1 Developer's guide to the React Ecosystem

This chapter covers

- Understanding the concept of React mastery
- Navigating the React ecosystem
- Introducing the React technology stack
- Creating a proper React stack

Greetings, and welcome to *React in Depth*, an essential travel companion for developers who are ready to deepen their expertise and keep pace with the dynamic React community. As you embark on this developmental odyssey, remember that mastering React is more than understanding the basics; it's also about embracing a universe of advanced methodologies, best practices, and evolving tools.

In this inaugural chapter, we chart the territory ahead. I'll introduce the concept of React mastery and guide you through the expansive React ecosystem. You'll learn about the essential components that make up the React technology stack and get a glimpse of how these elements interact to form effective solutions.

Although this chapter sets the stage for understanding, the rest of the book will take you on a deeper dive into selecting and crafting the right technologies and libraries for your projects. By blending theoretical knowledge with practical insights, I aim to equip you with the foundational understanding necessary to navigate the complex landscape of modern web development as a proficient React developer.

1.1 Navigating the React mastery journey

This book assumes that you have a solid understanding of React and are comfortable building simple applications. To explain in very simple terms, you have progressed from A to B in figure 1.1, and this book will take you from B to C. As you can see in the figure, you have a lot more to learn in this segment of the journey.

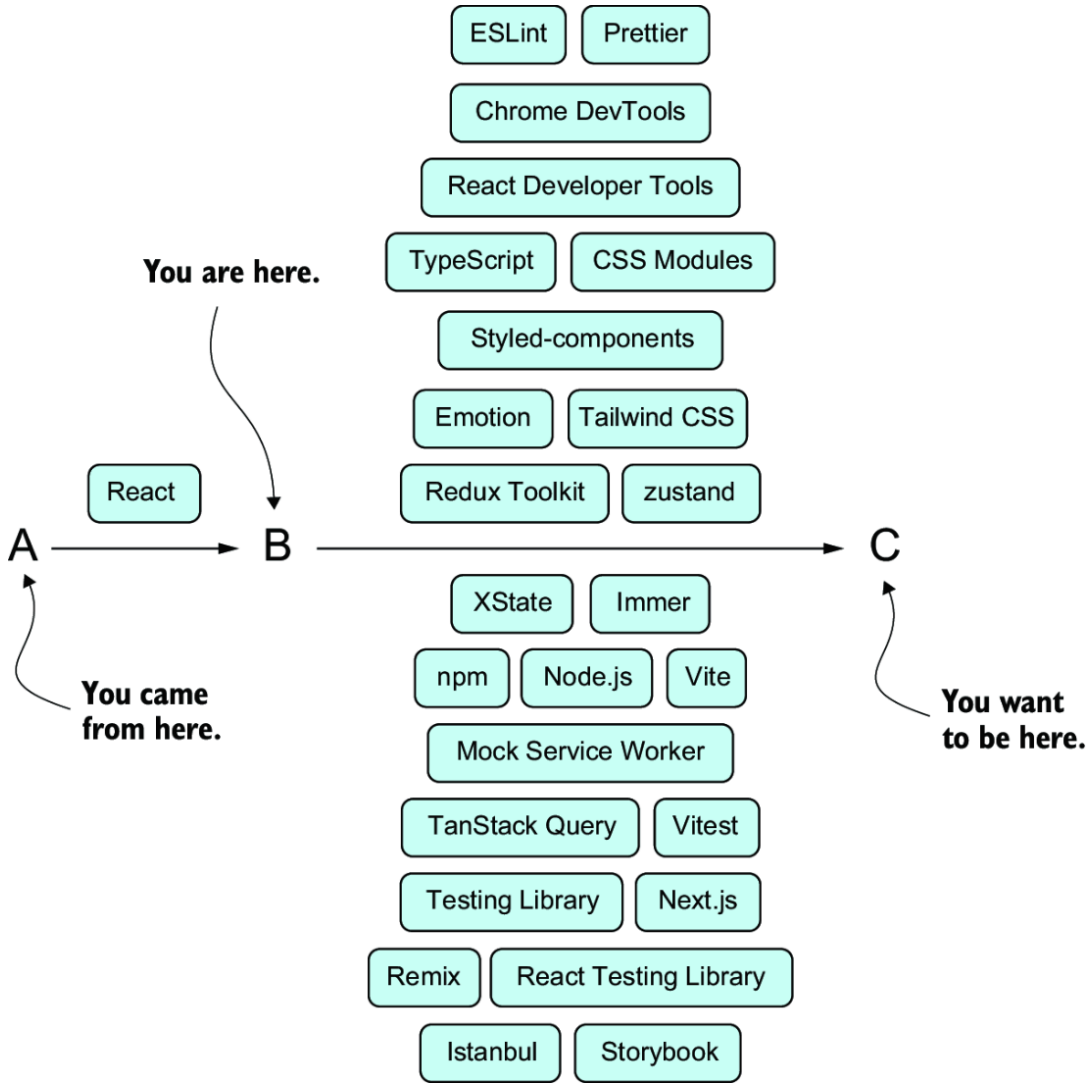

Figure 1.1 You knew how to write static websites using HTML, CSS, and JavaScript. And then you "just" had to learn React to get to your current level. Now you have to learn ESLint, Prettier, Chrome DevTools, React Developer Tools, TypeScript, CSS Modules, styled-components, Emotion, Tailwind CSS, Redux Toolkit, Zustand, XState, Immer, npm, Node.js, Vite, Mock Service Worker, TanStack Query, Vitest, Testing Library, Next.JS, Remix, React Testing Library, Storybook, and Istanbul. That doesn't sound too daunting, does it?

When you arrive at point B, you're probably capable of getting a good job as a junior or midlevel React developer and might even be able to make it to the senior level from there. But to kick-start your advancement and navigate your way forward quicker, let me introduce you to what I call React mastery.

