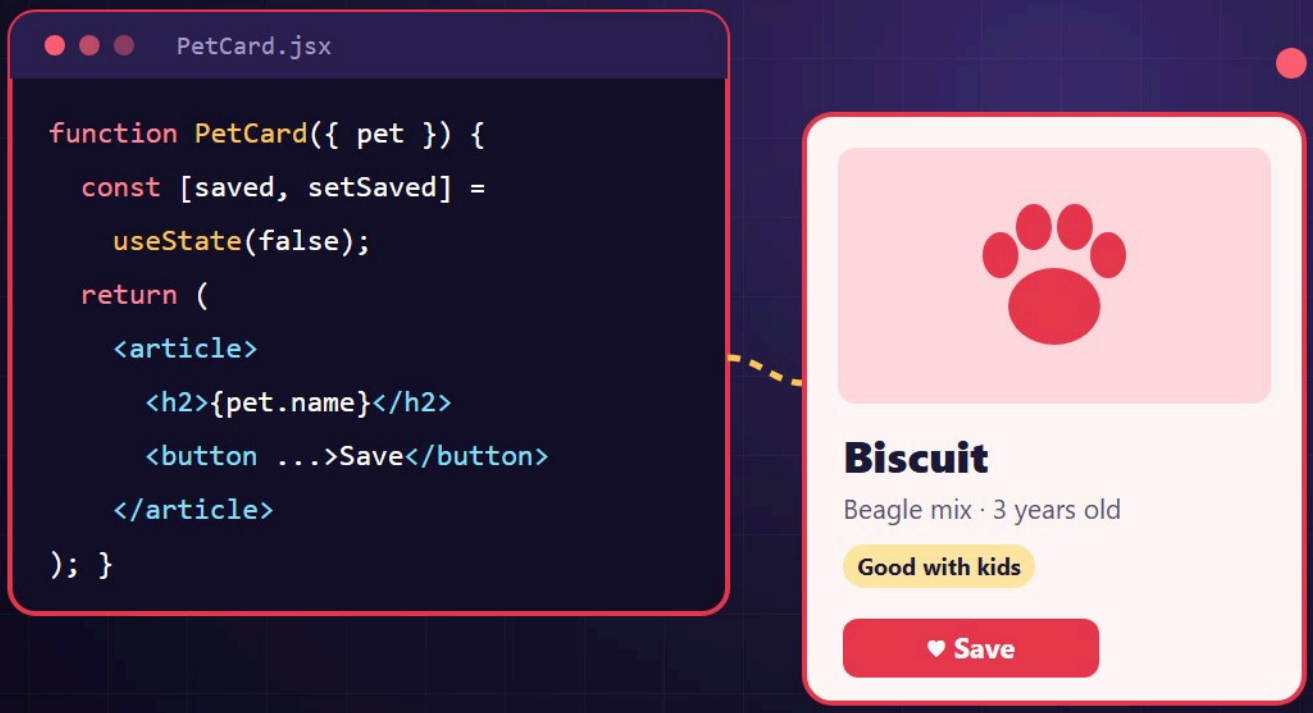


BUILD YOUR FIRST APPS

React for Complete Beginners

Build Your First Interactive Web Apps, Step by Step

No React experience needed · Components, state, effects, routing, and complete projects

COMPONENTS · PROPS · STATE · HOOKS

THENEXBOOKS

React for Complete Beginners

Build Your First Interactive Web Apps, Step by Step

First Edition

THENEXBOOKS

React for Complete Beginners
Build Your First Interactive Web Apps, Step by Step

First Edition, 2026

Copyright © 2026 thenexbooks (thenexbooks.gumroad.com). All rights reserved.

No part of this publication may be reproduced, distributed, or transmitted in any form or by any means without the prior written permission of the publisher, except for brief quotations in reviews and certain other non-commercial uses permitted by copyright law. The code in this book may be used freely in your own projects.

