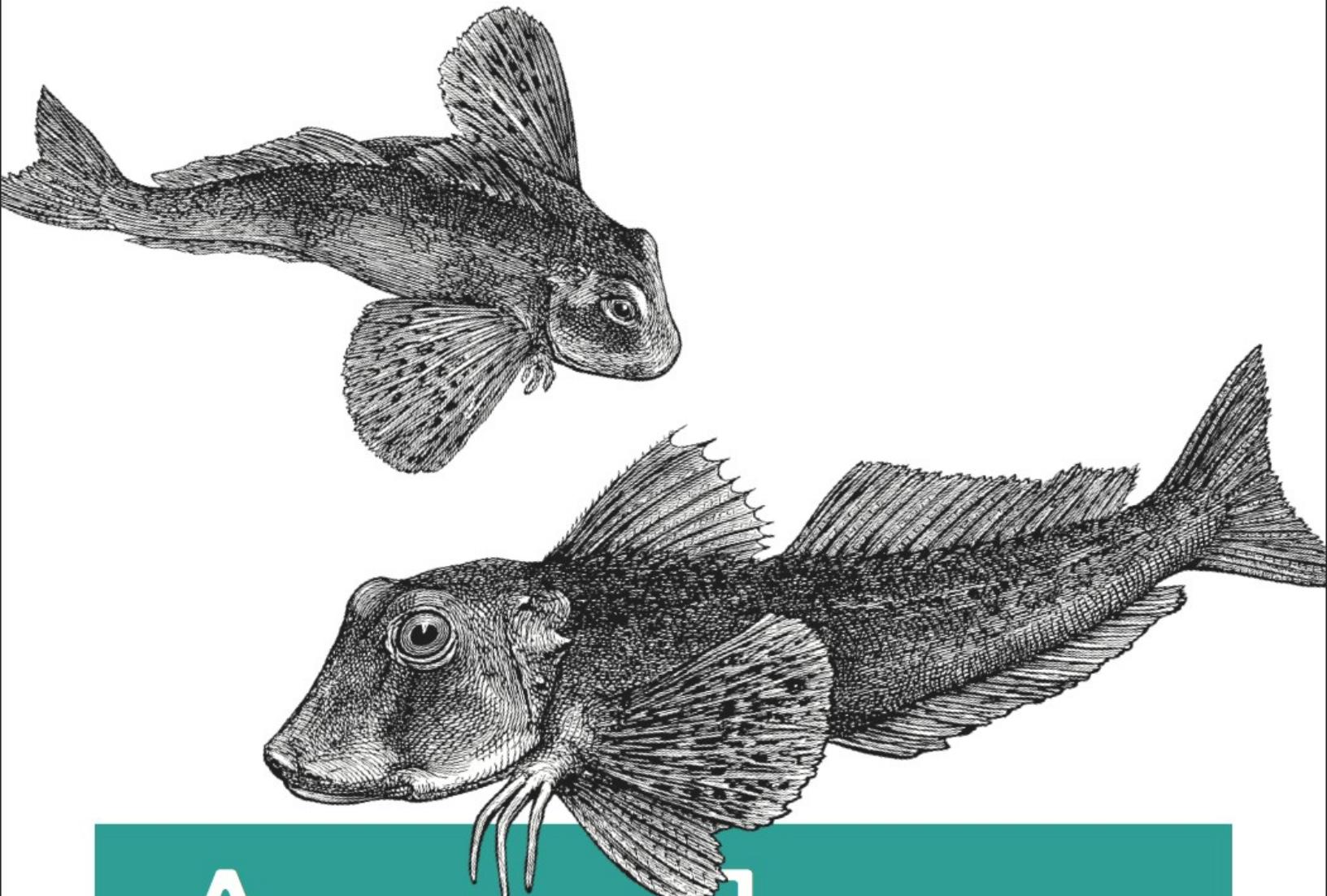


O'REILLY®



Angular Up & Running

LEARNING ANGULAR, STEP BY STEP

Shyam Seshadri

Angular: Up and Running

Angular: en funcionamiento

Learning Angular, Step by Step

Aprendiendo Angular, paso a paso

Shyam Seshadri

Shyam Seshadri

Angular: Up and Running

Angular: en funcionamiento

By Shyam Seshadri

Por Shyam Seshadri

Copyright © 2018 Shyam Seshadri. All rights reserved.

Copyright © 2018 Shyam Seshadri. Reservados todos los derechos.

Printed in the United States of America.

Impreso en los Estados Unidos de América.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North,
Sebastopol, CA 95472.

Publicado por O'Reilly Media, Inc., 1005 Gravenstein Highway North,
Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (<http://oreilly.com/safari>). For more information, contact our corporate/institutional sales department: 800-998-9938 or corporate@oreilly.com.

Los libros de O'Reilly se pueden comprar para uso educativo, comercial o promocional de ventas. También hay ediciones en línea disponibles para la mayoría de los títulos (<http://oreilly.com/safari>). Para obtener más información, comuníquese con nuestro departamento de ventas corporativo/institucional: 800-998-9938 o corporativo@oreilly.com.

