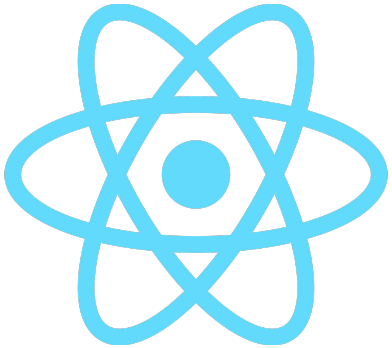


React

in practice

Prerequisites

- Basic knowledge of **HTML, CSS, JavaScript and TypeScript**
- **Node.js** (version ≥ 18)
- Integrated Development Environment (IDE) such as **Visual Studio Code**

https://www.typescriptlang.org/
https://nodejs.org/en
https://code.visualstudio.com/

Outline

- **Basic Concepts of Next.js**
  What is Next.js? - Optimizations - Rendering
- **Getting Started**
  Installation - Available scripts - Project Structure
- **Building an Application**
  Routing - Rendering - Data fetching - Styling
- **Dependencies**
- **Conclusions**

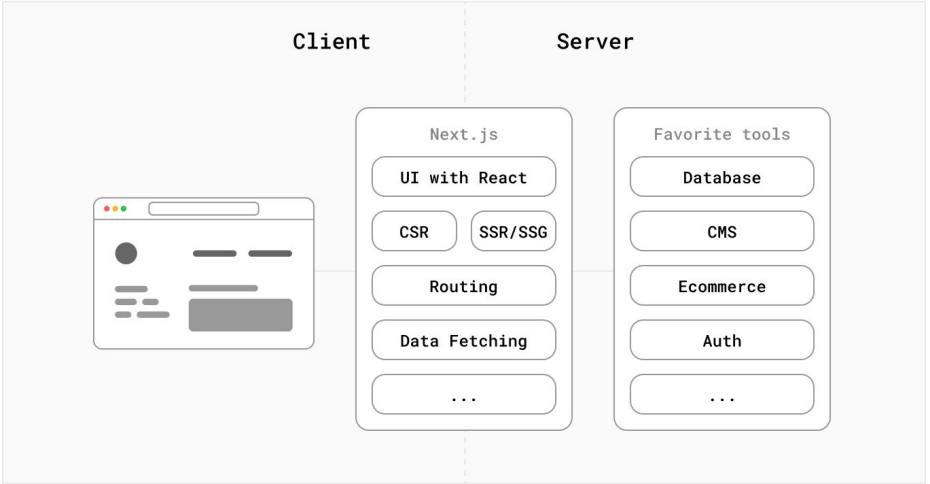
What is Next.js?

*Next.js is a React **framework** that gives you building blocks to create web applications.*
By framework, we mean Next.js handles the tooling and configuration needed for React, and provides additional structure, features, and optimizations for your application.

Main features of Next.js:

- routing (file-system based router)
- rendering (client-side and server-side rendering)
- data fetching
- styling
- optimizations