React mastery is more than just writing code that works. It involves understanding best practices, designing applications for scalability and maintainability, and working effectively in a team environment. It requires a deep understanding of React and its ecosystem, as well as knowledge of the latest tools and techniques for building modern web applications.

At its core, React mastery is about building applications that are easy to understand, easy to maintain, and easy to extend, which means writing clean, modular code that can be easily tested and refactored. It means designing applications that can scale to meet the needs of a growing user base without sacrificing performance or stability. It means working collaboratively with other developers to build applications that meet the needs of stakeholders and users alike.

To become a React master, you need to have a solid understanding of React's core concepts and be comfortable working with a wide range of libraries and tools. You should be familiar with common design patterns and architectural principles, and you should be able to apply them to real-world problems. You should also be able to work effectively in a team environment, using version-control systems and agile development methodologies.

This book will teach you all these things. When you've finished it, you will be equipped to build, maintain, and collaborate on high-quality, scalable React applications for organizations of all sizes in all domains.

This opening chapter will dive a bit deeper into the *raison d'être* of this book: why I believe this book needs to exist and why you need to read it. Then we will explore the concepts of the *ecosystem* and *technology*

stack—two topics that are very important when it comes to architecting React applications. We'll devote most of this chapter to discussing the ecosystem, how to navigate it, how to apply it, and how to understand any new technology that will inevitably spring to life even after this book has been set in stone (or paper, probably; I don't think we'll publish it on stone tablets).

In the rest of this book, we will explore the key concepts and tools that you need to become a React master. We will cover topics such as tooling, strong typing, data management, remote data access, unit testing, and website frameworks. We will also provide practical examples and real-world simulations to help you apply these concepts in your own work. So let's dive in and get on the road to React mastery!

1.2 Why a book on React mastery?

The truth is, many resources are available for learning React, from great books and online tutorials to official documentation. But few of these resources cover the practical aspects of building large-scale applications with React.

React in Depth is designed to fill this gap by providing a comprehensive guide. Whether you are a frontend developer looking to improve your React skills or a full-stack developer building complex applications, this book has something to offer.

Figure 1.2 shows the crucial knowledge areas that create a well-rounded React developer. What's unique about this book is that it addresses all these areas.

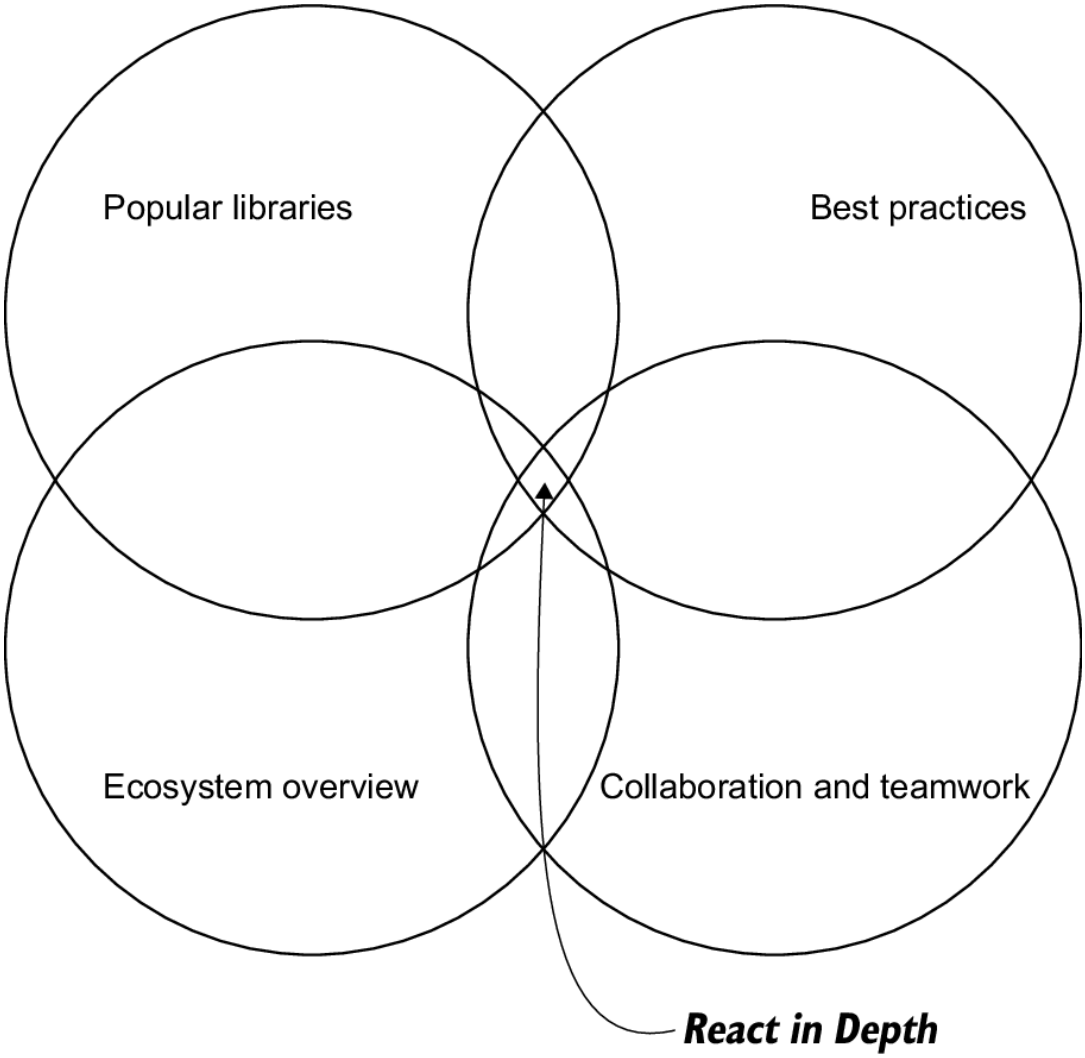

Figure 1.2 This book fits right in the intersection of popular libraries, best practices, ecosystem overview, and collaboration and teamwork.

