Web Application Architectures

Lab 4 - Create a Next.js app to demo the Student API

# **Initial Setup**

To complete this tutorial we will need the completed student API from week 2. The code for this is available on Brightspace in the Week 4 Lab section.

To do:

Step 1: Open the student api app in Brightspace and run the server on port 3000.

A screenshot of a computer program

AI-generated content may be incorrect.

Step 2: Create a Lab\_4 directory and open this in VS Code

A screenshot of a computer

AI-generated content may be incorrect.

# **What is Next.js?**

Next.js is a React framework built by Vercel

It adds powerful features on top of React such as:

* **File-based routing**: create routes by adding files in the app/ or pages/ directory.
* **Server-side rendering (SSR) & static site generation (SSG)**: pre-render content for speed and SEO.
* **API routes**: build backend endpoints alongside your frontend.
* **Full-stack React**: use both server and client components in the same project.
* **Performance optimizations**: image optimization, font optimization, caching, and bundling out of the box.

Source: <https://nextjs.org/learn/dashboard-app/getting-started>

# **Getting Started:**

Next.js recommends using **pnpm as the package manager**, as it's faster and more efficient than npm or yarn. If you don't have pnpm installed, you can install it globally by running:

|  |  |
| --- | --- |
| In the Lab\_4 project in VS Code, open a terminal window and run:  npm install -g pnpm |  |

Source: <https://nextjs.org/learn/dashboard-app/getting-started>

# **Create a new Next App**

To create a Next.js app, in the lab\_4 terminal in VS Code, run the following command:

npx create-next-app@latest student\_frontend --use-pnpm

This command uses create-next-app, a Command Line Interface (CLI) tool that sets up a Next.js application for you.

When prompted select the following options:

A computer screen with text and numbers

AI-generated content may be incorrect.

Source: <https://nextjs.org/learn/dashboard-app/getting-started>

## **Folder structure:**

|  |  |
| --- | --- |
| A screenshot of a computer  AI-generated content may be incorrect. | **/app**: Will contain all the routes, components, and logic for your application, this is where you'll be mostly working from.  **/app/lib**: Contains functions used in your application, such as reusable utility functions and data fetching functions.  **/app/ui**: Contains all the UI components for your application, such as cards, tables, and forms. To save time, we've pre-styled these components for you.  **/public**: Contains all the static assets for your application, such as images.  **Config Files**: You'll also notice config files such as next.config.ts at the root of your application. Most of these files are created and pre-configured when you start a new project using create-next-app. You will not need to modify them in this course. |

## **Run the server:**

|  |  |
| --- | --- |
| Cd into the new next app:  cd student\_frontend  Then run:  pnpm dev -p 4000 |  |

Open http://localhost:4000 on your browser. Your home page should look as follows:

A screenshot of a computer

AI-generated content may be incorrect.

# **Create the Environmental Variables**

## **Setup Environment Variables (Next Frontend project):**

Create a new file named .env.local in the student\_frontend directory

Add the following:

NEXT\_PUBLIC\_API\_URL=http://localhost:3000

## **Setup CORS for the Student API project:**

Make sure cors is included in the Student API app.

|  |  |
| --- | --- |
| npm i cors  import cors from "cors";  app.use(cors()); |  |

## **Test the Student API App, in terminal run:**

npm run dev

Make sure there is no errors in terminal.

# **What is React?**

JavaScript library for writing front end for web apps

Created by Facebook (Meta)

created by Jordan Walke (software engineer) at Meta, who released an early prototype of React called "FaxJS” in 2013

“React lets you build user interfaces out of individual pieces called components. Create your own React components like Thumbnail, LikeButton, and Video. Then combine them into entire screens, pages, and apps.” Source: https://react.dev/

React is a feature rich component based JavaScript UI library.

It offers features to help develop UI applications .

Offers advanced concepts, eg:

* State management
* Routing
* Etc…

First version: 0.3.0, May 2013

React follows Semantic Versioning (semver)

## **What is a Component?**

React Components

* With React components are the building blocks for the User Interface.
* The offer self-contained and reusable components that encapsulate specific behavior and functionality.
* A React application usually used a variety of different components.
* A React component can be a:
  + JS Class
  + Function
* The function is now the preferred way to create a component

**React Components** are the primary building blocks of the app.

It uses React Elements and JSX to design the UI.

The React component can be a JS Class (extends the React component class) or a pure JavaScript function.

A React Component can have:

* Properties (Props)
* State management
* Life Cycle
* Event handlers

A component can cater for simple or complex logic.

## **Functional Components**

* JS function that returns React elements.
* This basically dictates what will display on the screen.
* Appear to be stateless and they don’t manage their own internal state.
* Will receive data through props (properties).
* Naming convention will follow Pascal Casing.

Demo of a simple component

This is what a component looks like.

A screenshot of a computer

AI-generated content may be incorrect.