Acquisitions Editor: Mary Treseler
Editora de adquisiciones: Mary Treseler

Developmental Editor: Angela Rufino
Editora de Desarrollo: Ángela Rufino

Production Editor: Kristen Brown
Editora de producción: Kristen Brown

Copyeditor: Kim Cofer
Editor: Kim Cofer

Proofreader: Jasmine Kwityn
Correctora: Jasmine Kwityn

Indexer: Ellen Troutman-Zaig
Indexador: Ellen Troutman-Zaig

Interior Designer: David Futato
Diseñador de interiores: David Futato

Cover Designer: Ellie Volckhausen
Diseñador de portada: Ellie Volckhausen

Illustrator: Rebecca Demarest
Ilustradora: Rebecca Demarest

June 2018: First Edition
Junio de 2018: Primera edición

Revision History for the First Edition **Historial de revisiones de la primera edición**

- 2018-05-31: First release
2018-05-31: Primer lanzamiento

See <http://oreilly.com/catalog/errata.csp?isbn=9781491999837> for release details.

Consulte <http://oreilly.com/catalog/errata.csp?isbn=9781491999837> para obtener detalles sobre la versión.

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. *Angular: Up and Running*, the cover image, and related trade dress are trademarks of O'Reilly Media, Inc.

El logotipo de O'Reilly es una marca registrada de O'Reilly Media, Inc. Angular: Up and Running, la imagen de portada y la imagen comercial relacionada son marcas comerciales de O'Reilly Media, Inc.

The views expressed in this work are those of the author, and do not represent the publisher's views. While the publisher and the author have used good faith efforts to ensure that the information and instructions contained in this work are accurate, the publisher and the author disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

Las opiniones expresadas en este trabajo son las del autor y no representan las opiniones del editor. Si bien el editor y el autor han realizado esfuerzos de buena fe para garantizar que la información y las instrucciones contenidas en este trabajo sean precisas, el editor y el autor renuncian a toda responsabilidad por errores u omisiones, incluida, entre otras, la responsabilidad por los daños resultantes del uso o confianza en este trabajo. El uso de la información y las instrucciones contenidas en este trabajo es bajo su propio riesgo. Si algún ejemplo de código u otra tecnología que este trabajo contiene o describe está sujeto a licencias de código abierto o derechos de propiedad intelectual de otros, es su responsabilidad asegurarse de que su uso cumpla con dichas licencias y/o derechos.

978-1-491-99983-7

[LSI]

[LSI]

Introduction

Introducción

It's funny that we constantly over- or underestimate the impact of certain events and projects in our lives. I seriously believed that the last project I worked on at Google, Google Feedback, would end up completely changing how the company interacted with its customers. And I believed Angular (AngularJS at the time) would just be another flash-in-the-pan, yet-another-framework that would not outlive the Feedback project's admin interface.

Es curioso que constantemente sobreestimemos o subestimemos el impacto de ciertos eventos y proyectos en nuestras vidas. Creía seriamente que el último proyecto en el que trabajé en Google, Google Feedback, acabaría cambiando por completo la forma en que la empresa interactuaba con sus clientes. Y creí que Angular (AngularJS en ese momento) sería simplemente otro flash-in-the-pan, otro marco más que no sobreviviría a la interfaz de administración del proyecto Feedback.

And in hindsight, it was exactly the other way around. While Feedback still exists and is baked into a lot of Google products, it is Angular that has gone from a tiny project used by one internal team at Angular to now being used by thousands of developers and companies worldwide. And a lot of it stems from Misko, Igor, and the entire team around it, and their unerring dedication to improving how we develop web applications.

Y en retrospectiva, fue exactamente al revés. Si bien los comentarios todavía existen y están integrados en muchos productos de Google,

es Angular el que ha pasado de ser un pequeño proyecto utilizado por un equipo interno de Angular a ser utilizado ahora por miles de desarrolladores y empresas en todo el mundo. Y mucho de esto proviene de Misko, Igor y todo el equipo que lo rodea, y su infalible dedicación para mejorar la forma en que desarrollamos aplicaciones web.

What started off as a two-member project is now one of the largest open source communities on the web, and the framework has impacted and been a part of thousands of projects across the world. There are dozens of books, hundreds of tutorials, and thousands of articles on Angular, and Angular's adoption and support continues to grow each day.

Lo que comenzó como un proyecto de dos miembros es ahora una de las comunidades de código abierto más grandes en la web, y el marco ha impactado y ha sido parte de miles de proyectos en todo el mundo. Hay docenas de libros, cientos de tutoriales y miles de artículos sobre Angular, y la adopción y el soporte de Angular continúan creciendo cada día.

Some of the major concepts that were ahead of their time during the first version of Angular (like data binding, separation of concerns, dependency injection, etc.) are now common features of new frameworks.

Algunos de los conceptos principales que se adelantaron a su tiempo durante la primera versión de Angular (como enlace de datos, separación de preocupaciones, inyección de dependencia, etc.) ahora son características comunes de los nuevos marcos.

The biggest change to the AngularJS ecosystem has been the release of the new version of Angular (initially called Angular 2.0, now just called Angular). It was a drastic, non-backward-compatible change that almost divided an entire community. But with community engagement and an open, inclusive team, what could

have been a disastrous step turned out to be a much needed overhaul of Angular to bring it to the new age of web development.

El mayor cambio en el ecosistema AngularJS ha sido el lanzamiento de la nueva versión de Angular (inicialmente llamada Angular 2.0, ahora simplemente llamada Angular). Fue un cambio drástico, no compatible con versiones anteriores, que casi dividió a toda una comunidad. Pero con la participación de la comunidad y un equipo abierto e inclusivo, lo que podría haber sido un paso desastroso resultó ser una revisión muy necesaria de Angular para llevarlo a la nueva era del desarrollo web.