One of the key goals of this book is to help you become a more efficient and effective React developer. We will cover a range of topics that are essential for building high-quality React applications, from data management to testing to website frameworks. By mastering these topics, you will be able to build applications that are more scalable, maintainable, and robust.

Another goal of this book is to provide practical guidance for working in a team environment. Building large-scale React applications requires collaboration with other developers, designers, and stakeholders. We will cover best practices for coding guidelines and developer tooling often used on large teams, as well as introduce TypeScript, a new flavor of JavaScript that's used by more and more teams.

Finally, this book aims to provide a road map for ongoing learning and professional development. The React ecosystem is constantly evolving, with new tools and techniques emerging all the time. I will provide guidance on staying up to date with the latest developments in React and improving your skills over time. Whether you are just starting with React at an advanced level or are already an experienced developer, this book will help you take your skills to the next level.

1.3 How does this book teach React mastery?

The scope of this book is broad, covering a range of topics related to React development, including libraries commonly used with React, data management, remote data, unit testing, and website frameworks. I'll cover each of these topics in detail, giving you the knowledge and skills you need to tackle real-world projects.

The focus of this book is on practical, hands-on development. Although I'll provide some background and theory on each topic, my main goal is to help you learn by doing. Each chapter includes a series of examples, and the book concludes with three complex projects that allow you to apply what you've learned and build your own React applications from scratch.

Throughout the book, I'll also emphasize best practices and common pitfalls to avoid. I want you to come away with not just a deeper understanding of React but also with the ability to build high-quality, maintainable applications that can stand up to the demands of real-world development. You can see your journey ahead in figure 1.3.

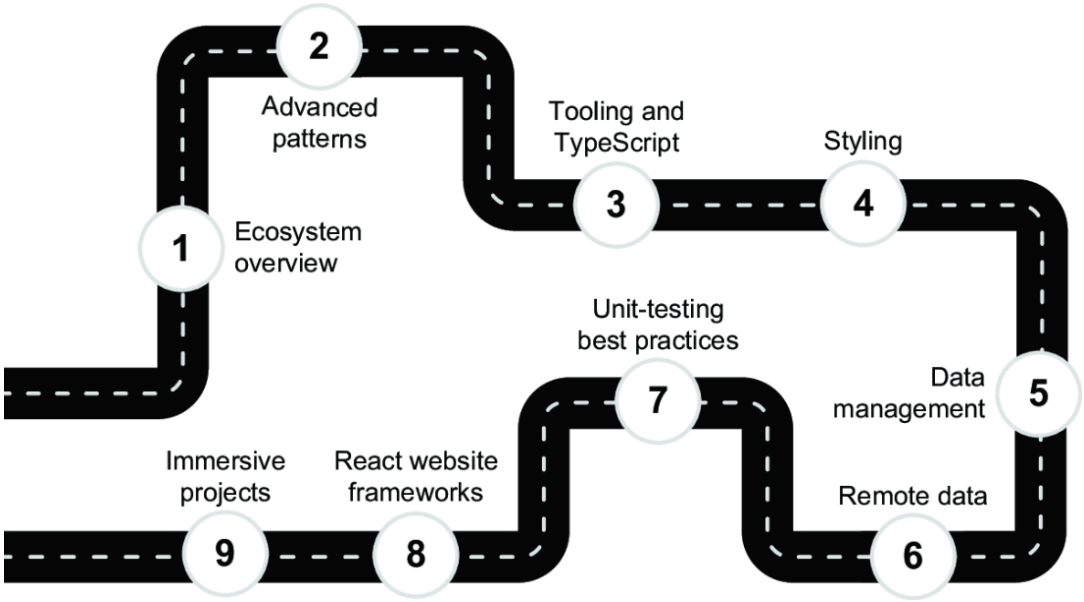

Figure 1.3 Your journey to React mastery starts with the first step. These steps represent nine realms of React that will be covered in this book, with one or two chapters dedicated to each.

Overall, this book is designed to help you become a master in the field of React development, equipped with the skills and knowledge you need to tackle any project with confidence.

1.4 The React ecosystem

The React ecosystem is a vast collection of libraries and tools built around the React library. As the popularity of React has grown, so has the number of tools and libraries that developers can use to enhance their workflows and improve the performance and functionality of their React applications.

One of the most significant advantages of using React is the sheer number of libraries available for it. These libraries cover a wide range of use cases, from data management and routing to animation and testing. Many of these libraries have become essential parts of the React developer's toolkit, and understanding how they work and how to use them effectively is a critical part of modern-day React development.

In this book, we will explore many of the most popular and useful libraries and tools in the React ecosystem. The book covers libraries for everything from styling to state management.

It’s worth noting that not all libraries in the React ecosystem are created equal, and not all of them are necessary for every project. For those reasons, we will focus on the most commonly used libraries and tools, providing guidance on how to evaluate and choose the right libraries for your specific project needs.

By the end of this book, you will have a solid understanding of the React ecosystem and the libraries and tools available within it. You will be able to confidently choose the right libraries for your projects, and you will have the skills and knowledge necessary to build complex, high-performance React applications.

1.4.1 What’s in the ecosystem?

To give you an idea of the enormity of the current React ecosystem, take a look at figure 1.4, which lists more than 150 tools and libraries currently being used with React. The tools are sorted into three main groups, depending on what they do and where they are used:

Figure 1.4 The React ecosystem is split into three main groups (UI, architecture, and productivity) and comprises hundreds of technologies, tools, and libraries. If you found the amount of technologies displayed in figure 1.1 daunting, this figure might break you.