The information in this book is provided "as is" without warranty of any kind. While every precaution has been taken in its preparation, the publisher assumes no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein. React, its tools, and its ecosystem change over time; consult the official documentation for the versions you use.

This book is an independent publication and is not affiliated with, sponsored by, or endorsed by Meta Platforms, Inc. React is a trademark of Meta Platforms, Inc. Vite, Node.js, Visual Studio Code, React Router, Vitest, Tailwind CSS, Netlify, Vercel, GitHub, and other product names are trademarks of their respective owners; their use here is for identification only. Whisker Haven, Maple Hollow, and all people and animals in the examples are fictional.

Preface

React is one of the most popular ways to build interactive websites. Instead of writing long scripts that reach into a page and change it piece by piece, you describe what the screen should look like for the current data, and React keeps the page in sync as that data changes. Once that idea clicks, building buttons that respond, lists that filter as you type, and pages that remember your choices becomes surprisingly natural.

Many React tutorials, though, assume you already know modern JavaScript, the command line, and a pile of tools. This book does not. It starts from the very beginning: installing the tools, writing the JavaScript that React relies on, and creating your first component. From there we add one idea at a time — props, lists, events, state, forms, effects, data fetching, routing, and context — and use each one immediately in a real app.

That app is **Whisker Haven Adopt**, a website for a small animal shelter. Chapter by chapter it grows from a single hard-coded pet card into a complete app with search and filters, saved favorites, a pet details page, an adoption application, and a volunteer shift planner. By the end you will have built something you can proudly show, and you will understand React well enough to build your own ideas.

Who This Book Is For

- **Complete beginners to React** who know basic HTML and CSS — JavaScript experience is helpful but not required.
- **Students and self-taught learners** who want a structured, patient path into modern front-end development.
- **Designers and career changers** who want to turn their layouts into working, interactive apps.
- **Developers from other fields** who want a gentle, practical introduction before tackling larger frameworks.

What You Will Learn

- How to set up a React project with Vite and the JavaScript features React uses every day.
- Components, JSX, props, lists, keys, conditional rendering, and styling with CSS and CSS Modules.
- Events, state with `useState`, controlled forms, and how to share state between components.
- Effects, fetching data, loading and error states, localStorage, custom hooks, and `useReducer`.
- Multiple pages with React Router, sharing data with context, refs, debugging, and testing.
- Two complete projects — a multi-step application form and a volunteer planner — and how to deploy your app.

How This Book Is Organized

Part I — Getting Ready (Chapters 1–5) sets up your tools, teaches the JavaScript you need, and builds your first component. **Part II — Components and Props** (Chapters 6–9) shows how to build a user interface from reusable pieces and style it. **Part III — State and Interactivity** (Chapters 10–14) makes the app respond to users and ends with a complete pet adoption browser. **Part IV — Effects and Data** (Chapters 15–19) loads data, saves it, and manages more complex state. **Part V — Growing Your App** (Chapters 20–24) adds pages, shared data, refs, debugging, and tests. **Part VI — Projects and Next Steps** (Chapters 25–28) completes two larger projects, deploys the app, and points the way forward. The appendices provide a cheat sheet, starter files, checklists, and a glossary.

Every chapter ends with **Try It Yourself** challenges and a **Key Takeaways** box.

Welcome to Whisker Haven

Whisker Haven Animal Shelter is a small, friendly (and fictional) shelter in the town of Maple Hollow. Grace Lin manages the shelter, Omar Haddad coordinates the volunteers, and Jess Carter helps families find the right pet. Right now they post adoptable animals on a paper noticeboard. Over the course of this book, you will build them a real web app — and meet Biscuit, Luna, Pepper, Mochi, and the rest of the residents along the way.

How to Use This Book

Type the code yourself instead of copying it; noticing details and fixing your own typos is one of the best ways to learn. Keep the development server running while you work so you can see each change in the browser immediately. When something breaks — and it will, for everyone — read the error message, compare your code with the book, and remember that debugging is a normal part of programming. Chapter 23 is devoted to it.

