Frontend System Design Essentials

Juntao Qiu

Frontend System Design Essentials

A practical guide to designing scalable, reliable, and maintainable frontend systems.

Juntao Qiu

This book is available at
https://leanpub.com/frontend-system-design-essentials

This version was published on 2025-12-23

Contents

Preface

I've always enjoyed learning new concepts, and more importantly, I want to see abstract ideas in action, in a tangible and practical way. Concise and beautiful abstractions only make sense to me *after* I've seen the details and understood them clearly in my head.

Before I graduated from university, I had the chance to work as an intern at a company. During that internship, my mentor Dong gave me a few tasks, and one of them was to implement a thread pool in Java.

Until that moment, thread pools were still a mystery to me—something I only knew from textbooks. I recognised the diagrams, had a rough idea of how they behaved, and had seen some pseudo-code. But watching Dong implement a thread pool using a simple `List` in Java completely changed how I understood programming concepts. It was one of those moments that reshaped how I learn.

Later, while preparing for system design interviews, I realised how difficult it is to find practical, hands-on explanations—especially in the frontend space. Many tutorials and whiteboard explanations make sense on the surface, but I often walked away feeling unsure, sometimes with even more questions than before. If I couldn't *see* how something worked in a real environment, I knew I didn't fully understand it.

That's the gap I want to fill with this book (or course, or series—whatever this eventually becomes). I want to teach these concepts through concrete examples: runnable, interactive code you can play with. Examples where you can add a `console.log`, change a line, observe a network request, and immediately understand what's happening.

I truly believe this kind of tangible experience is what helps you internalise concepts, explain them confidently in a system design interview, apply them correctly in your projects, and contribute to complex systems at scale.

A High-Level View of Frontend Work

If we simplify everything, most frontend work falls into two categories: **reading** data and **writing** data. Frontend and backend systems must stay in sync. You might change some data and need to see your updates reflected, or you might need to react to changes made by someone else.

To deliver a smooth user experience around these two tasks, we have several opportunities. Some improvements happen at build time, some at the infrastructure level, and others directly at runtime in our application code.

At build time, for example, we can minify JavaScript and CSS into smaller bundles to reduce transfer time. On top of that, we can configure the server to use gzip or other compression methods, add ETags to improve caching, and keep assets on the client for longer so we don't fetch them on every page load.

At the code level, we can optimise rendering. In React, that might mean memoising components with `useCallback` or `useMemo` to prevent unnecessary re-renders. Or we can preload data when the user hovers over an important element, like a user avatar.

All of this contributes to a smoother, less janky experience. But building applications like this isn't easy. It requires understanding many subtle but important concepts. Even in the AI era, where LLMs can write code or tests for us, knowing *what* to ask for still matters. The quality of your prompt—or even the request you send to an API—makes a huge difference. Models can only help you when you understand what you're doing. Otherwise, the best you'll produce is something mediocre.

Book Structure and Outline

This book is designed as a practical journey—from high-level architectural thinking to concrete, production-ready implementation. Each section builds on the previous one, using the same running example to show how real frontend systems evolve over time.

The content is organized into several modules, each with a clear focus:

- **The CCDAO Framework (Chapters 2–4)** This module introduces a lightweight thinking framework—**Collect information, Component**

structure, Data modeling, API design, Optimization strategies. You can treat it as a "fast-start mode" for frontend system design: a way to organize your thoughts quickly under pressure, whether you're in a design discussion, debugging a complex feature, or preparing for an interview.

