из сервиса по соседству

Сервер

Где и как попробовать?

The biggest change is that we introduced `async` / `await` as the primary way to do data fetching from Server Components. We also plan to support data loading from the client by introducing a new Hook called `use` that unwraps Promises. Although we can't support `async / await` in arbitrary components in client-only apps, we plan to add support for it when you structure your client-only app similar to how RSC apps are structured.

Now that we have data fetching pretty well sorted, we're exploring the other direction: sending data from the client to the server, so that you can execute database mutations and implement forms. We're doing this by letting you pass Server Action functions across the server/client boundary, which the client can then call, providing seamless RPC. Server Actions also give you progressively enhanced forms before JavaScript loads.

React Server Components has shipped in Next.js App Router. This showcases a deep integration of a router that really buys into RSC as a primitive, but it's not the only way to build a RSC-compatible router and framework. There's a clear separation for features provided by the RSC spec and implementation. React Server Components is meant as a spec for components that work across compatible React frameworks.

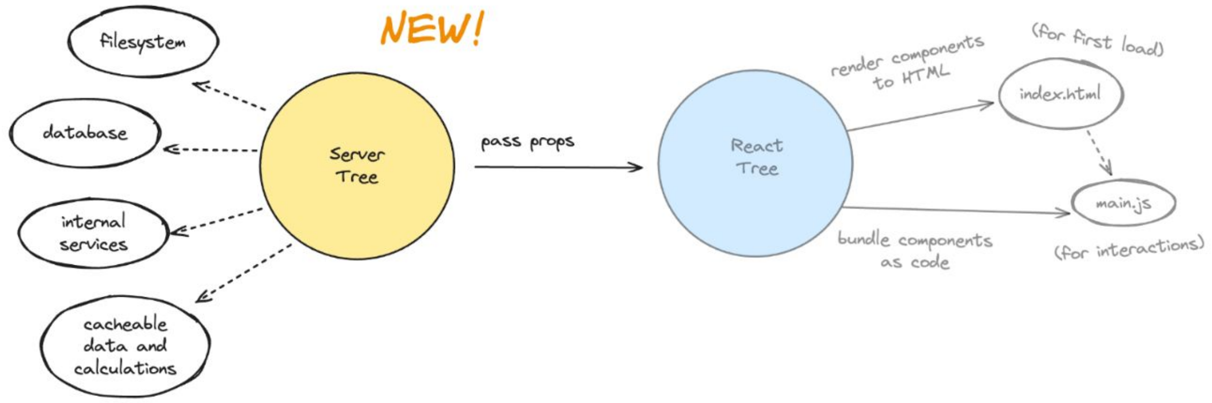
We generally recommend using an existing framework, but if you need to build your own custom framework, it is possible. Building your own RSC-compatible framework is not as easy as we'd like it to be, mainly due to the deep bundler integration needed. The current generation of bundlers are great for use on the client, but they weren't designed with first-class support for splitting a single module graph between the server and the client. This is why we're now partnering directly with bundler developers to get the primitives for RSC built-in.

Не Next.js единым

Why do Client Components get SSR'd to HTML? #4

gaearon on Jun 2, 2023 · 0 comments

RSC does not *change* that mental model, but it adds a new layer *before* any of that existing code runs:

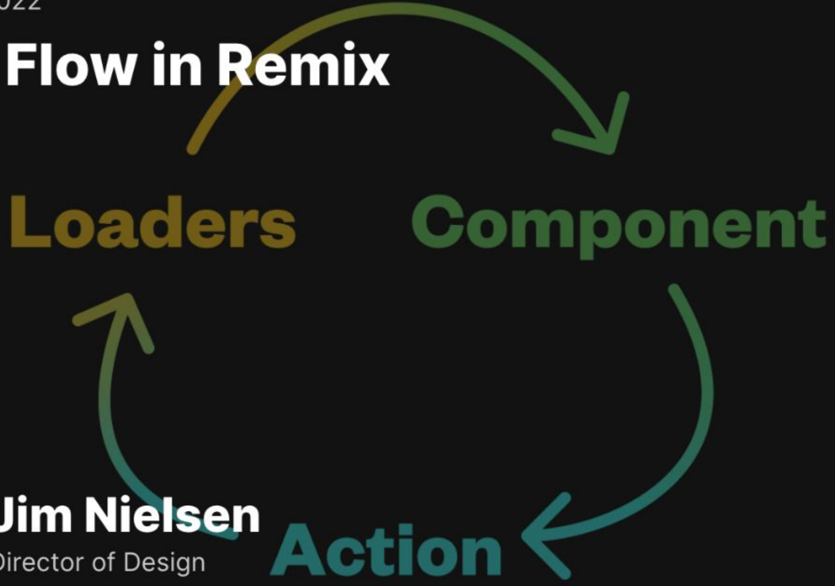


The RSC Server layer might remind you of Remix loaders, Astro templates, build-time scripts, and other code that runs ahead-of-time — but in the form of React components. To disambiguate, the "React you already knew" (and all its features) is called Client:

WTF?
Ryan Florence, Reactathon 2022
Reactathon
Reactathon
Vercel

JUNE 22, 2022

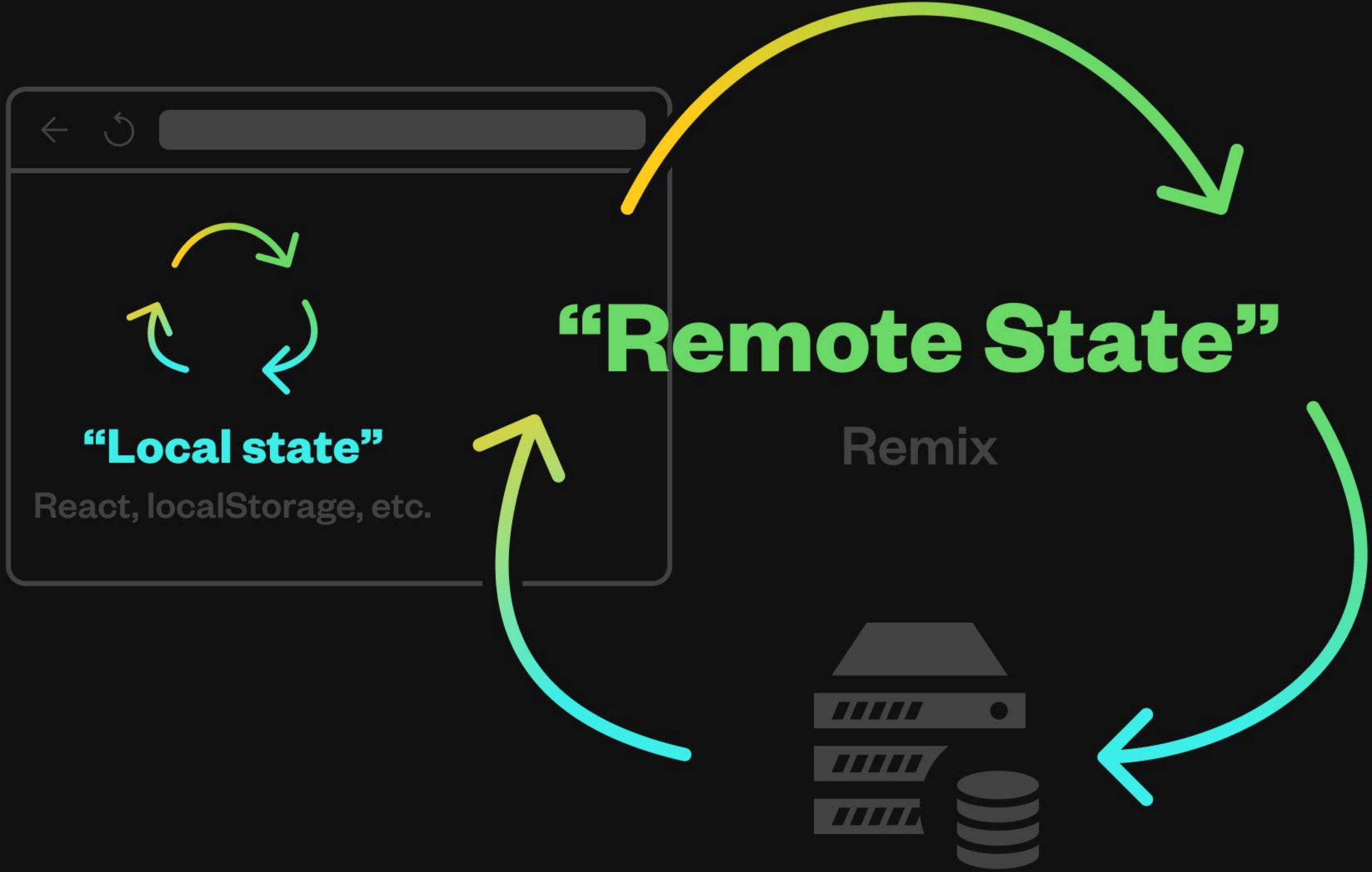
Data Flow in Remix

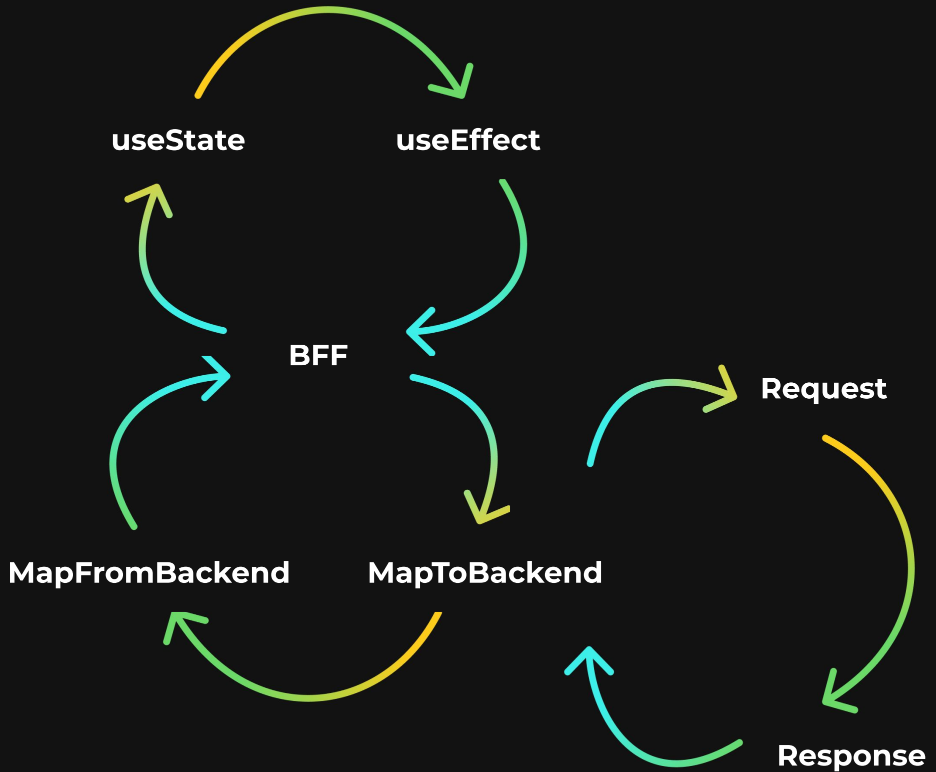


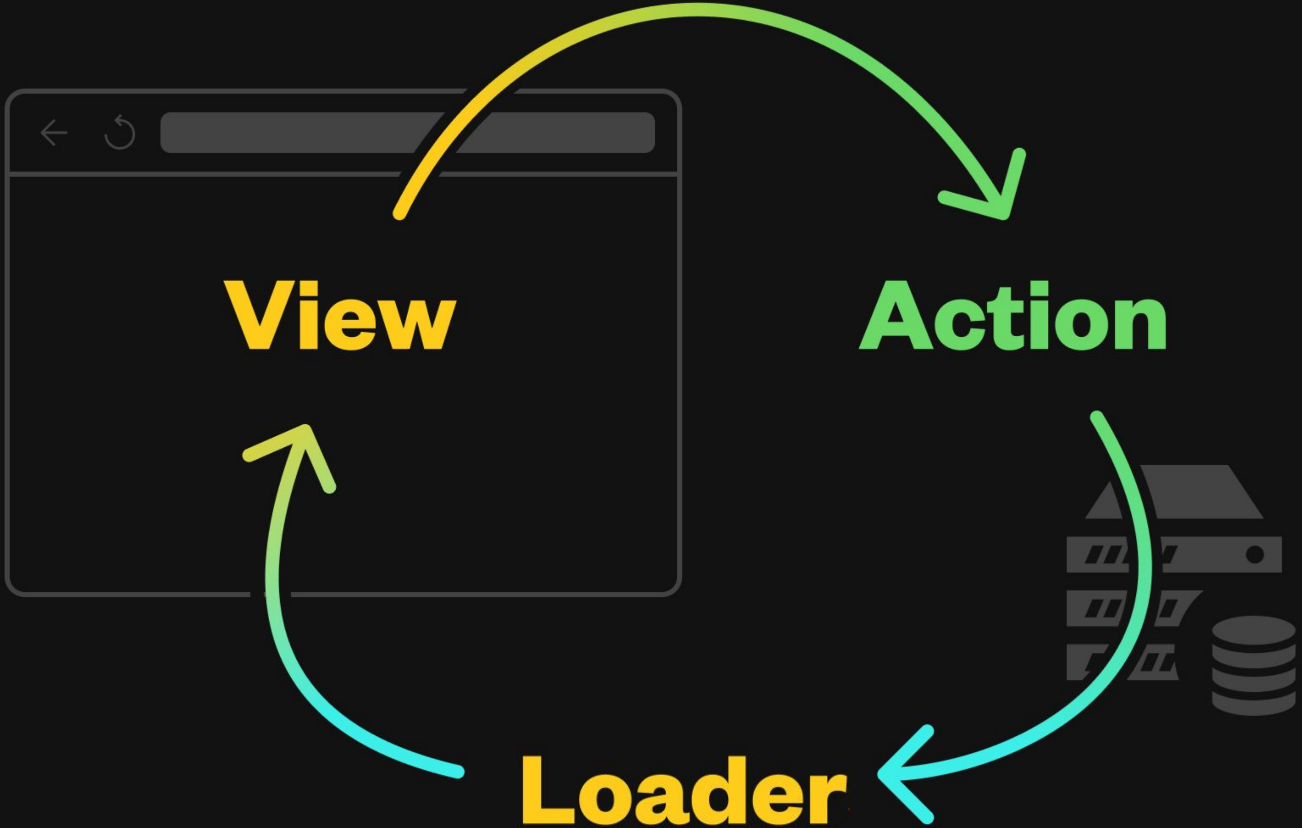
Jim Nielsen
Director of Design

When React first appeared on the scene, one of its most compelling features was its “one-way data flow”. This is still outlined in the React docs under the page “Thinking in React”