Conventions Used in This Book

> **TIP**
> A recommended practice or a shortcut that makes your work easier.

> **NOTE**
> Background information, version details, or a useful aside.

> **WARNING**
> A common mistake or something that can cause bugs.

Code appears in `monospace`. When a code block shows a file, its path appears above it — for example `src/components/PetCard.jsx`. Boxes labeled **What you see in the browser** describe the result on screen.

Contents

Preface

PART I — GETTING READY

1 What Is React, and Why Learn It?

2 Setting Up: Node.js, VS Code, and Your First React App

3 The JavaScript You Need for React, Part 1

4 The JavaScript You Need for React, Part 2

5 Your First Component and JSX

PART II — COMPONENTS AND PROPS

6 Building a UI from Components

7 Props: Passing Data to Components

8 Rendering Lists and Conditional Content

9 Styling React Apps

PART III — STATE AND INTERACTIVITY

10 Handling Events

11 State with useState

12 Forms and Controlled Inputs

13 Lifting State Up and Sharing Data

14 Project: The Pet Adoption Browser

PART IV — EFFECTS AND DATA

15 useEffect Explained Simply

16 Fetching Data from an API

17 Loading, Errors, and Empty States

18 Saving Data with localStorage and Custom Hooks

19 Managing Complex State with useReducer

PART V — GROWING YOUR APP

20 Multiple Pages with React Router

21 Sharing Data with Context

22 Refs, Focus, and Working with the DOM

23 Debugging React Apps

24 Testing Your Components

PART VI — PROJECTS AND NEXT STEPS

25 Project: A Multi-Step Adoption Application Form

26 Project: The Volunteer Shift Planner

27 Building and Deploying Your App

28 Next Steps: TypeScript, Next.js, and Keep Building

APPENDICES

A React Cheat Sheet

B JavaScript Refresher and Project Starter Files

C Checklists

D Glossary

PART I

Getting Ready

Meet React, set up your tools, learn the JavaScript React relies on, and build your very first component.

1 · What Is React, and Why Learn It?
2 · Setting Up Your First React App
3 · The JavaScript You Need, Part 1
4 · The JavaScript You Need, Part 2
5 · Your First Component and JSX

CHAPTER 1 · PART I — GETTING READY

What Is React, and Why Learn It?

Before we write a single line of React, we will get a clear picture of what React is, what problem it solves, and how it thinks about a web page. Along the way you will meet Whisker Haven Animal Shelter and preview the pet adoption app we will build together, one small step at a time.

Welcome, and What You Need to Start

Welcome to *React for Complete Beginners*. If you have built a web page or two with HTML and CSS, you already have everything you need to begin. You do not need to be a JavaScript expert. You do not need a computer science degree. You do not even need to remember much JavaScript, because Chapters 3 and 4 teach exactly the JavaScript that React uses, slowly and with plenty of examples.

This book is built around one idea: **you learn React by building something real**. Instead of a pile of disconnected examples, we will grow a single app from a blank folder into a complete, multi-page web application. Each chapter adds one new idea, shows it on a tiny example first, and then uses it in the app. By the end, you will have written every line yourself and understood why each line is there.

Here is what helps you get the most out of the book:

- **A computer** running Windows, macOS, or Linux. Any reasonably recent machine will do; React development is not demanding.
- **Some HTML and CSS.** You should recognize tags such as `<h1>`, `<p>`, `<button>`, and `<div>`, and know that CSS changes colors, sizes, and layout.
- **Curiosity and patience.** You will see error messages. Everyone does, every day, including people who have written React for years. Errors are not a sign that you are doing badly; they are the computer telling you exactly where to look.

Web Pages and Web Apps

Let us start with a question that sounds simple: what is the difference between a web page and a web app?