- **Data Modeling (Chapters 5–9)** Here we focus on the foundation of most frontend complexity: data. You'll explore domain-driven thinking and why **normalization** matters on the client side. By treating your frontend store more like a relational database, you establish a **single source of truth** that makes updates predictable and scalable.
- **Data Fetching and Requests (Chapters 10–11)** This module addresses the realities of asynchronous systems. You'll look at request lifecycles, cancellation, debouncing, and pagination strategies, including the trade-offs between **offset-based** and **cursor-based** pagination.
- **Performance Optimization (Chapters 12–15)** With a real Express backend in place, we move beyond component-level optimizations. You'll introduce **Server-Side Rendering (SSR)** to improve initial load performance, then layer in **code splitting** and **prefetching** to reduce perceived latency and keep interactions fast.
- **Mutation and Real-Time Sync (Chapters 16–18)** This section focuses on keeping the UI responsive while data changes. You'll implement **optimistic updates** to eliminate unnecessary waiting, and explore real-time synchronization using **Server-Sent Events (SSE)** and **WebSockets**.
- **Productionization (Chapters 19–22)** The final module shifts attention to long-term reliability. Topics include **HTTP caching**, **error boundaries**, **accessibility-first design**, and **performance monitoring** with Core Web Vitals—concerns that often determine whether a system holds up in production.

How to Make the Most of This Book

This book is most effective when approached with a **hands-on mindset**. The goal is not just to understand the ideas, but to make them part of how you think and work.

1. **Work with the Running Project** The entire book is built around a Trello-style board application. Don't treat it as a static example—clone the starter repository, run it locally, and use it as your playground.

2. **Use the Git Checkpoints** Each major step in the book is captured as a Git tag. You can jump to any checkpoint to compare implementations, recover from mistakes, or inspect a complete solution for a specific chapter.
3. **Break Things on Purpose** Because the project uses Mock Service Worker (MSW) or a local Express server, it's safe to experiment. Add `console.logs`, tweak request logic, introduce bugs, and observe how the system responds. This is where many of the insights become concrete.
4. **Apply the Mindset Beyond the Book** Treat the **CCDAO framework** as a reusable mental checklist. Use it when designing new features at work, reviewing pull requests, or reasoning about unfamiliar code—not just when following the examples here.

Summary

Frontend system design may sound abstract, but every concept in this book maps directly to your day-to-day work. Whether you're fetching data, updating state, handling errors, or improving performance, the ideas we cover will help you build smoother, more reliable experiences.

My goal is simple: to help you truly *see* these concepts by applying them, experimenting with them, and making them part of your toolkit. Once you understand how they work in practice, you can use them confidently—whether you're building complex features, reviewing designs, or explaining your decisions in an interview.

Chapter 1 — Evolving from Components to Systems

Modern frontend development isn't just about rendering pixels. It's about coordinating data, state, and behaviour across components that interact almost like small services. In this chapter, we'll start from something deceptively simple – a user selector dropdown – and use it to show why **system design thinking** matters when you want to build reliable applications (and explain your decisions clearly in interviews).

System design thinking Looking beyond a single component to consider data flow, state, constraints, and trade-offs across the whole application.

Analyzing how simple requests turn complex

You get a ticket:

> "We need a dropdown to select a user when assigning a task."

It feels straightforward. You give it a two-hour estimate. But when you finally ship it, two days have passed. What happened? Let's unpack it.