UI = f(state)


"Local state"
React, localStorage, etc.
"Remote State"
Remix


useState
useEffect
BFF
Request
MapFromBackend
MapToBackend
Response

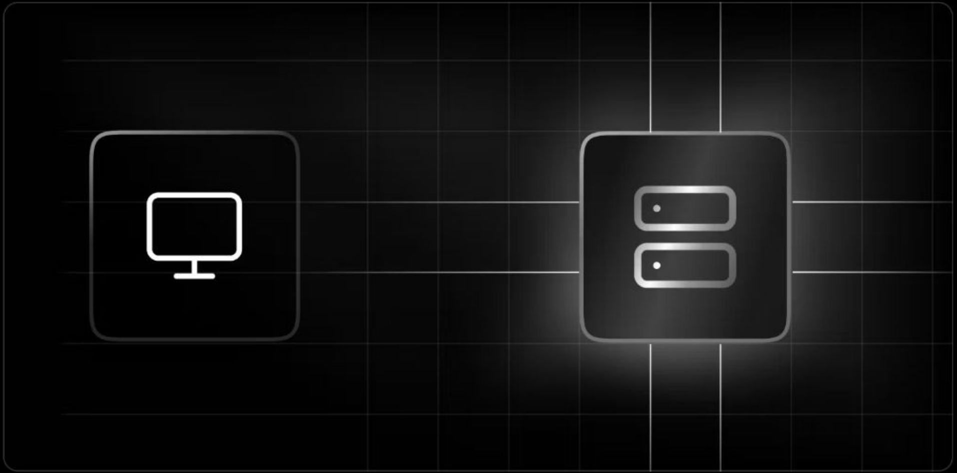

View
Action
Loader

← Back to Blog

Engineering Tuesday, August 1st 2023

Understanding React Server Components

Learn the fundamentals of React Server Components, to better understand why (and when) to adopt.



Posted by

Alice Alexandra Moore
Content Engineer

Related reading

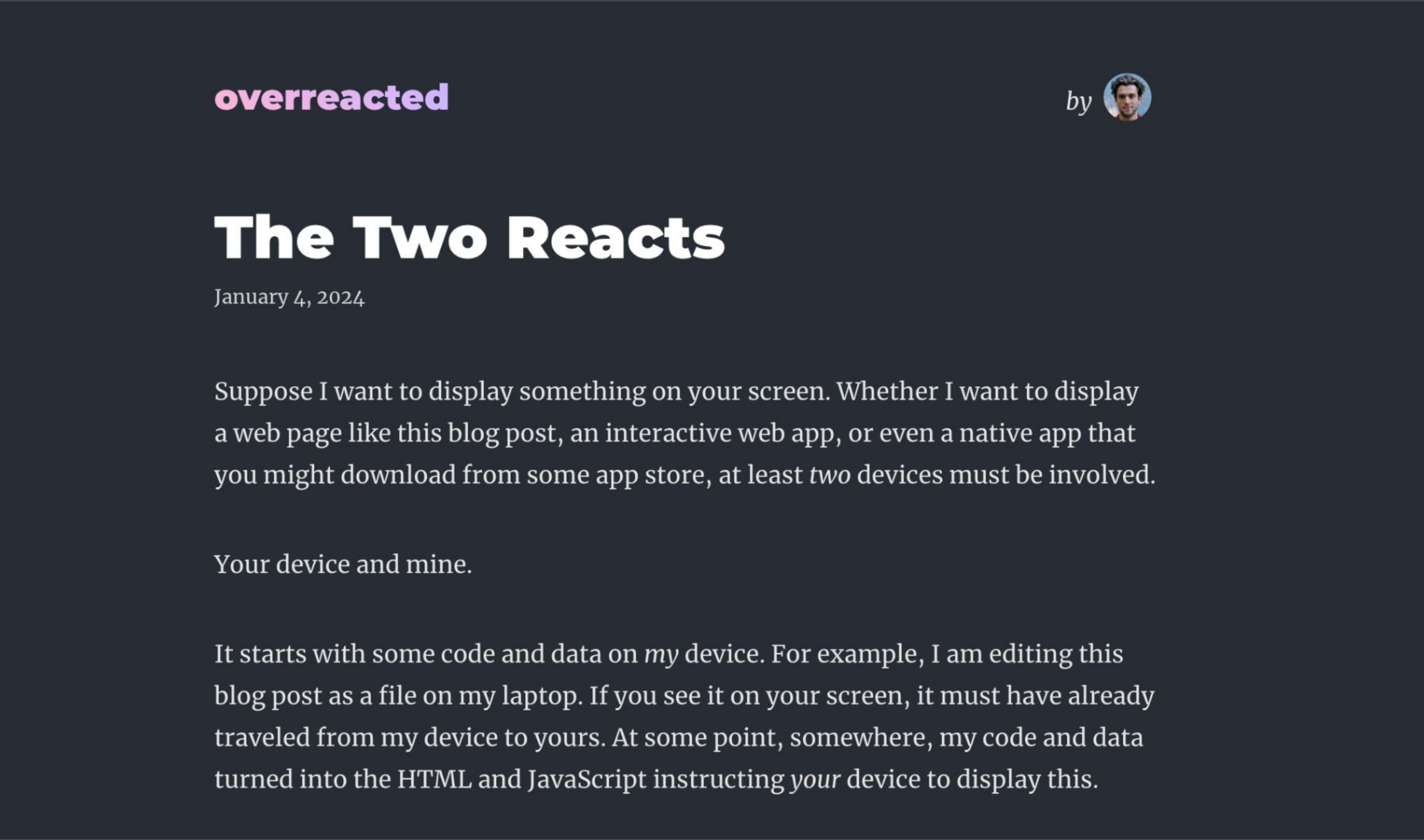
Less code, better UX: Fetching data faster with the Next.js 13 App Router

Alice Alexandra Moore, Ariel Kanter

Vercel Data Cache: A progressive cache, integrated with Next.js

Casey Gowrie, Luba Kravchenko,

UI = f(data)(state)



overreacted by

The Two Reacts

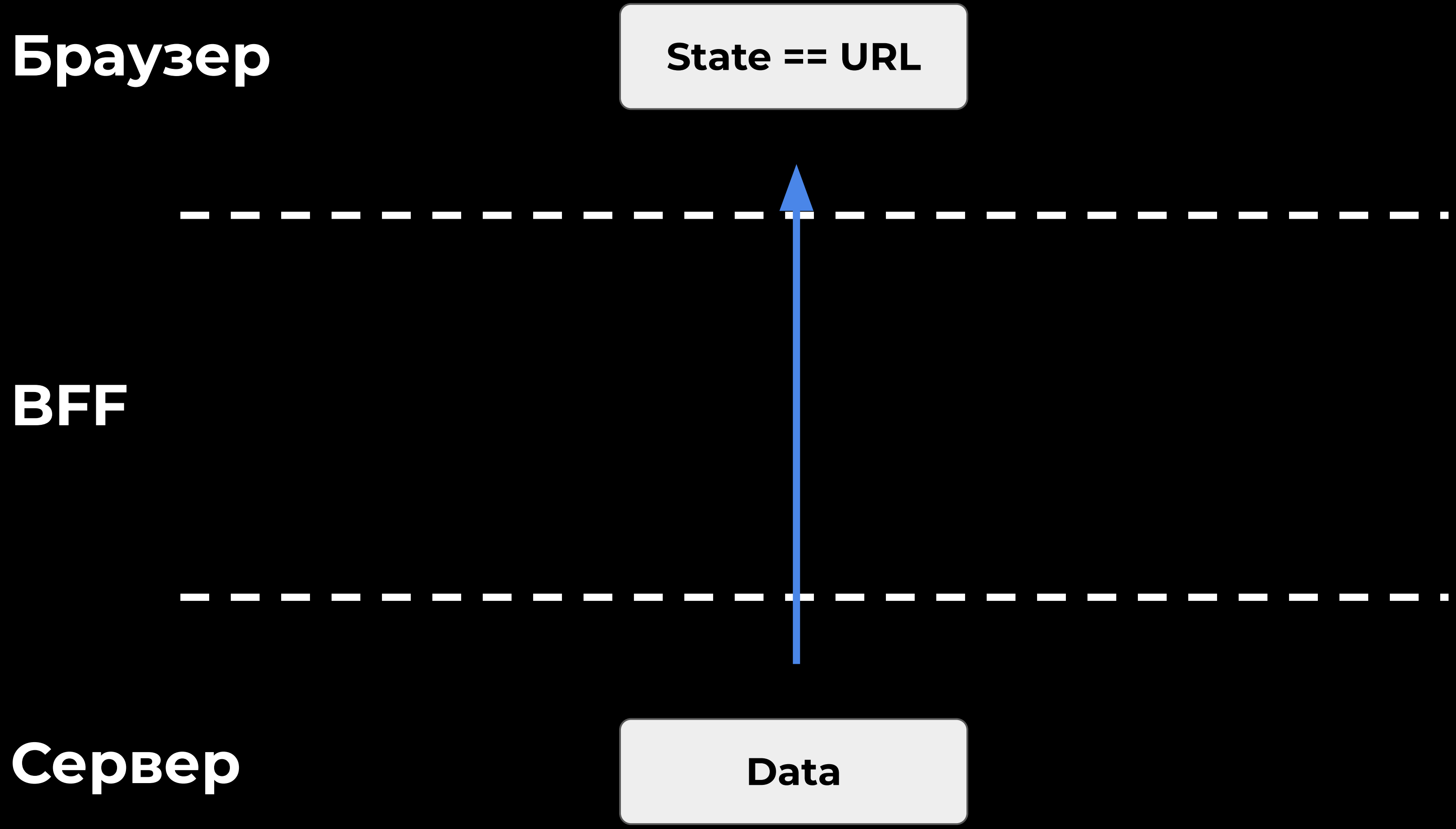
January 4, 2024

Suppose I want to display something on your screen. Whether I want to display a web page like this blog post, an interactive web app, or even a native app that you might download from some app store, at least *two* devices must be involved.

Your device and mine.

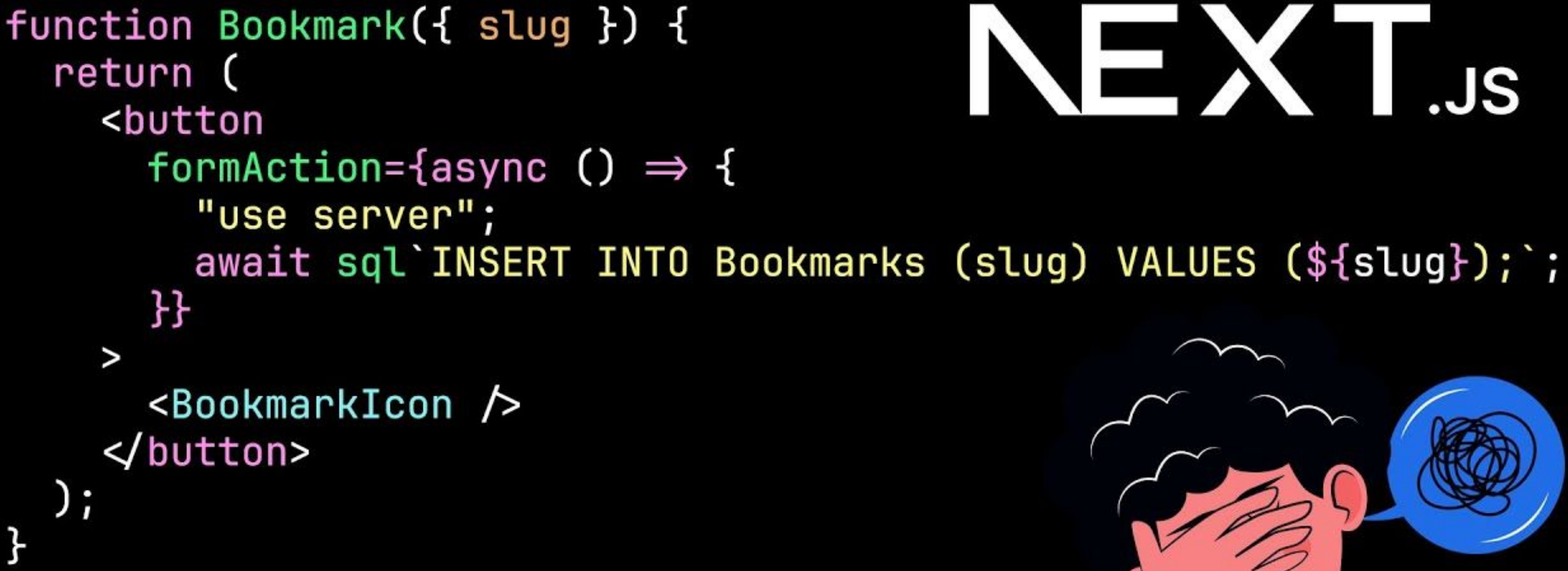
It starts with some code and data on *my* device. For example, I am editing this blog post as a file on my laptop. If you see it on your screen, it must have already traveled from my device to yours. At some point, somewhere, my code and data turned into the HTML and JavaScript instructing *your* device to display this.

