Introduction

You'll learn about the topics this chapter contains, which include using promises, async/await, and storing data on the browser.

In the 19th century, an international team of engineers embarked on one of the greatest engineering challenges of their day: laying a telegraph cable across the Atlantic Ocean. The project took several failed attempts and plenty of money and hours before it succeeded. In the end, you could send a message from Europe to the United States in an impressive 17 hours—much faster than the nearly two-week boat trip it used to take.

Fast communication can mean the difference between success and failure. JavaScript's resurgence is partially due to the fact that you can load a page once and then use JavaScript for all future communications to and from servers. Suddenly, you could experience websites as actual software instead of a series of discrete pages. When you skip page loads, you save time and resources for your users. They don't have to reload new images and other assets. They have less latency, and their experience is greatly improved. Accessing external data is crucial to so-called *Single Page web applications*.

What does this chapter include?

In this chapter, you'll learn how to *access* external data and how to use the data you receive. JavaScript is an *asynchronous* language, which means it *won't* block code execution while waiting for requested data. JavaScript can give you *speedy* websites, but asynchronous requests can be a little confusing to work with.

We'll start off by exploring how to use `fetch()` to access remote data. Next, we'll take a deep dive into *promises*, the JavaScript method for handling asynchronous requests. Then we'll use the new `async`/`await` syntax to make working with promises even more clear. Finally, you'll learn to *store* data on the browser so you can keep a user’s state without any server access.

You’ll never experience performance gains on the level of the transatlantic cable, but every second counts. Don’t be surprised when a mobile user leaves a site that requires a new page render on every action. You can’t remove the server entirely, but you can give your users experiences that make the server requests as painless as possible.

In the next tip, you’ll see how to use promises to perform asynchronous actions.