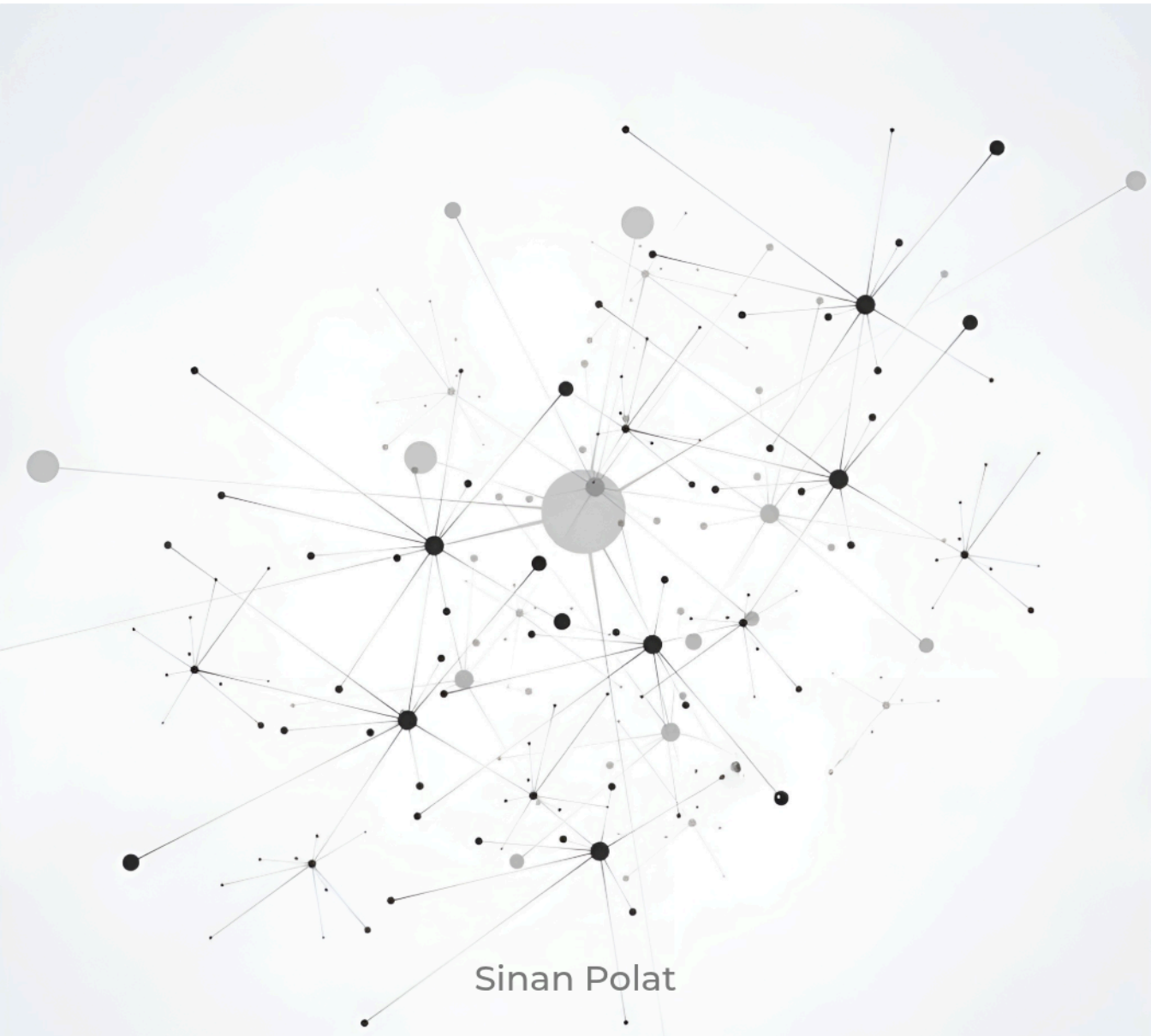
The Complete Guide

Solid Router



Sinan Polat

Solid Router: The Complete Guide

Sinan Polat

Solid Router: The Complete Guide by Sinan Polat

Table of Contents

CHAPTER 1

Introduction

Every modern web application needs navigation. Whether it is a small single-page app or a full-fledged platform with nested layouts, dynamic segments, and preloaded data, routing is the connective tissue that ties the user experience together. In the SolidJS world, that job is handled by Solid Router.

