assignment # check-off # front-end # nextjs

Check-Off 7 — Next.js Foundations

School of Engineering and Technology, University of Washington Tacoma
TCSS 460 — Client/Server Programming, Spring 2026

Due Date

Sunday, May 17, 2026, 11:59 PM

Description

This is your introduction to Next.js — the React-based framework you will use for the rest of the quarter. You will work through the **introduction overview** plus **Chapters 1–5** of the official Next.js Learn dashboard course. Along the way you will scaffold a real project (`acme-dashboard`), add global and component styling, optimize fonts and images, build nested routes with shared layouts, and replace `<a>` tags with `<Link>` for client-side navigation. The check-off is less about typing the code than about being able to point at a file and explain — in Next.js's vocabulary — what the App Router is doing for you.

You will **stop after Chapter 5: Navigating Between Pages**. Chapter 6 introduces a database, which is out of scope for this check-off.

Learning Objectives

By completing this check-off, you will:

- Scaffold and run a Next.js application locally using the App Router
- Identify how files inside `app/` become URL routes (file-based routing)
- Distinguish a **root layout** from a **nested layout** and explain when each renders
- Apply global styles, Tailwind utilities, and conditional class names with `clsx`
- Use `next/font` and `next/image` and explain the optimization each one provides
- Use `<Link>` for client-side navigation, and `usePathname()` to highlight the active route

- Recognize a **Client Component** by its `'use client'` directive and explain why one is needed

Course Learning Objectives

This check-off contributes to the following course learning objectives:

- **LO 4:** Build interactive front-end applications using a component-based framework (e.g., React/Next.js)
- **LO 6:** Transfer object-oriented programming skills to a new language and ecosystem (e.g., Java to TypeScript)

It also supports these course outcomes:

- **Inquiry and Critical Thinking** — reading a Next.js project and reasoning about why a file lives where it lives
- **Communication/Self-Expression** — explaining the App Router's behavior to your checker in framework vocabulary, not "this part of the page"

Before You Begin

Ensure you have:

- Node.js **18.18+** installed (Node 20 LTS or 22 LTS recommended — same setup you used for the Express labs)
- `git` installed
- A code editor (VS Code is what most of you have used)
- Check-Off 6 — React Tutorial completed, or equivalent comfort with components, props, and state

Guides for This Check-Off

- Next.js Fundamentals — App Router, file-based routing, layouts, `<Link>`, server vs client components
- React Fundamentals — components, props, state, hooks (carry-over from Week 6)
- The official Next.js Learn dashboard course is the primary reference

---

Project Setup

Requirement 1: Read the Course Introduction

> **Reference**
>
> Next.js Learn — Dashboard App Overview

Read the **introduction page** before you start clicking through chapters. It tells you what you are building (a financial dashboard with a home page, login, and a protected `/dashboard` area), what prerequisites the course assumes, and which features of Next.js the course will cover.

You do not need to type any code on this page. Read it, then click into Chapter 1.

Requirements

Requirement 2: Complete Chapter 1 — Getting Started

> **Reference**
>
> Chapter 1: Getting Started

Scaffold the starter project using the command the chapter gives you, install dependencies, and run the development server. By the end you should:

- Have the `nextjs-dashboard` project cloned and installed
- See `npm run dev` (or `pnpm dev`) running at `http://localhost:3000`
- Be able to open `app/page.tsx` in your editor and identify it as the file that renders the home route (`/`)

> **npm vs pnpm**
>
> The course uses `pnpm` by default. **`npm` works fine** for this check-off — substitute `npm install` and `npm run dev` if you prefer. Either is accepted.

Requirement 3: Complete Chapter 2 — CSS Styling

Reference

Chapter 2: CSS Styling

Work through the chapter end-to-end. You will:

- Import the global stylesheet from `app/layout.tsx`
- See Tailwind CSS utility classes used directly in JSX
- Use a CSS Module for a scoped style
- Use the `clsx` utility to conditionally apply class names

When you finish, the home page should have its styled hero section visible.

Requirement 4: Complete Chapter 3 — Optimizing Fonts and Images

Reference

Chapter 3: Optimizing Fonts and Images

Work through the chapter end-to-end. You will:

- Add the `Inter` font via `next/font/google` and apply it to the root layout
- Add the `Lusitana` font and apply it to a specific element
- Replace a plain `<img>` tag with `next/image` for the hero illustration
- See responsive desktop and mobile variants of the hero image

When you finish, your home page should show the custom font and the hero image at the right dimensions for your viewport width.

Requirement 5: Complete Chapter 4 — Creating Layouts and Pages

Reference

Chapter 4: Creating Layouts and Pages

Work through the chapter end-to-end. You will:

- Create the `/dashboard` route using `app/dashboard/page.tsx`
- Create nested routes for `/dashboard/customers` and `/dashboard/invoices`
- Add a **dashboard layout** in `app/dashboard/layout.tsx` that wraps every page under `/dashboard` (sidenav stays in place)
- Understand how the **root layout** (`app/layout.tsx`) differs from a **nested layout**

When you finish, navigating between `/dashboard`, `/dashboard/customers`, and `/dashboard/invoices` should show the sidenav persisting and only the right-hand panel changing.