NEXT.JS



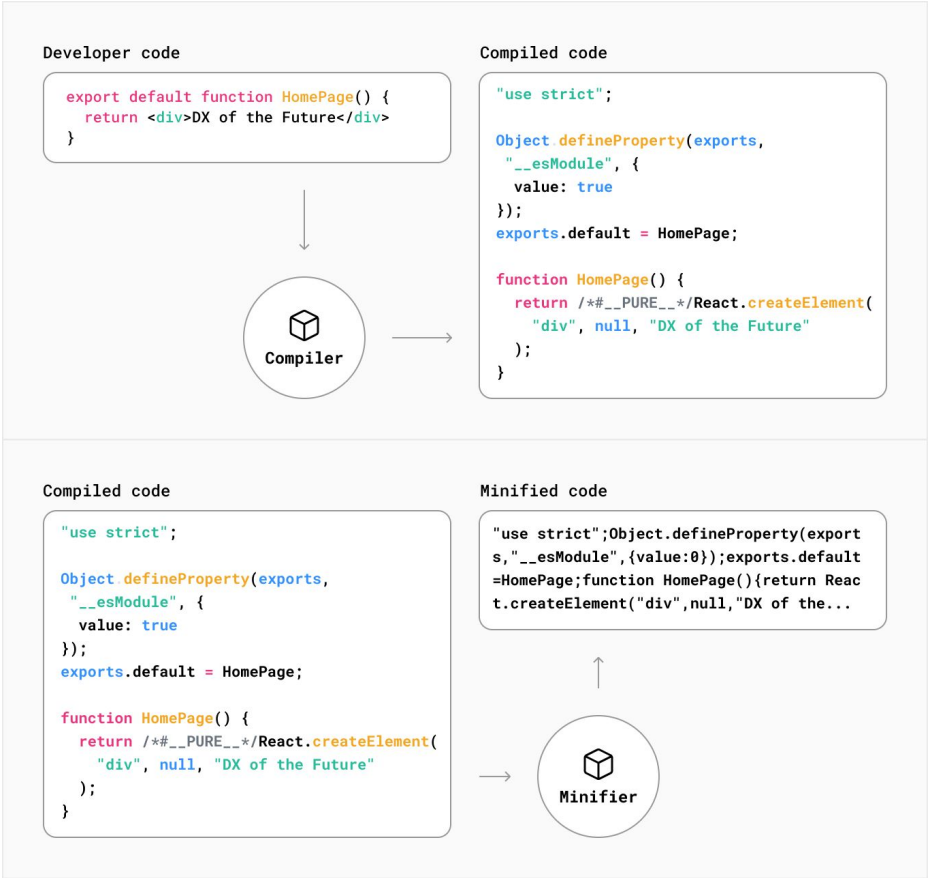
Optimizations (1/2)

Compiling

Developers write code in *developer-friendly* languages such as JSX, TypeScript, and modern versions of JavaScript.
While these languages improve the efficiency and confidence of developers, they need to be **compiled into JavaScript** before browsers can understand them.

Minifying

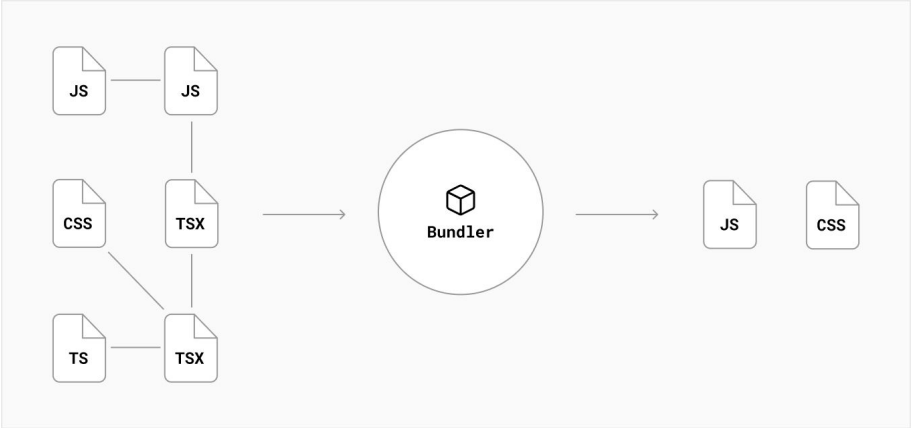
Minification is the process of **removing unnecessary code** formatting and comments without changing the code's functionality.
The goal is to improve the application's performance by decreasing file sizes.



Optimizations (2/2)

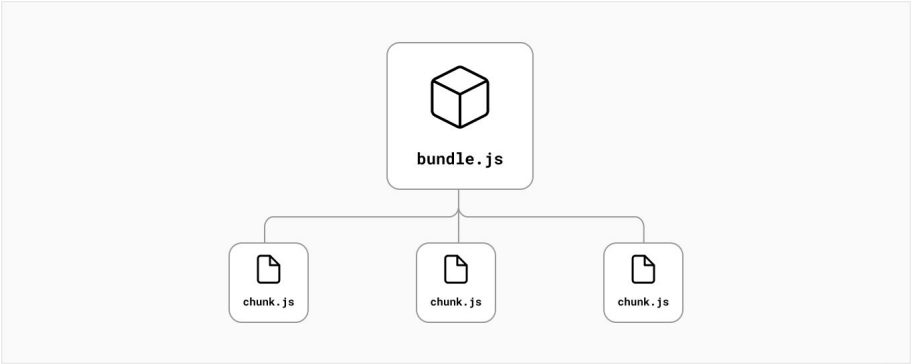
Bundling

Bundling is the process of resolving the web of dependencies and **merging (or 'packaging') the files** (or modules) into optimized bundles for the browser, with the goal of reducing the number of requests for files when a user visits a web page.