UI = f(data)(state)


Браузер
State == URL
BFF
Сервер
Data

UI = f(data)(state)

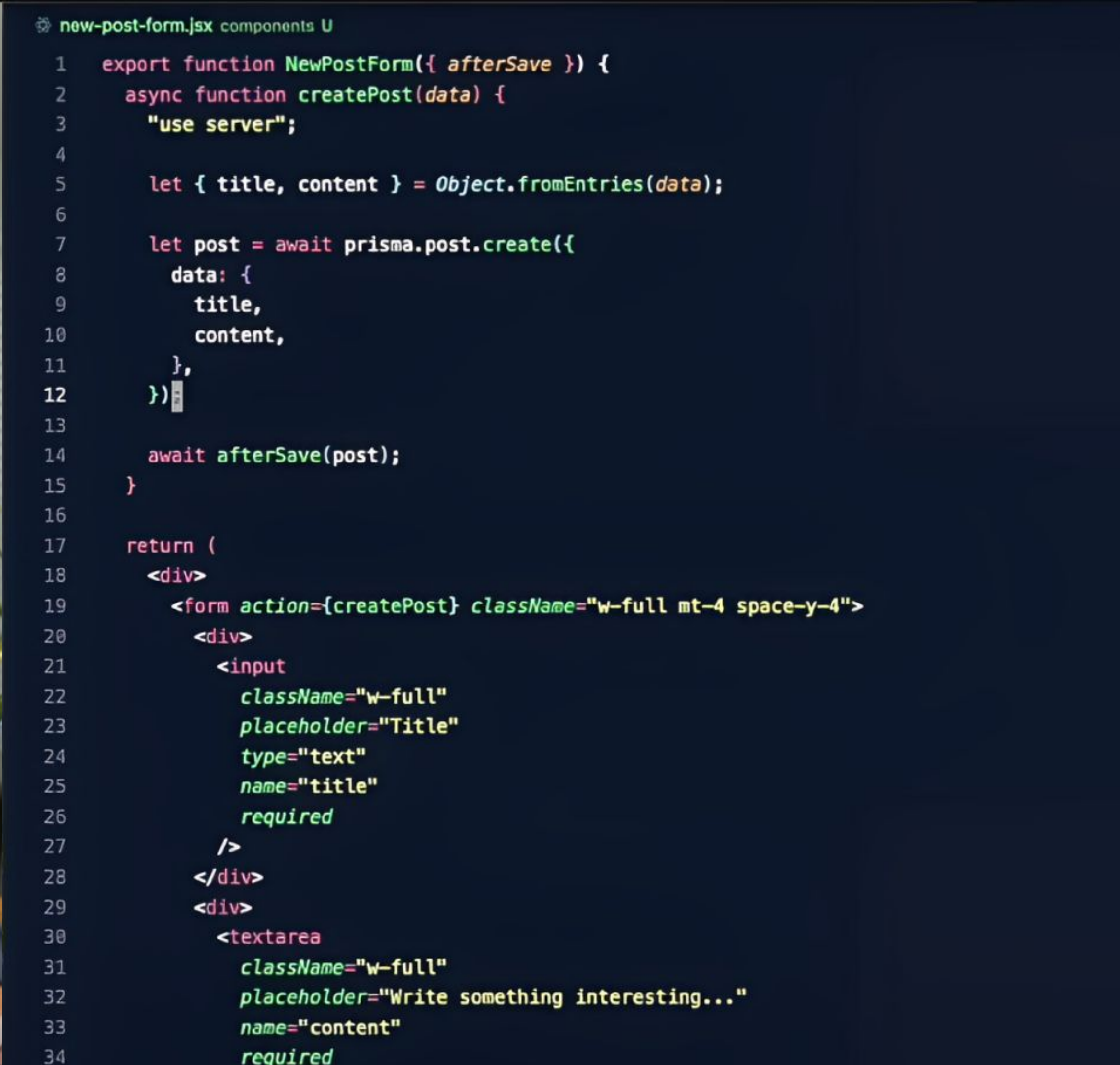
Ты фулстек, Гарри



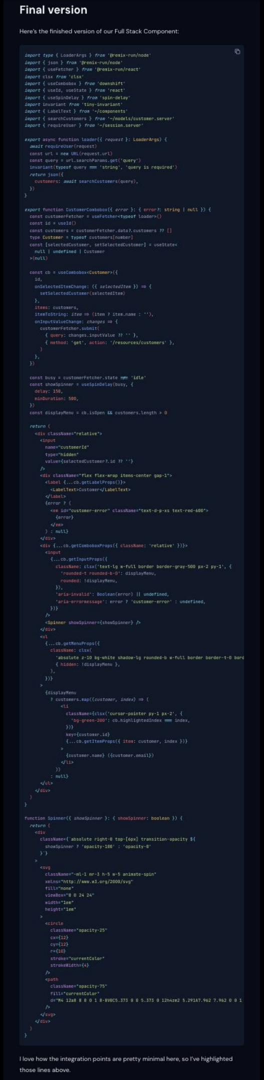
```
function Bookmark({ slug }) {
  return (
    <button
      formAction={async () => {
        "use server";
        await sql`INSERT INTO Bookmarks (slug) VALUES (${slug});`;
      }}
    >
      <BookmarkIcon />
    </button>
  );
}
```

NEXT.JS

kudos Pedro Pessoa



```jsx
export function NewPostForm({ afterSave }) {
  async function createPost(data) {
    "use server";

    let { title, content } = Object.fromEntries(data);

    let post = await prisma.post.create({
      data: {
        title,
        content,
      },
    });

    await afterSave(post);
  }

  return (
    <div>
      <form action={createPost} className="w-full mt-4 space-y-4">
        <div>
          <input
            className="w-full"
            placeholder="Title"
            type="text"
            name="title"
            required
          />
        </div>
        <div>
          <textarea
            className="w-full"
            placeholder="Write something interesting..."
            name="content"
            required
```



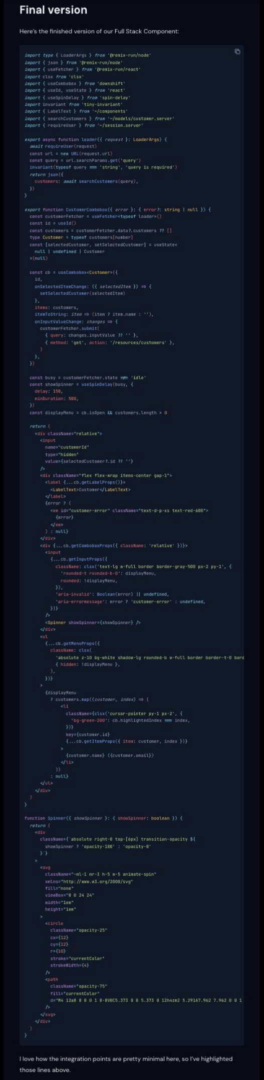
Final version

Here's the finished version of our Full Stack Component:

```tsx
import type { LoaderArgs } from '@remix-run/node'
import { json } from '@remix-run/node'
import { useFetcher } from '@remix-run/react'
import clsx from 'clsx'
import { useCombobox } from 'downshift'
import { useId, useState } from 'react'
import { useSpinDelay } from 'spin-delay'
import invariant from 'tiny-invariant'
import { LabelText } from '~/components'
import { searchCustomers } from '~/models/customer.server'
import { requireUser } from '~/session.server'

export async function loader({ request }: LoaderArgs) {
  await requireUser(request)
  const url = new URL(request.url)
  const query = url.searchParams.get('query')
  invariant(typeof query === 'string', 'query is required')
  return json({
    customers: await searchCustomers(query),
  })
}

export function CustomerCombobox({ error }: { error?: string | null }) {
  const customerFetcher = useFetcher<typeof loader>()
  const id = useId()
  const customers = customerFetcher.data?.customers ?? []
  type Customer = typeof customers[number]
  const [selectedCustomer, setSelectedCustomer] = useState<
    null | undefined | Customer
  >(null)

  const cb = useCombobox<Customer>({
    id,
    onSelectedItemChange: ({ selectedItem }) => {
      setSelectedCustomer(selectedItem)
    },
    items: customers,
    itemToString: item => (item ? item.name : ''),
    onInputValueChange: changes => {
      customerFetcher.submit(
        { query: changes.inputValue ?? '' },
        { method: 'get', action: '/resources/customers' },
      )
    },
  })

  const busy = customerFetcher.state !== 'idle'
  const showSpinner = useSpinDelay(busy, {
    delay: 150,
    minDuration: 500,
  })
  const displayMenu = cb.isOpen && customers.length > 0

  return (
    <div className="relative">
      <input
        name="customerId"
        type="hidden"
        value={selectedCustomer?.id ?? ''}
      />
      <div className="flex flex-wrap items-center gap-1">
        <label {...cb.getLabelProps()}>
          <LabelText>Customer</LabelText>
        </label>
        {error ? (
          <em id="customer-error" className="text-d-p-xs text-red-600">
            {error}
          </em>
        ) : null}
      </div>
      <div {...cb.getComboboxProps({ className: 'relative' })}>
        <input
          {...cb.getInputProps({
            className: clsx('text-lg w-full border border-gray-500 px-2 py-1', {
              'rounded-t rounded-b-0': displayMenu,
              rounded: !displayMenu,
            }),
            'aria-invalid': Boolean(error) || undefined,
            'aria-errormessage': error ? 'customer-error' : undefined,
          })}
        />
        <Spinner showSpinner={showSpinner} />
      </div>
      <ul
        {...cb.getMenuProps({
          className: clsx(
            'absolute z-10 bg-white shadow-lg rounded-b w-full border border-t-0 bord
            { hidden: !displayMenu },
          ),
        })}
      >
        {displayMenu
          ? customers.map((customer, index) => (
              <li
                className={clsx('cursor-pointer py-1 px-2', {
                  'bg-green-200': cb.highlightedIndex === index,
                })}
                key={customer.id}
                {...cb.getItemProps({ item: customer, index })}
              >
                {customer.name} ({customer.email})
              </li>
            ))
          : null}
      </ul>
    </div>
  )
}

function Spinner({ showSpinner }: { showSpinner: boolean }) {
  return (
    <div
      className={`absolute right-0 top-[6px] transition-opacity ${
        showSpinner ? 'opacity-100' : 'opacity-0'
      }`}
    >
      <svg
        className="-ml-1 mr-3 h-5 w-5 animate-spin"
        xmlns="http://www.w3.org/2000/svg"
        fill="none"
        viewBox="0 0 24 24"
        width="1em"
        height="1em"
      >
        <circle
          className="opacity-25"
          cx={12}
          cy={12}
          r={10}
          stroke="currentColor"
          strokeWidth={4}
        />
        <path
          className="opacity-75"
          fill="currentColor"
          d="M4 12a8 8 0 0 1 8-8V0C5.373 0 0 5.373 0 12h4zm2 5.291A7.962 7.962 0 0 1
        />
      </svg>
    </div>
  )
}
```

I love how the integration points are pretty minimal here, so I've highlighted those lines above.

View



Final version

Here's the finished version of our Full Stack Component:

```
import type { LoaderArgs } from '@remix-run/node'
import { json } from '@remix-run/node'
import { useFetcher } from '@remix-run/react'
import clsx from 'clsx'
import { useCombobox } from 'downshift'
import { useId, useState } from 'react'
import { useSpinDelay } from 'spin-delay'
import invariant from 'tiny-invariant'
import { LabelText } from '~/components'
import { searchCustomers } from '~/models/customer.server'
import { requireUser } from '~/session.server'

export async function loader({ request }: LoaderArgs) {
  await requireUser(request)
  const url = new URL(request.url)
  const query = url.searchParams.get('query')
  invariant(typeof query === 'string', 'query is required')
  return json({
    customers: await searchCustomers(query),
  })
}

export function CustomerCombobox({ error }: { error?: string | null }) {
  const customerFetcher = useFetcher<typeof loader>()
  const id = useId()
  const customers = customerFetcher.data?.customers ?? []
  type Customer = typeof customers[number]
  const [selectedCustomer, setSelectedCustomer] = useState<
    null | undefined | Customer
  >(null)

  const cb = useCombobox<Customer>({
    id,
    onSelectedItemChange: ({ selectedItem }) => {
      setSelectedCustomer(selectedItem)
    },
    items: customers,
    itemToString: item => (item ? item.name : ''),
    onInputValueChange: changes => {
      customerFetcher.submit(
        { query: changes.inputValue ?? '' },
        { method: 'get', action: '/resources/customers' },
      )
    },
  })

  const busy = customerFetcher.state !== 'idle'
  const showSpinner = useSpinDelay(busy, {
    delay: 150,
    minDuration: 500,
  })
  const displayMenu = cb.isOpen && customers.length > 0

  return (
    <div className="relative">
      <input
        name="customerId"
        type="hidden"
        value={selectedCustomer?.id ?? ''}
      />
      <div className="flex flex-wrap items-center gap-1">
        <label {...cb.getLabelProps()}>
          <LabelText>Customer</LabelText>
        </label>
        {error ? (
          <em id="customer-error" className="text-d-p-xs text-red-600">
            {error}
          </em>
        ) : null}
      </div>
      <div {...cb.getComboboxProps({ className: 'relative' })}>
        <input
          {...cb.getInputProps({
            className: clsx('text-lg w-full border border-gray-500 px-2 py-1', {
              'rounded-t rounded-b-0': displayMenu,
              rounded: !displayMenu,
            }),
            'aria-invalid': Boolean(error) || undefined,
            'aria-errormessage': error ? 'customer-error' : undefined,
          })}
        />
        <Spinner showSpinner={showSpinner} />
      </div>
      <ul
        {...cb.getMenuProps({
          className: clsx(
            'absolute z-10 bg-white shadow-lg rounded-b w-full border border-t-0 bord
            { hidden: !displayMenu },
          ),
        })}
      >
        {displayMenu
          ? customers.map((customer, index) => (
              <li
                className={clsx('cursor-pointer py-1 px-2', {
                  'bg-green-200': cb.highlightedIndex === index,
                })}
                key={customer.id}
                {...cb.getItemProps({ item: customer, index })}
              >
                {customer.name} ({customer.email})
              </li>
            ))
          : null}
      </ul>
    </div>
  )
}

function Spinner({ showSpinner }: { showSpinner: boolean }) {
  return (
    <div
      className={`absolute right-0 top-[6px] transition-opacity ${
        showSpinner ? 'opacity-100' : 'opacity-0'
      }`}
    >
      <svg
        className="-ml-1 mr-3 h-5 w-5 animate-spin"
        xmlns="http://www.w3.org/2000/svg"
        fill="none"
        viewBox="0 0 24 24"
        width="1em"
        height="1em"
      >
        <circle
          className="opacity-25"
          cx={12}
          cy={12}
          r={10}
          stroke="currentColor"
          strokeWidth={4}
        />
        <path
          className="opacity-75"
          fill="currentColor"
          d="M4 12a8 8 0 0 1 8-8V0C5.373 0 0 5.373 0 12h4zm2 5.291A7.962 7.962 0 0 1
        />
      </svg>
    </div>
  )
}
```

I love how the integration points are pretty minimal here, so I've highlighted those lines above.