Truly, what makes Angular a great technology and framework is the community around it—those who contribute to the core framework, or develop plug-ins for it, as well as those who use it on a day-to-day basis.

En verdad, lo que hace que Angular sea una gran tecnología y marco es la comunidad que lo rodea: aquellos que contribuyen al marco central o desarrollan complementos para él, así como aquellos que lo usan en el día a día.

As part of the community, I am truly excited to present this book, and contribute in my own way to what makes this community great.

Como parte de la comunidad, estoy realmente emocionado de presentar este libro y contribuir a mi manera a lo que hace grande a esta comunidad.

Who Should Read This Book

Quién debería leer este libro

This book is for anyone who is looking to get started with Angular (2.0 and onward), whether as a side project, as an additional tool, or for their main work. It is expected that readers are comfortable with

JavaScript and HTML before starting this book, but a basic knowledge of JavaScript should be sufficient to learn Angular. Knowledge of AngularJS 1.0 is not needed or expected.

Este libro es para cualquiera que esté buscando comenzar con Angular (2.0 y posteriores), ya sea como un proyecto paralelo, como una herramienta adicional o para su trabajo principal. Se espera que los lectores se sientan cómodos con JavaScript y HTML antes de comenzar este libro, pero un conocimiento básico de JavaScript debería ser suficiente para aprender Angular. No es necesario ni esperado tener conocimientos de AngularJS 1.0.

We will also use TypeScript, which is the recommended way of developing in Angular, but a preliminary knowledge is sufficient to read this book.

También usaremos TypeScript, que es la forma recomendada de desarrollar en Angular, pero un conocimiento previo es suficiente para leer este libro.

We will take it step by step, so relax and have fun learning with me.

Lo iremos paso a paso, así que relájate y diviértete aprendiendo conmigo.

Why I Wrote This Book

Por qué escribí este libro

Angular as a framework has grown immensely, and comes with a large set of features and capabilities. With a large community behind it, it also comes with an influx of helpful content. But the help content, tutorials, and guides are either focused only on particular topics, or sporadic and not necessarily useful for someone getting started.

Angular como marco ha crecido enormemente y viene con un gran conjunto de características y capacidades. Con una gran comunidad detrás, también viene con una gran cantidad de contenido útil. Pero el contenido de ayuda, los tutoriales y las guías se centran sólo en temas concretos o son esporádicos y no necesariamente útiles para alguien que está empezando.

The aim of this book is to provide a step-by-step guide on getting started with Angular. Each concept is provided in a logical, organized fashion, building on top of the previous one. With so many moving parts and an active community, this book does not intend to cover each and every aspect, but instead focuses on the core building blocks in a detailed fashion while letting readers discover the rest on their own.

El objetivo de este libro es proporcionar una guía paso a paso sobre cómo empezar a utilizar Angular. Cada concepto se proporciona de forma lógica y organizada, basándose en el anterior. Con tantas partes móviles y una comunidad activa, este libro no pretende cubrir todos y cada uno de los aspectos, sino que se centra en los componentes básicos de forma detallada, dejando que los lectores descubran el resto por sí mismos.

At the end of the book, you should be familiar with a majority of the Angular framework, and be able to use Angular to develop your own web applications and use it in your own projects.

Al final del libro, debería estar familiarizado con la mayor parte del marco Angular y poder utilizar Angular para desarrollar sus propias aplicaciones web y utilizarlo en sus propios proyectos.

A Word on Web Application Development

Today

Unas palabras sobre el desarrollo de aplicaciones web hoy

JavaScript has come a long way, to the point where it is one of the most widely used and adopted programming languages. Nowadays, it's rare for web developers to have to worry about browser inconsistencies and the like, which was the primary reason for frameworks like jQuery to have existed.

JavaScript ha recorrido un largo camino, hasta el punto de que es uno de los lenguajes de programación más utilizados y adoptados. Hoy en día, es raro que los desarrolladores web tengan que preocuparse por las inconsistencias del navegador y cosas similares, que fue la razón principal por la que existieron marcos como jQuery.

Frameworks (like Angular and React) are now a very common choice for developing frontend experiences, and it is rare for anyone nowadays to decide to build a frontend application without leveraging one.

Los marcos (como Angular y React) son ahora una opción muy común para desarrollar experiencias frontend, y hoy en día es raro que alguien decida crear una aplicación frontend sin aprovecharla.

The advantages of using frameworks are manifold, from reducing boilerplate code, to providing a consistent structure and layout for developing an application to many more. The primary intent is always to reduce the time spent on cruft, and focus more on the major functionality we want to provide. And if it works across browsers (and platforms, like Android and iOS, in addition to desktop), then more power to it.

Las ventajas de utilizar marcos son múltiples, desde reducir el código repetitivo hasta proporcionar una estructura y un diseño

consistentes para desarrollar una aplicación y muchos más. La intención principal es siempre reducir el tiempo dedicado a la cruft y centrarse más en la funcionalidad principal que queremos proporcionar. Y si funciona en todos los navegadores (y plataformas, como Android e iOS, además del escritorio), entonces tendrá más potencia.