## **Components Summary**

* Components are like building blocks, making them reusable.
* Components let you organize UI into small, focused pieces.
* Each component handles its own markup, styling, and logic.
* Each component manages its own state (data it needs to work).
* Components can be nested and combined to build complex UIs from simple parts.

# **What is React Markup?**

React uses JSX (JavaScript XML), a syntax extension for JavaScript that looks very similar to HTML. JSX allows you to write UI components declaratively inside JavaScript. It combines markup (HTML-like tags) with JavaScript expressions, making your UI dynamic and interactive.

## **JSX Basics**

**JSX Looks Like HTML**

* JSX uses HTML-like syntax inside JavaScript, but it’s actually compiled into React function calls.
* Eg. const element = <h1>Hello, world!</h1>;

**Embedding JavaScript Expressions**

* Use {} to insert variables, expressions, or function calls.
* Eg.

const name = "Joe";

const element = <p>Hello, {name}!</p>;

**Attributes in JSX**

* class → className
* for → htmlFor
* All other HTML attributes mostly stay the same.
* Eg.

<label htmlFor="email">Email:</label>

<input id="email" className="border p-2" />

**Self-closing Tags**

* In JSX, all void elements must self-close.
* Eg.

<img src="/logo.png" alt="Logo" />

<input type="text" />

<br />

**Event Handlers**

* Use camelCase for events (onClick, onChange, onSubmit).
* Pass a function reference instead of a string.
* eg. <button onClick={() => alert("Clicked!")}>Click Me</button>

**Conditional Rendering**

* Use ternary operators or logical expressions inside {}.
* Eg.

const isEnrolled = true;

<p>Status: {isEnrolled ? "Enrolled" : "Pending"}</p>

**Styling**

* use className with Tailwind/CSS.
* <div className="text-red-500 text-xl">Styled with Tailwind</div>

**JSX Must Have a Single Parent**

* Wrap multiple elements in a div or React Fragment.

<>

<h1>Title</h1>

<p>Paragraph</p>

</>

**JSX is Case-Sensitive**

* Components must start with a capital letter.
* HTML tags are lowercase; React treats capitalized tags as components.
* eg.

function StudentCard() {

return <div>Student Info</div>;

}

# **Update the Home Component**

To demonstrate components in our new Next project lets update the Home componemt

The Home component resides in app/page.tsx. This file represents the root route / of your application. In the project it currently looks as follows:

|  |  |
| --- | --- |
| import Image from "next/image";  export default function Home() {  return (  <div className="font-sans grid grid-rows-[20px\_1fr\_20px] items-center justify-items-center min-h-screen p-8 pb-20 gap-16 sm:p-20">  <main className="flex flex-col gap-[32px] row-start-2 items-center sm:items-start">  <Image  className="dark:invert"  src="/next.svg"  alt="Next.js logo"  width={180}  height={38}  priority  />  <ol className="font-mono list-inside list-decimal text-sm/6 text-center sm:text-left">  <li className="mb-2 tracking-[-.01em]">  Get started by editing{" "}  <code className="bg-black/[.05] dark:bg-white/[.06] font-mono font-semibold px-1 py-0.5 rounded">  src/app/page.tsx  </code>  .  </li>  <li className="tracking-[-.01em]">  Save and see your changes instantly.  </li>  </ol>  <div className="flex gap-4 items-center flex-col sm:flex-row">  <a  className="rounded-full border border-solid border-transparent transition-colors flex items-center justify-center bg-foreground text-background gap-2 hover:bg-[#383838] dark:hover:bg-[#ccc] font-medium text-sm sm:text-base h-10 sm:h-12 px-4 sm:px-5 sm:w-auto"  href="https://vercel.com/new?utm\_source=create-next-app&utm\_medium=appdir-template-tw&utm\_campaign=create-next-app"  target="\_blank"  rel="noopener noreferrer"  >  <Image  className="dark:invert"  src="/vercel.svg"  alt="Vercel logomark"  width={20}  height={20}  />  Deploy now  </a>  <a  className="rounded-full border border-solid border-black/[.08] dark:border-white/[.145] transition-colors flex items-center justify-center hover:bg-[#f2f2f2] dark:hover:bg-[#1a1a1a] hover:border-transparent font-medium text-sm sm:text-base h-10 sm:h-12 px-4 sm:px-5 w-full sm:w-auto md:w-[158px]"  href="https://nextjs.org/docs?utm\_source=create-next-app&utm\_medium=appdir-template-tw&utm\_campaign=create-next-app"  target="\_blank"  rel="noopener noreferrer"  >  Read our docs  </a>  </div>  </main>  <footer className="row-start-3 flex gap-[24px] flex-wrap items-center justify-center">  <a  className="flex items-center gap-2 hover:underline hover:underline-offset-4"  href="https://nextjs.org/learn?utm\_source=create-next-app&utm\_medium=appdir-template-tw&utm\_campaign=create-next-app"  target="\_blank"  rel="noopener noreferrer"  >  <Image  aria-hidden  src="/file.svg"  alt="File icon"  width={16}  height={16}  />  Learn  </a>  <a  className="flex items-center gap-2 hover:underline hover:underline-offset-4"  href="https://vercel.com/templates?framework=next.js&utm\_source=create-next-app&utm\_medium=appdir-template-tw&utm\_campaign=create-next-app"  target="\_blank"  rel="noopener noreferrer"  >  <Image  aria-hidden  src="/window.svg"  alt="Window icon"  width={16}  height={16}  />  Examples  </a>  <a  className="flex items-center gap-2 hover:underline hover:underline-offset-4"  href="https://nextjs.org?utm\_source=create-next-app&utm\_medium=appdir-template-tw&utm\_campaign=create-next-app"  target="\_blank"  rel="noopener noreferrer"  >  <Image  aria-hidden  src="/globe.svg"  alt="Globe icon"  width={16}  height={16}  />  Go to nextjs.org →  </a>  </footer>  </div>  );  } | A screenshot of a computer  AI-generated content may be incorrect. |

