14 Project: Develop a word game in React

This chapter covers

- Solving a complex challenge by breaking it down
- Choosing your own stack

This chapter is the end of the line, the big finale, the grand exam. It has been quite a journey, and we've covered a lot of ground. In this last challenge, you'll be mostly on your own. I will help with some of the meta decisions on tackling the project, but I won't discuss any code. But I have a lot of faith in you. You can do it! Go, team!

Note The source code for the example in this chapter is available at **https://reactlikea.pro/ch14**.

14.1 Building a game

You'll be building a certain five-letter word-guessing game. You've probably heard of it. This game could end up looking like figure 14.1.

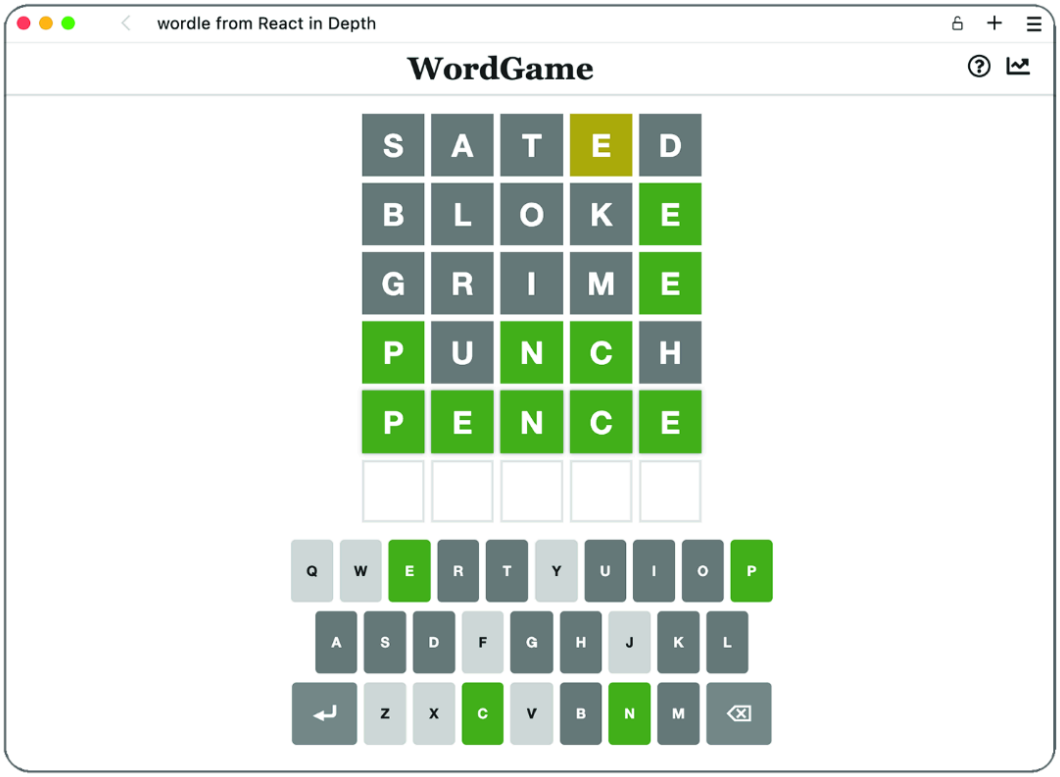

Figure 14.1 My implementation of the word game ended up looking something like this (but only in dark mode, this figure is an approximate conversion to light mode for print purposes). I didn't fare too well in this round and barely managed to get the answer in five words.

14.2 Choose your ambition

Before you start, it's a good idea to choose your ambition level. The application consists of two equally complex parts:

- The game engine, with a keyboard and a grid, allows you to guess a specified word.
- The web application around the game engine picks a daily word, remembers whether you've already played today (and if so, what happened), remembers past results, and provides statistics. It also welcomes new players.

That second part may look innocent or trivial, but the game engine of itself is lacking several things in order to become a good web application, which is what that second part does.