Angular (as well as other frameworks) provides this, primarily through some core fundamentals that are at the heart of the framework, including:

Angular (así como otros marcos) proporciona esto, principalmente a través de algunos fundamentos básicos que están en el corazón del marco, que incluyen:

- Powerful templating syntax driven by declarative programming

Potente sintaxis de plantillas impulsada por programación declarativa

- Modularity and separation of concerns

Modularidad y separación de preocupaciones.

- Data binding, and through it, data-driven programming

Enlace de datos y, a través de él, programación basada en datos.

- Testability and awesome testing support

Capacidad de prueba y excelente soporte de pruebas

- Routing and navigation

Enrutamiento y navegación

- And a host of other features, from server-side rendering, to the ability to write native mobile applications, and much more!

¡Y una serie de otras características, desde renderizado del lado del servidor hasta la capacidad de escribir aplicaciones móviles

nativas y mucho más!

With the help of Angular, we can focus on building amazing experiences, while managing complexity and maintainability in a seamless fashion.

Con la ayuda de Angular, podemos centrarnos en crear experiencias increíbles y, al mismo tiempo, gestionar la complejidad y la mantenibilidad de forma fluida.

Navigating This Book

Navegando por este libro

This book aims to walk a developer through each part of Angular, step by step. Each chapter that introduces a new concept will be immediately followed by a chapter on how we can unit test it. The book is roughly organized as follows:

Este libro tiene como objetivo guiar al desarrollador a través de cada parte de Angular, paso a paso. Cada capítulo que introduce un nuevo concepto será seguido inmediatamente por un capítulo sobre cómo podemos realizar pruebas unitarias. El libro está organizado aproximadamente de la siguiente manera:

- **Chapter 1, *Introducing Angular*,** is an introduction to Angular as well as the concepts behind it. It also covers what it takes to start writing an Angular application.

El Capítulo 1, Introducción a Angular, es una introducción a Angular y a los conceptos detrás de él. También cubre lo que se necesita para comenzar a escribir una aplicación Angular.

- **Chapter 2, *Hello Angular*,** walks through creating a very simple Angular application, and diving into how the pieces work together. It also introduces the Angular CLI.

El Capítulo 2, Hola Angular, explica cómo crear una aplicación Angular muy simple y profundiza en cómo funcionan juntas las piezas. También presenta Angular CLI.

- **Chapter 3, *Useful Built-In Angular Directives*,** digs into the basic built-in Angular directives (including `ngFor`, `ngIf`, etc.) and when and how to use them.

El Capítulo 3, Directivas angulares integradas útiles, profundiza en las directivas angulares integradas básicas (incluidas `ngFor`, `ngIf`, etc.) y cuándo y cómo usarlas.

- **Chapter 4, *Understanding and Using Angular Components*,** covers Angular components in more detail, as well as the various options available when creating them. It also covers the basic lifecycle hooks available with components.

El Capítulo 4, Comprensión y uso de componentes angulares, cubre los componentes angulares con más detalle, así como las diversas opciones disponibles al crearlos. También cubre los ganchos básicos del ciclo de vida disponibles con componentes.

- **Chapter 5, *Testing Angular Components*,** introduces how to unit test angular components using Karma and Jasmine, along with the Angular testing framework.

El Capítulo 5, Prueba de componentes angulares, presenta cómo realizar pruebas unitarias de componentes angulares utilizando Karma y Jasmine, junto con el marco de pruebas Angular.

- **Chapter 6, *Working with Template-Driven Forms*,** covers creating and working with forms in Angular, specifically template-driven forms.

El Capítulo 6, Trabajar con formularios basados en plantillas, cubre la creación y el trabajo con formularios en Angular, específicamente formularios basados en plantillas.

- **Chapter 7, Working with Reactive Forms**, covers the other way of defining and working with forms, which is how to create and develop reactive forms.

El Capítulo 7, Trabajar con formularios reactivos, cubre la otra forma de definir y trabajar con formularios, que es cómo crear y desarrollar formularios reactivos.

- **Chapter 8, Angular Services**, covers Angular services, which includes how to use built-in Angular services, as well as how and when to define our own Angular services.

El Capítulo 8, Servicios Angular, cubre los servicios Angular, que incluye cómo utilizar los servicios Angular integrados, así como cómo y cuándo definir nuestros propios servicios Angular.

- **Chapter 9, Making HTTP Calls in Angular**, moves into the server communication aspect of Angular, and delves into making HTTP calls, as well as some advanced topics like interceptors and the like.

El Capítulo 9, Realizar Llamadas HTTP en Angular, avanza hacia el aspecto de comunicación del servidor de Angular y profundiza en la realización de llamadas HTTP, así como en algunos temas avanzados como interceptores y similares.

- **Chapter 10, Unit Testing Services**, takes a step back and covers unit testing again, but this time with a focus on unit testing services. This includes testing simple services and slightly harder cases like asynchronous flows as well as services and components that make HTTP calls.

El Capítulo 10, Servicios de pruebas unitarias, da un paso atrás y cubre las pruebas unitarias nuevamente, pero esta vez con un enfoque en los servicios de pruebas unitarias. Esto incluye probar servicios simples y casos un poco más difíciles, como flujos asincrónicos, así como servicios y componentes que realizan llamadas HTTP.

- **Chapter 11, *Routing in Angular***, goes in depth into how we can accomplish routing in an Angular application and covers the Angular routing module in detail as well as a majority of its features.

