MERN STACK

Pro MERN Stack is for full-stack developers, architects, and team leads wanting to learn about this stack built with Mongo, Express, React, and Node. It was in late December 2016 that I finished the manuscript of the first edition. But within just a few months after the book was out, it was outdated. React Router released a new version, 3.0, and this alone was enough to warrant a new edition. I tried to communicate to readers via GitHub issues, asking them to use the older version of React Router. It worked for a while, but this was less than satisfactory. That was not all. Within another few months, there was a new Node.js LTS version, then React 16, and then a new version of MongoDB, and then Babel. Almost every module used in the MERN stack was upgraded and soon Pro MERN Stack became outdated. This pace of change amazed me. Although every new change was great, it did no good for technical authors like me. I did wait for things to stabilize a bit before initiating the second edition of Pro MERN Stack. And I am glad I did, because I believe the current set of tools that form the MERN stack have reached a good level of maturity and therefore will be reasonably stable for some time to come. Compared to the first edition, there are the expected newer versions of all the tools and libraries. But importantly, I have introduced more modern ways of doing things. I have replaced REST APIs with GraphQL, which I believe is a superior way of implementing APIs. There is a new architecture that separates the user interface and the APIs into two different servers. Instead of using in-memory sessions to track logged-in users, I used JSON Web Tokens. I have simplified server rendering by using a global store instead of the deprecated React Context. Finally, I introduced a new chapter on deployment using Heroku. I also changed many things that one needs to do as an author to enhance the readers' experiences. I added illustrations and more explanations to code snippets and increased the granularity of the sections. All these changes, in my opinion, will make Pro MERN Stack, Second Edition, far superior to the first edition. It's not just an up-to-date version of the first edition, but it enhances the learning experience significantly. 1 V. Subramanian, Pro MERN Stack, https://doi.org/10.1007/978-1-4842-4391-6_1 CHAPTER 1 Introduction Web application development is not what it used to be, even a few years back. Today, there are so many options, and the uninitiated are often confused about what's good for them. There are many choices; not just the broad stack (the various tiers or technologies used), but also for tools that aid in development. This book stakes a claim that the MERN stack is great for developing a complete web application and takes the reader through all that is necessary to get that done. In this chapter, I'll give a broad overview of the technologies that the MERN stack consists of. I won't go into details or examples in this chapter, instead, I'll just introduce the high-level concepts. I'll focus on how these concepts affect an evaluation of whether MERN is a good choice for your next web application project. What's MERN? Any web application is built using multiple technologies. The combinations of these technologies is called a "stack," popularized by the LAMP stack, which is an acronym for Linux, Apache, MySQL, PHP, which are all open-source software. As web development matured and their interactivity came to the fore, Single Page Applications (SPAs) became more popular. An SPA is a web application paradigm that avoids fetching the contents of an entire web page from the server to display new contents. It instead uses lightweight calls to the server to get some data or snippets and changes the web page. The result looks quite nifty as compared to the old way of reloading the page entirely. This brought about a rise in front-end frameworks, since a lot of the work was done on the front-end. At approximately the same time, though completely unrelated, NoSQL databases also started gaining popularity. The MEAN (MongoDB, Express, AngularJS, Node.js) stack was one of the early open-source stacks that epitomized this shift toward SPAs and adoption of NoSQL. AngularJS, a front-end framework based on the Model View Controller (MVC) design pattern, anchored this stack. MongoDB, a very popular NoSQL database, was used for persistent data storage. Node.js, a server-side JavaScript runtime environment, and Express, a

web-server built on Node.js, formed the middle-tier, or the web server. This stack was arguably the most popular stack for any new web application until a few years back. Not exactly competing, but React, an alternate front-end technology created by Facebook, has been gaining popularity and offers an alternative to AngularJS. It thus replaces the “A” with an “R” in MEAN, to give us the MERN Stack. I said “not exactly” since React is not a full-fledged MVC framework. It is a JavaScript library for building user interfaces, so in some sense, it’s the View part of the MVC. Although we pick a few defining technologies to define a stack, these are not enough to build a complete web application. Other tools are required to help the process of development, and many libraries are needed to complement React. This book is about building a complete web application based on the MERN stack and all these related tools and libraries. Chapter 1 ■ Introduction 2 Who Should Read