A **web page** is mostly something you read. A restaurant menu, a blog post, or a page listing a shelter's opening hours are good examples. The content is the same for everyone, and once the page loads, it mostly sits still. You scroll, you read, you click a link to go to another page, and the browser loads that new page from the server.

A **web app** is something you use. Think of an email inbox, an online map, a to-do list, or a shopping cart. The page reacts to what you do while you are on it. You type in a search box and the results narrow down instantly. You click a heart icon and a counter in the corner goes up. You drag an item and the list reorders itself. The page changes *in place*, without loading a brand new page each time.

An example: saving favorite pets

Imagine a shelter website that lists adoptable pets. Each pet has a small card with a photo, a name, and a heart button labeled "Save". At the top of the page there is a counter: "Favorites: 0".

When a visitor clicks the heart on a pet, several things should happen at once:

1. The heart on that card changes to show it is saved.
2. The favorites counter at the top goes up by one.
3. If there is a "Favorites" list somewhere else on the page, the pet appears in it.

Clicking the heart again should reverse all three. None of this is hard to describe in words. The challenge is keeping every part of the screen correct as the visitor clicks around, filters the list, and changes their mind. With a handful of pets it is easy. With dozens of pets, a search box, filters, and several pages, it becomes surprisingly easy to end up with a counter that says 3 while only two hearts are filled in. React exists to make that kind of bug much less likely.

So, What Is React?

React is a JavaScript library for building user interfaces. That one sentence contains three important words, so let us unpack each.

- **JavaScript:** React is written in JavaScript and you use it by writing JavaScript. It runs in the web browser, the same place your HTML and CSS end up.
- **Library:** A library is a collection of ready-made code you can call from your own code. React focuses on one job and leaves other jobs, such as page navigation or talking to a server, to you or to other libraries. We will add a routing library in Chapter 20, for example.
- **User interfaces:** The "UI" is everything the user sees and interacts with: headings, buttons, forms, lists, and cards. React's whole purpose is to help you describe the UI and keep it up to date.

React was created at Facebook (now Meta) and released as open source in 2013. It is free to use, and its source code is public. Over the years it has changed a lot. This book uses **React 19** and the modern style of writing React with *function components* and *hooks*. You may come across older tutorials that use "class components"; that is an older style you will not need for this book.

> **NOTE**
>
> React is a trademark of Meta Platforms, Inc. Other product names mentioned in this book, such as Node.js, Vite, and Visual Studio Code, belong to their respective owners and are used only to identify the tools.