Error Handling

Validation

State handling

New Button

View

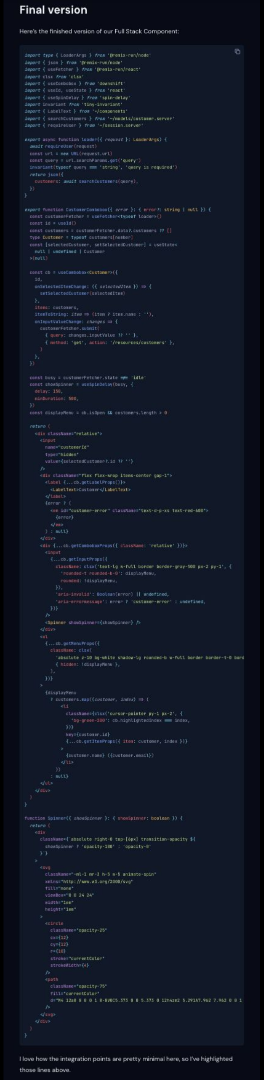
CSS

Translation

API

Validation

Mapping



Final version

Here's the finished version of our Full Stack Component:

```
import type { LoaderArgs } from '@remix-run/node'
import { json } from '@remix-run/node'
import { useFetcher } from '@remix-run/react'
import clsx from 'clsx'
import { useCombobox } from 'downshift'
import { useId, useState } from 'react'
import { useSpinDelay } from 'spin-delay'
import invariant from 'tiny-invariant'
import { LabelText } from '~/components'
import { searchCustomers } from '~/models/customer.server'
import { requireUser } from '~/session.server'

export async function loader({ request }: LoaderArgs) {
  await requireUser(request)
  const url = new URL(request.url)
  const query = url.searchParams.get('query')
  invariant(typeof query === 'string', 'query is required')
  return json({
    customers: await searchCustomers(query),
  })
}

export function CustomerCombobox({ error }: { error?: string | null }) {
  const customerFetcher = useFetcher<typeof loader>()
  const id = useId()
  const customers = customerFetcher.data?.customers ?? []
  type Customer = typeof customers[number]
  const [selectedCustomer, setSelectedCustomer] = useState<
    null | undefined | Customer
  >(null)

  const cb = useCombobox<Customer>({
    id,
    onSelectedItemChange: ({ selectedItem }) => {
      setSelectedCustomer(selectedItem)
    },
    items: customers,
    itemToString: item => (item ? item.name : ''),
    onInputValueChange: changes => {
      customerFetcher.submit(
        { query: changes.inputValue ?? '' },
        { method: 'get', action: '/resources/customers' },
      )
    },
  })

  const busy = customerFetcher.state !== 'idle'
  const showSpinner = useSpinDelay(busy, {
    delay: 150,
    minDuration: 500,
  })
  const displayMenu = cb.isOpen && customers.length > 0

  return (
    <div className="relative">
      <input
        name="customerId"
        type="hidden"
        value={selectedCustomer?.id ?? ''}
      />
      <div className="flex flex-wrap items-center gap-1">
        <label {...cb.getLabelProps()}>
          <LabelText>Customer</LabelText>
        </label>
        {error ? (
          <em id="customer-error" className="text-d-p-xs text-red-600">
            {error}
          </em>
        ) : null}
      </div>
      <div {...cb.getComboboxProps({ className: 'relative' })}>
        <input
          {...cb.getInputProps({
            className: clsx('text-lg w-full border border-gray-500 px-2 py-1', {
              'rounded-t rounded-b-0': displayMenu,
              rounded: !displayMenu,
            }),
            'aria-invalid': Boolean(error) || undefined,
            'aria-errormessage': error ? 'customer-error' : undefined,
          })}
        />
        <Spinner showSpinner={showSpinner} />
      </div>
      <ul
        {...cb.getMenuProps({
          className: clsx(
            'absolute z-10 bg-white shadow-lg rounded-b w-full border border-t-0 bord
            { hidden: !displayMenu },
          ),
        })}
      >
        {displayMenu
          ? customers.map((customer, index) => (
              <li
                className={clsx('cursor-pointer py-1 px-2', {
                  'bg-green-200': cb.highlightedIndex === index,
                })}
                key={customer.id}
                {...cb.getItemProps({ item: customer, index })}
              >
                {customer.name} ({customer.email})
              </li>
            ))
          : null}
      </ul>
    </div>
  )
}

function Spinner({ showSpinner }: { showSpinner: boolean }) {
  return (
    <div
      className={`absolute right-0 top-[6px] transition-opacity ${
        showSpinner ? 'opacity-100' : 'opacity-0'
      }`}
    >
      <svg
        className="-ml-1 mr-3 h-5 w-5 animate-spin"
        xmlns="http://www.w3.org/2000/svg"
        fill="none"
        viewBox="0 0 24 24"
        width="1em"
        height="1em"
      >
        <circle
          className="opacity-25"
          cx={12}
          cy={12}
          r={10}
          stroke="currentColor"
          strokeWidth={4}
        />
        <path
          className="opacity-75"
          fill="currentColor"
          d="M4 12a8 8 0 0 1 8-8V0C5.373 0 0 5.373 0 12h4zm2 5.291A7.962 7.962 0 0 1
        />
      </svg>
    </div>
  )
}
```

I love how the integration points are pretty minimal here, so I've highlighted those lines above.

New Error

Error Handling

New conditions

State handling

New action

SRP? Нет, не слышал

- ➔ Гибкость
- ➔ Тестируемость
- ➔ Модульность
- ➔ Поддерживаемость
- ➔ Разделение ответственности
- ➔ Дешевая стоимость изменения

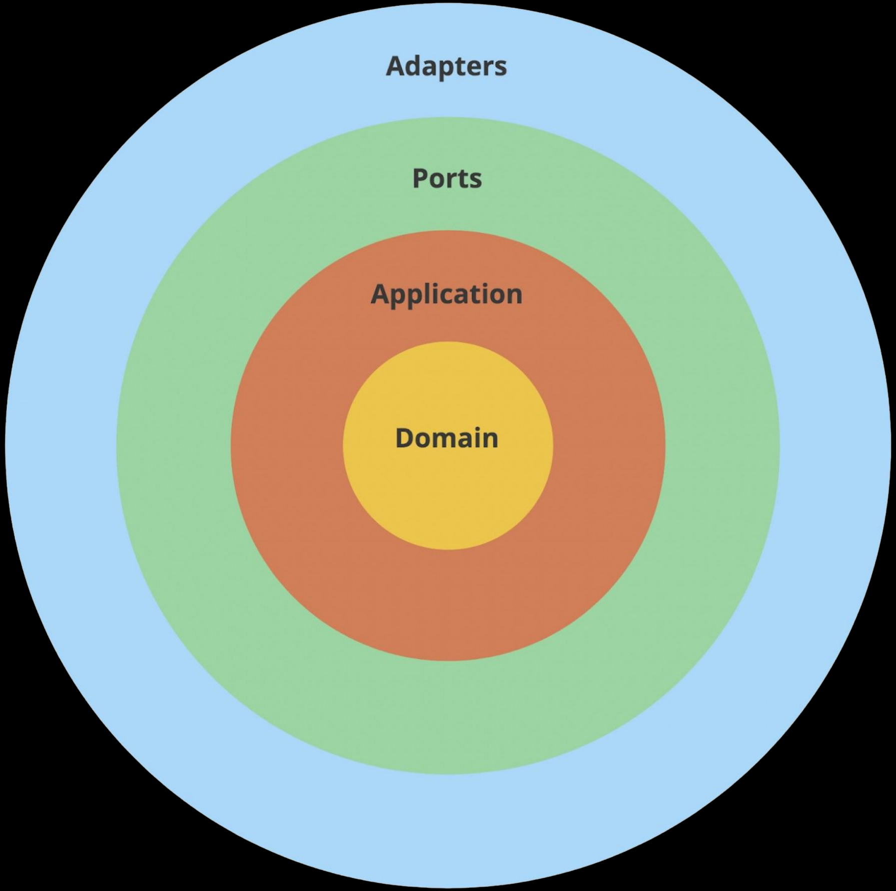
```
"use server"

export default async function () {
  const result = await
get_Grouped_Products_For_Listing_Page_From_All_Suppliers_OrError();

  if (result.isError) {
    return <Error />
  }

  return <View props={result.value} />
}
```

```
"use server"

export default async function (searchParams: SearchParams) {
  const result = await …All_Suppliers_OrError(searchParams);

  if (result.isError) {
    return <Error />
  }

  return <View props={result.value} />
}
```

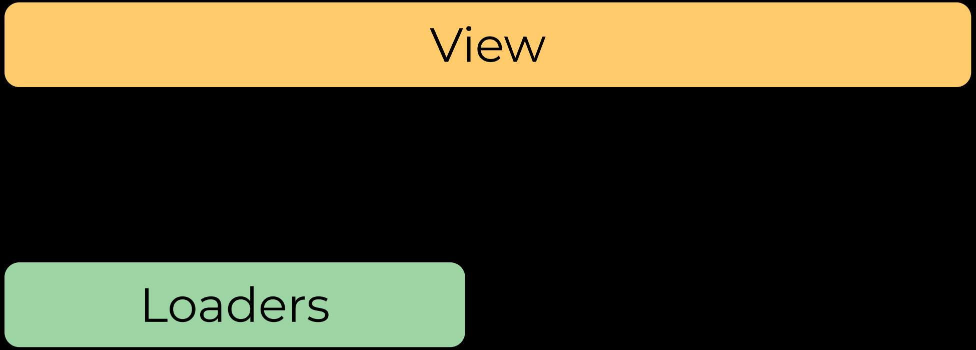
Well yes, but actually no

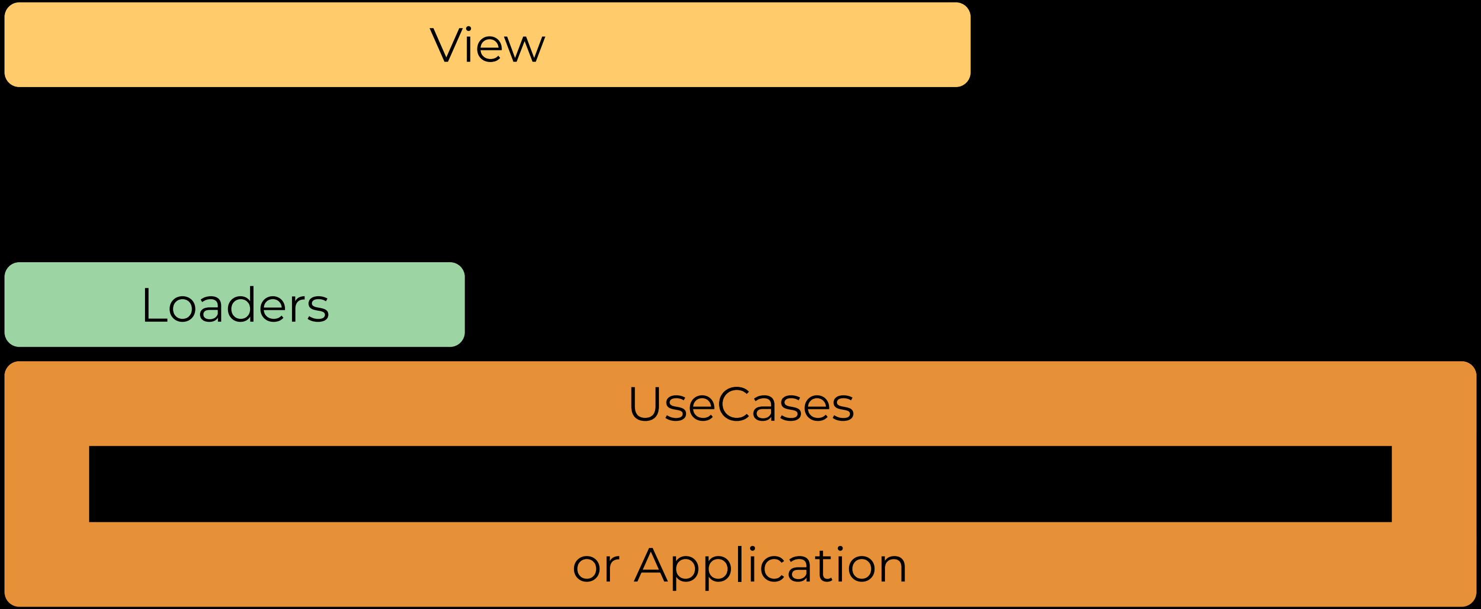
```
"use server"

export default async function (searchParams: SearchParams) {

  const result: OurInterface = await …OrError(searchParams);

  …
}
```