- *User interface* —Includes things like UI libraries and animation tools
- *Architecture* —Includes data management tools, authentication libraries, and many others
- *Productivity*—Covers build tools, testing libraries, and more

The React ecosystem is fairly complex, with many tools, libraries, and technologies involved. Not all the technologies are specific to React, but many are. In this book, we'll at least partially cover the technologies highlighted in figure 1.5. I chose these technologies because they're popular and stable, representing either the diversity or flexibility within their categories. We'll get into more detail about how the technologies are located within their categories in each chapter.

You'll notice in figure 1.5 that most of the focus is on the productivity group because these tools are the most essential and also the most diverse. As an example, you can use any UI library from the UI library category to achieve the same result, but not all build tools are the same; neither are they interchangeable.

A lot of technologies are connected to an existing brand. This connection happens when some company decides to open up its internal tools or libraries to the public by open sourcing them. This situation is fairly common. Adobe appears three times in figure 1.4, for example, because React Spectrum is a UI library, React Aria is an accessibility toolbox, and React Stately is a state flow library. All three are contained with the overall React Spectrum architecture, and all are created, maintained, and (mostly) controlled by Adobe. In the same way, you can find libraries created by Airbnb, Facebook, Microsoft, Amazon, Google, Twitter, Palantir, IBM, Hewlett-Packard, Pinterest, and quite a few others. The following sections discuss each group and list all the technologies included in figure 1.4.

USER INTERFACE

This group is related to visual elements used in React applications. These elements are ready-to-use component collections (UI libraries), tools to help you write CSS in your React components more easily (CSS-in-JS), libraries to display complex data in beautiful ways using charts or interactive diagrams (data visualization), and packages to help you create animations and transitions (animation and effects):

- *UI libraries* —Tailwind CSS, Material UI, Ant Design, Chakra UI, Blueprint, Fluent, Semantic UI, Circuit UI, Headless UI, Grommet,

Evergreen, Rebass, PrimeReact, React Spectrum, and Gestalt

- *CSS-in-JS*—Emotion, CSS Modules, styled-components, JSS, styled-JSX, Linaria, PostCSS-JS, Theme UI, Styled System, Stitches, vanilla-extract, Twin, nano-css, Fela, and Astroturf
- *Data visualization*—D3, Victory, Recharts, Nivo, Chart.js, Semiotic, Frappe Charts, react-vis, FusionCharts, ApexCharts, and visx

Figure 1.5 The React ecosystem with all the technologies not covered in this book removed. As you can see, this book aims for breadth, not depth, so it covers a bit of everything, with most of the focus on the productivity group.

- *Animation and effects* —React Spring, Framer Motion, Remotion, React Move, GreenSock Animation Platform, Popmotion, React AOS, react-tsparticles, Lottie, and React AnimaKit

ARCHITECTURE

The architecture group contains mostly technologies related to the overall structure of your application or complex parts within the application. These logic-based packages help with managing data, controlling the flow of the application, working with remote data, managing complex forms, handling authentication and authorization, or ensuring accessibility and inclusivity:

- *Data management* —Redux (and RTK), zustand, Immer, MobX, Jotai, Recoil, Akita, Elf, Rematch, Hookstate, react-easy-state, Unstated, RxJS, Effector, Valtio, Easy Peasy, Kea, storeon, nanostores, PouchDB, Firebase, Supabase, Synergies, Flux, and Fluxible
- *State flow* —XState, Robot, React Stately, Overmind, useStateMachine, Zag, Little State Machine, and React Transition State
- *Remote data* —TanStack Query, SWR, Relay, Apollo Client, React-Fetching-Library, Redux Saga, Appsync, Axios, GraphQL, FeathersJS, Swagger, and GraphQL-Tools
- *Authentication and authorization* —JWT, Firebase Authentication, OAuth2, AWS Amplify, Auth0, Okta, Passport.js, OpenID, AWS Cognito, Keycloak, and React Auth Kit
- *Accessibility* —React Aria, axe-core, Reakit, and Downshift
- *Form management* —Formik, Yup, React Hook Form, React Final Form, and React-JSONSchema

PRODUCTIVITY

This final group is not directly included in the actual application; it's related to creating, working with, or validating the application. The group includes libraries that create the basic setup and bundling, run React on the server, ensure a smooth developer experience across even large teams, test applications automatically, analyze and monitor application health, and document the application both internally and externally:

- *Build and bundle* —TypeScript, Create React App, webpack, Vite, esbuild, SWC, react-hot-loader, react-loadable, Loadable Components, Babel, Parcel, Snowpack, Rollup, Terser, TSdx, Neutrino, React Next Boilerplate, Rekit, Sucrase, npm, and Node.js
- *Server-side rendering* —Remix, Next.js, Redwood.JS, astro.build, Nest.js, React Helmet, React Router, Razzle, Qwik, Blitz, and After.js
- *Testing libraries* —Testing Library, Jest, Mock Service Worker, React Testing Library, Vitest, Enzyme, Mocha, Chai, Cypress, Sinon, Puppeteer, Playwright, Karma, Jasmine, and Selenium
- *Developer experience* —ESLint, Chrome DevTools, Prettier, Redux DevTools, React Developer Tools, typescript-eslint, and Editorconfig
- *Analytics and monitoring* —Istanbul, React Performance Devtool, and Bugsnag
- *Documentation* —Storybook, Docz, Styleguidist, Docusaurus, and Pitsby

1.4.2 Navigating the ecosystem

Remember that you don't need to know every single item in the ecosystem. Nobody does, not even me. But it's a very good idea to know about all the categories and their purposes and to know at least a few technologies within each category.