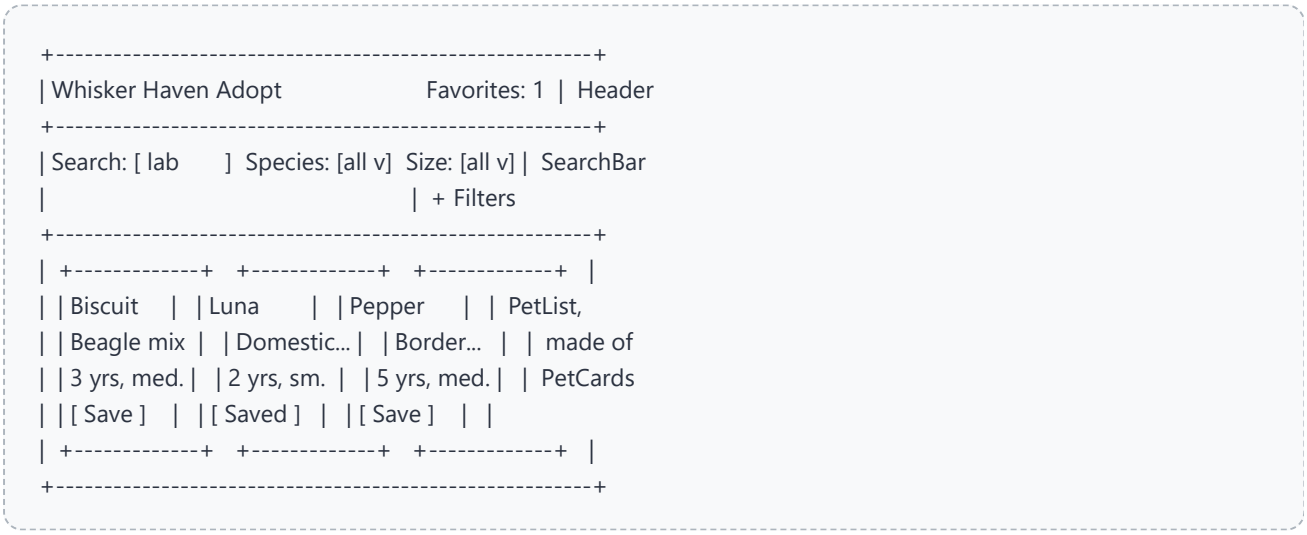
Big Idea 1: Components

The first big idea in React is the **component**. A component is a reusable piece of the user interface that you define once and use as many times as you like.

If you have ever built with toy bricks, you already understand components. You do not build a castle from one giant block. You build it from many small, standard pieces that snap together. Some pieces are simple, like a single flat tile. Others are small assemblies, like a window made of a frame and a pane, that you can reuse all over the castle.

A web page can be broken down the same way. Look at the rough sketch below of the adoption page we will eventually build.

A rough sketch of the Whisker Haven Adopt pets page



We can see at least five different pieces here:

- A **Header** across the top with the site title and a favorites counter.
- A **SearchBar** where visitors type a name or breed.
- A set of **Filters** for species and size.
- A **PetList** that holds all the cards.
- Many copies of a **PetCard**, one for each pet.

In React, each of these becomes its own component. The most interesting one is the `PetCard`. We define what a pet card looks like just once, and then React creates one card for every pet in our data. If we later decide the card needs to show the pet's size in bold, we change one component and every card

on the page updates.

Components can contain other components. The `PetList` contains many `PetCard` components. The whole page is itself a component, usually called `App`, which contains the `Header`, the `SearchBar`, the `Filters`, and the `PetList`. This creates a tree shape, a bit like a family tree:

The component tree

```
App
|-- Header
|-- SearchBar
|-- Filters
`-- PetList
   |-- PetCard (Biscuit)
   |-- PetCard (Luna)
   |-- PetCard (Pepper)
   `-- ... one PetCard per pet
```

Why does this matter? Because small pieces are easier to understand than big ones. When the favorites counter looks wrong, you know to look in the `Header`. When cards look wrong, you open the `PetCard`. Each file stays short and focused. We will practice breaking designs into components in Chapter 6.

A first look at a component

You do not need to understand the code below yet. Just look at its shape. Here is roughly what a simple pet card component looks like in React:

A preview of a PetCard component (we will write the real one later)

```
function PetCard({ pet }) {
  return (
    <article className="pet-card">
      <h2>{pet.name}</h2>
      <p>{pet.breed} · {pet.age} years old</p>
      <button>{pet.isFavorite ? "♥ Saved" : "♡ Save"}</button>
    </article>
  );
}
```

If you squint, it looks like a JavaScript function that returns some HTML. That is exactly what it is. The HTML-like part is called **JSX**, and it lets us write the structure of a component in a familiar way. The curly braces, such as `{pet.name}`, are places where we drop JavaScript values into the markup. When React runs this component for Biscuit, it produces ordinary HTML like this:

The HTML React produces for Biscuit

```
<article class="pet-card"><h2>Biscuit</h2><p>Beagle mix · 3 years old</p>
<button>♡ Save</button></article>
```

Notice that `className` in the component became `class` in the final HTML. That is one of the small JSX differences we will learn in Chapter 5. For now, the key point is simple: **a component is a function that describes a piece of the page**.

Big Idea 2: Declarative UI

The second big idea is a little more abstract, but it is the real reason React feels different. React lets you write your user interface in a **declarative** way instead of an **imperative** way.