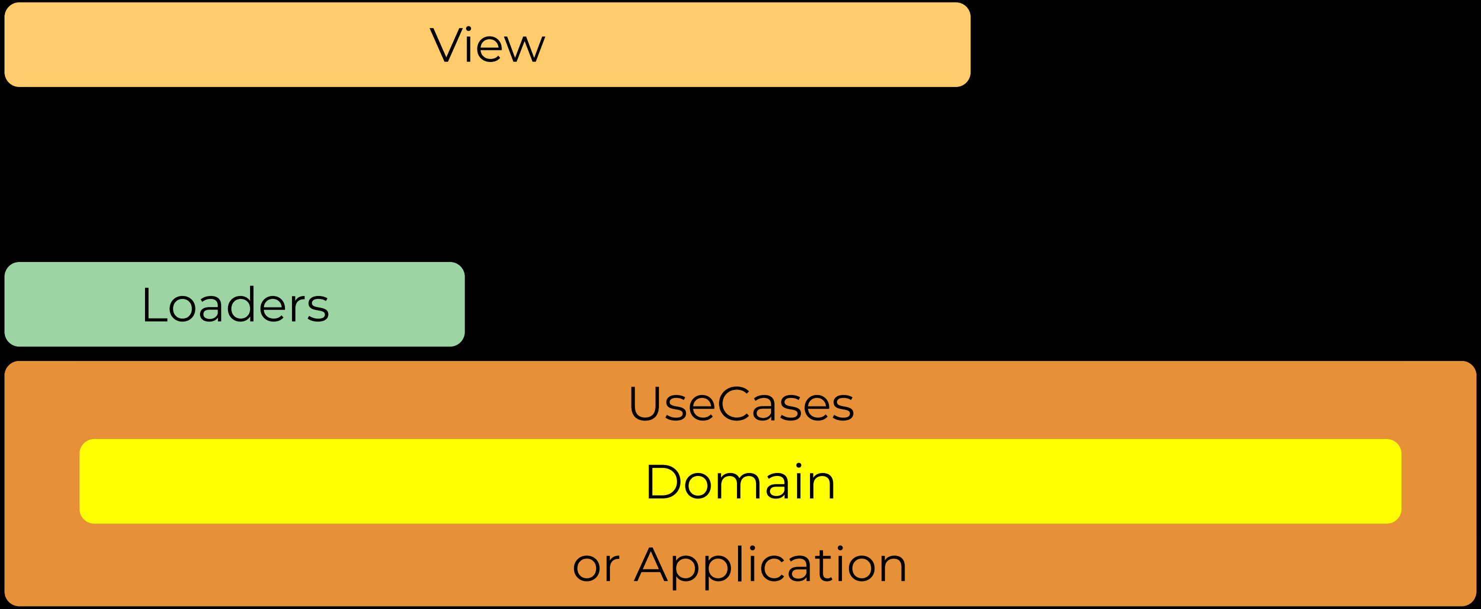
get_Grouped_Products_For_Listing_Page_From_All_Suppliers_OrError


Adapters
Ports
Application
Domain

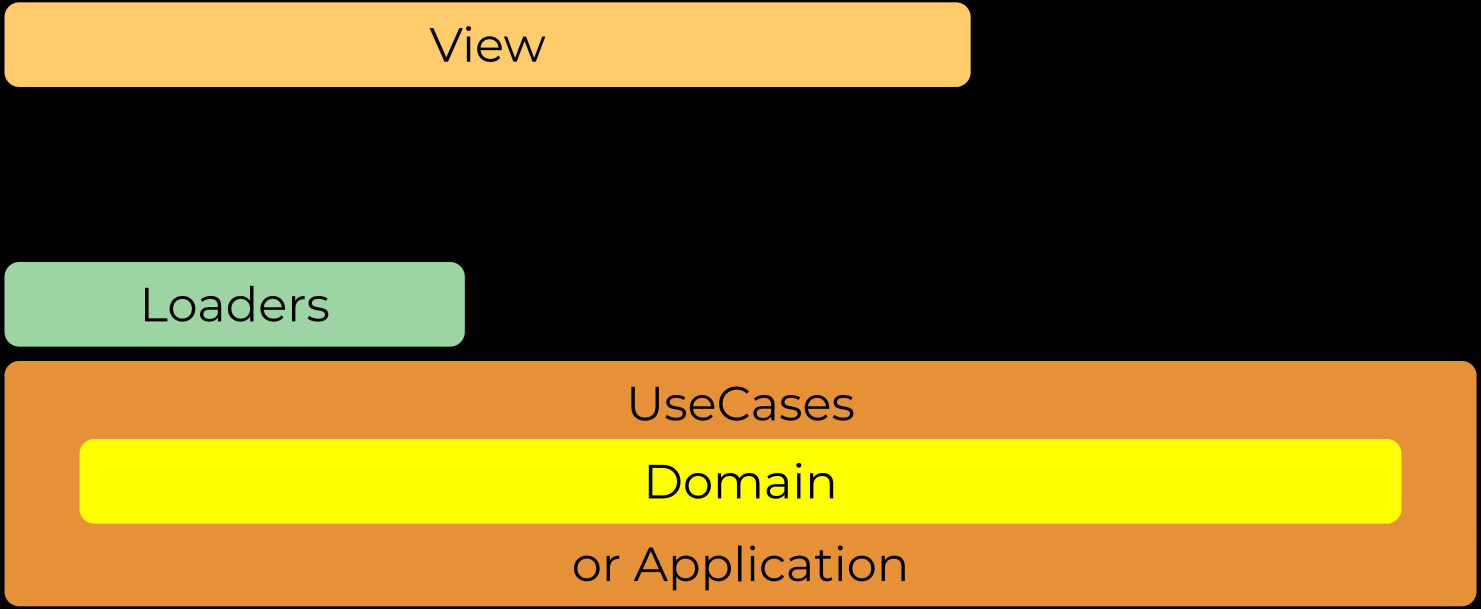
Loaders


View
Loaders

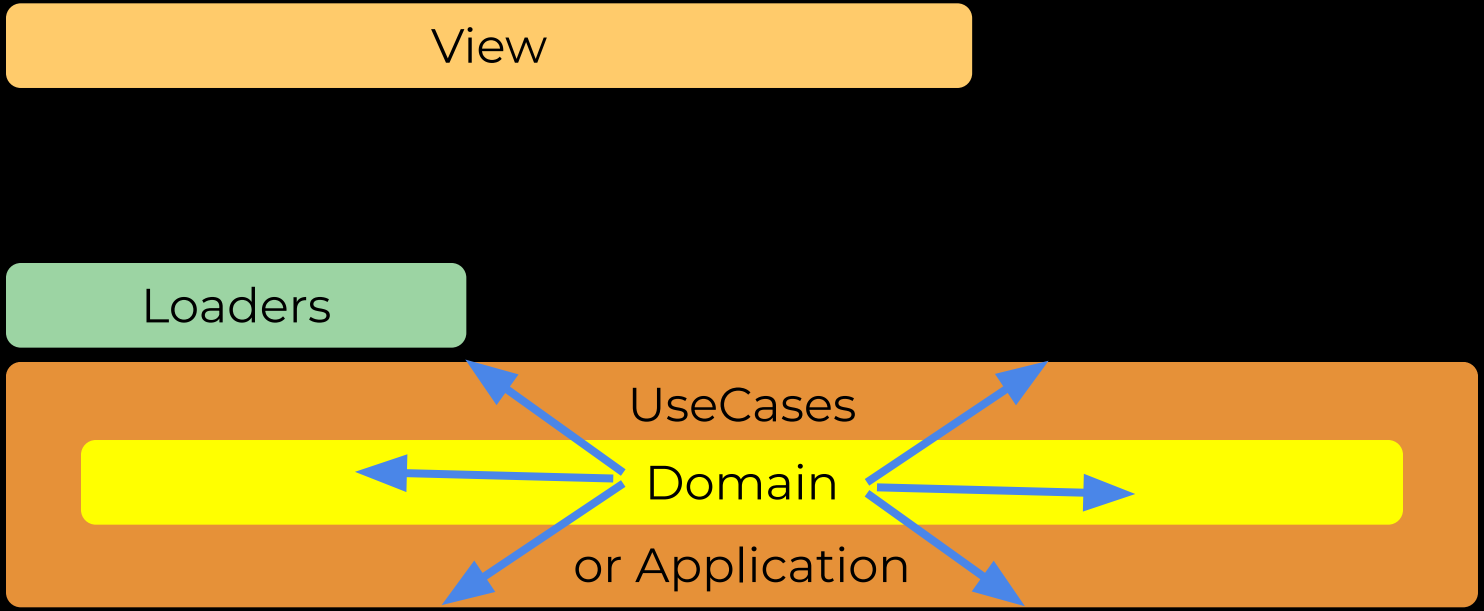

View
Loaders
UseCases
or Application


View
Loaders
UseCases
Domain
or Application

```
export default async function LongNamedUseCase ( … ) {
  const part1 = await get_Products_From_One_Supplier_OrError( … );
  const part2 = await get_Products_From_Other_Supplier_OrError( … );

  if (
    isValid_Products(part1)
    && isValid_Products(part2)
  ) {
    return transform_To_Groups(part1, part2)
  }

  throw new Error( … )
}
```

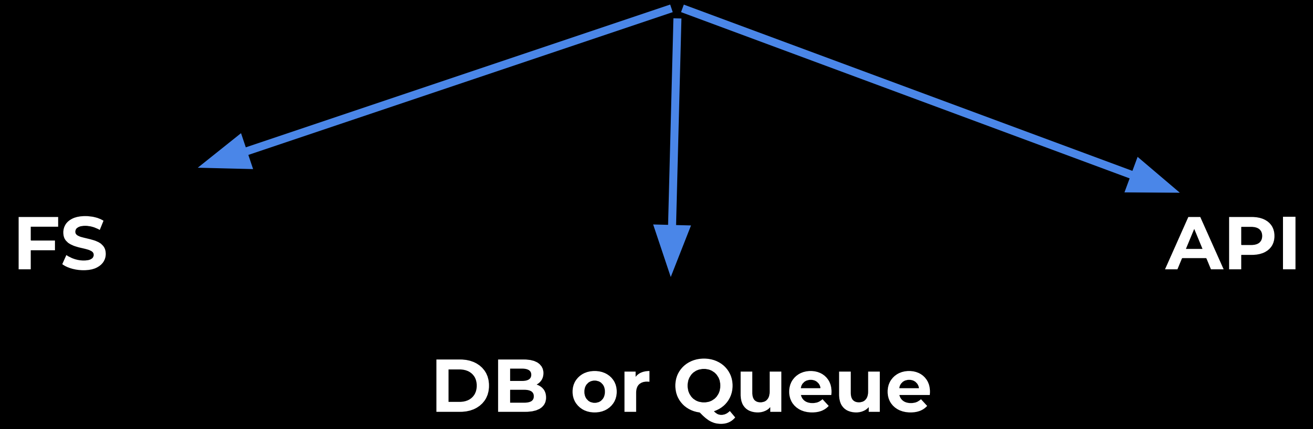

View
Loaders
UseCases
Domain
or Application

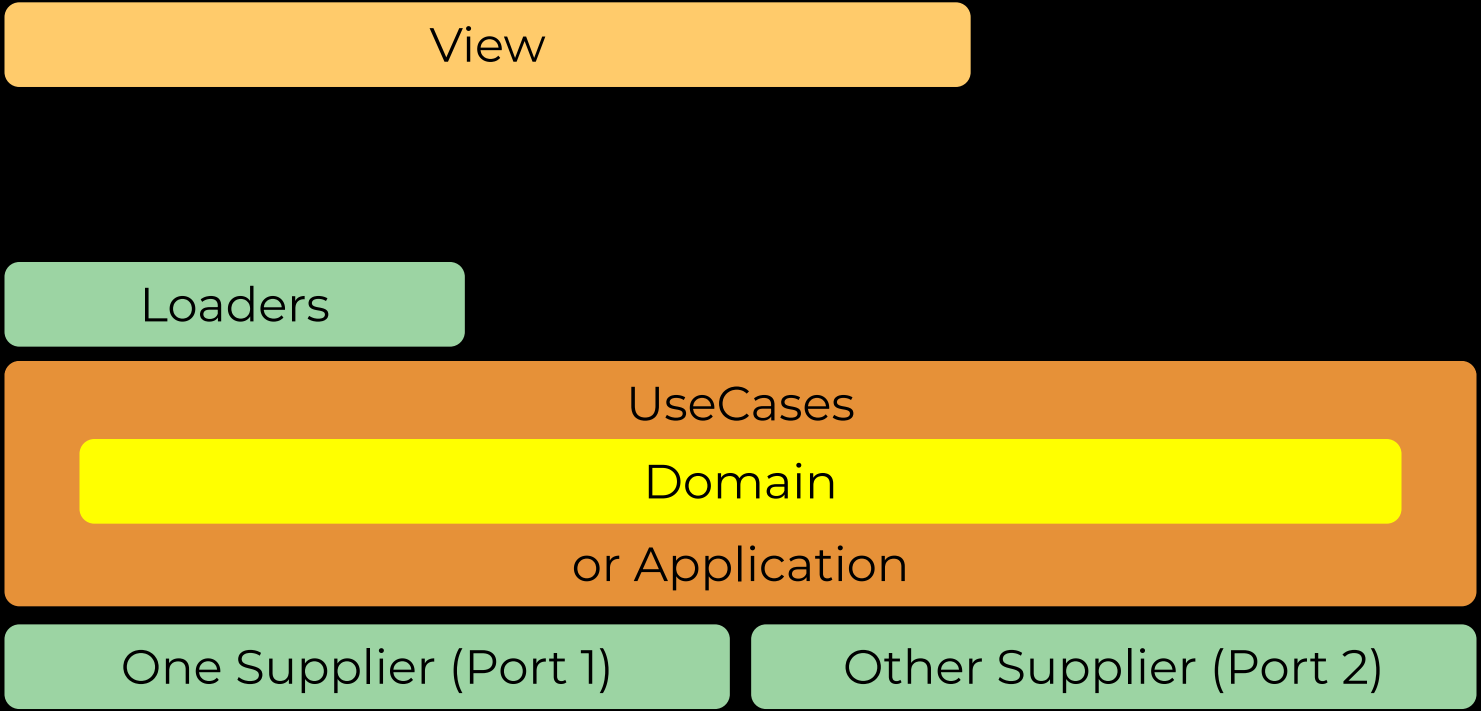
```
export type Product = {
    …
}

export function isValidProduct(product: Product): boolean = {
    …
}

export function mapProductToSome(product: Product): Some = {
    …
}
```


View
Loaders
UseCases
Domain
or Application

```
export default async function LongNamedUseCase ( ... ) {

    get_Products_From_One_Supplier_OrError( ... );

    get_Products_From_Other_Supplier_OrError( ... );
}
```

```
export default async function LongNamedUseCase ( … ) {

    get_Products_From_One_Supplier_OrError( … );

    get_Products_From_Other_Supplier_OrError( … );
}
```