Requirement 6: Complete Chapter 5 — Navigating Between Pages

Reference

Chapter 5: Navigating Between Pages

Work through the chapter end-to-end. You will:

- Replace the sidenav's `<a>` tags with the `<Link>` component from `next/link`
- Observe that the page no longer fully reloads when you navigate between dashboard routes
- Add a `'use client'` directive to the `NavLinks` component
- Use `usePathname()` from `next/navigation` and `clsx` to highlight the active nav link

When you finish, clicking nav items should feel like a single-page app — no flash of a fresh page load — and the active link should be visually distinct.

Requirement 7: Be Ready to Explain the Core Concepts

The point of the check-off is the conversation, not the keystrokes. Before you find a checker, make sure you can answer each of the following in your own words while pointing at your code:

- **File-based routing:** Which file makes `/dashboard/invoices` work? What would you create to add a `/dashboard/reports` route?
- **Root vs. nested layout:** What is in `app/layout.tsx` that is **not** in `app/dashboard/layout.tsx`? When you navigate from `/dashboard` to `/dashboard/customers`, which layout re-renders and which one stays put?

- **`<Link>` vs `<a>`:** What changes in the browser when you click a `<Link>` compared to a plain `<a>` pointing at the same URL? Why does Next.js want you to use `<Link>` for internal navigation?
- **`next/image` vs `<img>`:** Name at least one thing `next/image` is doing for you that `<img>` is not.
- **`'use client'`:** Why does `NavLinks` need the `'use client'` directive? What would happen if you removed it?

Peer Check-Off

Find a classmate to verify your work.

> **For the Checker**
>
> The checker has 10 points to award. There is no rubric — use your judgment. Did they do the work? Can they explain it? Award points based on completeness and understanding — not perfection.

Check-Off Checklist

#	Verify	✓
1	Student runs `npm run dev` (or `pnpm dev`) and the app renders at `http://localhost:3000` with the styled hero section	☐
2	The hero image is rendered via `next/image` (student can open the source and point at the `<Image>` component)	☐
3	The page uses a custom font loaded via `next/font/google` — student can point at the import in `app/layout.tsx` (or wherever they put it)	☐
4	The `/dashboard` route renders, and so do `/dashboard/customers` and `/dashboard/invoices`	☐

#	Verify	✓
5	Navigating between dashboard routes keeps the sidenav in place — student can point at `app/dashboard/layout.tsx` and explain why	☐
6	Sidenav links use `<Link>` from `next/link`, not `<a>` tags (student opens the nav component to show this)	☐
7	The currently-active nav link is visually distinct — student can show the `usePathname()` call and the `clsx` (or equivalent) that styles it	☐
8	Student can point at the `'use client'` directive in the `NavLinks` component and explain in their own words why it is required	☐
9	Student can answer the **file-based routing** question: name the file that makes `/dashboard/invoices` work, and what file they would create to add a new `/dashboard/reports` route	☐
10	Student can answer the `<Link>` **vs** `<a>` question: name at least one concrete difference in browser behavior between the two	☐
11	Student can answer **one** of: what `next/image` does that `<img>` doesn't, **or** the difference between the root layout and the dashboard layout	☐

How to Submit

1. Complete the check-off requirements above.
2. Find a classmate to be your checker.
3. Demo your work — walk them through each item on the checklist.
4. The checker has 10 points to award. There is no rubric — use your judgment. Did they do the work? Can they explain it? Award points based on completeness and understanding — not perfection.
5. Both of you fill in the shared spreadsheet linked from the Canvas assignment: the student enters their name in Column A, the checker enters their name in Column B, and the checker enters the score in Column C.

Canvas — Check-Off 7: Next.js Foundations

Guide Reference

Guide / Resource	What It Covers
Next.js Learn — Dashboard App	The official course; chapters 1–5 are the focus of this check-off
Next.js Fundamentals (course guide)	App Router, file-based routing, layouts, `<Link>`, server vs client components
React Fundamentals (course guide)	Components, props, state, hooks — the React layer underneath Next.js

Gen AI & Learning: Using AI for This Check-Off

You are welcome to use AI coding assistants to ask "why did the tutorial do it this way?" or to debug an error — that is a great use of an AI tutor. You should **not** paste the chapter prompts into an AI, copy its output into your project, and call it done. Your checker will ask you to point at the lines you wrote and explain *why* — if you cannot, the check-off is not complete. The point of this check-off is the conversation, and the conversation only works if you wrote (or fully understand) the code.

This assignment is part of TCSS 460 — Client/Server Programming, School of Engineering and Technology, University of Washington Tacoma.