Now we will update the Home page with our own student focused page

Replace the content of **/src/app/page.tsx** with the following. We aren’t explaining the content of this file, you can go through this yourself and see how it relates to the JSX description detailed above. The styling of the JSX is provided via Tailwind. The classNames map to existing Tailwind style rules.

|  |  |
| --- | --- |
| import Link from "next/link";  export default function Home() {  return (  <main className="min-h-screen flex flex-col items-center justify-center bg-gradient-to-br from-blue-50 via-white to-blue-100 p-6">  {/\* Header \*/}  <header className="w-full max-w-5xl flex justify-between items-center mb-16">  <h1 className="text-3xl font-bold text-blue-700">Student Manager</h1>  <nav className="space-x-6 text-lg font-medium">  <Link href="/about" className="text-gray-600 hover:text-blue-600">  About  </Link>  <Link href="/features" className="text-gray-600 hover:text-blue-600">  Features  </Link>  <Link href="/login" className="text-blue-600 hover:underline">  Login  </Link>  </nav>  </header>  {/\* Hero Section \*/}  <section className="grid lg:grid-cols-2 gap-12 items-center max-w-5xl w-full">  {/\* Text \*/}  <div className="text-center lg:text-left">  <h2 className="text-5xl font-extrabold text-gray-900 leading-tight mb-6">  Manage Students <span className="text-blue-600">Effortlessly</span>  </h2>  <p className="text-gray-600 text-lg mb-8">  A modern student management app to track performance, attendance,  and academic records — all in one place.  </p>  <div className="flex gap-4 justify-center lg:justify-start">  <Link  href="/signup"  className="px-6 py-3 bg-blue-600 text-white rounded-xl shadow hover:bg-blue-700 transition"  >  Get Started  </Link>  <Link  href="/demo"  className="px-6 py-3 border border-blue-600 text-blue-600 rounded-xl hover:bg-blue-50 transition"  >  Live Demo  </Link>  </div>  </div>  {/\* Hero Image \*/}  <div className="flex justify-center lg:justify-end">    </div>  </section>  {/\* Features Section \*/}  <section className="mt-20 grid gap-8 sm:grid-cols-2 lg:grid-cols-3 max-w-5xl w-full">  <div className="p-6 bg-white rounded-2xl shadow hover:shadow-lg transition">  <h3 className="text-xl font-semibold mb-2 text-blue-700">Student Profiles</h3>  <p className="text-gray-600">  Create and manage student profiles with academic records, grades, and attendance tracking.  </p>  </div>  <div className="p-6 bg-white rounded-2xl shadow hover:shadow-lg transition">  <h3 className="text-xl font-semibold mb-2 text-blue-700">Class Management</h3>  <p className="text-gray-600">  Organize classes, assign teachers, and keep everything structured and accessible.  </p>  </div>  <div className="p-6 bg-white rounded-2xl shadow hover:shadow-lg transition">  <h3 className="text-xl font-semibold mb-2 text-blue-700">Reports & Analytics</h3>  <p className="text-gray-600">  Generate performance reports and gain insights into student progress in seconds.  </p>  </div>  </section>  {/\* Footer \*/}  <footer className="mt-20 text-gray-500 text-sm">  © {new Date().getFullYear()} Student Manager. All rights reserved.  </footer>  </main>  );  } |  |

Save the changes.

Run the project (in terminal make sure you are in the student\_frontend directory):

npm run dev

A screenshot of a computer

AI-generated content may be incorrect.

# **Add styling to the project**

Currently the home page is styled via the project setup boilerplate.

We will look at the different ways to style the Next.js app.

## **Global styles**

We can cater for global styling in the project. The globals.css file provide this for us and the resides in the app directory.

In our project setup we selected to include **Tailwind as our CSS library**.

Use global.css to add CSS rules to all the routes in your application - such as CSS reset rules, site-wide styles for HTML elements like links, and more.

You can import global.css in any component in your application, but it's usually good practice to add it to your top-level component.