You will never need something from every category on every project. But when you do come across that new project that happens to need something special, it's a big help to at least have an idea about the landscape in advance.

It's also a good idea to be able to use the ecosystem in figure 1.4 in reverse. When a new technology arises (which is bound to happen a few seconds after this book has been printed and then every week going forward), you should be able to look at the new piece of tech, quickly determine what it does, and mentally place it in the ecosystem diagram.

Another good thing to keep in mind is that technologies within a single category aren't necessarily equivalent alternatives. Often, the technologies are used together and augment one another. Some projects

might use several technologies within a given category. You might use Jest, React Testing Library, Puppeteer, and Karma in a single project, and all of them are in the testing category.
At other times, the technologies and libraries are direct competitors. It wouldn't make sense to include both Material UI and Ant Design, for example, as they are two completely different and mostly overlapping UI libraries that for the most part have the same responsibility in an application. Then again, on a large project, you might have different libraries used in different parts; those libraries might not work together directly, but at least they don't collide too terribly when they're used separately.

The manner in which libraries group together or function separately is also a critical aspect to consider when evaluating new technologies. A new technology might replace an existing item, such as zustand replacing Redux in the data management category (if you want to use zustand instead of Redux). But it might be a new augmentation to the existing libraries, such as Immer, that can be used with zustand or Redux (or many other data management libraries), as it's a tool used within data management to write simpler immutable code, not the entire data management itself.

1.5 The technology stack

The technologies used in a given project are often referred to as the *technology stack*, *solution stack*, or *stack* for that project. It's a common term that you might even find listed directly in a job posting or on a startup's website.

We'll cover where the concept of stacks comes from, as well as the contents of the Frontend React stack, how to quickly understand a stack as you join an existing project, and how to create your own stack for a new application.

One interesting side note: *Technology stack* is the concept from which the term *fullstack* originates. A *fullstack developer* is someone who

works on the entire technology stack, from frontend to backend, in a given project.

1.5.1 Why do we talk about a tech stack?

In the realm of software development, the technology stack serves as a blueprint, guiding the construction of robust and efficient applications. This comprehensive outline details the combination of technologies, frameworks, and tools that form the foundation of a software project. Although it's commonly employed in various domains, frontend development in particular benefits from a well-defined technology stack des-cription to streamline collaboration, ensure consistency, and maximize productivity.

The creation of a technology stack involves carefully selecting the frontend technologies that best align with project requirements and goals. This task may include choosing a JavaScript framework like React, Angular, or Vue.js alongside supporting libraries, build tools, and testing frameworks. By documenting these choices clearly and concisely, development teams can communicate their technological preferences effectively and facilitate seamless collaboration among team members.

A technology stack serves as a reference point for stakeholders in frontend development projects. It provides valuable insights into the tools and technologies employed, enabling effective project planning, resource allocation, and decision-making. For project managers, designers, developers, and quality assurance (QA) teams, a shared understanding of the technology stack description fosters a cohesive and efficient development process.

For frontend development, a technology stack holds particular significance. It outlines the essential components required for UI creation, data management, state handling, routing, and more. This description encapsulates the frontend ecosystem, encompassing frontend frameworks, UI libraries, data management solutions, build tools, and other specialized technologies. By defining the frontend technology stack,

developers can ensure consistency, scalability, and maintainability throughout the project's life cycle.

It's crucial to acknowledge that the technology stack description is not static; it's an evolving blueprint. This goes double for frontend projects, as the ecosystem advances at a much faster pace than most backend technology landscapes. As a frontend project progresses and matures, new requirements emerge, and technological advancements become available. Consequently, the technology stack needs to adapt to accommodate these changes. This dynamic nature reflects the agility required in modern frontend development. Teams must remain flexible, continuously evaluating and adjusting their technology stack to embrace innovation and ensure that their frontend product remains competitive and aligned with evolving user needs. In this way, the technology stack becomes a living document that grows and evolves alongside the frontend project it supports, empowering developers to harness the latest tools and techniques to deliver exceptional user experiences.

1.5.2 Viewing the anatomy of a React stack

You'll often see React stacks boiled down to these five core layers (also illustrated in figure 1.6):

- Foundation layer (build tools, bundler, TypeScript, and so on)
- Data layer (data management and state flow)
- API layer (fetching libraries)
- UI layer (UI libraries and CSS-in-JS libraries)
- Testing layer (unit, integration, and end-to-end-testing)

Optional parts of the React stack include information about developer experience tools and various utility libraries that are large enough to warrant their own mention.