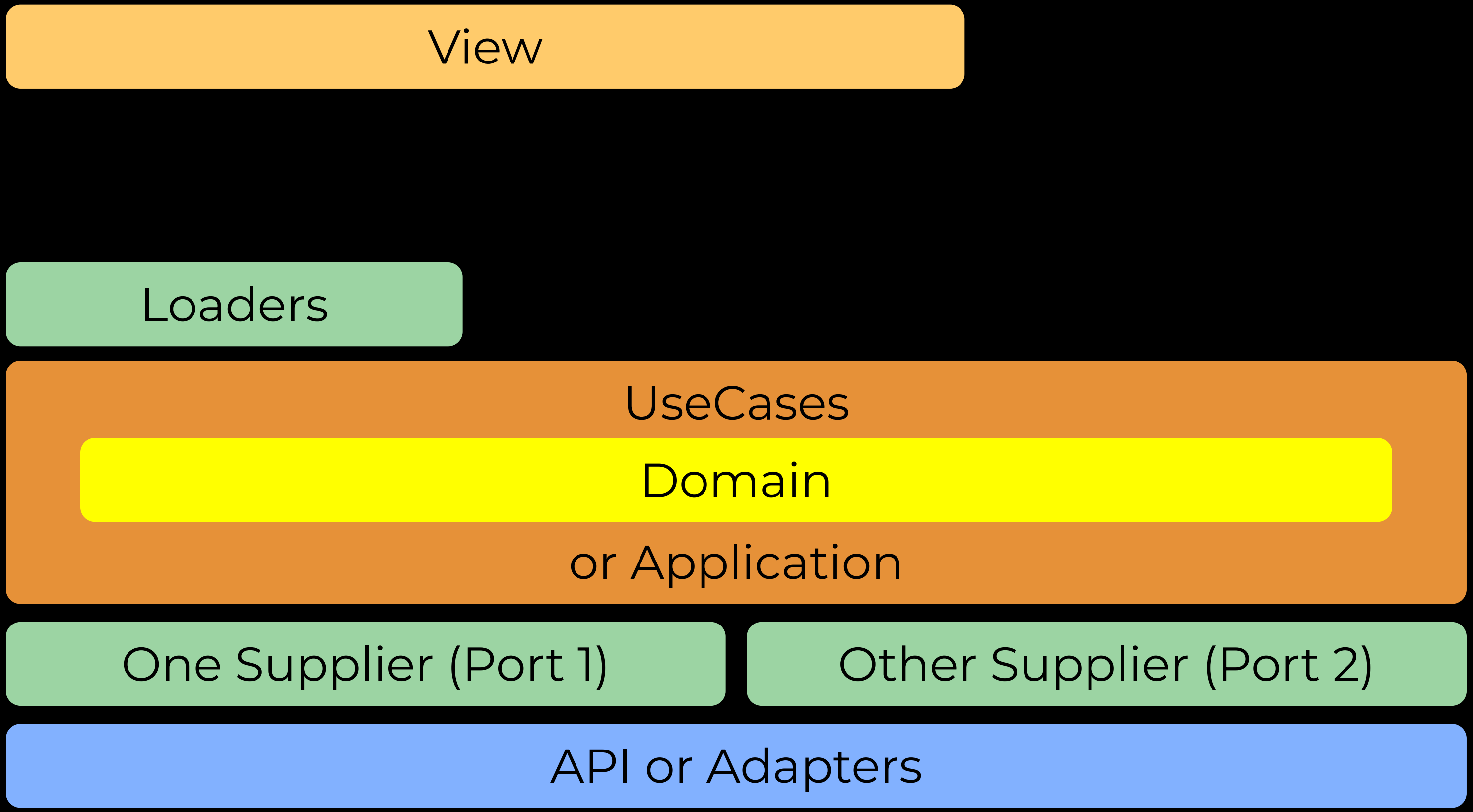

View
Loaders
UseCases
Domain
or Application
One Supplier (Port 1)
Other Supplier (Port 2)

Port

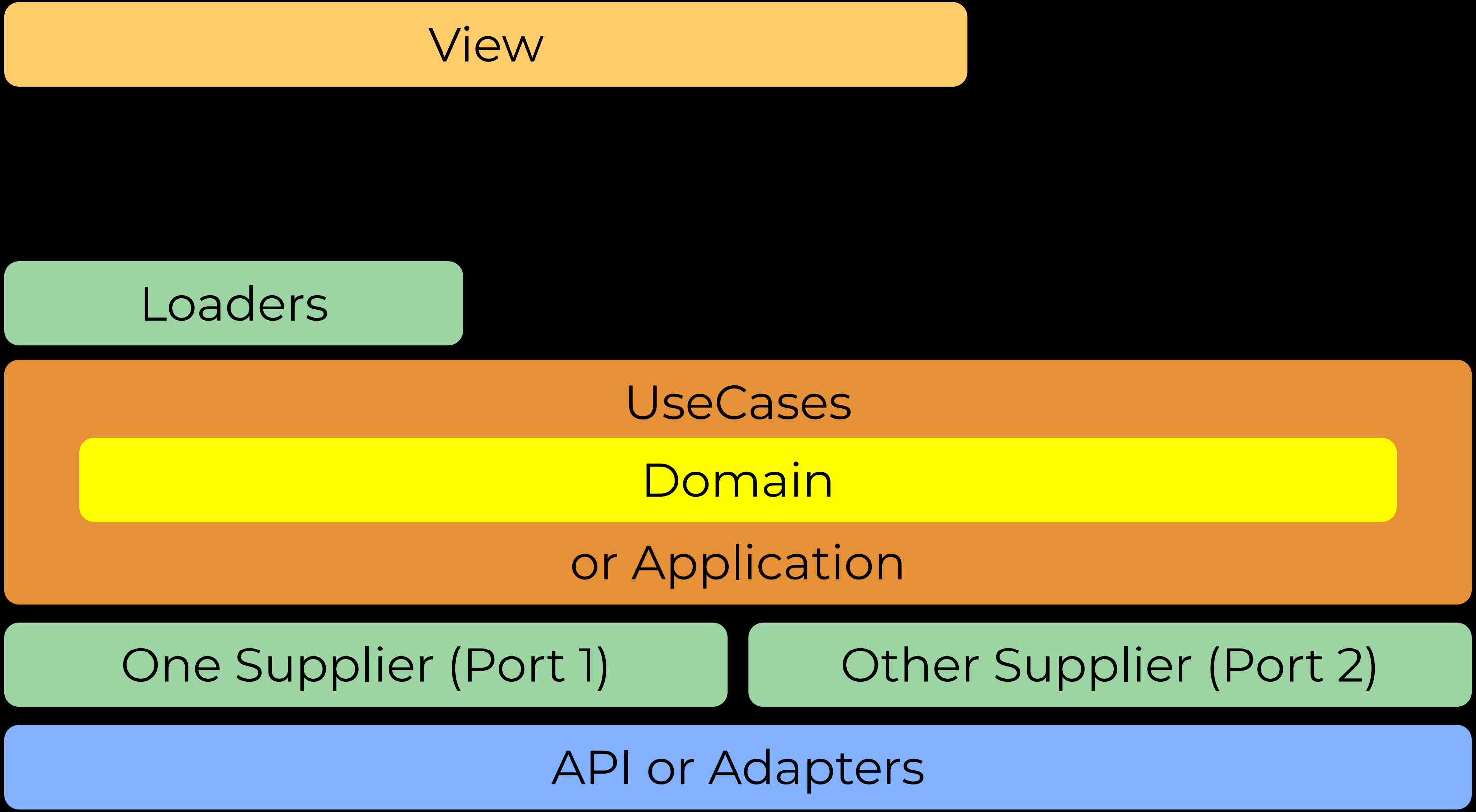
```
         example.com ➔ ExamplePort
  /api/products/red ➔ ExamplePort.getRedProducts()
/api/products/green ➔ ExamplePort.getGreenProducts()
```

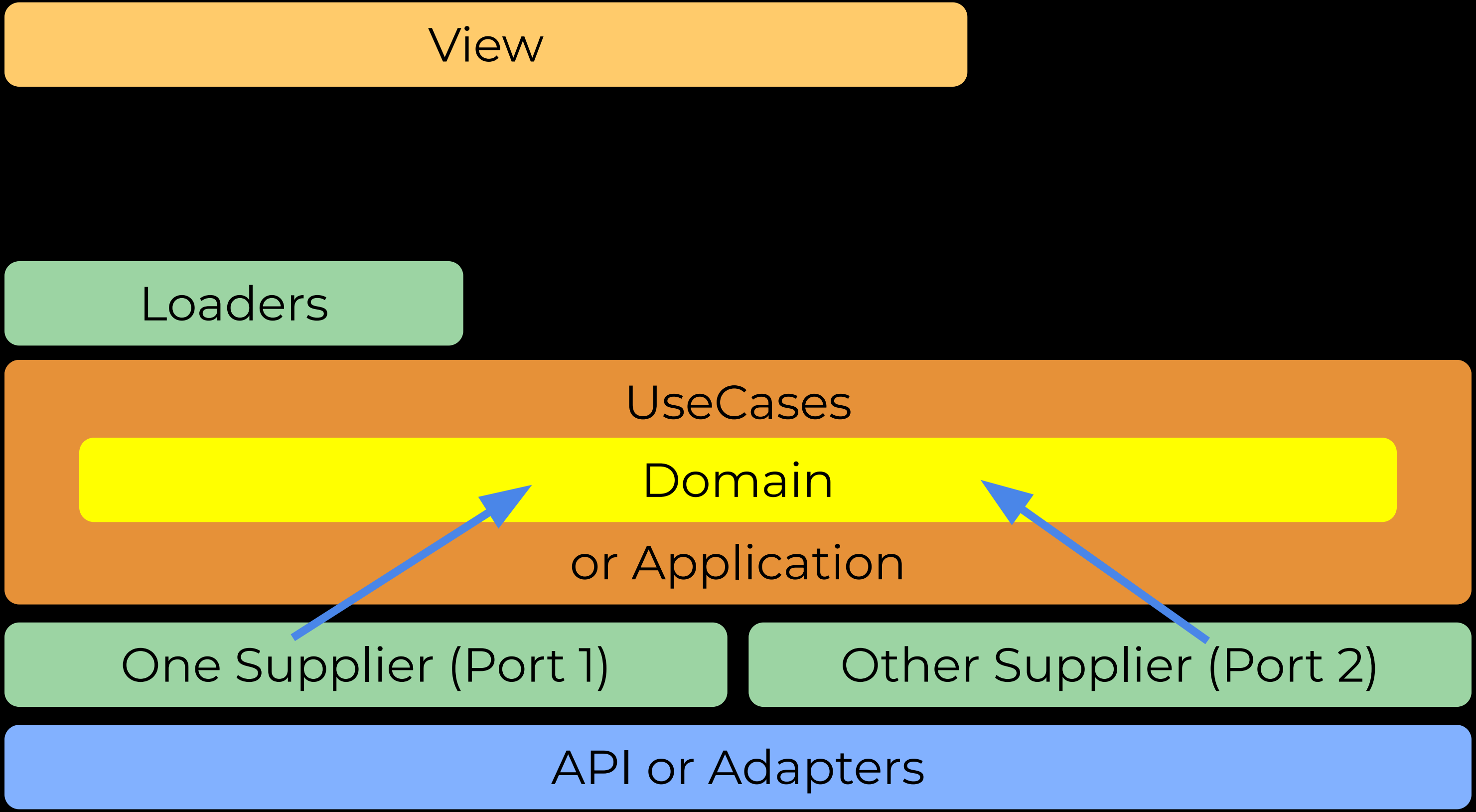
```
export default async function getProductsFromOneSupplier ( … ) {

    return oneSupplierPort.someMethod(
        await oneSupplierPort.someOtherMethod( … )
    );

}
```

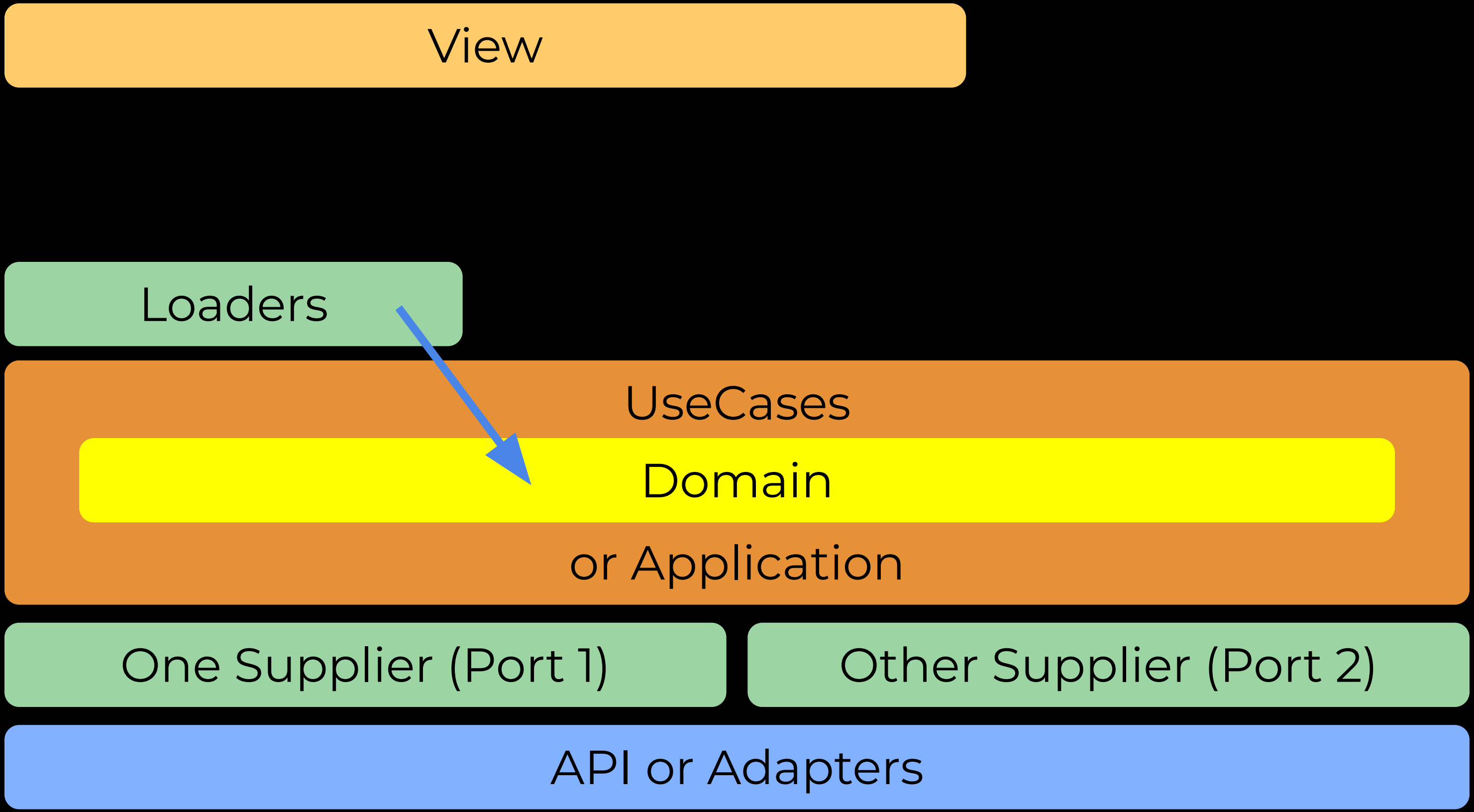
```
export default async function getProductsFrom_One_Supplier ( ... ) {
    const getData() { … }
    const postData() { … }

    return someMethod() {
        return await getData( ... )
    }
}


export default async function getProductsFrom_Other_Supplier ( ... ) {
    const getData() { … }
    const postData() { … }

    return someOtherMethod() {
        return await getData( ... )
    }
}
```