Code-splitting

Code-splitting is the process of **splitting the application's bundle into smaller chunks** required by each entry point. The goal is to improve the application's initial load time by only loading the code required to run that page.



Rendering

Client-Side Rendering

In a standard React application, the browser receives an **empty HTML** shell from the server along with the **JavaScript instructions to construct the UI**.

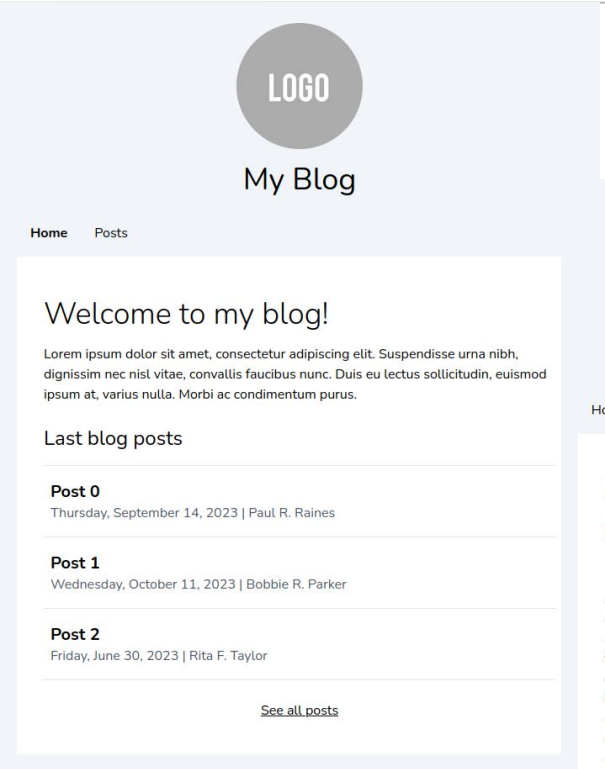
Pre-Rendering

Next.js **pre-renders** every page by default: this means that the HTML is generated in advance, on a server, instead of having it all done by JavaScript on the user's device.

- **Server-Side Rendering:** the HTML of the page is generated on a server for **each** request.
- **Static Site Generation:** the HTML is generated on the server, but there is no server at runtime. Instead, content is generated once, at **build time**.

Today's Goals

- Understand base features of Next.js
- Learn the difference between Client and Server Components
- Discover best practices for Next.js applications
- Implement a very simple blog web application



https://github.com/franksacco/react-in-practice

Installation

The recommend way for starting a new Next.js app is using create-next-app, which sets up everything automatically. To create a project, run:

```
npx create-next-app@latest
```

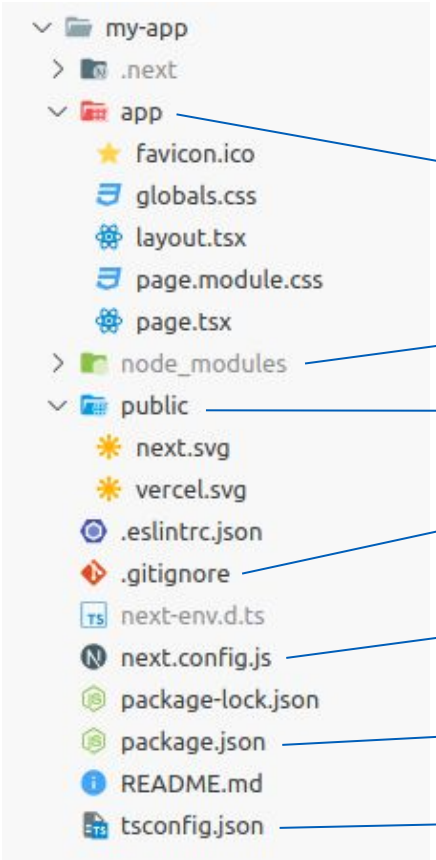
On installation, you'll see the following prompts:

```
What is your project named? my-app
Would you like to use TypeScript? No / Yes
Would you like to use ESLint? No / Yes
Would you like to use Tailwind CSS? No / Yes
Would you like to use `src/` directory? No / Yes
Would you like to use App Router? (recommended) No / Yes
Would you like to customize the default import alias (@/*)? No / Yes
What import alias would you like configured? @/*
```