Those are fancy words for a simple difference:

- **Imperative** means giving step-by-step instructions: "Find the button. Change its text to Saved. Find the counter. Read its number. Add one. Write the new number back."
- **Declarative** means describing the result you want: "The button says Saved if this pet is a favorite, otherwise Save. The counter shows how many favorites there are."

A helpful comparison is ordering food. The imperative approach is walking into the kitchen and telling the cook every step: "Take out the pan, heat it, crack two eggs..." The declarative approach is telling the server, "Two scrambled eggs, please," and trusting the kitchen to handle the steps. You describe *what*, and someone else figures out *how*.

The imperative way, with plain JavaScript

Here is how the favorite button might be written with plain JavaScript that talks directly to the page. Again, you do not need to understand every line; notice how many small steps there are.

Plain JavaScript: updating the page step by step

```
let count = 0;
const button = document.getElementById("heart");
const counter = document.getElementById("favorites-count");

button.addEventListener("click", () => {
  if (button.textContent === "Save") {
    button.textContent = "Saved";
    count = count + 1;
  } else {
    button.textContent = "Save";
    count = count - 1;
  }
  counter.textContent = count;
});
```

We tried this code in a simulated browser page and clicked the button twice. After the first click, the button said "Saved" and the counter said 1. After the second click, the button said "Save" and the counter was back to 0. So it works.

But look at what we had to do. The code has to *remember to update every piece of the page by hand*. If we add a third place that depends on favorites, such as a list of saved pets, we must remember to update that too, in every place that changes favorites. Forget one line, and the page quietly shows the wrong thing. With one button this is fine. With twelve pet cards, a search box, filters, and several pages, keeping everything in sync by hand becomes a real headache.

The declarative way, with React

In React we think differently. We keep our important information, such as "which pets are favorites", in one place. Then we write components that *describe* what the page should look like for that information. The button's label is not something we change by hand; it is simply calculated from the data every time:

The React way: describe the result

```
<button>{pet.isFavorite ? "♥ Saved" : "♡ Save"}</button>
```

This single line says: "If this pet is a favorite, the button shows Saved; otherwise it shows Save." When the data changes, React runs our component again, compares the new description with what is already on the screen, and updates only the parts that actually changed. We never write "find the button and change its text". React does that part for us.

You can summarize the idea with a tiny formula you will see often in React circles:

The core idea of React

```
UI = component(data)
```

The user interface is what you get when you run your components with your current data. Change the data, and the UI follows. This is why React apps tend to stay consistent: there is one source of truth for each piece of information, and the screen is always a fresh description of that truth. We will explore this carefully when we meet **state** in Chapter 11.

> **TIP**
> When you get stuck later in the book, come back to this formula and ask two questions: "What data does this component depend on?" and "What should the screen look like for that data?" Most React problems become clearer once you can answer both.

How React Updates the Page

You might wonder how React knows which parts of the page to change. The details are clever, but the basic idea is easy to follow.

The browser keeps a live model of the page called the **DOM** (Document Object Model). Every heading, paragraph, and button on the screen is an object in the DOM. Changing the DOM is how any JavaScript, React included, changes what you see.

When your data changes, React calls your components again to get a fresh description of what the UI should be. It then compares this new description with the previous one and works out the smallest set of changes needed. Maybe only one button label and one number are different. React then updates just those parts of the real DOM and leaves everything else alone. This comparison process is often called **reconciliation**. You will also hear people mention a "virtual DOM", which is an informal name for the lightweight descriptions React compares.

The good news is that you almost never need to think about this process. You describe the UI; React handles the updates. The main thing to remember is that **your component functions may run many times**, every time the data they depend on changes. That fact will matter in later chapters, and we will point it out when it does.

Why Learn React?

There are many ways to build interactive websites, and React is one popular option among several. Here are some honest reasons it is a good one to learn, especially as your first UI library.