A screenshot of a computer program

AI-generated content may be incorrect.

Add global styles to your application by navigating to /app/layout.tsx and importing the global.css file:

A screen shot of a computer program

AI-generated content may be incorrect.

Global styling will be configured through the root layout component (app/layout.tsx). This should be setup by default as part of the project install.

## **Images**

* The <Image> component in Next.js builds on the standard HTML <img> tag, but adds built-in optimization features to improve performance and user experience:
* No layout shift → it automatically reserves space so the page doesn’t jump when images load.
* Responsive resizing → delivers images sized appropriately for the user’s device instead of always sending large files.
* Lazy loading → images only load when they scroll into view, reducing initial load time.
* Modern formats → serves images in WebP or AVIF when supported, falling back to standard formats if not.

Add a hero image to the landing page. Download the image from Brightspace (students.png) and add it to the Public directory. Source: <https://pixabay.com/vectors/pixel-cells-lecture-lecture-hall-3976296/>

Add the following Image component to the Home page.

|  |  |
| --- | --- |
| import Image from "next/image";  <Image  src="/students.png"  alt="Illustration of students using the app"  width={500}  height={400}  className="rounded-2xl shadow-lg"  priority // ensures hero image loads quickly  /> |  |

Note: don’t forget the import. We need to import every component we wish to use on our pages.

A screenshot of a computer

AI-generated content may be incorrect.

# **Next.js Routing**

Routing is how Next.js decides which page (or component) to render when a user visits a specific URL in your application.

Eg.

/ → shows the Home page.

/students → shows the Students page.

/students/123 → shows the details page for student with ID 123.

Next.js handles this automatically using the file and folder structure of your project, no extra router library is required.

In Next.js version 15 the App Router provides routing functionality.

* Uses the app/ directory.
* Every folder and file in app/ defines a route.
* Supports layouts, templates, and advanced data-fetching.

## **Special File Conventions (App Router)**

|  |  |
| --- | --- |
| page.tsx | defines the UI for a route |
| layout.tsx | shared layout across routes (e.g., navbar, sidebar) |
| loading.tsx | loading UI while a page is fetching data |
| error.tsx | error boundary for that route |
| not-found.tsx | 404 handling for that route |

# **Student Frontend**

The frontend is a React-based web application built with Next.js. It will connect to the Express backend API to manage students. The homepage for students will show a table listing all student records, with options to view, edit, or delete each entry. Clicking “View” will open a details page for that student, showing their personal information in a clean card layout. The “Edit” link will open a form where the student’s current details are pre-filled, allowing changes to be saved back to the backend. The “Delete” button will remove a student and updates the list instantly without refreshing the whole page.

There is also an “Add New Student” page that displays a form for creating a new record. After submitting, the user is redirected back to the list, where the new student appears immediately.

All navigation uses Next.js’ client-side routing, so moving between pages feels fast and seamless. The forms and actions communicate with the backend API using fetch, and the UI updates dynamically based on the results.

# **Define the Types**

We will create two interfaces to define the shape of data your frontend works with when talking to the backend API.

The Student interface represents a full student record as it exists in the database. It includes all properties, including the id (which the backend generates). This is what you use when reading or displaying students (e.g. on the list, details, or edit pages).

The StudentCreate interface represents the data needed to create a new student. Notice it does not include the id because at creation time, the frontend doesn’t know it yet — the backend will assign it. You’ll use this type when building forms and sending POST requests to add students.

In the student\_frontend create a new directory named types

In the types directory create a new file named student.ts

Add the following interfaces to student.ts:

|  |  |
| --- | --- |
| export interface Student {  id: string;  firstName: string;  lastName: string;  studentNumber: string;  email: string;  course: string;  }  export interface StudentCreate {  firstName: string;  lastName: string;  studentNumber: string;  email: string;  course: string;  } |  |

# **Create a Service Layer**

We will create a service layer or API client module for our Next.js frontend. Instead of calling fetch directly inside every page or component, we will centralize all communication with the backend API in one place.

This module acts as the bridge between the frontend and backend, enforcing types, reducing duplication, and keeping the app maintainable and clean.

In the student\_frontend create a new directory named services

In the types directory create a new file named studentAPI.ts

Add the following interfaces to studentAPI.ts:

|  |  |
| --- | --- |
| import { Student, StudentCreate } from "../types/student";  const API\_URL = process.env.NEXT\_PUBLIC\_API\_URL || "http://localhost:3000";  const BASE\_URL = `${API\_URL}/v1/students`; |  |

We will add to this as we implement our Student CRUD operations in the following steps.

This code sets up the base configuration for interacting with the student API. It imports the TypeScript types for a student and a student creation object. It defines a main API URL by reading an environment variable, defaulting to a local server if the variable is not set, and then constructs a base URL for all student-related endpoints. This base URL is used throughout the project to make HTTP requests to the backend for operations like fetching, creating, updating, or deleting students.