Available Scripts

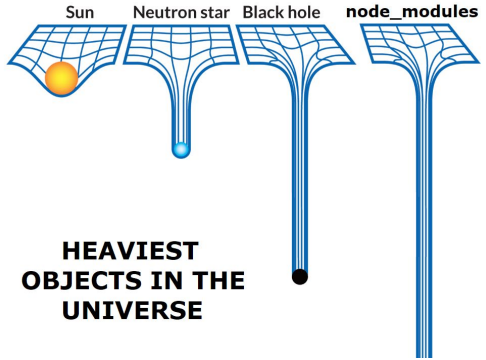
- `npm run dev`
  Start the application in **development mode** with hot-code reloading, error reporting, ...
  The application will start at `http://localhost:3000` by default. The default port can be changed with `-p`
  You can also set the hostname to be different from the default of `0.0.0.0`, this can be useful for making the application available for other devices on the network. The default hostname can be changed with `-H`
- `npm run build`
  Create an **optimized production build** of your application.
- `npm start`
  Start the application in **production mode**.
- `npm lint`
  Run ESLint for all files in the `pages/`, `app/`, `components/`, `lib/`, and `src/` directories.

Project Structure



App Router

The folder where npm (or Yarn) installs all the **dependencies**



Static assets (images, fonts, ...) to be served

Git files and folders to ignore

Configuration file for Next.js

Project dependencies and scripts

Configuration file for TypeScript

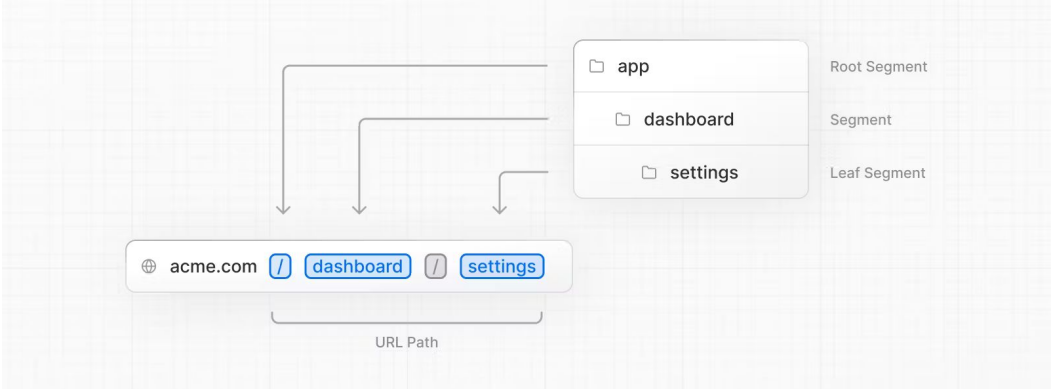
Defining Routes

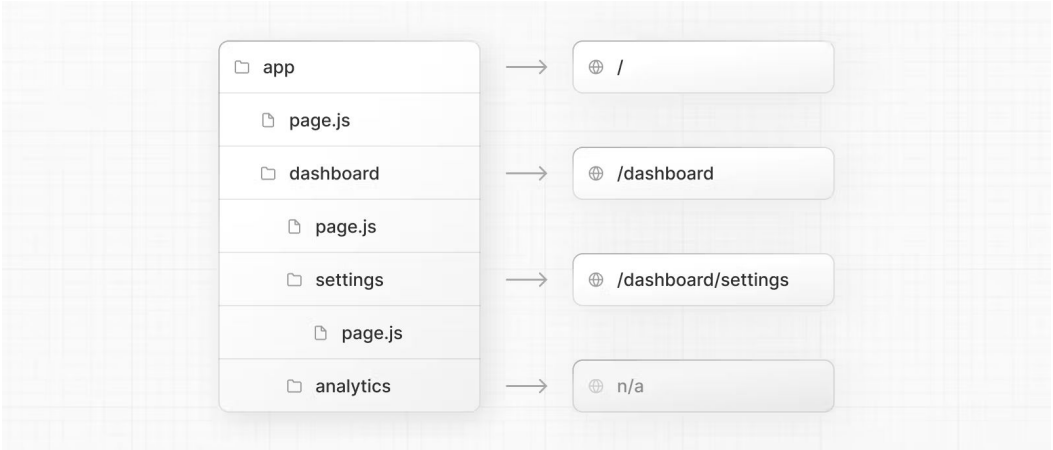
Next.js uses a **file-system based router** where **folders** are used to define routes.