- **The ideas transfer.** Components, data flowing into the UI, and describing results instead of steps are ideas you will find in many modern tools. Learning them well with React makes other tools easier to pick up later.
- **It is "just JavaScript".** React adds a small number of concepts on top of the language. Most of what you write is plain JavaScript: functions, objects, arrays. As you get better at React, you get better at JavaScript too.
- **A large ecosystem.** React has been around for many years, so there are libraries for routing, forms, testing, charts, animations, and much more. When you have a common problem, there is often a well-known tool for it.
- **Room to grow.** Once you know React, you can explore frameworks built on top of it for larger sites, or React Native for mobile apps. We will look at some of these options in Chapter 28.

We will not make claims about job numbers or how many companies use React; those figures change constantly and are hard to measure fairly. What we can promise is that the skills in this book are practical, and that by the end you will have a real project you understand from top to bottom.

What React Does Not Do

It is just as useful to know what React is *not*, so you are not surprised later.

- **React does not replace HTML and CSS.** You will still use headings, lists, and buttons, and you will still style them with CSS. Chapter 9 covers styling.

- **React does not include page navigation.** Moving between "Home", "Pets", and "Favorites" pages uses a separate library. We will add React Router in Chapter 20.
- **React does not store data on a server.** React runs in the browser. It can fetch data from a server, and we will do exactly that in Chapter 16, but databases and servers are outside the scope of this beginner book.

> **WARNING**
> Searching the web for React help will turn up examples written in many different styles and for many different versions. Some use class components, some use older setup tools, and some are written for frameworks with extra rules. If an example looks very different from this book, check its date and version before copying it into your project. When in doubt, follow the book and the official documentation.

Meet Whisker Haven Animal Shelter

Every good project needs a reason to exist. Ours is **Whisker Haven Animal Shelter**, a small, friendly (and fictional) animal shelter in the fictional town of **Maple Hollow**. The shelter takes in dogs, cats, rabbits, and guinea pigs, cares for them, and helps them find new homes.

Three people at the shelter will pop up throughout the book:

Name	Role	What they care about
Grace Lin	Shelter manager	A clear, friendly site that shows every adoptable pet
Omar Haddad	Volunteer coordinator	An easy way for volunteers to sign up for shifts
Jess Carter	Adoptions counselor	Complete, well-organized adoption applications

Some useful facts about the shelter, which we will use in examples and in the app:

- **Opening hours:** Tuesday to Sunday, 10 AM to 6 PM. The shelter is closed on Mondays.
- **Adoption fees:** dogs $150, cats $95, and small animals (rabbits and guinea pigs) $40.

Grace has a problem that many small organizations share. Visitors want to browse the animals from home, narrow them down ("Which small pets are good with kids?"), keep a shortlist, and then apply. Omar needs volunteers to sign up for shifts without endless back-and-forth messages. Jess wants applications that arrive complete, not half-filled. A few static pages cannot do all this well. It is a perfect job for a React app.

What We Will Build: Whisker Haven Adopt

Throughout this book you will build **Whisker Haven Adopt**, a web app for browsing adoptable pets, saving favorites, applying to adopt, and scheduling volunteer shifts. We will build it gradually. Each part of the book adds a layer:

Part	What you learn	What the app gains
I — Getting Ready	Tools, JavaScript for React, JSX	A running project and a first pet card
II — Components and Props	Components, props, lists, styling	A styled grid of every pet
III — State and Interactivity	Events, state, forms, shared state	Search, filters, sorting, favorites
IV — Effects and Data	Effects, fetching, loading, storage	Pets loaded from a file; saved favorites
V — Growing Your App	Routing, context, refs, debugging, tests	Multiple pages and a pet details page
VI — Projects and Next Steps	Bigger projects, deployment	Adoption form, volunteer planner, live site

When the app is finished, a visitor will be able to:

- Browse all adoptable pets as a tidy grid of cards.
- Search by name or breed, and filter by species and size.
- Sort pets by name or age.
- Save favorites with a heart button and see a favorites count in the header, with favorites remembered even after closing the browser.
- Open a details page for each pet with its own web address.
- Fill in a multi-step adoption application with helpful validation.
- Sign up for volunteer shifts in a weekly planner, with limits on how many people each shift can hold.