Part I — How a small component grows

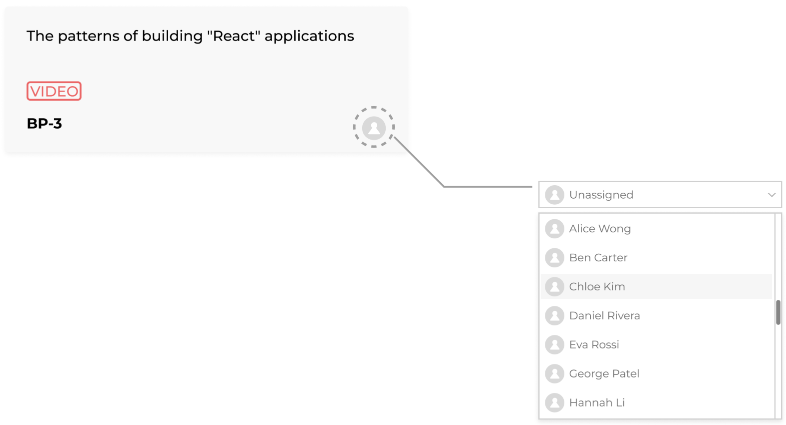

Figure 1. UserSelect component

Version 1 — The happy-path implementation

You might begin with the bare minimum: fetch some data and pass it into the dropdown.

```
const UserSelect = () => {
  const [users, setUsers] = useState<User[]>([]);

  useEffect(() => {
    fetch("/api/users")
      .then((r) => r.json())
      .then((data) => setUsers(data));
  }, []);

  return <Select options={users} defaultValue="Unassigned" />;
};
```

This version works when everything goes right: the network is fast, the API behaves, and the data set is small. It's the version you might write in a coding exercise, or as a first pass.

Real production environments rarely stay that perfect for long.

Version 2 — Handling loading and errors

On a slower connection, the gaps start to show. When you open the dropdown, nothing happens for a moment. If the network fails, the whole component breaks. To make it robust, you add loading and error state:

```
const [isLoading, setLoading] = useState(false);
const [error, setError] = useState<Error | null>(null);
```

You show a spinner when `isLoading` is true, and a friendly message when `error` is not null. The code isn't complicated, but the component now has more states to manage. Your "simple dropdown" is already tracking several transitions instead of just "data or no data".

Version 3 — Scaling to large datasets

Everything seems fine until a customer with 2,000 employees reports that opening the dropdown freezes the UI. Rendering large lists and filtering them on the client side becomes noticeably expensive.

To make it scale, you introduce pagination and adjust the API:

```
const [page, setPage] = useState(0);
fetch(`/api/users?page=${page}`);
```

This isn't just a local tweak anymore. You've changed the backend contract. A problem that started inside one React component now affects the API and possibly multiple teams.

Version 4 — Adding search

Next, customers ask for a search box so they can quickly find users.

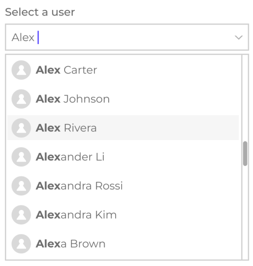

Figure 2. UserSelect component searchable

You add a controlled input and use its value in your request:

```
const [query, setQuery] = useState("");
fetch(`/api/users?query=${query}&page=${page}`);
```

Typing "Alice" now sends a request on every keypress. That's unnecessary load and can slow both the client and the server.

To fix it, you add debouncing.

Debounce Debouncing delays a function call until a certain time has passed without new input. It's commonly used to avoid sending too many requests while the user is typing.

```
const debouncedQuery = useDebounce(query, 300);
fetch(`/api/users?query=${debouncedQuery}&page=${page}`);
```

Now the request only fires after the user pauses typing for 300ms.

Version 5 — Avoiding race conditions

As the component becomes more dynamic, timing issues appear. QA finds a bug: they type “Alice”, then quickly change it to “Bob”. If the “Bob” request returns last, the UI is correct. But if the “Alice” response arrives later, it overwrites the results with stale data.

To avoid this, you cancel previous requests using `AbortController`:

```
useEffect(() => {
  const controller = new AbortController();
  const url = `/api/users?query=${debouncedQuery}&page=${page}`;

  fetch(url, { signal: controller.signal })
    .then((r) => r.json())
    .then((data) => setUsers(data))
    .catch((error) => {
      if (error.name !== "AbortError") {
        setError(error);
      }
    });

  return () => controller.abort();
}, [debouncedQuery, page]);
```

AbortController A browser API that lets you cancel in-flight fetch requests, which is essential for avoiding race conditions in dynamic UIs.

This ensures only the latest request is allowed to update the UI. You’ve introduced another moving part, but it’s necessary to handle real-world user behaviour.