Each folder represents a **route** segment that maps to a **URL** segment.

To create a nested route, you can nest folders inside each other.

A special `page.js` file is used to make route segments publicly accessible.

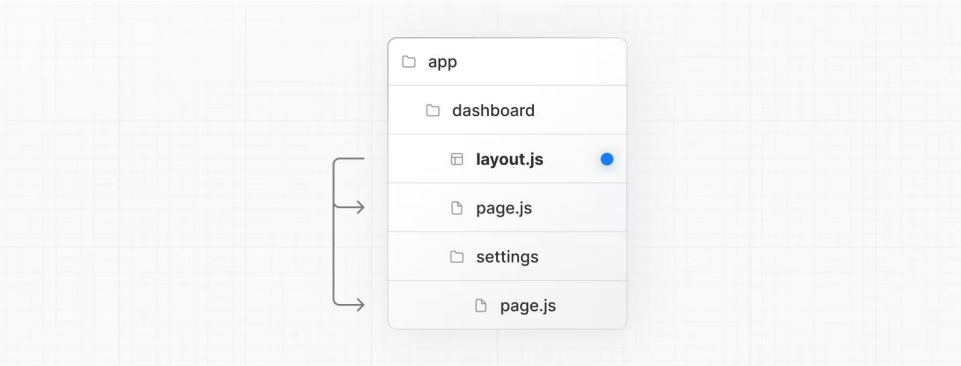




Pages and Layouts

A page is UI that is **unique** to a route.
You can define pages by exporting a component from a `page.js` file.
Use nested folders to define a route and a `page.js` file to make the route publicly accessible.

A layout is UI that is **shared** between multiple pages. On navigation, layouts preserve state, remain interactive, and do not re-render.
Layouts can also be nested.
The **root layout**, that is defined at the top level of the `app` directory and applies to all routes, is **required**.



Linking and Navigating

<Link> is a built-in component that extends the HTML <a> tag to provide prefetching and client-side navigation between routes. It is the primary way to navigate between routes in Next.js.

```
import Link from 'next/link'
export default function Page() {
  return <Link href="/dashboard">Dashboard</Link>
}
```

You can use usePathname() in client components to determine if a link is active.

```
'use client'
import { usePathname } from 'next/navigation'
export default function Nav() {
  const pathname = usePathname()
  return (...)
}
```