# **Define Pages / Components**

We will use the App Router to cater for our Student Routes

## **Structure:**

/app

/students

page.tsx — list all students

[id]

page.tsx — view one student

/edit

page.tsx — form to update student

/new

page.tsx — form to create new student

layout.tsx

page.tsx — homepage or redirect to /students

Inside the pages / components, we will use fetch (server components or client components). In Next.js 15, the default caching behavior for fetch is no caching (i.e. no-store) so your API calls will always fetch fresh data, unless we override.

|  |  |
| --- | --- |
| In student\_frontend/src/app create a new directory named students  **Ie. In the apps directory create a directory named students** |  |

# **Create the Student List All**

First we will add a get\_students function to the Student services.

**Add the following to services/studentAPI.ts**

|  |  |
| --- | --- |
| export async function getStudents(): Promise<Student[]> {    const res = await fetch(BASE\_URL, { cache: "no-store" });    if (!res.ok) throw new Error(`Failed to fetch students: ${res.status} ${res.statusText}`);    const data = await res.json();    return data.value;  } |  |

This function is a TypeScript async helper that fetches the list of students from the backend API. It uses the fetch API to make a GET request to the BASE\_URL.

The { cache: "no-store" } option ensures the request always fetches fresh data instead of using cached responses.

If the request fails (status not OK), it throws an error with the status code and text.

If successful, it parses the JSON response and returns the value property, which is expected to be an array of Student objects.

**In the app/students directory create a file named page.tsx**

|  |  |
| --- | --- |
| "use client";  import { useEffect, useState } from "react";  import { getStudents } from "../../../services/studentAPI"  import { Student } from "../../../types/student";  import Link from "next/link";  export default function StudentsPage() {      const [students, setStudents] = useState<Student[]>([]);      const fetchStudents = async () => {          const data = await getStudents();          setStudents(data);      };      useEffect(() => {          fetchStudents();      }, []);      return (          <main className="min-h-screen bg-gradient-to-br from-blue-50 via-white to-blue-100 p-6">              <div className="max-w-6xl mx-auto">                  {/\* Page Header \*/}                  <header className="flex justify-between items-center mb-10">                      <h1 className="text-3xl font-bold text-blue-700">Students</h1>                      <Link                          href="/students/new"                          className="px-4 py-2 bg-blue-600 text-white rounded-lg shadow hover:bg-blue-700 transition"                      >                          + Add Student                      </Link>                  </header>                  {/\* Students Table or Empty State \*/}                  {students.length === 0 ? (                      <p className="text-gray-500 text-center bg-white rounded-lg shadow p-6">                          No students found.                      </p>                  ) : (                      <div className="overflow-x-auto bg-white rounded-lg shadow border border-gray-200">                          <table className="min-w-full text-gray-700">                              <thead className="bg-gray-100 text-sm font-semibold">                                  <tr>                                      <th className="px-4 py-3 text-left">Name</th>                                      <th className="px-4 py-3 text-left">Student #</th>                                      <th className="px-4 py-3 text-left">Email</th>                                      <th className="px-4 py-3 text-left">Course</th>                                      <th className="px-4 py-3 text-center">Actions</th>                                  </tr>                              </thead>                              <tbody className="divide-y divide-gray-200 text-sm">                                  {students.map((s) => (                                      <tr key={s.id} className="hover:bg-gray-50">                                          <td className="px-4 py-3">                                              {s.firstName} {s.lastName}                                          </td>                                          <td className="px-4 py-3">{s.studentNumber}</td>                                          <td className="px-4 py-3">{s.email}</td>                                          <td className="px-4 py-3">{s.course}</td>                                          <td className="px-4 py-3 text-center space-x-3">                                              <Link                                                  href={`/students/${s.id}`}                                                  className="text-blue-600 hover:underline"                                              >                                                  View                                              </Link>                                              <Link                                                  href={`/students/${s.id}/edit`}                                                  className="text-green-600 hover:underline"                                              >                                                  Edit                                              </Link>                                              <button                                                  className="text-red-600 hover:underline"                                              >                                                  Delete                                              </button>                                          </td>                                      </tr>                                  ))}                              </tbody>                          </table>                      </div>                  )}              </div>          </main>      );  } |  |

This code defines a client-side Students page in a Next.js application. It uses React’s useState and useEffect hooks to fetch student data from an API after the component mounts and stores it in state.

The page displays a header with a title and a “Add Student” button styled consistently with the Home page. Below the header, it either shows an empty state if there are no students, or renders the list of students in a styled table with their name, student number, email, and course. Each student row also includes action buttons for viewing, editing, or deleting the student.

The "use client" directive ensures this page runs entirely in the browser, allowing interactive features like live updates or button clicks to work without a full page reload.

Run the server: pnpm dev -p 4000

A screen shot of a computer

AI-generated content may be incorrect.

Go to: <http://localhost:4000/students>

A screenshot of a computer

AI-generated content may be incorrect.