This Book Developers and architects who have prior experience in any web app stack other than the MERN stack will find the book useful for learning about this modern stack. Prior knowledge of how web applications work is required. Knowledge of JavaScript is also required. It is further assumed that the reader knows the basics of HTML and CSS. It will greatly help if you are also familiar with the version control tool git; you could try out the code just by cloning the git repository that holds all the source code described in this book and running each step by checking out a branch. The code in the book uses the latest features of JavaScript (ES2015+), and it is assumed that you wellversed in these features such as classes, fat-arrow functions, const keyword, etc. Whenever I first use any of these modern JavaScript features, I will point it out using a note so that you are aware that it is a new feature. In case you are not familiar with a particular feature, you can read up on it as and when you encounter it. If you have decided that your new app will use the MERN stack, then this book is a perfect enabler for you that lets you quickly get off the ground. Even if you have not, reading the book will get you excited about MERN and equip you with enough knowledge to make a choice for a future project. The most important thing you will learn is to put together multiple technologies and build a complete, functional web application, and you can be called a full-stack developer or architect on MERN. Structure of the Book Although the focus of the book is to let you learn how to build a complete web application, most of the book revolves around React. That’s just because, as is true of most modern SPAs, the front-end code forms the bulk. And in this case, React is used for the front-end. The tone of the book is tutorial-like and designed for learning by doing. We will build a web application during the course of the book. I use the term “we” because you will need to write code just as I show you the code that is to be written as part of the plentiful code listings. Unless you write the code yourself alongside me and solve the exercises, you will not get the full benefit of the book. I encourage you not to copy-paste; instead please type out the code. I find this very valuable in the learning process. There are very small nuances—e.g., types of quotes—that can cause a big difference. When you type out the code, you are much more conscious of this than when you are just reading it. Sometimes, you may run into situations where what you typed in doesn’t work. In such cases, you may want to copy-paste to ensure that the code is correct and overcomes any typos you may have made. In such cases, do not copy-paste from the electronic version of the book, as the typesetting may not be faithful to the actual code. I have created a GitHub repository at https://github.com/vasansr/pro-mern-stack-2 for you to compare, and in unavoidable circumstances, to copy-paste from. I have also added a checkpoint (a git branch in fact) after every change that can be tested in isolation so that you can look at the exact diffs between two checkpoints, online. The checkpoints and links to the diffs are listed in the home page (the README) of the repository. You may find this more useful than looking at the entire source, or even the listings in the text of this book, as GitHub diffs are far more expressive than what I can show in print. Rather than cover one topic or technology per section, I have adopted a more practical and problemsolving approach. We will have developed a full-fledged working application by the end of the book, but we’ll start small with a Hello World example. Just as in a real project, we will add more features to the application as we progress. When we do this, we’ll encounter tasks that need additional concepts or knowledge to proceed. For each of these, I will introduce the concept or technology that can be used, and I’ll discuss that in detail. Thus, you may not find every chapter or section devoted purely to one topic or technology. Some chapters

may focus on a technology and others may address a set of goals we want to achieve in the application. We will be switching between technologies and tools as we progress. Chapter 1 ■ Introduction 3 I have included exercises wherever possible, which makes you either think or look up various resources on the Internet. This is so that you know where to get additional information for things that are not covered in the book, typically very advanced topics or APIs. I have chosen an issue-tracking application as the application that we'll build together. It's something most developers can relate to, at the same time has many of the attributes and requirements that any enterprise application will have, commonly referred to as a "CRUD" application (CRUD stands for Create, Read, Update, Delete of a database record). Conventions Many of the conventions used in the book quite obvious, so I'll not explain all of them. I'll only cover some conventions with respect to how the sections are structured and how changes to the code are shown because it's not very obvious. Each chapter has multiple sections, and each section is devoted to one set of code changes that results in a working application that can be run and tested. A section can have multiple listings, but each of them may not be testable by itself. Every section will also have a corresponding entry in the