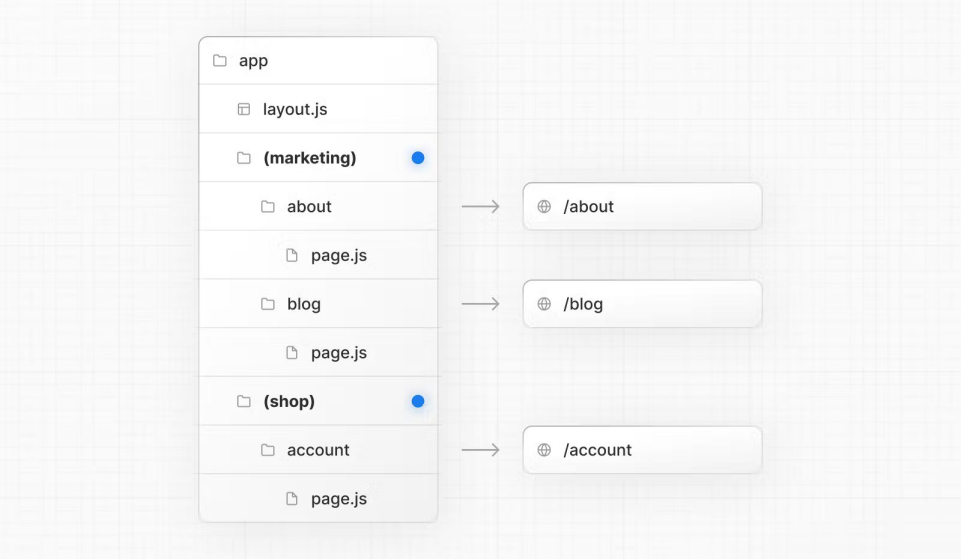
Route Groups

In the app directory, nested folders are normally mapped to URL paths. However, you can mark a folder as a **Route Group** to prevent the folder from being included in the route's URL path.

This allows you to organize your route segments and project files into logical groups without affecting the URL path structure.

Route groups are useful for:

- **organizing routes** into groups e.g. by site section, intent, or team;
- enabling **nested layouts** in the same route segment level.



Dynamic Routes

When you want to create routes from **dynamic data**, you can use Dynamic Segments that are filled in at **request time** or **pre-rendered at build time**.

A Dynamic Segment can be created by wrapping a folder's name in square brackets: **`[folderName]`**. For example, `[id]` or `[slug]`.
Dynamic Segments are passed as the **`params`** prop to `layout`, `page`, `route`, and `generateMetadata` functions.

The **`generateStaticParams`** function can be used in combination with dynamic route segments to **statically generate** routes at build time instead of on-demand at request time.

`[folder]`	Dynamic route segment
`[...folder]`	Catch-all route segment
`[[...folder]]`	Optional catch-all route segment

Routing File Conventions

layout	`.js .jsx .tsx`	A **root layout** is the top-most layout in the root `app` directory. It is used to define the `<html>` and `<body>` tags and other globally shared UI.
page	`.js .jsx .tsx`	A **page** is UI that is unique to a route.
loading	`.js .jsx .tsx`	A **loading** file can create instant loading states built on Suspense.
not-found	`.js .jsx .tsx`	The **not-found** file is used to render UI when the `notFound` function is thrown within a route segment.
error	`.js .jsx .tsx`	An **error** file defines an error UI boundary for a route segment.
global-error	`.js .jsx .tsx`	To specifically handle errors in root `layout.js`, use a variation of `error.js` called `app/global-error.js` located in the root `app` directory.
route	`.js .ts`	Route Handlers allow you to create custom request handlers for a given route using the Web `Request` and `Response` APIs.
template	`.js .jsx .tsx`	A **template** file is similar to a layout in that it wraps each child layout or page. Unlike layouts that persist across routes and maintain state, templates create a new instance for each of their children on navigation.
default	`.js .jsx .tsx`	Parallel route fallback page.

Rendering

- **Server Components** allow you to write UI that can be rendered and optionally cached on the server. There are three different server rendering strategies: **static rendering**, **dynamic rendering** and **streaming**.
- **Client Components** allow you to write interactive UI that can be rendered on the client at request time.

By default, Next.js uses Server Components.
This allows you to automatically implement server rendering with no additional configuration, meaning **you have to explicitly decide what components React should render on the client**.

Benefits of Server Components:

- Data Fetching
- Security
- Caching
- Bundle Sizes
- Initial Page Load and First Contentful Paint (FCP)
- Streaming
- Search Engine Optimization and Social Network Shareability

Benefits of Client Components:

- Interactivity
- Browser APIs

When to use Server and Client Components?

What do you need to do?	Server Component	Client Component
Fetch data	✅	❌
Access backend resources (directly)	✅	❌
Keep sensitive information on the server (access tokens, API keys, etc)	✅	❌
Keep large dependencies on the server / Reduce client-side JavaScript	✅	❌
Add interactivity and event listeners (`onClick()`, `onChange()`, etc)	❌	✅
Use State and Lifecycle Effects (`useState()`, `useReducer()`, `useEffect()`, etc)	❌	✅
Use browser-only APIs	❌	✅
Use custom hooks that depend on state, effects, or browser-only APIs	❌	✅
Use React Class components	❌	✅

https://nextjs.org/docs/app/building-your-application/rendering/composition-patterns

Data Fetching

1. **On the server, with `fetch`:** Next.js extends the native **`fetch` Web API** to allow you to configure the caching and revalidating behavior for each fetch request on the server.
   React extends `fetch` to automatically memoize fetch requests while rendering a React component tree.