# **View an individual Student**

First we will add a get\_student function to the Student services.

**Add the following to services/studentAPI.ts**

|  |  |
| --- | --- |
| export async function getStudent(id: string): Promise<Student> {      const res = await fetch(`${BASE\_URL}/${id}`, { cache: "no-store" });      if (!res.ok) throw new Error("Student not found");      return res.json();  } |  |

This function is a utility for retrieving the details of a single student from the backend API.

It takes a student’s unique identifier as an argument, builds the request URL by appending that identifier to a base API endpoint, and makes an asynchronous fetch request. The cache: "no-store" option ensures that the response is always fresh and not cached.

If the response is unsuccessful (for example, if the student does not exist), it throws an error with the message “Student not found.” Otherwise, it parses the JSON from the response and returns it as a Student object.

Each student in the backend can be fetched by their unique identifier through the API route that looks like /v1/students/{id}.

In Next.js, when you have a dynamic route such as a page under app/students/[id]/page.tsx, you can use the params that Next.js gives you to access that id.

Inside the page component, you can call fetch with the URL that includes the id. For example, if the id is in params.id, you append it to the API base URL.

**In the students directory create a new directory named [id]. Ie. Include the square brackets. In the new [id] directory create a file named page.tsx**

Add the following code to [id] -> page.tsx

|  |  |
| --- | --- |
| "use client";  import { useEffect, useState } from "react";  import { useParams, useRouter } from "next/navigation";  import { getStudent } from "../../../../services/studentAPI"  import { Student } from "../../../../types/student";  import Link from "next/link";  export default function ViewStudentPage() {      const params = useParams();      const router = useRouter();      const studentId = Array.isArray(params.id) ? params.id[0] : params.id;      const [student, setStudent] = useState<Student | null>(null);      const [loading, setLoading] = useState(true);      useEffect(() => {          if (!studentId) return;          getStudent(studentId)              .then((data) => setStudent(data))              .catch(() => setStudent(null))              .finally(() => setLoading(false));      }, [studentId]);      if (!studentId) return <div>Invalid student ID.</div>;      if (loading) return <div>Loading...</div>;      if (!student) return (          <main className="min-h-screen bg-gradient-to-br from-blue-50 via-white to-blue-100 p-6 flex items-center justify-center">              <div className="max-w-xl w-full bg-white rounded-2xl shadow-lg p-8 text-center">                  <h1 className="text-2xl font-bold text-blue-700 mb-4">Student not found</h1>                  <p className="text-gray-500 mb-6">The requested student does not exist.</p>                  <Link                      href="/students"                      className="px-4 py-2 bg-gray-200 text-gray-700 rounded-lg hover:bg-gray-300 transition"                  >                      Back to list                  </Link>              </div>          </main>      );      return (          <main className="min-h-screen bg-gradient-to-br from-blue-50 via-white to-blue-100 p-6 flex justify-center">              <div className="max-w-3xl w-full bg-white rounded-2xl shadow-lg p-8">                  <h1 className="text-3xl font-bold text-blue-700 mb-6">                      {student.firstName} {student.lastName}                  </h1>                  <div className="space-y-4 text-blue-700">                      <p><span className="font-medium">Student #:</span> {student.studentNumber}</p>                      <p><span className="font-medium">Email:</span> {student.email}</p>                      <p><span className="font-medium">Course:</span> {student.course}</p>                  </div>                  <div className="mt-4 flex gap-4">                      <Link                          href={`/students/${student.id}/edit`}                          className="px-4 py-2 bg-green-600 text-white rounded-lg hover:bg-green-700 transition"                      >                          Edit                      </Link>                      <Link                          href="/students"                          className="px-4 py-2 bg-gray-200 text-gray-700 rounded-lg hover:bg-gray-300 transition"                      >                          Back to list                      </Link>                  </div>              </div>          </main>      );  } |  |

This page is a dynamic route in Next.js (/students/[id]). That means when a user visits a URL like /students/123, Next.js passes the route parameter (123) to the page component via params.

In this implementation, the props are typed using a StudentPageProps interface, which declares that params will not be a plain object but rather a Promise<{ id: string }> — meaning the route parameters must be awaited before use. That’s why you see const { id } = await params;. This ensures the function safely extracts the id before using it in the API request.

Once the id is available, the function makes a fetch request to the backend API, using the id to get the student’s details. If the response is unsuccessful, the page shows a “student not found” message styled consistently with the home page. If successful, it parses the response into a Student type (defined elsewhere for strong typing) and renders the student’s profile, with options to edit or go back to the list.

Tet the app.

Run the server: pnpm dev -p 4000

A screen shot of a computer

AI-generated content may be incorrect.

Go to: <http://localhost:4000/students>

Click View for one of the Students:

A screenshot of a computer

AI-generated content may be incorrect.

# **Delete a Student**

First we will add a deleteStudent function to the Student services.