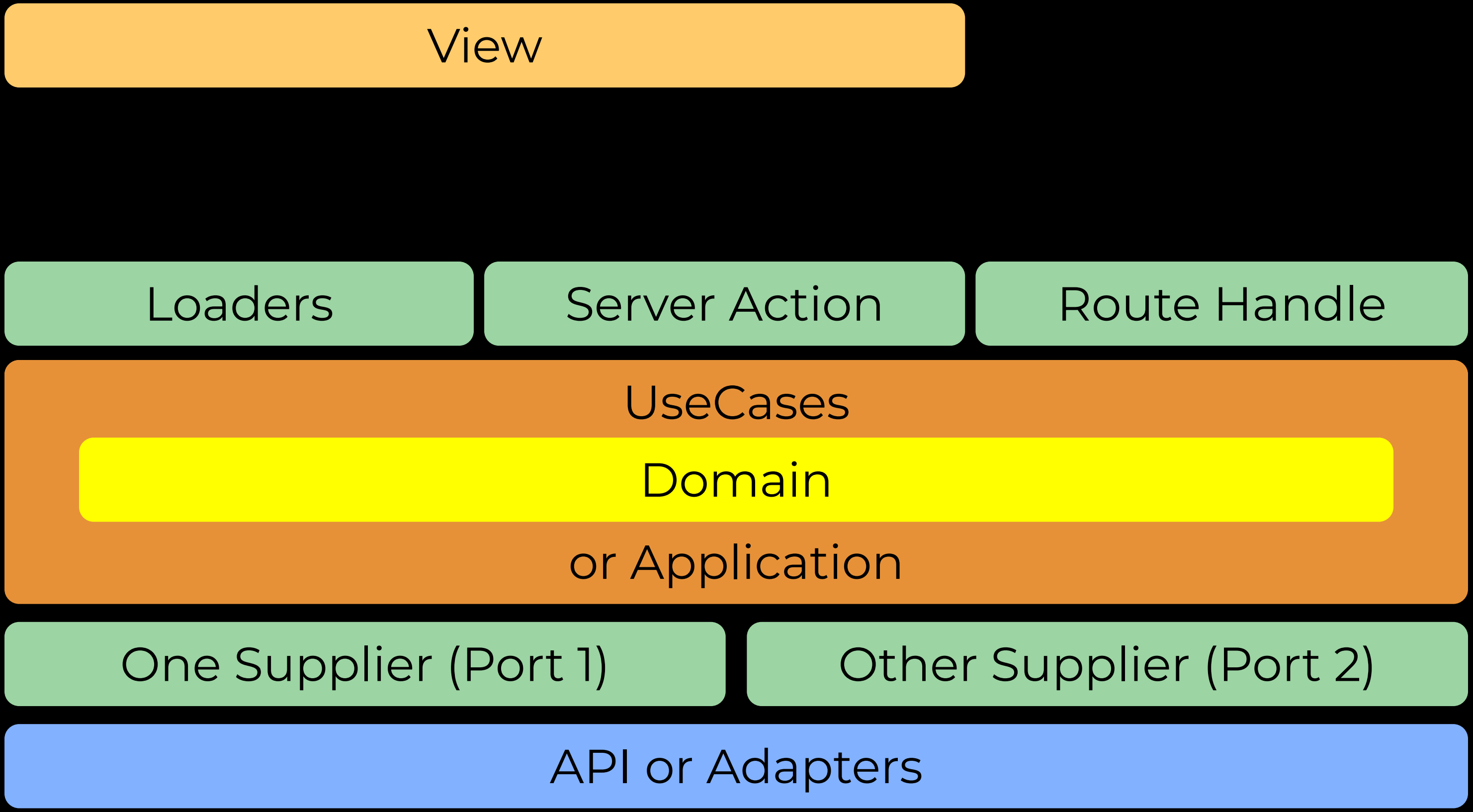

View
Loaders
UseCases
Domain
or Application
One Supplier (Port 1)
Other Supplier (Port 2)
API or Adapters

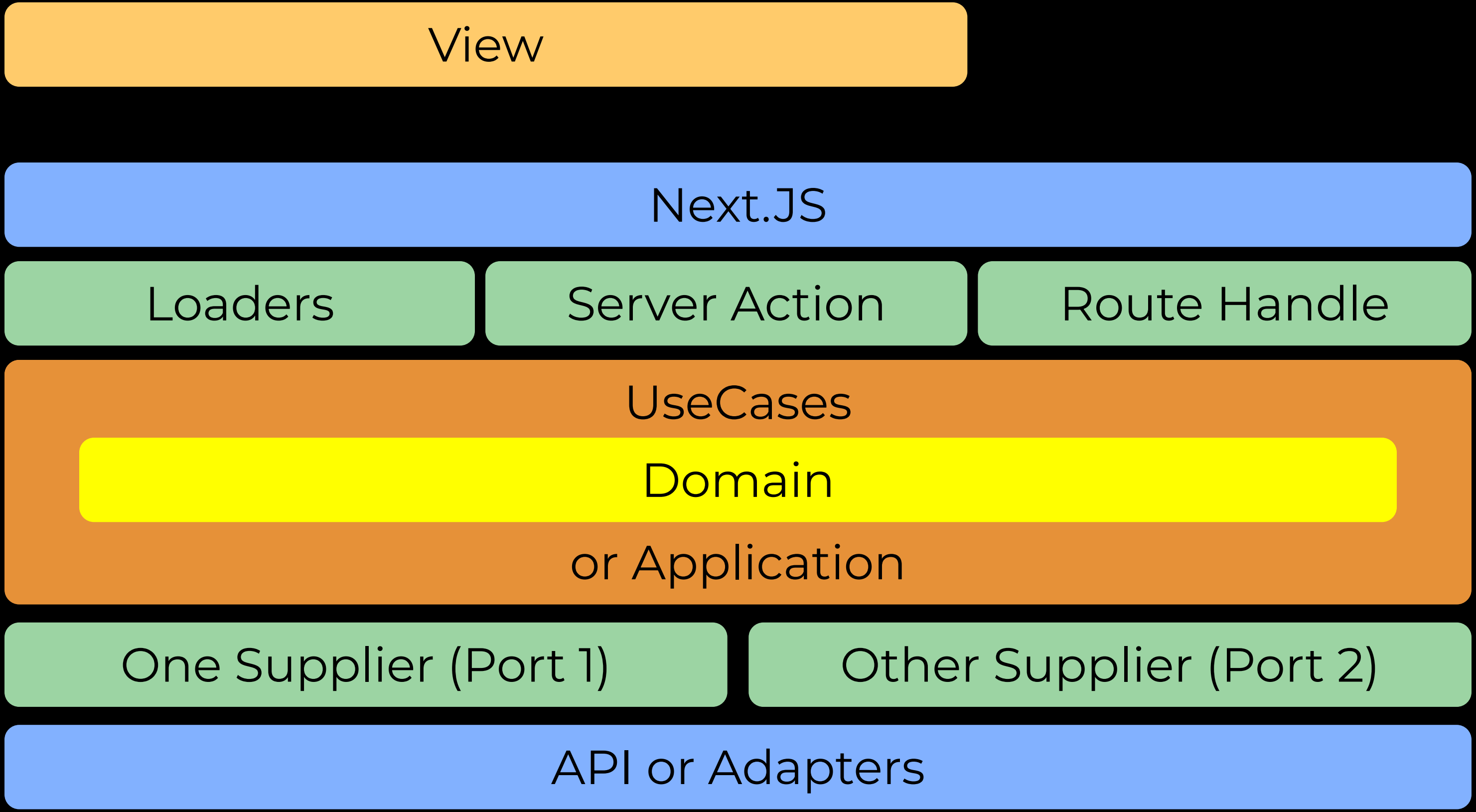
```
export default async function getProductsFrom_One_Supplier ( ... ) {
    const api: API;

    return someMethod() {
        return api.get(“/api/one/supplier”);
    }
}


export default async function getProductsFrom_Other_Supplier ( ... ) {
    const api: API;

    return someOtherMethod() {
        return api.get(“/api/other/supplier”);
    }
}
```

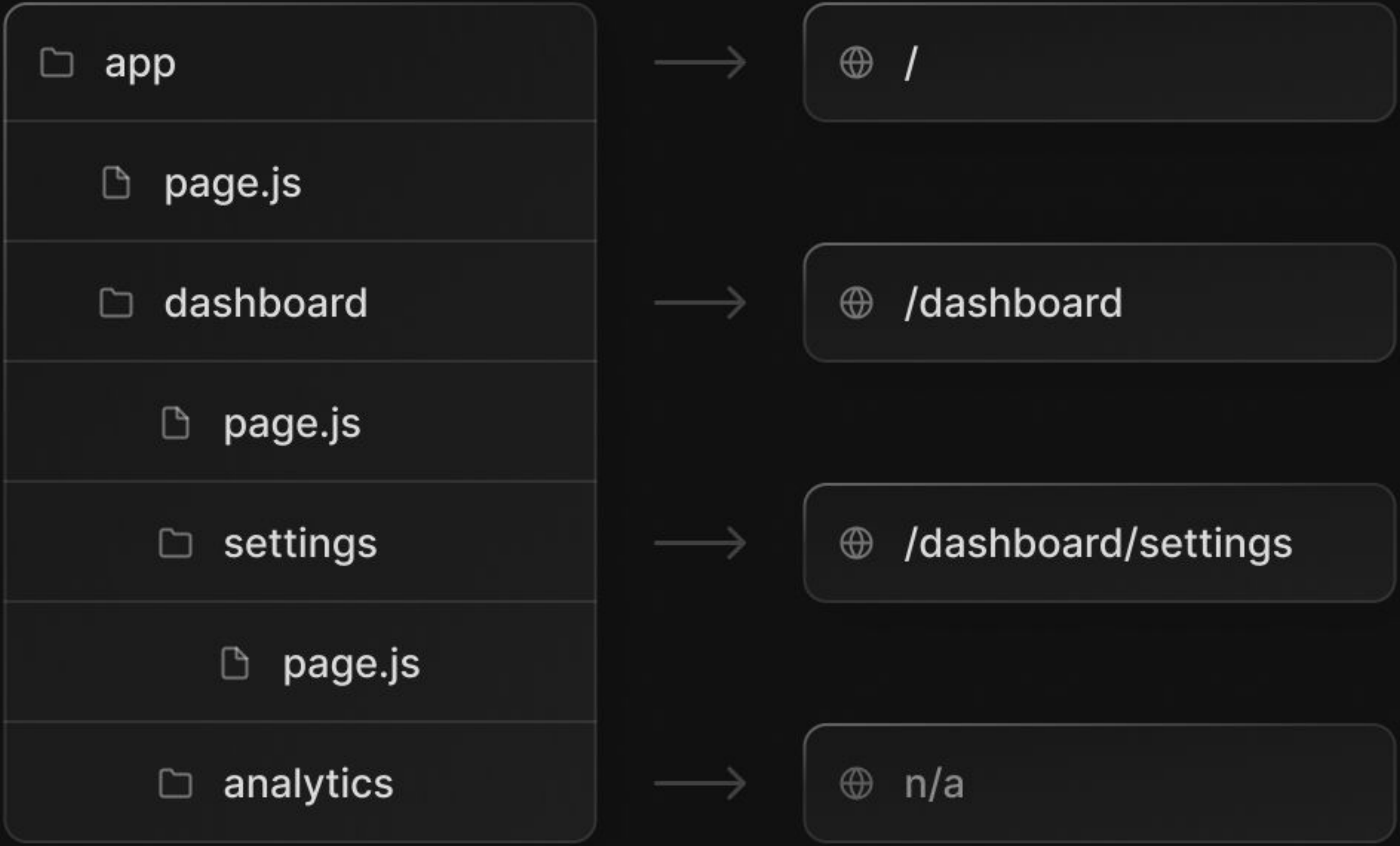

View
Loaders
UseCases
Domain
or Application
One Supplier (Port 1)
Other Supplier (Port 2)
API or Adapters