GitHub repository, where you can see the complete source of the application as of the end of that section, as well as the differences between the previous and the current section. You will find the difference view very useful to identify the changes made in the section. All code changes will appear in listings within the section, but please do not rely on their accuracy. The reliable and working code can be found in the GitHub repository, which could even have undergone last-minute changes that couldn't make it to the printed book in time. All listings will have a listing caption, which will include the name of the file being changed or created. You may use the GitHub repository to report problems in the printed book. But before you do that, do check the existing list of issues to see if anyone else has reported the same. I will monitor the issues and post resolutions and, if necessary, correct the code in the GitHub repository. A listing is a full listing if it contains a file, a class, a function, or an object in its entirety. A full listing may also contain two or more classes, functions, or objects, but not multiple files. In such a case, if the entities are not consecutive, I'll use ellipses to indicate chunks of unchanged code. Listing 1-1 is an example of a full listing, the contents of an entire file. Listing 1-1. server.js: Express Server const express = require('express'); const app = express(); app.use(express.static('static')); app.listen(3000, function () { console.log('App started on port 3000'); }); A partial listing, on the other hand, will not list complete files, functions, or objects. It will start and end with an ellipsis, and may have ellipses in the middle to skip chunks of code that have not changed. The new code added will be highlighted in bold, and the unchanged code will be in the normal font. Listing 1-2 is an example of a partial listing that has small additions. Chapter 1 ■ Introduction 4 Listing 1-2. package.json: Adding Scripts for Transformation ... "scripts": { "compile": "babel src --presets react --out-dir static", "watch": "babel src --presets react --out-dir static --watch", "test": "echo \"Error: no test specified\" && exit 1" }, ... Deleted code will be shown using strikethrough, like in Listing 1-3. Listing 1-3. package.json: Changes for Removing Polyfill ... "devDependencies": { "babel-polyfill": "^6.13.0", ... } ... Code blocks are used within regular text to extract changes in code for discussion and are usually a repetition of code in listings. These are not listings and often are just a line or two. Here is an example, where the line is extracted out of a listing and one word is highlighted: ... const contentNode = ... All commands that need to be executed on the console will be in the form a code block starting with $. Here is an example: $ npm install express All commands used in the book can also be found in the GitHub repository in a file called commands.md. This is so that errors in the book can be corrected after the book has been published, and also to be a more reliable source for copy-pasting. Again, you are encouraged not to copy-paste these commands, but if you are forced to because you find something is not working, then please copy-paste from the GitHub repository rather than the book's text. In later chapters in the book, the code will be split across two projects or directories. To distinguish in which directory the command should be issued, the command block will start with a cd. For example, to execute a command in the directory called api, the following will be used: $ cd api $ npm install dotenv@6 All commands that need to be executed within a MongoDB shell will be in the form of a code block starting with >. For example: > show

collections These commands are also collected in one file, called mongoCommands.md in the GitHub repository. Chapter 1 ■ Introduction 5 What You Need You will need a computer that can run your server and do other tasks such as compilation. You will also need a browser to test your application. I recommend a Linux-based computer such as Ubuntu, or a Mac as your development server, but with minor changes, you could also use a Windows PC. Running Node.js directly on Windows will also work, but the code samples in this book assume a Linux-based PC or Mac. If you choose to run directly on a Windows PC, you may have to make appropriate changes, especially when running commands in the shell, using a copy instead of using soft links, and in rare cases, to deal with \ vs. / in path separators. One option would be to try to run an Ubuntu server Virtual Machine (VM) using Vagrant (https:// www.vagrantup.com/). This is helpful because you will eventually deploy your code on a Linux-based server, and it is best to get used to that environment from the beginning. But you may find it difficult to edit files, because in an Ubuntu server, you only have a console. An Ubuntu desktop VM may work better for you, but it will need more memory. Further, to keep the book concise, I have not included installation instructions for packages, and they are different for different operating systems. You will need to follow the installation instructions from the package providers' websites. And in many cases, I have not included direct links to websites and I ask you to look them up. This is due to a couple of reasons. The first is to let you learn by yourself how to search for these. The second is that any link I provide