Version 6 — Accessibility considerations

An accessibility review reveals another gap. Keyboard users can’t reliably navigate the dropdown, and screen readers don’t announce options correctly. Fixing it means:

- Adding appropriate ARIA attributes
- Handling focus and blur events

- Supporting keyboard navigation (ArrowUp, ArrowDown, Enter, Escape)

Accessibility rarely means "just add a few attributes". It often reshapes how the component is structured and how events are handled.

Version 7 — Internationalization

As the app expands to new regions, translations come into play. Labels need to be pulled from a translation system. Text might become longer, affecting layouts. Some languages require right-to-left layout support.

The same dropdown now has to respect locale, formatting, and text direction, on top of everything you've already added.

The real shape of the "simple dropdown"

If we pause here, the picture is clear. Our dropdown now deals with:

- Loading and error states
- Pagination and performance
- Search and debouncing
- Race condition prevention
- Accessibility
- Internationalization

What looked like a two-hour task quietly turned into something much larger. This isn't over-engineering. It's the natural shape of real-world requirements.

Zooming out to the system perspective

So far, we've only looked at one piece of UI. But in a real application, this user selector is only part of a bigger flow.

Changing the assignee of a task might:

- Update a card on a board and change its avatar
- Re-sort a list of tasks

- Append an entry to an activity log
- Trigger a notification
- Update a “My tasks” view on the assignee’s dashboard
- Push a real-time update to other connected clients

One tiny interaction ripples across many components and screens.

As soon as you zoom out, new questions appear:

- Should we update the UI immediately for responsiveness, or wait for server confirmation to avoid inconsistencies?
- If the request fails, do we roll back the UI or ask the user to retry?
- Where should we keep shared state: in a global store, React context, or local component state?
- How do we handle two users making conflicting updates at the same time?

These are not rare edge cases. They’re everyday concerns once you treat the frontend as a system with multiple sources of truth, asynchronous operations, and interconnected components.

Frontend as a system Once you zoom out, the frontend behaves like a distributed system: multiple views, shared state, network latency, partial failures, and concurrent updates.

This is where system design begins.

Developing system design thinking

Frontend has evolved from inline scripts to reusable components, and now to coordinated systems that must stay reliable under real pressure.

We deal with:

- Data modelling and shared state
- Data fetching and caching
- Data mutation and optimistic UI
- Performance and bundling strategies
- Accessibility and internationalization

- Error handling and resilience

None of these topics live in isolation. They shape each other and create trade-offs you need to reason about.

To make this easier to learn, this book is organised around a few core building blocks of frontend architecture:

- Data modelling and state management
- Data fetching
- Data mutation
- Performance optimisation
- Productionalisation (making things robust in real environments)

Optimisation strategies In this book, “optimisation” is not just about speed. It includes reducing complexity, improving reliability, and designing APIs and data flows that scale.

As you work through the chapters, you’ll build a mental model for how modern frontends are structured and how data moves through them. The goal is to help you reason about design decisions clearly – both in real projects and in interviews.

A note on CCDAO and the interview lens

Before we dive deep into each building block, the next two chapters take a slightly different angle.

If you’re preparing for system design interviews and don’t have time to absorb everything at once, it’s helpful to have a simple structure you can lean on under pressure. For that, we’ll use a lightweight framework called CCDAO.

CCDAO A practical framework for structuring frontend system design answers: Collect information, Component structure, Data modeling, API design, Optimisation strategies.

CCDAO is not the structure of this book, but it’s a useful lens for walking through open-ended questions. We’ll use two concrete examples and apply

CCDAO step by step, so you can see how to turn vague requirements into a clear, structured explanation.

Think of those chapters as a fast-start mode: a way to speak about frontend systems confidently, even before you've internalised all the details that follow.

Summary — The invisible work

Frontend complexity doesn't come from over-engineering. It comes from real-world conditions: slow and unreliable networks, large datasets, many users updating the same data, and the need to support different devices, languages, and accessibility requirements.