2. **On the server, with third-party libraries** that doesn't support or expose `fetch` (for example, a database, CMS, or ORM client), you can configure the caching and revalidating behavior of those requests using the Route Segment Config Option and React's `cache` function.

3. **On the client, via a Route Handler:** if you need to fetch data in a client component, you can call a Route Handler from the client.

4. **On the client, with third-party libraries:** you can also fetch data on the client using a third-party library such as SWR or React Query. These libraries provide their own APIs for memoizing requests, caching, revalidating, and mutating data.

https://nextjs.org/docs/app/building-your-application/data-fetching/fetching-caching-and-revalidating

Styling

Next.js supports different ways of styling your application, including:

- **Global CSS**: Simple to use and familiar for those experienced with traditional CSS, but can lead to larger CSS bundles and difficulty managing styles as the application grows.
- **CSS Modules**: Create locally scoped CSS classes to avoid naming conflicts and improve maintainability.
- **Tailwind CSS**: A utility-first CSS framework that allows for rapid custom designs by composing utility classes.
- **Sass**: A popular CSS preprocessor that extends CSS with features like variables, nested rules, and mixins.
- **CSS-in-JS**: Embed CSS directly in your JavaScript components, enabling dynamic and scoped styling.

https://nextjs.org/docs/app/building-your-application/styling

Some Useful Functions

- **`notFound()`**
  Invoking the `notFound()` function throws a `NEXT_NOT_FOUND` error and terminates rendering of the route segment in which it was thrown.
  The **Not Found UI** defined for the route segment is rendered instead.
- **`redirect(path, type)`**
  The `redirect` function allows you to **redirect the user to another URL**. `redirect` can be used in Server Components, Client Components, Route Handlers, and Server Actions.
- **`usePathname()`**
  `usePathname` is a **Client Component** hook that lets you read the current URL's **pathname**.

https://nextjs.org/docs/app/api-reference/functions

Install a dependency

You can install a package using:

```
npm install <package>
# Alternatively you may use yarn:
yarn add <package>
```

Your package manager will update the lock and `package.json` files accordingly.

The npm registry is a **public collection of packages** of open-source code for Node.js, front-end web apps and the JavaScript community at large.

https://docs.npmjs.com/cli/v8/commands/npm-install

https://www.npmjs.com/

Material UI

Material UI is a library of React UI components that implements Google's **Material Design**.

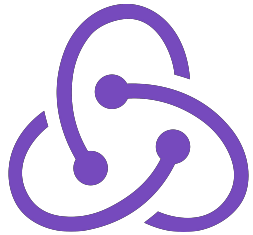
It includes a comprehensive **collection of prebuilt components** that are ready for use in production right out of the box.

```
npm install @mui/material @emotion/react @emotion/styled
# Robot Font
npm install @fontsource/roboto
# Material Icons
npm install @mui/icons-material
```

https://mui.com/material-ui/getting-started/installation/

https://m3.material.io/

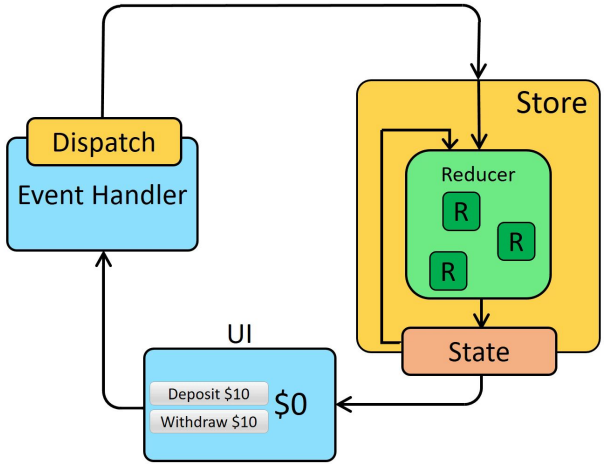
Redux Toolkit



Redux is a pattern and library for managing and updating **application state**, using events called "**actions**".