The first part is integral to the word game, but you could skip the second part to make the project a bit easier. The first part is more fun, in

my experience, so I recommend that if you feel a bit daunted by this challenge, ignore storage, persistence, and the welcome screen. Make a game component that takes a word and allows you to guess it. Regardless of whether you implement only the game or the application envelope as well, the challenge is still not set in stone. Many facets of the game and the application are optional, allowing you to fine-tune the challenge based on your ambition level. In the game engine, you have the following things to consider:

- Do you want to support hard mode? Check the original game if you don't know what it does.
- Which word database are you going to use? Do you want to use the same one as the original or something else?
- The grid and keyboard have a distinct style and color scheme. Do you want to copy them or develop your own?
- Do you want to include animations and transitions to make the game look a bit nicer, or do you want to skip them for simplicity?
- Do you want to support light mode only, light and dark mode, or dark mode only?
- Do you want to support color-blind mode with increased contrast?
- Do you want to add translations for various messages to make them more accessible?
- How do you want to handle errors—similarly to the original or differently?

If you're building the second part for the full web application experience, you have the following things to consider:

- You probably have to implement both a welcome/help dialog box and a result dialog box. Do you want to do the same thing as the existing game or something different? You can skip the welcome/help dialog box to make the project easier.
- Do you want to build a menu at the top, and if so, what should go in the menu?
- Do you want to track and display statistics on the results screen or only display the result of a single game?

- How will the game pick a daily word, and how will it make sure that the word is the same for everyone on a given day (if that's important)?

MY CHOICE

I chose to implement the following features of the word game:

- One new word per day based on the local time zone
- The same word database but a different word-selection algorithm
- Statistics saved in a similar fashion as the original
- Slightly different welcome and results dialog boxes
- No share button
- Keyboard and grid display similar to the original, including reveal and win animations
- Errors displayed with a wiggle and alert messages in the same style as the original
- No hard mode, no color-blind mode, dark mode only, no translations, and no settings

14.3 Choose your stack

After you figure out what you want to build, figure out how you want to build it. If you want to build it as a static website with URL routing and search engine optimization (SEO) performance, you probably want to use Next or Remix. If you want to work fast and not worry much about design, you might pull in an existing design library to make prototyping go faster. Although no design library is capable of facilitating a word game–like onscreen keyboard or word game–like letterwise reveals, you may be able to find smaller libraries that do those things. You need to make choices in the following categories:

- *Base framework*—Which base framework will you build this project on? You can use Vite, which we've done many times in this book, but you're also free to choose a different base, such as Next or Remix.
- *Developer experience (DX)*—Do you want to use tools to assist in the development experience, such as TypeScript, formatters, and

linters?

- *Styling*—Do you want to use an existing UI library, a styling library, or plain CSS? Do you want to use transitions and animations? React has some nice libraries for handling those elements.
- *State and flow management*—How do you want to store and manipulate data in your application: with `useState`, with reducers, or with some external library (or combination of libraries)?
- *Testing*—Do you want to write any tests? If so, what type of tests and using which libraries?

MY CHOICE

I chose the following stack:

- *Main build tool*—Vite
- *DX*
  - *TypeScript* for improving code clarity and robustness
  - *Prettier* for uniform formatting
  - *ESLint* for maintaining proper React style
- *Styling*
  - *@emotion/styled* for CSS-in-JS. (This library is similar to styled-components, but it's a bit newer and thus a bit faster.)
  - *react-icons/fa* for icons for the menu
  - *Plain CSS* for the transitions and animations
- *State and flow management*
  - *XState* for elegant flow management through the different stages of the various components and overall game
  - *Immer* inside XState for simplified state updates
  - *React Context* for distributing global state and providing overlay utility
- *Testing*
  - *Storybook* for visual testing
  - React Testing Library for component testing
  - *Vitest* as the main test runner

You are by no means required to use the same tools, but I like the ones in this stack. Feel free to pick your own tools or to copy some or all of what I did.

14.4 My implementation

You can see what I did in the repository.