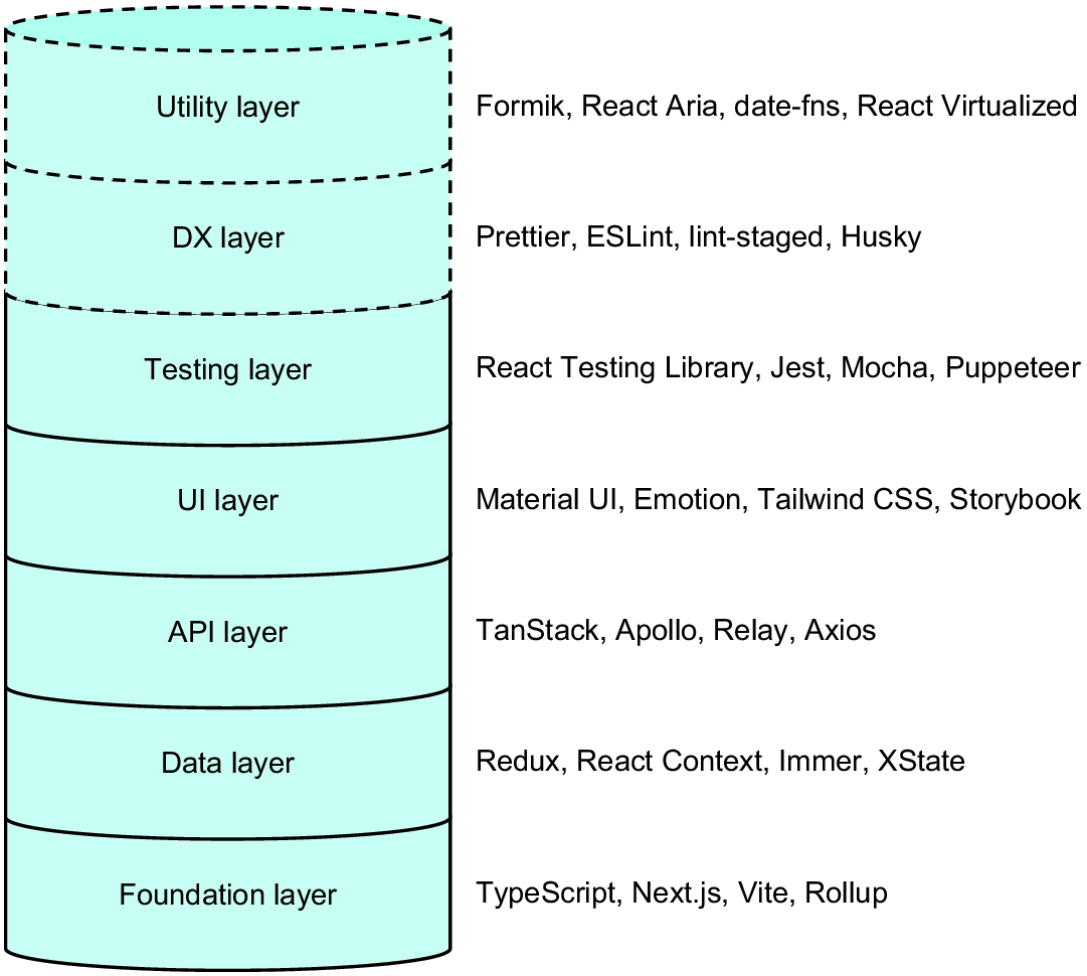

Figure 1.6 The overall layers of a React application, with several examples of each. Note the dashed lines around the top two layers, which are the most likely ones to be left out.

Note that each layer in the stack as described here may include more than one technology. When you're describing the data layer, for example, you might have zustand, Immer, and XState in the mix in different parts of an application. And the foundation layer in particular will often include many different technologies.

The level of detail you go into when describing a stack is up to you. The more detail, the denser the information is to your audience. If you want to give a high-level overview in a job listing, for example, saying "Next.js and TypeScript with Redux and Material UI" might be just fine. But if you're introducing said new hire to the entire application, you'd want to add a lot more detail to get them to understand the system.

If you're using a fullstack React setup with server-side rendering (SSR), you might also include information about the database layer and de-

velopment operations (colloquially known as *DevOps*) items as the bottom-most layers in the graphic. Figure 1.7 illustrates this stack. We'll look at some example stacks later in this section, where we'll get more specific.