El Capítulo 11, Enrutamiento en Angular, profundiza en cómo podemos lograr el enrutamiento en una aplicación Angular y cubre el módulo de enrutamiento Angular en detalle, así como la mayoría de sus características.

- **Chapter 12, *Productionizing an Angular App***, finally brings together all the concepts and covers taking the developed Angular application to production and the various concerns and techniques involved in the same.

El Capítulo 12, Producción de una aplicación Angular, finalmente reúne todos los conceptos y cubre cómo llevar la aplicación Angular desarrollada a producción y las diversas preocupaciones y técnicas involucradas en la misma.

The entire code repository is hosted on GitHub, so if you don't want to type in the code examples from this book, or want to ensure that you are looking at the latest and greatest code examples, visit the repository and **grab the contents**.

Todo el repositorio de código está alojado en GitHub, por lo que si no desea escribir los ejemplos de código de este libro, o desea asegurarse de que está viendo los mejores y más recientes ejemplos de código, visite el repositorio y obtenga el contenido.

This book uses AngularJS version 5.0.0 for all its code examples.

Este libro utiliza AngularJS versión 5.0.0 para todos sus ejemplos de código.

Online Resources

Recursos en línea

The following resources are a great starting point for any AngularJS developer, and should be always available at your fingertips:

Los siguientes recursos son un excelente punto de partida para cualquier desarrollador de AngularJS y siempre deben estar disponibles a su alcance:

- [The Official Angular API Documentation](#)
La documentación oficial de Angular API
- [The Official Angular Quickstart Guide](#)
La guía oficial de inicio rápido de Angular
- [The Angular Heroes Tutorial App](#)
La aplicación tutorial de Angular Heroes

Conventions Used in This Book

Las convenciones usadas en este libro

The following typographical conventions are used in this book:

En este libro se utilizan las siguientes convenciones tipográficas:

Italic Itálico

Indicates new terms, URLs, email addresses, filenames, and file extensions.

Indica nuevos términos, URL, direcciones de correo electrónico, nombres de archivos y extensiones de archivos.

Constant width Ancho constante

Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

Se utiliza para listados de programas, así como dentro de párrafos para hacer referencia a elementos del programa, como nombres de funciones o variables, bases de datos, tipos de datos, variables de entorno, declaraciones y palabras clave.

Constant width bold

Shows commands or other text that should be typed literally by the user.

Muestra comandos u otro texto que el usuario debe escribir literalmente.

Constant width italic

Shows text that should be replaced with user-supplied values or by values determined by context.

Muestra texto que debe reemplazarse con valores proporcionados por el usuario o por valores determinados por el contexto.

**TIP
CONSEJO**

This element signifies a tip or suggestion.

Este elemento significa un consejo o sugerencia.

NOTE NOTA

This element signifies a general note.

Este elemento significa una nota general.

WARNING ADVERTENCIA

This element indicates a warning or caution.

Este elemento indica una advertencia o precaución.

Using Code Examples

Usando ejemplos de código

Supplemental material (code examples, exercises, etc.) is available for download at <https://github.com/shyamseshadri/angular-up-and-running>.

El material complementario (ejemplos de código, ejercicios, etc.) está disponible para descargar en <https://github.com/shyamseshadri/angular-up-and-running>.

This book is here to help you get your job done. In general, if example code is offered with this book, you may use it in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting

example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

Este libro está aquí para ayudarle a realizar su trabajo. En general, si se ofrece código de ejemplo con este libro, puede utilizarlo en sus programas y documentación. No es necesario que se comunique con nosotros para solicitar permiso a menos que esté reproduciendo una parte importante del código. Por ejemplo, escribir un programa que utilice varios fragmentos de código de este libro no requiere permiso. Vender o distribuir un CD-ROM con ejemplos de libros de O'Reilly requiere permiso. Responder una pregunta citando este libro y citando código de ejemplo no requiere permiso. Incorporar una cantidad significativa de código de ejemplo de este libro en la documentación de su producto requiere permiso.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example:

"Angular: Up and Running by Shyam Seshadri (O'Reilly). Copyright 2018 Shyam Seshadri, 978-1-491-99983-7."

Apreciamos la atribución, pero no la exigimos. Una atribución suele incluir el título, el autor, la editorial y el ISBN. Por ejemplo: "Angular: en funcionamiento de Shyam Seshadri (O'Reilly). Copyright 2018 Shyam Seshadri, 978-1-491-99983-7."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at permissions@oreilly.com.

Si cree que su uso de ejemplos de código queda fuera del uso legítimo o del permiso otorgado anteriormente, no dude en contactarnos en permisos@oreilly.com.

O'Reilly Safari

O'Reilly Safari

Safari (formerly Safari Books Online) is a membership-based training and reference platform for enterprise, government, educators, and individuals.

Safari (anteriormente Safari Books Online) es una plataforma de referencia y capacitación basada en membresías para empresas, gobiernos, educadores e individuos.

Members have access to thousands of books, training videos, Learning Paths, interactive tutorials, and curated playlists from over 250 publishers, including O'Reilly Media, Harvard Business Review, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Adobe, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett, and Course Technology, among others.

Los miembros tienen acceso a miles de libros, videos de capacitación, rutas de aprendizaje, tutoriales interactivos y listas de reproducción seleccionadas de más de 250 editoriales, incluidas O'Reilly Media, Harvard Business Review, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que , Peachpit Press, Adobe, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett y Course Tecnología, entre otros.

For more information, please visit <http://oreilly.com/safari>.