System design thinking gives you tools to understand these challenges, communicate your choices, and build software that holds up under pressure. When you learn to think in systems, you design more reliable products and explain your decisions more clearly.

In the rest of this book, we'll work from components up to systems, focusing on the key building blocks: modelling data, fetching it, mutating it, keeping it fast, and making it robust in production. Step by step, you'll see how all the pieces connect – and how to reason about them with confidence.

Chapter 2 — Introducing the CCDAO Framework

Preparing for a system design interview can feel daunting, even for experienced frontend engineers. Many developers spend most of their time refining features within a single product, not architecting systems from the ground up. That's why interview questions about designing scalable, maintainable, and performant applications can feel unfamiliar. But the gap isn't about intelligence or experience–it's about structure.

A structured framework gives you a way to reason through complex problems and communicate your thinking clearly. It ensures that you don't overlook key areas while also helping interviewers follow your logic. More importantly, it mirrors the way real frontend systems should be designed–methodically, with awareness of both technical trade-offs and user experience. That's where the **CCDAO** framework comes in.

CCDAO stands for **Collect Information, Component Structure, Data Modeling, API Design, and Optimization Strategies**. It's a practical, five-step framework you can use to approach any frontend system design challenge–whether in an interview or in your day-to-day work.

Collecting information for requirement clarity

Every strong design begins with understanding the problem. Before drawing diagrams or naming components, take a few minutes to clarify what you're building. Ask questions that reveal both the *functional* and *cross-functional* requirements.

Start with the **core functionality**. What exactly should the system do? What's the minimal set of features–often called the *steel thread*–that defines the essential user journey? Then move to **cross-functional needs**: performance goals, accessibility standards, security constraints, and expected scale. Will it handle thousands of concurrent users? Does it need to support real-time updates or offline access?

These clarifications shape every decision that follows. They help you focus on what matters most and demonstrate to interviewers that you're thinking beyond the happy path. In practice, developers who start by collecting information tend to make better architectural trade-offs later on.

When collecting information, focus first on the language of the domain, not the shape of the UI.

A "board app" that only talks about *columns* and *cards* is usually an oversimplification. Real systems quickly introduce richer concepts: users, filters, sprints, priorities, blockers, risk levels, tags, and more.

Your goal at this stage is not to design everything, but to extract enough domain vocabulary to have meaningful conversations. These terms will later influence your data model, APIs, and component boundaries.

This is where a **Domain-Driven Design** mindset helps: listen carefully to how the problem is described, reuse the same words in your design, and let the domain—not the UI—drive your structure.

Designing modular component structures

Once you understand the requirements, it's time to structure the interface. Imagine how the user interacts with the system and translate that into components. Even if you don't have a mockup, a quick sketch—on paper or mentally—is enough to identify the main building blocks.

Break the UI into logical parts: what belongs together, what can be reused, and how data should flow between components. Consider state ownership carefully—should a component manage its own state, or should it rely on a shared store or context? In interviews, describing these choices out loud helps the interviewer see how you reason about modularity and scalability.

In real projects, this step connects to architecture patterns like container-presentational separation or headless component design. Structuring components thoughtfully ensures flexibility as the system grows.

Modeling data for predictability and efficiency

After defining your component structure, you need to understand the data that powers it. What entities exist in your system, and how do they relate to each other? For example, in a task management app, you might have users, tasks, and boards, each with relationships such as “a user owns many tasks.”

Good data modeling makes state management predictable and efficient. Decide whether to **normalize** or **denormalize** your data. Normalized structures avoid duplication and simplify updates, while denormalized ones can be faster to render but harder to maintain. Explain the trade-offs clearly if you’re in an interview.

Think about how the data flows: where it’s fetched, cached, and updated. Does your system support pagination for large datasets? Should it work offline with local storage or IndexedDB? These details not only show depth of understanding but also reflect real-world challenges frontend systems face daily.

Designing stable API contracts

Data modeling naturally leads to how your frontend communicates with the backend. This is where **API Design** comes into play. Define how the client retrieves, updates, and synchronizes data.