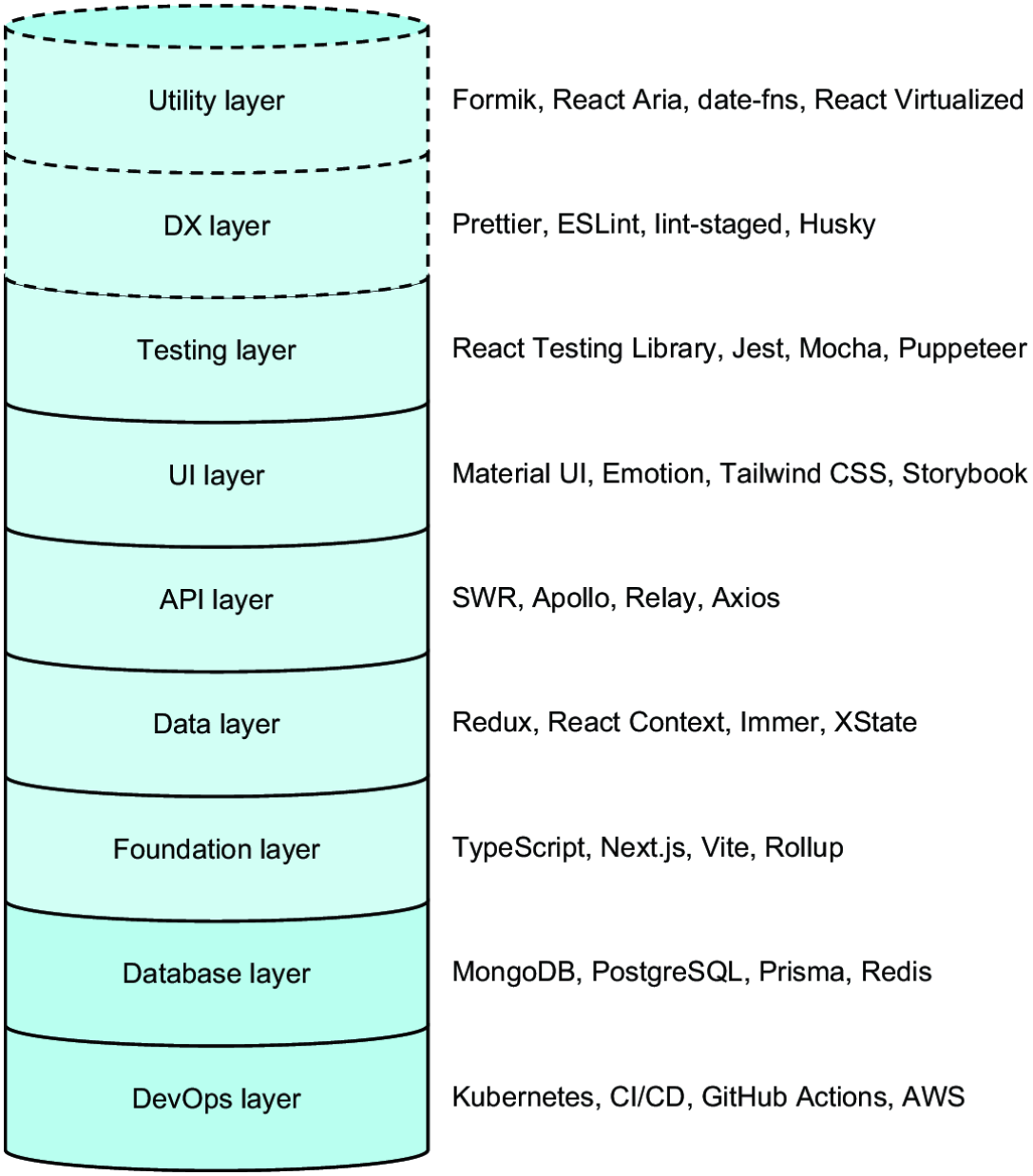

Figure 1.7 We've added two new layers in darker gray at the bottom of the stack for SSR setup: the database layer and the DevOps layer. Note that we're not restricting the database layer to actual databases but to anything that functions *in that realm*. The DevOps layer includes anything from hosting to deployment pipeline tools.

1.5.3 Joining a project

A well-defined technology stack serves as a guiding light for onboarding new team members, especially in the realm of frontend development. When a new developer joins the team, they are often confronted

with a codebase that may seem complex and unfamiliar. In such situations, the technology stack acts as a compass, providing clear direction and an overview of the tools, frameworks, and libraries that power the project. New hires can use this document to orient themselves quickly, understanding the key components they'll be working with and how those components fit into the broader development ecosystem.

The technology stack also plays a crucial role in streamlining the learning curve for new team members. By offering insights into the chosen technologies and their purposes, it accelerates the process of getting familiar with the codebase. New developers can refer to this outline to gain an understanding of the frontend framework in use, the styling methodology employed, the data management solutions implemented, and more. This foundational knowledge allows them to dive into the codebase with greater confidence, knowing where to focus their efforts and how various components interact. In essence, the technology stack becomes an essential tool for reducing the time it takes for new hires to contribute effectively to the project.

As a new hire stepping into a team with a well-defined technology stack, a strategic approach to onboarding can significantly expedite your assimilation into the project. Don't dive headfirst into the codebase; rather, begin by studying the technologies mentioned in the stack. Familiarize yourself with each technology, especially if you already have experience with related technologies. If you're well versed in Redux but the project uses zustand, understanding that zustand serves as a Redux alternative gives you a head start. Instead of delving into zustand within the context of the specific codebase, you can focus on grasping the nuances of zustand itself, comparing its approach to that of Redux. This proactive knowledge-building approach empowers you to adapt swiftly to the technology stack in use.

Additionally, exploring the documentation, tutorials, and resources related to the technologies before diving into the codebase can be immensely beneficial. These external resources offer valuable insights and best practices for working with the chosen technologies. By gaining a solid understanding of the stack components independently, you

enhance your ability to navigate the codebase effectively. When you eventually delve into the code, you'll find it easier to identify where and how these technologies are implemented, thanks to your existing knowledge.

Moreover, this proactive approach allows you to be a quick learner when it comes to new technologies introduced within the project. When you encounter unfamiliar stack elements, you can draw on your foundational knowledge to grasp their purpose and utility. This knowledge enables you to adapt swiftly to changes or enhancements made to the stack as the project progresses. In essence, by preparing yourself with a strong foundation in the stack technologies, you position yourself as an agile learner who can readily explore and adapt to new tools and frameworks introduced in the project.

It's essential to prepare strategically for rapid learning and efficient onboarding in a project with a well-defined technology stack. To help you get started, here's a practical to-do list that can significantly enhance your readiness:

- *Review stack components* —Begin by thoroughly reviewing the technology stack components mentioned in the stack; understand their roles and how they interact.
- *Review related technologies* —If you have experience with technologies related to those in the stack, such as Redux or MobX in the context of zustand, take time to refresh your knowledge.
- *Explore documentation* —Explore the official documentation and tutorials for the stack technologies. Familiarize yourself with their core concepts and use patterns.
- *Check online resources* —Seek online resources, blogs, or video tutorials that provide insights into best practices and common challenges related to the stack.
- *Seek external learning* —If certain stack elements are new to you, consider completing online courses or tutorials that focus specifically on those technologies.
- *Create practice projects* —Experiment with small practice projects using the stack components independently. Create a simple app with React and zustand, for example, to gain hands-on experience.

- *Study coding standards*—Study any coding standards and practices mentioned in the technology stack to align your coding approach with the team's expectations.
- *Have peer discussions*—Engage in discussions with your peers within the team to gather insights into how the stack is used in the project. Seek clarification on any queries or doubts you may have.
- *Bookmark documentation*—Organize and bookmark relevant sections of the stack documentation for quick reference during your initial work on the project.
- *Seek mentorship*—Don't hesitate to reach out to experienced team members for guidance and mentorship as you prepare to dive into the codebase.

By systematically tackling this to-do list, you'll enter the project well prepared to use your knowledge of the technology stack, making the process of familiarizing yourself with the codebase smoother and more efficient.

1.5.4 Creating a stack from scratch

Choosing the right stack for a React project can be challenging, as many options are available in the ecosystem. In this section, we'll discuss some of the factors to consider when choosing stack components by looking at some example scenarios and suggesting specific technologies for them.