Para obtener más información, visite <http://oreilly.com/safari>.

How to Contact Us

Cómo contactarnos

Please address comments and questions concerning this book to the publisher:

Por favor dirija sus comentarios y preguntas sobre este libro al editor:

O'Reilly Media, Inc.
O'Reilly Media, Inc.

1005 Gravenstein Highway North
1005 Gravenstein Carretera Norte

Sebastopol, CA 95472
Sebastopol, CA 95472

800-998-9938 (in the United States or Canada)
800-998-9938 (en Estados Unidos o Canadá)

707-829-0515 (international or local)
707-829-0515 (internacional o local)

707-829-0104 (fax)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at
<http://bit.ly/angularUR>.

Tenemos una página web para este libro, donde enumeramos erratas, ejemplos y cualquier información adicional. Puede acceder a esta página en <http://bit.ly/angularUR>.

To comment or ask technical questions about this book, send email to bookquestions@oreilly.com.

Para comentar o hacer preguntas técnicas sobre este libro, envíe un correo electrónico a bookquestions@oreilly.com.

For more information about our books, courses, conferences, and news, see our website at <http://www.oreilly.com>.

Para obtener más información sobre nuestros libros, cursos, conferencias y noticias, consulte nuestro sitio web en <http://www.oreilly.com>.

Find us on Facebook: <http://facebook.com/oreilly>

Encuéntrenos en Facebook: <http://facebook.com/oreilly>

Follow us on Twitter: <http://twitter.com/oreillymedia>

Síguenos en Twitter: <http://twitter.com/oreillymedia>

Watch us on YouTube: <http://www.youtube.com/oreillymedia>

Míranos en YouTube: <http://www.youtube.com/oreillymedia>

Acknowledgments

Expresiones de gratitud

This book is dedicated to my wife, Sanchita, and my parents and grandmom who were my rock as well as my motivation to write this book in the best manner I could, all the while balancing my own fledgling startup in its most precarious time (the beginning!).

Este libro está dedicado a mi esposa, Sanchita, y a mis padres y mi abuela, quienes fueron mi apoyo y mi motivación para escribir este libro de la mejor manera posible, mientras equilibraba mi propia incipiente startup en su momento más precario (el icomienzo!).

I'd also like to thank my reviewers, Yakov Fain and Victor Mejia, who had to read and review my unedited ramblings and make sure I got my point across in the most succinct and understandable terms.

También me gustaría agradecer a mis revisores, Yakov Fain y Víctor Mejía, quienes tuvieron que leer y revisar mis divagaciones sin editar y asegurarse de que expresara mi punto de vista en los términos más concisos y comprensibles.

This book of course wouldn't be possible without the faith and efforts of the awesome team at O'Reilly, especially Angela and Kristen!

Por supuesto, este libro no sería posible sin la fe y los esfuerzos del increíble equipo de O'Reilly, especialmente Angela y Kristen!

And finally, thank you to the amazing Angular community for all their contributions, feedback, and support, and for teaching us how to use and make it better.

Y finalmente, gracias a la increíble comunidad de Angular por todas sus contribuciones, comentarios y apoyo, y por enseñarnos cómo usarlo y mejorarlo.

Chapter 1. Introducing Angular

Capítulo 1. Introducción a Angular

Our expectations of what we can perform on the web (and by web here, I mean both desktop as well as the mobile web) has increased to the point where what used to be full-fledged native desktop applications are run on the browser. Web applications now resemble desktop native applications in scope and complexity, which also results in added complexity as a developer.

Nuestras expectativas sobre lo que podemos realizar en la web (y por web aquí me refiero tanto a la web de escritorio como a la web móvil) han aumentado hasta el punto en que lo que solían ser aplicaciones de escritorio nativas y completas se ejecutan en el navegador. Las aplicaciones web ahora se parecen a las aplicaciones nativas de escritorio en alcance y complejidad, lo que también resulta en una mayor complejidad como desarrollador.

Furthermore, Single-Page Applications (SPAs) have become a very common choice in building out frontend experiences, as they allow for great customer experiences in terms of speed and responsiveness. Once the initial application has loaded into a customer's browser, further interactions only have to worry about loading the additional data needed, without reloading the entire page as was the norm with server-side rendered pages of the past.

Además, las aplicaciones de una sola página (SPA) se han convertido en una opción muy común a la hora de crear experiencias frontend, ya que permiten ofrecer excelentes experiencias a los clientes en términos de velocidad y capacidad de respuesta. Una vez que la aplicación inicial se ha cargado en el navegador del cliente, las interacciones posteriores solo tienen que preocuparse por cargar los datos adicionales necesarios, sin recargar toda la página como era la norma con las páginas renderizadas del lado del servidor en el pasado.

AngularJS was started to first bring structure and consistency to single-page web application development, while providing a way to quickly develop scalable and maintainable web applications. In the time since it was released, the web and browsers have moved forward by leaps and bounds, and some of the problems that AngularJS was solving weren't as relevant anymore.

AngularJS se inició para aportar estructura y coherencia al desarrollo de aplicaciones web de una sola página, al tiempo que proporcionaba una forma de desarrollar rápidamente aplicaciones web escalables y mantenibles. Desde su lanzamiento, la web y los navegadores han avanzado a pasos agigantados, y algunos de los problemas que AngularJS estaba resolviendo ya no eran tan relevantes.