This book provides a focused and practical guide to Solid Router.

The content presented here is adapted from my larger work, *SolidJS: The Complete Guide*. In that book, routing is covered alongside the broader fundamentals of SolidJS, including reactivity, component design, state management, server-side rendering. Because routing is a distinct and essential subject, I have extracted the original chapter to serve as a standalone reference for developers who wish to study this topic in depth.

If you are already comfortable with SolidJS basics and want to dive straight into building real applications with routing, this book is for you. It is lean, focused, and ready to get you up and running.

If you are new to SolidJS altogether, this book assumes just enough knowledge of components and reactivity to keep you moving. For a complete learning journey, you can always consult the parent book, where routing is explained in context as part of the broader SolidJS ecosystem.

My hope is that this slim volume serves as a practical companion—something you can keep by your side while building apps, and a direct way to level up your routing skills without unnecessary detours.

By working through this book, you will learn how to:

- Correctly configure and integrate Solid Router into a SolidJS project.
- Define routes, navigate between them, and work with route parameters.
- Build and manage layouts, including nested and dynamic routes.
- Use navigation techniques such as links, redirects, and programmatic navigation.
- Access route-related data, query parameters, and transitions.
- Implement preloading and data loading strategies.
- Handle asynchronous fetching, form submissions, and validation.

- Apply performance optimization techniques to ensure fast, efficient routing in applications of any size.
- Develop a mental model of routing that scales from simple to complex applications.

Requirements

This book is not an introduction to SolidJS itself. It assumes familiarity with the following:

- SolidJS fundamentals: signals, stores, components, and JSX usage.
- Reactivity: understanding how SolidJS tracks dependencies and updates the DOM.
- TypeScript: while not strictly required, examples and best practices in this book use TypeScript for type safety and clarity.
- Basic web development concepts: including HTML, CSS, and JavaScript modules.

Beyond This Book

This volume addresses only routing. If you wish to acquire a complete and progressive understanding of SolidJS, I recommend consulting the parent book. That work contains:

- A thorough introduction to SolidJS fundamentals.
- Practical coverage of routing with Solid Router (the basis of this book).
- Advanced topics such as SolidStart for building full-stack applications.
- Numerous examples that build on one another to create a cohesive learning path.

In short, this book is designed as a concise reference for routing. For a comprehensive view of SolidJS development—from first principles through advanced application architecture—*SolidJS: The Complete Guide* is the resource to explore next.

Code Examples

All code examples used in this book are available on GitHub:

https://github.com/solid-courses/solidjs-the-complete-guide

Navigate to the `examples/ch34/` folder to find the examples for this book. You are encouraged to download, run, and experiment with the code as you follow along in the text.

In general, you are free to use the example code provided with this book in your own programs and documentation. Permission is not required unless you intend to reproduce a substantial portion of the code. For example, you may use several snippets in your programs without restriction, and you

may also cite the book or quote short examples when answering a question. However, selling or redistributing the examples requires prior permission, as does incorporating a significant amount of code into your product's documentation.

Contact and Feedback

To share feedback, suggestions, or corrections, please visit the repository:

https://github.com/solid-courses/solidjs-the-complete-guide

You can open an issue to provide your input. Your contributions will help improve both this book and the learning experience of future readers.

CHAPTER 34

Solid Router

A router helps manage application complexity by leveraging the browser's address bar. It interprets the URL to determine what should be displayed by mapping parts of the URL to route parameters and components.