**Add the following to services/studentAPI.ts**

|  |  |
| --- | --- |
| export async function deleteStudent(id: string): Promise<void> {      const res = await fetch(`${BASE\_URL}/${id}`, { method: "DELETE" });      if (!res.ok) throw new Error("Failed to delete student");  } |  |

This function is responsible for deleting a student record from the backend.

It takes the student’s unique ID, sends a DELETE request to the API, and checks if the request was successful. If the response is not OK, it throws an error saying “Failed to delete student.”

Since the operation doesn’t return any data, the function’s return type is Promise<void>. In short, it provides a simple way to remove a student by ID while handling errors gracefully.

Update app/students/page.tsx. This is the list all page, we will create a delete link foe each student in the list.

**Add the following to page.tsx.**

Update this services import to include the deleteStudent function:

import { getStudents, **deleteStudent** } from "../../services/studentAPI"

Add the following function to the page (just before the return block in students/page.tsx)

|  |  |
| --- | --- |
| const handleDelete = async (id: string) => {      if (confirm("Are you sure?")) {        await deleteStudent(id);        fetchStudents();      }    }; |  |

This function will be called when the delete button is clicked.

Update the delete button to look as follows:

|  |  |
| --- | --- |
| <button  className="px-2 py-1 bg-red-500 text-white rounded"      onClick={() => handleDelete(s.id)}  >  Delete  </button> |  |

This code defines and uses a delete action for students:

The handleDelete function takes a student’s ID.

It shows a confirmation dialog (confirm("Are you sure?")).

If the user confirms, it calls the deleteStudent API function to remove the student from the backend.

After deletion, it refreshes the list of students by calling fetchStudents().

The <button> is styled as a red delete button, and when clicked, it triggers handleDelete with the selected student’s ID (s.id).

Run the server: pnpm dev -p 4000

A screen shot of a computer

AI-generated content may be incorrect.

Go to: <http://localhost:4000/students>

A screenshot of a computer

AI-generated content may be incorrect.

Click a delete button to delete a student.

# **Create a shared form**

In the student\_frontend directory **create a directory named components**

In the components directory create a **new file named StudentForm.tsx**

Add the following to StudentForm.tsx

|  |  |
| --- | --- |
| "use client";  import { useState } from "react";  import { StudentCreate } from "../types/student";  interface Props {    initialData?: StudentCreate;    onSubmit: (data: StudentCreate) => void;  }  export default function StudentForm({ initialData, onSubmit }: Props) {    const [firstName, setFirstName] = useState(initialData?.firstName || "");    const [lastName, setLastName] = useState(initialData?.lastName || "");    const [studentNumber, setStudentNumber] = useState(initialData?.studentNumber || "");    const [email, setEmail] = useState(initialData?.email || "");    const [course, setCourse] = useState(initialData?.course || "");    const handleSubmit = (e: React.FormEvent) => {      e.preventDefault();      onSubmit({ firstName, lastName, studentNumber, email, course });    };    return (      <form onSubmit={handleSubmit} className="flex flex-col gap-2 max-w-md">        <input value={firstName} onChange={e => setFirstName(e.target.value)} placeholder="First Name" />        <input value={lastName} onChange={e => setLastName(e.target.value)} placeholder="Last Name" />        <input value={studentNumber} onChange={e => setStudentNumber(e.target.value)} placeholder="Student Number" />        <input value={email} onChange={e => setEmail(e.target.value)} placeholder="Email" />        <input value={course} onChange={e => setCourse(e.target.value)} placeholder="Course" />        <button type="submit" className="px-4 py-2 bg-blue-500 text-white rounded">Submit</button>      </form>    );  } |  |

This code defines a shared StudentForm component that can be reused across the project for both creating and editing student records.

What it does:

* It’s a client component ("use client") because it manages state and user interactions.
* It accepts two props:
  + initialData → optional data to pre-fill the form (useful when editing an existing student).
  + onSubmit → a callback function that runs when the form is submitted.
* It uses React’s useState hooks to track form field values (firstName, lastName, studentNumber, email, course).
* When the form is submitted, it prevents the default page refresh and calls onSubmit with the form data as a StudentCreate object.
* The UI is a simple styled form with inputs for all student fields and a submit button.

We will use this component for creating and editing a student.

# **Create a New Student**

First we will add a createStudent function to the Student services.