Consider whether the application would benefit more from **REST** or **GraphQL**. REST is simple and well-understood, but GraphQL allows for more flexibility in querying exactly what the UI needs. If the product requires real-time updates, discuss technologies like **WebSockets** or **Server-Sent Events** and how they integrate into your architecture.

Security is another critical aspect. Mention how you’ll handle authentication and authorization–using tokens like JWT or OAuth–and how you’ll protect the frontend from common vulnerabilities such as XSS or CSRF. Even brief comments on these areas show maturity and awareness of full-stack concerns.

Implementing optimization strategies early

Finally, you reach the stage where performance, resilience, and user experience come together. Optimization isn’t an afterthought–it’s an ongoing mindset that runs through the entire design process.

Think about **performance** at multiple levels. On the client side, techniques like **code splitting**, **lazy loading**, and **server-side rendering** can drastically improve load times. On the network level, caching strategies–both client-side and server-side–reduce unnecessary requests. For large datasets, consider pagination or virtualization to keep the UI responsive.

Beyond raw performance, focus on **resilience and user experience**. How does your system handle network errors or server downtime? Are there loading states, skeleton screens, or retries in place? Accessibility and internationalization should also be treated as optimization problems: they expand your system's reach and usability.

In interviews, this is often where you can differentiate yourself by showing you've thought beyond functionality–considering how real users experience the application.

Applying CCDAO in an Interview

In a 45-minute system design interview, you won't have time to cover every detail. The goal is to demonstrate clarity of thought. Spend the first few minutes collecting information and clarifying requirements. Then move through component structure and data modeling, explaining how your design scales with complexity. Use the remaining time to discuss API design and optimization, showing awareness of performance and failure handling.

The exact time distribution doesn't matter as much as the flow. A good interview feels like a conversation guided by this framework, where you and the interviewer explore trade-offs naturally rather than following a rigid checklist.

Why It Matters Beyond Interviews

The CCDAO framework isn't just for interviews–it's a way of thinking that mirrors how robust frontend systems are built in practice. Each step represents a discipline that teams grapple with daily: gathering requirements, designing modular components, modeling data efficiently, defining stable APIs, and optimizing for performance and reliability.

If you invest time to study the concepts in this book–the patterns, trade-offs, and real-world examples–you'll go far beyond interview preparation. You'll develop the instincts of a true frontend architect: someone who can reason about complexity, communicate clearly, and design systems that stand the test of scale and time.

Before we move on, the next two chapters will show CCDAO in action. We'll walk through two concrete component designs step by step, applying the framework in real scenarios so you can see how each part translates into actual decisions in code.

Chapter 3 — Applying CCDAO: Designing a Typeahead Search Box

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

1. Collect Information

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

2. Component Structure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

3. Data Modeling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

4. API Design

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

5. Optimization Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Closing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Chapter 4 — Applying CCDAO: Designing a Scalable Feed List

1. Collect Information

2. Component Structure

3. Data Modeling

4. API Design

5. Optimization Strategies

Closing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Chapter 5 — Data Modeling: Understanding the Domain and the UI

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Defining the domain clearly through business rules

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Example 1 — Chat Application

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Example 2 — Online Course Platform

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Example 3 — Board Application

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Designing models for specific UI consumption patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

From Domain to Design

Chapter 6 — Case Study: Implementing Sidebar Entitlements

Step 1 — The Starting Point: Logic in the UI

Step 2 — When the Rules Multiply

Step 3 — Moving Business Logic to the Backend

Step 4 — Where Frontend Logic Still Belongs

Step 5 — Tailoring APIs with GraphQL or a BFF

Lessons from the Sidebar

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Chapter 7 — Setting Up the Project Environment

Why We Use a Starter Project

Prerequisites

Getting the Project Running

Useful Scripts

Configuring the Mock API with MSW

The service worker

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Mock Endpoints Provided

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

`GET /api/users`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

`GET /api/board/:id?q=<text>`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