To illustrate, consider an e-commerce application with four key sections: the home page, a catalog page, a product details page, and a shopping cart. These sections can be mapped to distinct URLs:

```
<Router>
  <Route path="/" component={Home} />
  <Route path="/products" component={Products} />
  <Route path="/product/:id" component={Product} />
  <Route path="/cart" component={Cart} />
</Router>
```

The router matches the current URL to the defined routes and renders the corresponding component:

- `/` loads the home page.
- `/products` displays a list of products.
- `/product/:id` shows details for a specific product, where `:id` represents a product's unique identifier.
- `/cart` presents the shopping cart, listing items the user has added.

With this setup, the application dynamically updates the displayed content based on the active URL. For instance, visiting `/product/42` loads the details for the product with the ID of `42`, while navigating to `/cart` brings up the shopping cart.

This routing mechanism enhances usability by ensuring each section of the application has a distinct, shareable URL. Users can bookmark pages, share links, and return to the same view.

Beyond determining which view to render, the router plays a crucial role in managing application state. It treats the active URL as the single source of truth, parsing it into structured data that influences the application's behavior.

To maintain consistency, the router listens for URL changes and responds appropriately—whether the user enters a URL manually, clicks a link, uses the browser's back and forward buttons, or triggers programmatic navigation. Handling all these navigation methods seamlessly ensures that the application remains predictable and behaves consistently as paths and query parameters change.

A router also handles cases where the requested path does not match any registered route. In such cases, it can display a fallback view, like a 404 page, or redirect users to a default path.

Setting Up Development Environment

Before diving into the details of Solid Router, let's first establish our development environment.

Solid Router can be used in client-only applications as well as full-stack projects. For examples that run entirely on the client side, we'll use a standard Vite-based setup. This will help us focus on the core features of Solid Router without involving server-side logic.

However, in most cases, Solid Router is used within SolidStart projects, Solid's official meta-framework that supports isomorphic rendering. SolidStart allows routes to be handled on both the server and client sides, enabling features like server-side rendering (SSR), API routes, and more.

To cover both scenarios, we'll set up environments for client-only and SolidStart projects. You can pick the approach that best matches your needs and stick with it as you work through the examples in this chapter. If you're not familiar with SolidStart or want to ensure you can confidently run the SolidStart examples, you'll find the next chapter especially useful—we cover the SolidStart framework in depth, complete with detailed explanations and practical examples.

Client-Only Development Environment

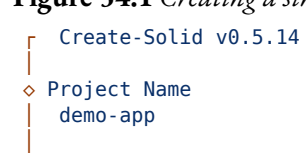
For this chapter, we're going to use readily available templates. Feel free to use your preferred package manager—whether it's `npm`, `yarn`, or `pnpm`. In this guide, we'll be using `pnpm`.

Start by creating a new Solid project using the following command:

```
pnpm create solid
```

This will launch an interactive prompt to guide you through the setup as shown in Figure 34.1.

Figure 34.1 *Creating a single-page Solid application using a template.*



```
◇ Is this a SolidStart project?
│  No
│
◇ Template
│  ts
│
◇ Use TypeScript?
│  Yes
│
◇ Project successfully created! 🎉
│
◇ To get started, run: ─┐
│                       │
│  cd demo-app          │
│  pnpm install         │
│  pnpm dev             │
│                       │
└───────────────────────┘
```

For a more detailed walkthrough on setting up a development environment from scratch, check out Chapter 2.

Now, move into your newly created project directory:

```
cd demo-app
```

Next, install the Solid Router library:

```
pnpm i @solidjs/router
```

Open the `App.tsx` file and replace its contents with the code from Listing 34.3.

```
import { render } from "solid-js/web";
import { Route, Router, RouteSectionProps } from "@solidjs/router";
import { Component } from "solid-js";

const Home: Component<RouteSectionProps> = () => {
  return (
    <div>
      <h1>Home</h1>
      <nav><a href="/about">About</a></nav>
    </div>
  )
};

const About: Component<RouteSectionProps> = () => {
  return (
    <div>
      <h1>About</h1>
      <nav><a href="/">Home</a></nav>
    </div>
  )
```

```
};

function App() {
  return (
    <Router>
      <Route path="/" component={Home} />
      <Route path="/about" component={About} />
    </Router>
  );
}

render(() => <App />, document.body);
```

This adds two components for two static paths, each with a link to navigate between them.