It serves as a centralized store for a **global state** (used across your entire application), with rules ensuring that the state can only be updated in a predictable fashion.

The patterns and tools provided by Redux make it easier to understand **when, where, why, and how the state in your application is being updated**, and how your application logic will behave when those changes occur.



https://redux-toolkit.js.org/

React Hook Form + Yup

```
import { useForm } from 'react-hook-form'

export const App = () => {
  const { register, handleSubmit, formState: { errors } } = useForm()
  const onSubmit = data => console.log(data)
  return (
    <form onSubmit={handleSubmit(onSubmit)}>
      <input type="text" placeholder="Name" {...register("Name", {required: true, maxLength: 80})} />
      <input type="email" placeholder="Email" {...register("Email", {required: true, pattern: /^\S+@\S+$/i})} />
      <input type="submit" />
    </form>
  )
}
```

https://react-hook-form.com/

https://github.com/jquense/yup

react-i18next

Internationalization framework for React which is based on i18next.

```
import i18n from "i18next"
import { useTranslation, initReactI18next } from "react-i18next"

i18n
  .use(initReactI18next) // passes i18n down to react-i18next
  .init({
    // [...]
  })

export const App = () => {
  const { t } = useTranslation()
  return <h2>{t('Welcome to React')}</h2>
}
```

https://react.i18next.com/

Other noteworthy packages

- **Axios** - https://axios-http.com/
  Promise based HTTP client for the browser and node.js
- **Lodash** - https://lodash.com/
  A modern JavaScript utility library delivering modularity, performance & extras
- **Luxon** - https://moment.github.io/luxon/
  Library for dealing with dates and times in JavaScript
- **React Draggable** - https://github.com/react-grid-layout/react-draggable
  A simple component for making elements draggable
- **React Motion** - https://github.com/chenglou/react-motion
  Popular animation library

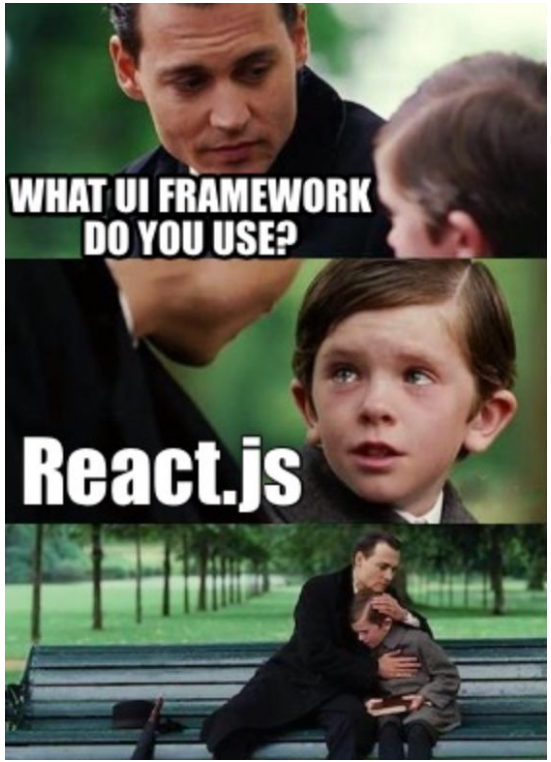
Good coding!

You can find these slides and the code of the sample application on the following GitHub repository:

https://github.com/franksacco/react-in-practice

For doubts or further information, please contact:

- Francesco Saccani - francesco.saccani@unipr.it
- Gabriele Penzotti - gabriele.penzotti@unipr.it



Project & Thesis Proposals

- **Sensors for soil and environmental data acquisition**
  - Embedded programming (Rust & C)
  - **TinyML**: Machine Learning in constrained devices
  - ***On-Device Learning***
- **Microservices architectures**
  - Study and use of containerization technologies such as Kubernetes and Docker
  - Development in constrained systems at the edge
- **GeoSpatial Machine Learning**
  - Machine Learning applied to satellite maps (Sentinel 2, …) and time series
  - Dataset definition, model training, data analysis, …
- **Web service with Dashboard**

  Visualization of sensory and geospatial data in the Smart Farming context