Angular then was basically a completely new rewritten version of the framework, built for the new-age web. It leveraged a lot of the newer advances, from modules to web components, while improving the existing features of AngularJS, like dependency injection and templating.

Angular entonces era básicamente una versión reescrita completamente nueva del marco, creada para la web de la nueva era. Aprovechó muchos de los avances más recientes, desde módulos hasta componentes web, al tiempo que mejoró las

características existentes de AngularJS, como la inyección de dependencias y las plantillas.

TIP
CONSEJO

From now on, when I say AngularJS, I refer to the original AngularJS framework, the 1.0 version. Whenever I mention Angular, it refers to the newer framework, from 2.0 onward. This is primarily because Angular 2.0 onward does not predicate itself to using only JavaScript, but also supports writing applications in TypeScript.

De ahora en adelante, cuando digo AngularJS, me refiero al framework original de AngularJS, la versión 1.0. Siempre que menciono Angular, me refiero al marco más nuevo, desde 2.0 en adelante. Esto se debe principalmente a que Angular 2.0 en adelante no se limita a usar solo JavaScript, sino que también admite la escritura de aplicaciones en TypeScript.

Why Angular

Por qué angular

Angular as a framework provides a few significant advantages while also providing a common structure for developers on a team to work with. It allows us to develop large applications in a maintainable manner. We will dig into each one of these in more detail in the following chapters:

Angular como marco proporciona algunas ventajas significativas y al mismo tiempo proporciona una estructura común para que los desarrolladores de un equipo trabajen. Nos permite desarrollar grandes aplicaciones de forma mantenible. Profundizaremos en cada uno de estos con más detalle en los siguientes capítulos:

Custom components

Componentes personalizados

Angular allows you to build your own declarative components that can pack functionality along with its rendering logic into bite-sized, reusable pieces. It also plays well with web components.

Angular le permite crear sus propios componentes declarativos que pueden empaquetar la funcionalidad junto con su lógica de representación en piezas reutilizables del tamaño de un bocado. También funciona bien con componentes web.

Data binding

El enlace de datos

Angular allows you to seamlessly move your data from your core JavaScript code to the view, and react to view events without having to write the glue code yourself.

Angular le permite mover sin problemas sus datos desde su código JavaScript principal a la vista y reaccionar para ver eventos sin tener que escribir el código adhesivo usted mismo.

Dependency injection

Inyección de dependencia

Angular allows you to write modular services, and have them injected wherever they are needed. This greatly improves the testability and reusability of the same.

Angular le permite escribir servicios modulares e injectarlos donde sea necesario. Esto mejora enormemente la capacidad de prueba y reutilización del mismo.

Testing

Pruebas

Tests are first-class citizens, and Angular has been built from the ground up with testability in mind. You can (and should!) test every part of your application.

Las pruebas son ciudadanos de primera clase y Angular se creó desde cero teniendo en cuenta la capacidad de prueba. Puede (y debe!) probar cada parte de su aplicación.

Comprehensive

Integral

Angular is a full-fledged framework, and provides out-of-the-box solutions for server communication, routing within your application, and more.

Angular es un marco completo y proporciona soluciones listas para usar para la comunicación del servidor, el enrutamiento dentro de su aplicación y más.

NOTE NOTA

Angular as a framework has adopted semantic versioning for all new releases. Furthermore, the core team has an aggressive roadmap, with a new major release planned every six months. Thus, what started off as Angular 2 is now referred to as just Angular, since we don't want to call them Angular 2, Angular 4, Angular 5, and so on.

Angular como marco ha adoptado versiones semánticas para todos los lanzamientos nuevos. Además, el equipo central tiene una hoja de ruta agresiva, con un nuevo lanzamiento importante planificado cada seis meses. Por lo tanto, lo que comenzó como Angular 2 ahora se conoce simplemente como Angular, ya que no queremos llamarlos Angular 2, Angular 4, Angular 5, etc.

That said, unlike AngularJS to Angular, upgrading between versions of Angular (say 2 to 4, etc.) is an incremental step, and more often than not an almost trivial upgrade. So you don't need to worry about having to do a major upgrade every few months with drastic code changes.

Dicho esto, a diferencia de AngularJS a Angular, la actualización entre versiones de Angular (digamos 2 a 4, etc.) es un paso incremental y, en la mayoría de los casos, una actualización casi trivial. Por lo tanto, no necesita preocuparse por tener que realizar una actualización importante cada pocos meses con cambios drásticos en el código.

What This Book Will Not Cover

Lo que este libro no cubrirá

While Angular as a framework is quite large, the community around it is even larger. A lot of great features and options for use with Angular in fact stem from this community. This makes life harder as an author to figure out how to write a book that preps you, the reader, as an Angular developer, while still limiting the scope to what I think are the essentials.

Si bien Angular como marco es bastante grande, la comunidad que lo rodea es aún mayor. De hecho, muchas características y opciones excelentes para usar con Angular provienen de esta comunidad. Esto hace que la vida como autor sea más difícil para descubrir cómo escribir un libro que lo prepare a usted, el lector, como desarrollador de Angular, y al mismo tiempo limitar el alcance a lo que creo que son lo esencial.

To that extent, while Angular can be extended in so many ways, from writing native mobile apps using Angular (see [NativeScript](#)), rendering your Angular application on the server (see [Angular Universal](#)), using Redux as a first-class option in Angular (multiple options; see [ngrx](#)), and many more, the initial version of the book will only focus on the core Angular platform and all the capabilities it provides. It will also strive to focus on the more common cases rather than cover every single feature and capability of Angular, as such a book would run into thousands of pages.