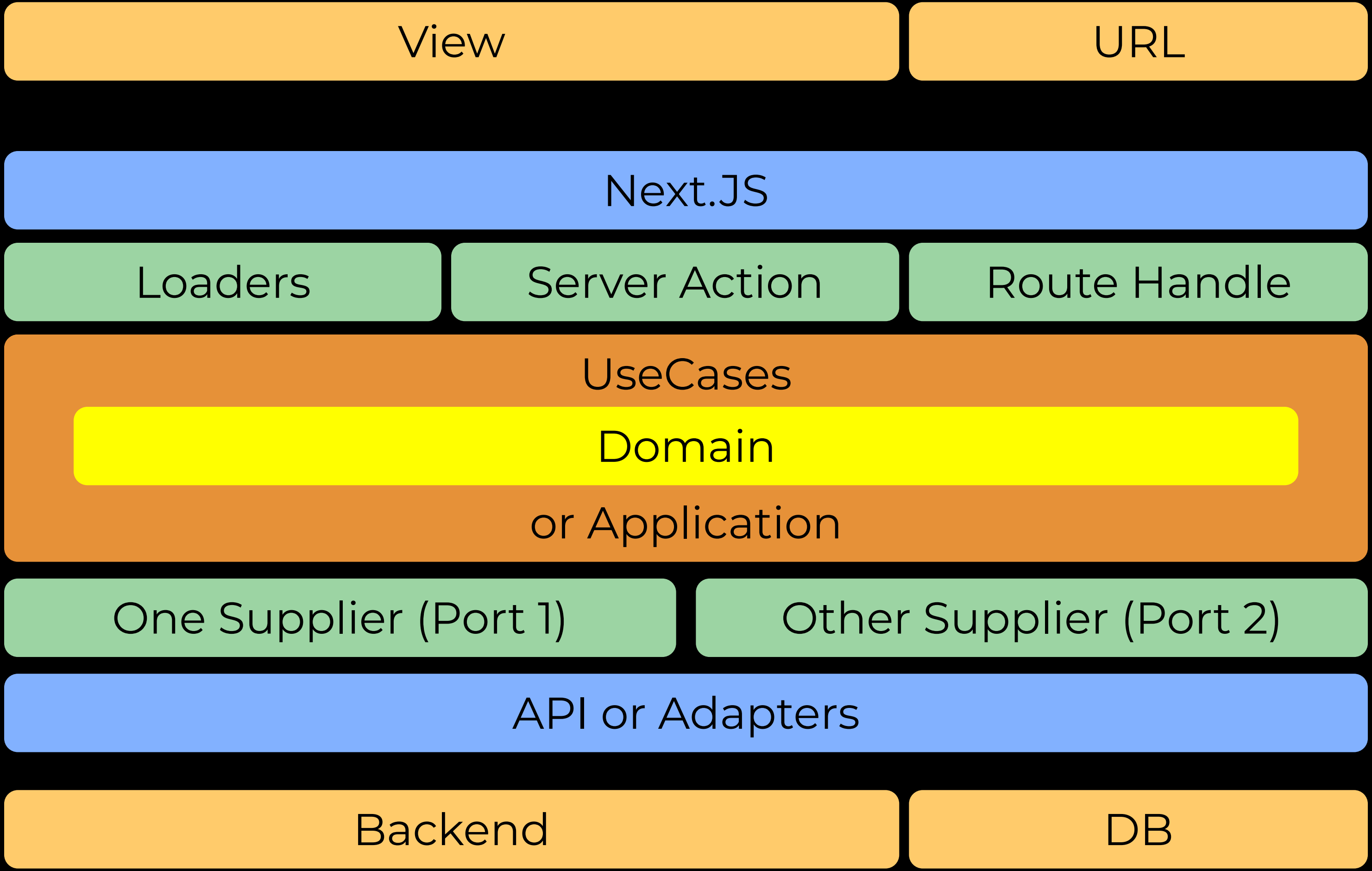

View
Loaders
UseCases
Domain
or Application
One Supplier (Port 1)
Other Supplier (Port 2)
API or Adapters


View
Loaders
UseCases
Domain
or Application
One Supplier (Port 1)
Other Supplier (Port 2)
API or Adapters


View
Loaders
Server Action
Route Handle
UseCases
Domain
or Application
One Supplier (Port 1)
Other Supplier (Port 2)
API or Adapters


View
Next.JS
Loaders
Server Action
Route Handle
UseCases
Domain
or Application
One Supplier (Port 1)
Other Supplier (Port 2)
API or Adapters


app
page.js
dashboard
page.js
settings
page.js
analytics
/
/dashboard
/dashboard/settings
n/a

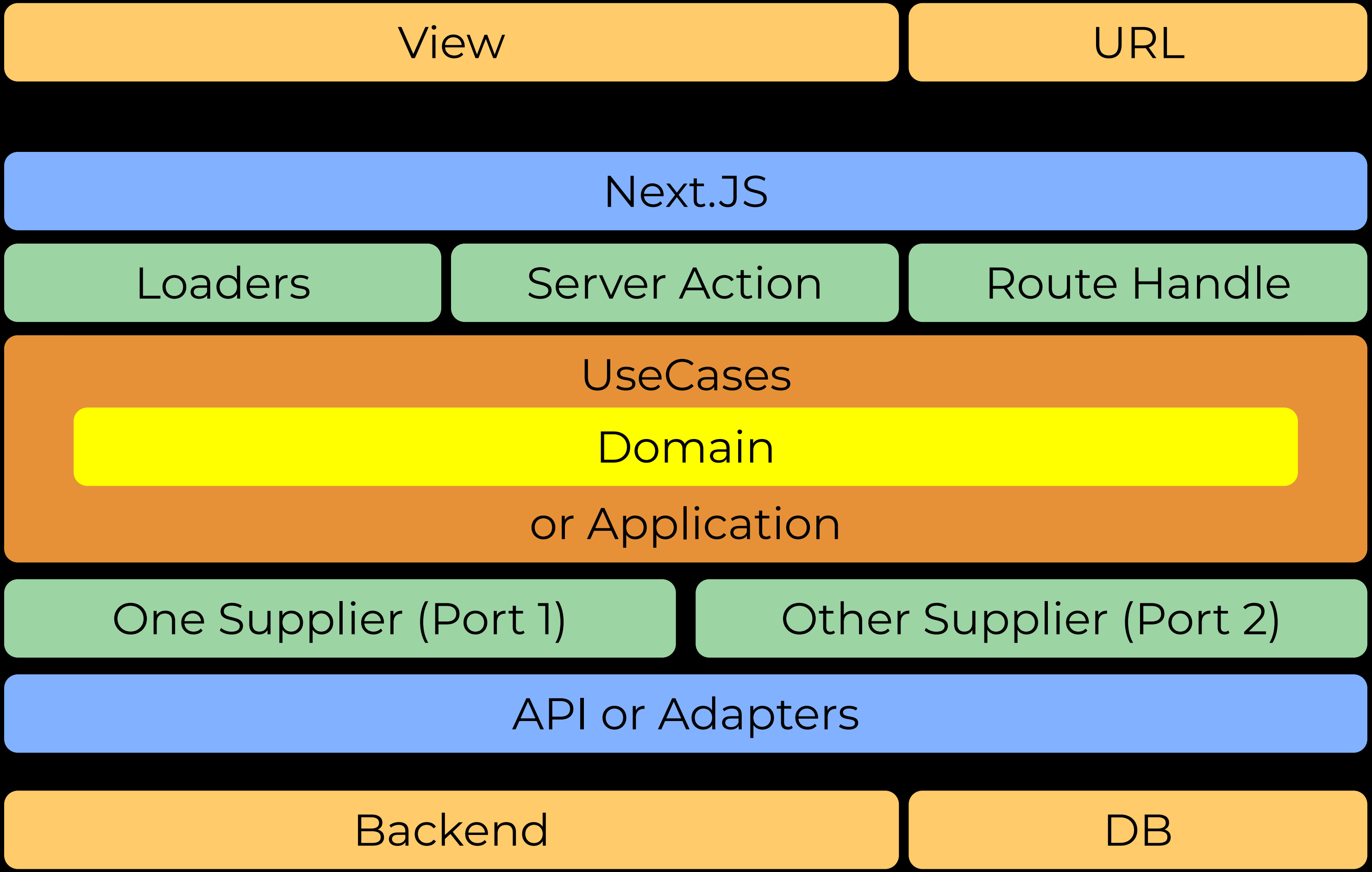

View
URL
Next.JS
Loaders
Server Action
Route Handle
UseCases
Domain
or Application
One Supplier (Port 1)
Other Supplier (Port 2)
API or Adapters
Backend
DB

Что еще имеется в виду?

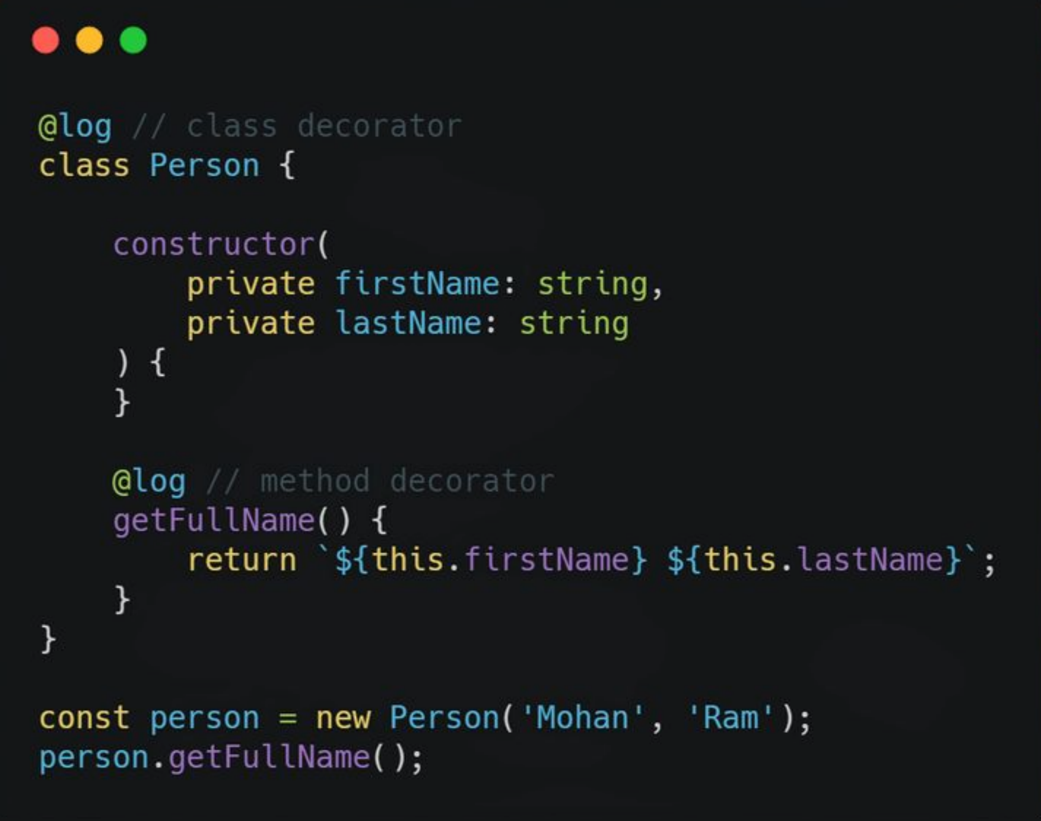
```
import { get_Products_From_One_Supplier_OrError } from “ … ”
import { get_Products_From_Other_Supplier_OrError } from “ … ”
export default async function LongNamedUseCase ( … ) {
  const part1 = await get_Products_From_One_Supplier_OrError( … );
  const part2 = await get_Products_From_Other_Supplier_OrError( … );

  if (
    isValid_Products(part1)
    && isValid_Products(part2)
  ) {
    return transform_To_Groups(part1, part2)
  }

  throw new Error( … )
}
```

```
export default async function LongNamedUseCase (
    get_Products_From_One_Supplier_OrError: OneInterface,
    get_Products_From_Other_Supplier_OrError: OtherInterface
) {
  const part1 = await get_Products_From_One_Supplier_OrError( ... );
  const part2 = await get_Products_From_Other_Supplier_OrError( ... );

  if (
    isValid_Products(part1)
    && isValid_Products(part2)
  ) {
    return transform_To_Groups(part1, part2)
  }

  throw new Error( ... )
}
```

- InversifyJS
- typedi
- tsyringe
- injectX


View
URL
Next.JS
Loaders
Server Action
Route Handle
UseCases
Domain
or Application
One Supplier (Port 1)
Other Supplier (Port 2)
API or Adapters
Backend
DB

- Validation
- Error handling
- Logging
- Tracing



```
@log // class decorator
class Person {

    constructor(
        private firstName: string,
        private lastName: string
    ) {
    }

    @log // method decorator
    getFullName() {
        return `${this.firstName} ${this.lastName}`;
    }
}

const person = new Person('Mohan', 'Ram');
person.getFullName();
```

```
const result = await Pipe(data)
    .to(x => log(x))
    .to(x => trackErrors(x))
    .to(x => validate(x))
    .exec()
```

SOL (Interface Segregation) D

```
export default async function ExamplePort ( … ) {
    const api: API;

    return {
        getProducts: () ⇒ api.get(“/api/products”),
        getCart: () ⇒ api.get(“/api/cart”),
        getFavorites: () ⇒ api.get(“/api/favorites”),
    }
}
```

```
export default async function ExampleUseCase ( ... ) {
    examplePort: ExamplePort;

    return () ⇒ {
        const cart = examplePort.getCart()
        const products = examplePort.getProducts()

        cart.put(products)
    }
}
```

```
interface ProductsGetter {
    getProducts(): Product[]
}

interface CartGetter {
    getCart(): Cart
}

export default async function ExampleUseCase ( … ) {
    examplePort: ProductsGetter & CartGetter;

    return () ⇒ {
        const cart = examplePort.getCart()
        const products = examplePort.getProducts()

        cart.put(products)
    }
}
```

Плюсы и минусы

- ➔ Писать тесты – одно удовольствие, когда везде четкий контракт и нет нестабильных импортов

```
export interface UseCase<TInput, TOutput = void> {
    execute(input?: TInput): Promise<TOutput>;
}
```

- ➔ Убрали нестабильные импорты. (Sentry, React-Intl, Axios и прочие). Все обернуто в контракт и следовательно легко заменяемо.

- ➔ Стоимость изменения системы не растет. Поскольку блоки взаимозаменяемы. Сложность проекта предсказуема, что дает уверенность бизнесу планировать новые фичи.

- ➔ Рефакторинг больше не больно, а еженедельная рутина, опять же, потому что все взаимозаменяемо по контракту.

- ➔ Чисто – значит много бойлерплейта

➔ Кривая обучения и онбординга

- ➔ Тюнинг контракта между слоями

Спасибо, я лайкнул

t.me/thxilikeit