Start the development server with:

```
pnpm run dev
```

Open the URL provided in your terminal and use the links to navigate between pages. Everything should work as expected. Feel free to clean up files and folders as you like. We'll tweak this setup for client-side-only examples or add similar examples later, as needed.

SolidStart Development Environment

With the client-only setup complete, let's now configure a SolidStart project, which offers a more robust foundation for full-stack applications.

To get started, run the Solid project initializer:

```
pnpm create solid
```

The prompt will guide you through creating a SolidStart project, as in Figure 34.2.

Figure 34.2 *Creating a SolidStart application using a template.*

```
┌  Create-Solid v0.5.14
│
◇ Project Name
│  demo-app
│
◇ Is this a SolidStart project?
│  Yes
│
◇ Which template would you like to use?
│  bare
│
◇ Use TypeScript?
│  Yes
│
```

```
◇ Project successfully created! 🎉
│
◇ To get started, run: ─┐
│                       │
│  cd demo-app          │
│  pnpm install         │
│  pnpm dev             │
│                       │
└───────────────────────┘
```

We'll go deeper into SolidStart and learn how to set it up properly in its own chapter. For now, we just need a simple SolidStart app that uses Solid Router for navigation.

Installing Solid Router

Since SolidStart doesn't include a router by default, we need to install it explicitly, just as we did earlier:

```
pnpm i -D @solidjs/router
```

Next, we'll open the App.tsx file and replace its contents with the code from Listing 34.3 as we did earlier.

Finally, we can start the development server:

```
pnpm run dev
```

Open the URL shown in your terminal and try navigating between pages using the links. Feel free to tidy up the project files and structure them however you prefer.

Now that we have both client-only and SolidStart environments ready, let's dive into the core routing strategies used across different setups.

Routing Strategies

Routers can be categorized based on where they handle navigation: server-only, client-only, or isomorphic. Each approach has distinct trade-offs that affect performance, SEO, and user experience.

A server-only router processes every navigation request on the server. When a user navigates to a new page, the browser makes a request to the server, which returns a fully rendered HTML page. This approach, common in traditional multi-page applications (MPAs), ensures good SEO and works without JavaScript but can feel slower due to full-page reloads.

In contrast, a client-only router handles navigation entirely on the client side. It intercepts link clicks and updates the view without making full page requests, allowing for smooth transitions and fast navigation. However, since the initial page is loaded via JavaScript, it may require additional work for SEO and first-load performance optimization, such as pre-rendering or hydration techniques.

An isomorphic (or universal) router combines the best of both worlds. It renders the initial page on the server for fast first paint and SEO benefits, then switches to client-side navigation for subsequent interactions. This allows applications to achieve fast initial loads while maintaining a smooth, single-page app (SPA) experience.

Solid Router is designed to work in an isomorphic fashion when paired with SolidStart, enabling developers to choose whether specific routes should be pre-rendered on the server or dynamically handled on the client.

In the context of server-side rendering, a router takes on additional responsibilities to manage the server's response lifecycle effectively. It sits on the rendering path, where it intercepts incoming requests, parses the URL to extract relevant information such as route parameters and query strings, and evaluates the URL against defined routes to locate the corresponding handler component. Based on the handler's return value, the router then constructs and delivers the appropriate response to the client.

Another key benefit of using a router is the ability to optimize application performance through features like code splitting and lazy loading. By loading only the code necessary for the current view, the router reduces the application's initial load time and improves its responsiveness. This is especially valuable for larger applications with numerous routes.

Solid Router also introduces powerful utilities that enhance data fetching and state management:

Request Deduplication
: Prevents redundant requests to the server, reducing load.

Form Actions
: Enable server-side mutations directly through form submissions, improving interactivity while keeping logic centralized.

Solid Router also offers additional functionalities that enhance the overall developer experience:

Lazy Loading
: Defers loading of components and data until the corresponding route is accessed, reducing initial load times.

Pre-Fetching