EXAMPLE: WORDLE

This example is in the `wordle` folder. You can use that example by running this command in the source folder:

```
$ npm run dev -w ch14/wordle
```

Alternatively, you can go to this website to browse the code, see the example in action in your browser, or download the source code as a zip file: **https://reactlikea.pro/ch14-wordle**.

I'll also show you a few previews to get you curious. First, I developed all the base components and hooked them up with Storybook so you can see and explore how everything works. Figure 14.2 shows some screenshots from Storybook.

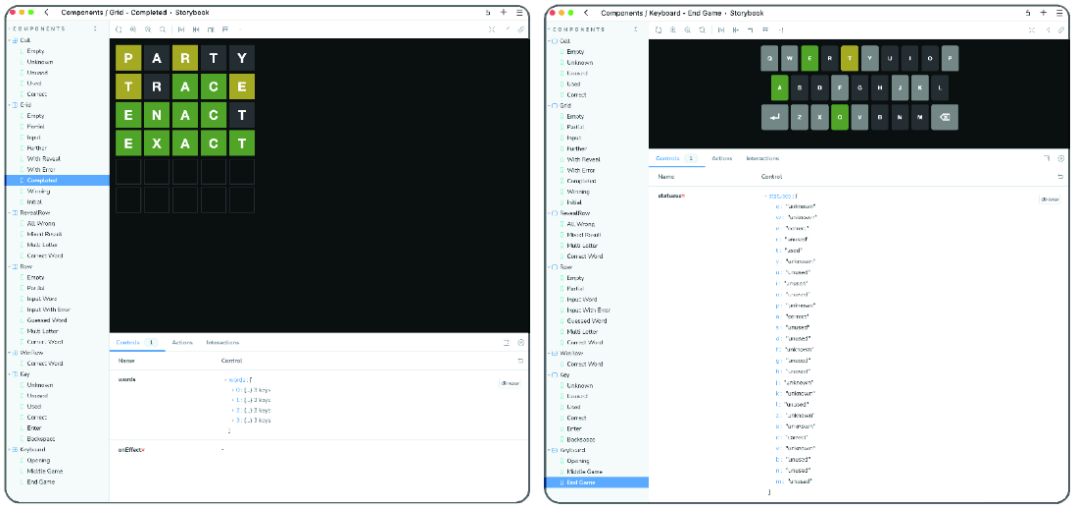

Figure 14.2 On the left is the grid display with rows of various states passed in. On the right is the keyboard with some custom state passed in. Notice all the components and variants on the left menu.

I built the main game logic in XState. Figure 14.3 shows the flow diagram.

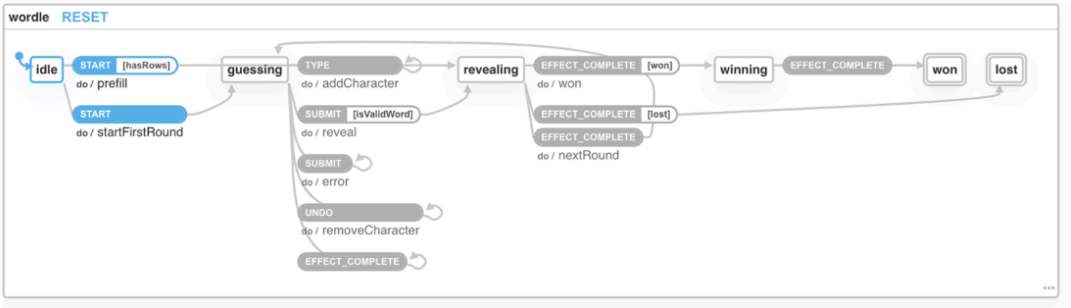

Figure 14.3 The XState flow diagram is surprisingly simple and is this complex only because of the reveal and win animations. Without those animations, it's even simpler. Guards and actions associated with the arrows make the game work as intended.

14.5 Share your result

I'm dying to see how you approach this challenge, and I want to see your results! Feel free to share your wins, frustrations, and any other experiences with me in the liveBook forums on Manning.com or on LinkedIn; feel free to @tag me at **https://linkedin.com/in/barklund**.