**Add the following to services/studentAPI.ts**

|  |  |
| --- | --- |
| export async function createStudent(student: StudentCreate): Promise<Student> {      const res = await fetch(BASE\_URL, {          method: "POST",          headers: { "Content-Type": "application/json" },          body: JSON.stringify(student),      });      if (!res.ok) throw new Error("Failed to create student");      return res.json();  } |  |

|  |  |
| --- | --- |
| Create a new directory in app/students named new  In the new directory create a file named page.tsx |  |

Add the following to new/page.tsx

|  |  |
| --- | --- |
| "use client";  import { useRouter } from "next/navigation";  import StudentForm from "../../../../components/StudentForm"  import { createStudent } from "../../../../services/studentAPI";  import { StudentCreate } from "../../../../types/student";  export default function NewStudentPage() {    const router = useRouter();    const handleCreate = async (data: StudentCreate) => {      await createStudent(data);      router.push("/students");    };    return (      <main className="min-h-screen bg-gradient-to-br from-blue-50 via-white to-blue-100 p-6 flex items-center justify-center">        <div className="max-w-3xl w-full bg-white rounded-2xl shadow-lg p-8 text-blue-700">          <h1 className="text-3xl font-bold text-blue-700 mb-6">Add Student</h1>          <StudentForm onSubmit={handleCreate} />        </div>      </main>    );  } |  |

This code defines a client-side page in Next.js for adding a new student.

It imports Next.js router to handle navigation, a shared StudentForm component for the input fields, and the createStudent service function to send data to the backend.

The handleCreate function runs when the form is submitted: it calls createStudent with the entered data and then redirects the user back to the /students list.

The return statement builds the UI:

* A full-page gradient background that matches the home page style.
* A centered card (max-w-3xl, white background, rounded corners, shadow) that contains the form.
* A header with "Add Student" styled consistently with the rest of the app.
* The reusable StudentForm component where the user enters student details.

Run the server: pnpm dev -p 4000

A screen shot of a computer

AI-generated content may be incorrect.

Go to: <http://localhost:4000/students>

A screenshot of a computer

AI-generated content may be incorrect.

# **Update an individual Student**

First we will add a createStudent function to the Student services.

**Add the following to services/studentAPI.ts**

|  |  |
| --- | --- |
| export async function updateStudent(id: string, student: StudentCreate): Promise<Student> {      const res = await fetch(`${BASE\_URL}/${id}`, {          method: "PUT",          headers: { "Content-Type": "application/json" },          body: JSON.stringify(student),      });      if (!res.ok) throw new Error("Failed to update student");      return res.json();  } |  |

|  |  |
| --- | --- |
| **Create a new directory in app/students/[id] named edit**  In the edit directory **create a file named page.tsx** |  |

Updating a student in your Next.js app follows the same pattern as creating one, but with a few differences.

* Fetch the current student data from the API.
* Pre-fill the form fields with that data.
* Submit updates with a PUT request to your backend.

Add the following code to edit/page.tsx

|  |  |
| --- | --- |
| "use client";  import { useRouter, useParams } from "next/navigation";  import { useEffect, useState } from "react";  import StudentForm from "../../../../../components/StudentForm";  import { getStudent, updateStudent } from "../../../../../services/studentAPI";  import { StudentCreate } from "../../../../../types/student";  export default function EditStudentPage() {    const router = useRouter();    const params = useParams();    const studentId = Array.isArray(params.id) ? params.id[0] : params.id;    const [student, setStudent] = useState<StudentCreate | null>(null);    const [loading, setLoading] = useState(true);    // Fetch student data when the component mounts    useEffect(() => {      if (studentId) {        getStudent(studentId)          .then((data) => {            setStudent({              firstName: data.firstName,              lastName: data.lastName,              studentNumber: data.studentNumber,              email: data.email,              course: data.course,            });            setLoading(false);          })          .catch(() => setLoading(false));      }    }, [studentId]);    // Handle update submission    const handleUpdate = async (data: StudentCreate) => {      if (!studentId) return;      await updateStudent(studentId, data);      router.push("/students");    };    // UI states    if (!studentId) return <div>Invalid student ID.</div>;    if (loading) return <div>Loading...</div>;    if (!student) return <div>Student not found.</div>;    return (      <main className="min-h-screen bg-gradient-to-br from-blue-50 via-white to-blue-100 p-6 flex items-center justify-center">        <div className="max-w-3xl w-full bg-white rounded-2xl shadow-lg p-8 text-blue-700">          <h1 className="text-3xl font-bold text-blue-700 mb-6">Edit Student</h1>          <StudentForm initialData={student} onSubmit={handleUpdate} />        </div>      </main>    );  } |  |

This code defines a client-side page in Next.js for editing a student’s details. When the page loads, it reads the student ID from the URL and fetches the corresponding student data from the backend API. While the data is being fetched, it shows a loading state, and if the ID is invalid or the student is not found, it displays appropriate messages. Once the data is loaded, it pre-fills a shared form component with the student’s existing information. When the user submits the form, the updated data is sent to the backend, and upon success, the page navigates back to the list of students. The page is styled with a centered card-like layout and uses a gradient background to match the overall design of the project.

Run the server: pnpm dev -p 4000

A screen shot of a computer

AI-generated content may be incorrect.

Go to: <http://localhost:4000/students>

A screenshot of a computer

AI-generated content may be incorrect.

# **Additional Work**

We should now have a Next.js application that runs on the client side and works with the student API to offer CRUD functionality. The pages are following Client Side Rendering (CSR).

The styling of the CRUD pages are minimal, can you update the styling to match the Home Page theme.