The most important decision is the foundation layer of your application. This decision is the hardest to change at a later stage in development but of course still doable then, so don't get too stuck on it. Building on said foundation, you expand with technologies that solve your particular pain points in the easiest way possible.

One crucial balancing act rests at the center of creating a stack for a new project: familiarity matters, but new technologies surpass existing ones. If your team knows TanStack, it doesn't matter if some other data-fetching library might technically be a better fit for a given new project. Your team will work a lot faster in a library they're familiar with than in a completely new one. On the other hand, if you don't

challenge yourself, you'll never evolve, and as technologies go stale or become outdated, you'll never learn about the new and simpler/better/faster tools replacing them.
Dancing on this balancing beam is at the center of the role of the team architect who's creating the technology stack. Where should we stick with what we know, and when should we challenge ourselves? These decisions don't exist just at the birth of a new project; they pop up continuously. Sometimes, you have to replace a given technology in your stack with a new one despite the growing pains it will inevitably introduce, simply because you have to keep up with the times. Let's go through some scenarios and see how we can solve them with a good choice of technologies.

SCENARIO: A MEDIUM-SIZE E-COMMERCE PLATFORM

You're tasked with revamping a medium-size e-commerce platform to improve performance, scalability, and user experience. For this kind of project, you want a stack that's proved itself time and time again, and you want to move fast and build things quickly. One good choice for a project like this one would be what we might call *The Popular Stack*:

- Next.js as the foundation
- RTK as the data management library
- TanStack as the data-fetching library
- Material UI with MUI as the styling library

The Popular Stack offers the familiarity of Next.js for server-side rendering, TanStack for efficient data fetching, RTK for state management, and Material UI for a polished UI. This stack provides the tools needed to enhance the platform's speed and user interface, ensuring an excellent shopping experience.

SCENARIO: A PERSONAL PORTFOLIO WEBSITE

You're a solo developer looking to create a personal portfolio website that showcases your skills and projects. For this kind of project, you're free to play around, but you also want to use the latest and greatest

tools out there, both to show off your skills and to stay ahead of the competition. The stack for such a project will always be changing with the times, but one possible candidate is what we'll call *The Indie Stack*:

- Vite at the foundation
- Zustand as the data management library
- Tailwind CSS as the styling library

The Indie Stack is perfect for this scenario. Vite offers rapid development with blazing-fast bundling, zustand simplifies state management, and Tailwind CSS allows for quick and attractive styling. This stack empowers you to showcase your work efficiently and aesthetically.

SCENARIO: MAINTAINING A LEGACY ENTERPRISE DASHBOARD

Your team is responsible for maintaining a legacy enterprise dashboard built using older technologies. The stack for such a project was determined a long time ago, and you just have to play along. Refactoring this to a new stack is nigh on impossible as the project is huge. For now, you're stuck with *The Old School Stack*:

- Create React App (CRA) as the foundation
- Redux as the data management library
- Axios as the data-fetching library
- CSS Modules as the styling library

The Old School Stack, with CRA for stability, Axios for data fetching, Redux for state management, and CSS Modules for maintainable styling, is still a good choice. It ensures compatibility with existing code while leaving an open door for gradual modernization.

SCENARIO: A FINANCIAL SERVICES WEB APPLICATION

You're developing a comprehensive web application for a financial services company. For this scenario, you want something that's trustworthy, secure, and scalable and that won't ruffle any feathers in senior management. You can't go wrong with *The Enterprise Stack*:

- Pure React plus TypeScript as the foundation
- Apollo as the data-fetching and management library
- Styled-components as the styling library

The Enterprise Stack, combining React with TypeScript for strong typing, Apollo for managing complex data fetching and storage, and styled-components for consistent and maintainable styling, ensures robustness and scalability for handling financial transactions securely.

SCENARIO: RAPID PROTOTYPING OF A COLLABORATIVE TASK MANAGEMENT TOOL

Your startup is building a collaborative task management tool, and you need to prototype the core features quickly to attract potential investors and users. For this task, you want something that has a lot of magic built in and can very easily scale up and move fast. You don't mind if the design looks a bit derivative, as speed and features—and making early investors satisfied with the progress—are more important than a unique user interface. You should check out *The Prototype Stack*:

- Remix as the foundation
- Supabase as the backend
- Stale-While-Revalidate (SWR) as the data-fetching library
- Ant Design as the styling library

The Prototype Stack, featuring Remix for fast server rendering, Supabase for rapid backend development, SWR for scalable data fetching, and Ant Design for a polished, feature-rich UI, allows you to create a functional prototype swiftly, demonstrating the product's potential to stakeholders.

Summary

- This book addresses the gaps in practical, large-scale React application development, covering essential topics for improved skills, teamwork, and continuous learning in the evolving React ecosystem.

- This book offers comprehensive scope, hands-on focus, and emphasis on best practices to teach advanced React development, guiding readers through nine realms of React to become confident, skilled professionals.
- The React ecosystem is extensive, comprising various libraries and tools for enhancing React applications' performance and functionality.
- This book explores popular libraries and tools in the React ecosystem, emphasizing practical, hands-on development and best practices.
- Readers will gain a comprehensive understanding of the React ecosystem, allowing them to select the right tools for their projects and build high-quality React applications.
- A technology stack (also known as a *solution stack* or simply *stack*) is central to software development, guiding the selection of technologies, frameworks, and tools for a project.
- A well-defined technology stack enhances collaboration, reduces onboarding time for new team members, and provides a blueprint for efficient project development, with specific layers that encompass foundational elements, data management, API use, UI components, and testing.
- The choice of a React stack components depends on project requirements and familiarity with the technologies, with options ranging from established stacks like The Popular Stack for e-commerce platforms to innovative stacks like The Indie Stack for personal portfolio websites, each tailored to specific project goals and constraints.