That may sound like a lot for a beginner, and it is a real app. But you will get there one small step at a time. Chapter 14, halfway through the book, is a milestone where the searchable, filterable pet browser comes together. Many readers find that the moment when things start to "click".

Meet the Pets

Our app needs data to display. Whisker Haven currently has twelve adoptable animals. Each pet is described by the same set of fields, which we will store in JavaScript. Here is Biscuit, written the way our app will store him:

One pet, as a JavaScript object

```
{
  id: 1,
  name: "Biscuit",
  species: "dog",
  breed: "Beagle mix",
  age: 3,
  sex: "male",
  size: "medium",
  goodWithKids: true,
  goodWithPets: true,
  adopted: false,
  photo: "/pets/biscuit.jpg",
  description: "A cheerful sniffer who loves long walks and belly rubs."
}
```

Each line is a piece of information: a **name** on the left, a **value** on the right. Words in quotes are text, numbers are numbers, and `true` or `false` answer yes-or-no questions such as "Is this pet good with kids?". Do not worry about the exact punctuation yet. Chapter 3 explains objects like this step by step.

A few rules keep the data consistent. The `species` is always one of "dog", "cat", "rabbit", or "guinea pig". The `size` is "small", "medium", or "large". The `age` is a number of years, and it can be a fraction: Olive the kitten is 0.5, meaning about six months old. Every pet has a unique `id` number, which React will use to tell the pets apart.

Here are all twelve pets you will get to know:

Id	Name	Species	Breed	Age	Sex	Size
1	Biscuit	dog	Beagle mix	3	male	medium
2	Luna	cat	Domestic shorthair	2	female	small
3	Pepper	dog	Border collie	5	female	medium
4	Mochi	rabbit	Holland lop	1	female	small
5	Captain	dog	Labrador mix	7	male	large
6	Olive	cat	Tabby	0.5	female	small
7	Bruno	dog	Boxer mix	4	male	large
8	Clover	guinea pig	American	2	male	small
9	Willow	cat	Maine coon mix	6	female	large
10	Nugget	dog	Chihuahua mix	2	male	small
11	Hazel	rabbit	Lionhead	3	female	small
12	Shadow	cat	Bombay mix	4	male	medium

In Chapter 7 we will put all twelve pets into a file named `src/data/pets.js`, and from then on our components will read from it. Later, in Chapter 16, we will move the same data into a `pets.json` file and load it the way real apps load data from a server.

The Tools We Will Use

You will install these tools in Chapter 2. For now, here is a quick introduction so the names are familiar.

Tool	What it is	Why we need it
Node.js	A program that runs JavaScript outside the browser	Runs our development tools and lets us test JavaScript quickly
npm	The package manager that comes with Node.js	Downloads React and other libraries into our project
Vite	A fast build tool and development server	Creates the project and shows changes instantly in the browser
VS Code	A free code editor	Where we write and organize our files
A web browser	Chrome, Firefox, Edge, or Safari	Shows the app and includes developer tools

How to Read This Book

A few conventions will make the rest of the book easier to follow.

Code boxes and file labels

Code appears in shaded boxes. When the code belongs in a specific file, a small label above the box tells you which one, such as `src/components/PetCard.jsx`. When a box shows what appears in the browser or terminal instead of code, the label says so, for example "What you see in the browser" or "Console output". Those boxes show results; you do not type them.

Tips, Notes, and Warnings

Colored boxes point out extra information:

- A **Tip** is a helpful shortcut or habit that makes your work easier.
- A **Note** adds background or detail that is good to know but not essential right now.
- A **Warning** flags a common mistake or something that could cause confusing problems.