`GET /api/cards/:id`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Data Shapes You'll Work With

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Board Payload

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Users

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Project Structure (High-Level)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Troubleshooting

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

1. “unsupported MIME type ‘text/html’ for mockServiceWorker.js”

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

2. 404 errors for `/api/*` during development

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

3. Port conflicts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

What Comes Next

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Chapter 8 — Implementing Data Normalization

Why Normalisation Matters

A Real Example: Inconsistent User Data

Establishing a consistent source of truth

Transforming nested board payloads into flat tables

What this function achieves

Example output

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Holding the normalized store in React Context

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

The provider

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Ingesting data on load

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Updating a user

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Hydrating data during the rendering process

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Chapter 9 — Drawing Inspiration from Backend Databases

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

A Familiar Pattern From Databases

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

How Databases Reconstruct Data

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

The Frontend Equivalent

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Why This Matters

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

A Good Place to Pause

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Chapter 10 — Managing Requests and Data Fetching

Showing Real Assignees in the UI

Extending the Backend API

Updating the Board on Selection

Understanding Race Conditions in Search

Fixing Race Conditions with Request Cancellation

Reducing Request Volume with Debouncing and Throttling

Summary

Chapter 11 — Implementing Pagination Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Demonstrating Pagination in `UserSelect`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

The Pagination Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

A Simple Users Table

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Offset Pagination

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Cursor Pagination

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Choosing the Right Strategy for `UserSelect`

UI Patterns for Pagination

1. Numbered Pagination

2. Infinite Scroll

3. Infinite Loading

Summary

Chapter 12 — Migrating to an Express Backend

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

A Brief Introduction to Express

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Setting up a standalone mock API server

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Proxying Frontend Requests to The Express Server

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Chapter 13 — Implementing Server-Side Rendering

Rendering strategies

How SSR works

Coordinating hydration between server and client

Creating the client entry (hydration)

Creating the server entry (rendering on the server)

Creating separate entry points

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Configuring Vite to build dual entries

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Generating HTML on the server

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Bringing it together in the Express SSR route

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

1. Fetch initial data

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

2. Begin streaming HTML

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

3. Send the HTML shell immediately

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

4. Pipe the stream into the response

5. When rendering completes, inject data and scripts

Why consistency matters

From SSR to SPA behaviour

Summary

Case Study — Understanding Bundlers Through Code Splitting

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

What Problem Are We Solving?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

What a Bundler Actually Does

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Tree Shaking: Removing Unused Exports

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Where We Are Now

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Code Splitting: Deferring Code Until It's Needed

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

A Subtle Caveat: Tree Shaking Stops at Dynamic Boundaries

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

What This Case Study Teaches Us

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Chapter 14 – Implementing Code Splitting and Lazy Loading

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Why Lazy Loading Helps

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

The Building Blocks of Lazy Loading

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Dynamic import

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

React.lazy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Suspense

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Implementing Lazy Loading in Our Board Application

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Step 1: Extract the component

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Step 2: Lazy load the component

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Step 3: Wrap with Suspense

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Step 4: Coordinate SSR and hydration using `useHydrated`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Another Example: Lazy Loading the List View

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Step 1: Move `ListView` into its own file

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Step 2: Convert the import to `React.lazy`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Step 3: Wrap the lazy component with `Suspense`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Understanding the Build Output After Code Splitting

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

What Happens at Runtime

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

How Code Splitting Changes the Bundle

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Chapter 15 – Implementing Data Prefetching

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

How Prefetch Works Conceptually

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

The Prefetch Function Inside QueryProvider

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Exposing Prefetch Through a Hook

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Prefetching Users in Card.tsx

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Explaining the Cache Key

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

The Fetch Function

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Triggering Prefetch on Hover