may have moved to another location due to the fast-paced changes that the MERN stack was experiencing at the time of writing this book. MERN Components I'll give a quick introduction to the main components that form the MERN stack, and a few other libraries and tools that we'll be using to build our web application. I'll just touch upon the salient features and leave the details to other chapters where they are more appropriate. React React anchors the MERN stack. In some sense, this is the defining component of the MERN stack. React is an open-source JavaScript library maintained by Facebook that can be used for creating views rendered in HTML. Unlike AngularJS, React is not a framework. It is a library. Thus, it does not, by itself, dictate a framework pattern such as the MVC pattern. You use React to render a view (the V in MVC), but how to tie the rest of the application together is completely up to you. Not just Facebook itself, but there are many other companies that use React in production like Airbnb, Atlassian, Bitbucket, Disqus, Walmart, etc. The 120,000 stars on its GitHub repository is an indication of its popularity. I'll discuss a few things about React that make it stand out. Why Facebook Invented React The Facebook folks built React for their own use, and later they made it open-source. Now, why did they have to build a new library when there are tons of them out there? React was born not in the Facebook application that we all see, but in Facebook's Ads organization. Originally, they used a typical client-side MVC model to start with, which had all the regular two-way data binding and templates. Views would listen to changes on models, and they would respond to those changes by updating themselves. Soon, this got pretty hairy as the application became more and more complex. What would happen was that a change would cause an update, that would cause another update (because something changed Chapter 1 ■ Introduction 6 due to that update), which would cause yet another, and so on. Such cascading updates became difficult to maintain, because there would be subtle differences in the code to update the view, depending on the root cause of the update. Then they thought, why do we need to deal with all this, when all the code to depict the model in a view is already there? Aren't we replicating the code by adding smaller and smaller snippets to manage transitions? Why can't we use the templates (that is, the views) themselves to manage state changes? That's when they started thinking of building something that's declarative rather than imperative. Declarative React views are declarative. What this really means is that you, as a programmer, don't have to worry about managing the effect of changes in the view's state or the data. In other words, you don't worry about transitions or mutations in the DOM caused by changes to the view's state. Being declarative makes the views consistent, predictable, easier to maintain, and simpler to understand. It's someone else's problem to deal with transitions. How does this work? Let's compare how things work in React and how things work in the conventional approach, say, using jQuery. A React component declares how the view looks, given the data. When the data changes, if you are used to

the jQuery way of doing things, you'd typically do some DOM manipulation. If, for example, a new row has been inserted in a table, you'd create that DOM element and insert it using jQuery. But not in React. You just don't do anything! The React library figures out how the new view looks and renders that. Won't this be too slow? Will it not cause the entire screen to be refreshed on every data change? Well, React takes care of this using its Virtual DOM technology. You declare how the view looks and React builds a virtual representation, an in-memory data structure, out of it. I'll discuss more about this in Chapter 2, but for now, just think of the Virtual DOM as an intermediate representation, somewhere between an HTML and the actual DOM. When things change, React builds a new virtual DOM based on the new truth (state) and compares it with the old (before things changed) virtual DOM. React then computes the differences in the old and the changed Virtual DOM, then applies these changes to the actual DOM. Compared to manual updates as you would have performed in the jQuery way of doing things, this adds very little overhead because the algorithm to compute the differences in the Virtual DOM has been optimized to the hilt. Thus, we get the best of both worlds: not having to worry about implementing transitions and also the performance of minimal changes. Component-Based The fundamental building block of React is a component that maintains its own state and renders itself. In React, all you do is build components. Then, you put components together to make another component that depicts a complete view or page. A component encapsulates the state of data and the view, or how it is rendered. This makes writing and reasoning about the entire application easier, by splitting it into components and focusing on one thing at a time. Components talk to each other by sharing state information in the form of read-only properties to their child components and by callbacks to their parent components. I'll dig deeper into this concept in a later chapter, but the gist of it is that components in React are very cohesive, but the coupling with one another is minimal.