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

How Prefetch and useQuery Work Together

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

The Full Runtime Experience

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

When Prefetch Helps

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

When Prefetch Doesn’t Help

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Chapter 16 – Implementing Optimistic Updates

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Creating a New Card (Server-First)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Introducing Optimistic Updates

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Adding a Card Optimistically

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Step-by-step

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Deleting a Card Optimistically

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Things to Keep in Mind

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Chapter 17 – Exploring Real-time Update Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Polling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Server-Sent Events (SSE)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

WebSockets

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Scaling Considerations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Chapter 18 – Implementing Real-time Updates with SSE

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

The pub-sub pattern

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Using the pub-sub pattern on the server

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Using the event emitter with SSE

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Emitting events from the update API

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Frontend: consuming SSE with EventSource

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Demonstration and behaviour

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Case Study — Implementing WebSockets for Real-time Updates

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Server Side — Setting Up WebSockets

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Installing the WebSocket Library

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Tracking Connections per Board

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Attaching the WebSocket Server

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Handling WebSocket Connections

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Emitting a Domain Event

Broadcasting to WebSocket Clients

Client Side — Receiving WebSocket Events

Encapsulating WebSocket Logic in a Hook

Wiring It into the Board

Observing the Behaviour

A Real-World Bug: Duplicate Cards

Making the State Update Idempotent

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Chapter 19 — Optimizing Performance with HTTP Caching

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Why caching matters

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Cache-Control — how long a response stays fresh

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Detailed breakdown: What does Cache-Control actually include?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Using stale-while-revalidate

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

ETag — detecting whether content changed

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Server-side: Returning an ETag with your response

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Browser's next request:

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Server logic: respond quickly if no change

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Why ETag matters

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Last-Modified — timestamp-based validation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Server-side example

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Browser request:

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Server-side validation:

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

When Last-Modified works well

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

ETag vs Last-Modified

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Vary — preventing incorrect caching

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Real-world problem: search queries

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Correct implementation with Express:

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Real-world scenario: user-specific data

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Benefits of Vary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Putting it all together: a realistic request flow

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Initial board load

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Subsequent load within the cache window

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Load after the cache expires

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

A write operation changes the resource

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

The next read reflects the change

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Chapter 20 — Handling Runtime Errors in React

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

A quick comparison to `try/catch`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Understanding Error Boundaries

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Wrapping a fragile component

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Cascading error boundaries in a column

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Application-level fallback

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Summary

Chapter 21 — Designing for Accessibility

Why we need to consider accessibility at all

Accessibility-first system design principles

Semantic HTML as a foundation

Progressive enhancement

Keyboard-first interaction

Accessibility checks as part of the testing architecture

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Why test accessibility in unit tests

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Setting up axe-jest

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Writing accessibility assertions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Semantic HTML in real components

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Refactoring the Card component

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Managing focus after destructive actions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Keyboard navigation as a system concern

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

A more complex example: accessible drag and drop

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Exposing the action through the context menu

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Announcing the new position

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

My typical accessibility workflow

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Quick feedback with Lighthouse and Chrome DevTools

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Enforcing guarantees with axe-based unit tests

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Preventing regressions with ESLint rules

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Validating real interactions manually

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Why this layered approach works

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Chapter 22 — Implementing Performance Monitoring

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Why performance monitoring matters

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Understanding bundle size in practice

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Visualising the bundle

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Making performance limits explicit with budgets

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Defining a performance budget

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Enforcing the budget after build

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Measuring real user experience with Web Vitals

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Collecting Web Vitals

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Performance as a continuous system concern

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontend-system-design-essentials.

The Architectural Roadmap: From Components to Systems

Module 1: The System Design Mindset (Chapters 1–4)

Module 2: Building the Data Foundation (Chapters 5–9)

Module 3: Managing Data Flow and Connectivity (Chapters 10–12)

Module 4: Rendering and Performance (Chapters 13–15)

Module 5: Mutations, Real-Time Sync, and Resilience (Chapters 16–22)

Designing System Step By Step