Hasta ese punto, si bien Angular se puede ampliar de muchas maneras, desde escribir aplicaciones móviles nativas usando Angular (ver [NativeScript](#)), renderizar su aplicación Angular en el servidor (ver [Angular Universal](#)), usar Redux como una opción de primera clase en Angular (múltiples opciones; consulte [ngrx](#)) y muchas más,

la versión inicial del libro solo se centrará en la plataforma central Angular y todas las capacidades que proporciona. También se esforzará por centrarse en los casos más comunes en lugar de cubrir todas las características y capacidades de Angular, ya que un libro de este tipo tendría miles de páginas.

The intention is to focus on the parts that will be necessary and useful to all Angular developers, rather than focus on bits and parts that would be useful to a subset.

La intención es centrarse en las partes que serán necesarias y útiles para todos los desarrolladores de Angular, en lugar de centrarse en partes y partes que serían útiles para un subconjunto.

Getting Started with Your Development Environment

Comenzando con su entorno de desarrollo

Angular expects you to do a fair bit of groundwork to be able to develop seamlessly on your computer. Certain prerequisites need to be installed that we will cover in this section.

Angular espera que usted haga un poco de trabajo preliminar para poder desarrollarse sin problemas en su computadora. Es necesario instalar ciertos requisitos previos que cubriremos en esta sección.

Node.js

Nodo.js

While you will never be coding in Node.js, Angular uses Node.js as its base for a large part of its build environment. Thus, to get started with Angular, you will need to have Node.js installed on your

environment. There are multiple ways to install Node.js, so please refer to the [Node.js Download Page](#) for more instructions.

Si bien nunca codificará en Node.js, Angular usa Node.js como base para gran parte de su entorno de compilación. Por lo tanto, para comenzar con Angular, necesitará tener instalado Node.js en su entorno. Hay varias formas de instalar Node.js, así que consulte la página de descarga de Node.js para obtener más instrucciones.

WARNING ADVERTENCIA

On macOS, installing Node.js through Homebrew has been known to cause some issues. So try installing it directly if you run into any problems.

En macOS, se sabe que la instalación de Node.js a través de Homebrew causa algunos problemas. Intente instalarlo directamente si tiene algún problema.

You need to install version 6.9.0 or above of Node.js, and version 3.0.0 or above of npm. You can confirm your versions after installing by running the following commands:

Debe instalar la versión 6.9.0 o superior de Node.js y la versión 3.0.0 o superior de npm. Puede confirmar sus versiones después de la instalación ejecutando los siguientes comandos:

```
node --version
npm --v
```

TypeScript

Mecanografiado

TypeScript adds a set of types to the JavaScript code that we write, allowing us to write JavaScript that is easier to understand, reason

about, and trace. It ensures that the latest proposed ECMAScript features are also available at the tip of our fingers. At the end of the day, all your TypeScript code compiles down to JavaScript that can run easily in any environment.

TypeScript agrega un conjunto de tipos al código JavaScript que escribimos, lo que nos permite escribir JavaScript que es más fácil de entender, razonar y rastrear. Garantiza que las últimas funciones ECMAScript propuestas también estén disponibles al alcance de nuestros dedos. Al final del día, todo su código TypeScript se compila en JavaScript que puede ejecutarse fácilmente en cualquier entorno.

TypeScript is not mandatory for developing an Angular application, but it is highly recommended, as it offers some syntactic sugar, as well as makes the codebase easier to understand and maintain. In this book, we will be using TypeScript to develop Angular applications.

TypeScript no es obligatorio para desarrollar una aplicación Angular, pero es muy recomendable, ya que ofrece algo de azúcar sintáctico y hace que el código base sea más fácil de entender y mantener. En este libro, usaremos TypeScript para desarrollar aplicaciones Angular.

TypeScript is installed as an NPM package, and thus can be simply installed with the following command:

TypeScript se instala como un paquete NPM y, por lo tanto, se puede instalar simplemente con el siguiente comando:

```
npm install -g typescript
```

Make sure you install at least version 2.4.0 or above.

Asegúrese de instalar al menos la versión 2.4.0 o superior.

While we will be covering most of the basic features/concepts that we use from TypeScript, it is always a good idea to learn more from the [official TypeScript documentation](#).

Si bien cubriremos la mayoría de las características/conceptos básicos que utilizamos de TypeScript, siempre es una buena idea aprender más de la documentación oficial de TypeScript.

Angular CLI

CLI angular

Unlike AngularJS, where it was easy to source one file as a dependency and be up and running, Angular has a slightly more complicated setup. To this extent, the Angular team has created a command-line interface (CLI) tool to make it easier to bootstrap and develop your Angular applications.

A diferencia de AngularJS, donde era fácil obtener un archivo como dependencia y estar en funcionamiento, Angular tiene una configuración un poco más complicada. En este sentido, el equipo de Angular ha creado una herramienta de interfaz de línea de comandos (CLI) para facilitar el arranque y el desarrollo de sus aplicaciones Angular.

As it significantly helps making the process of development easier, I recommend using it at the very least for your initial projects until you get the hang of all the things it does and are comfortable doing it yourself. In this book, we will cover both the CLI command as well as the actions it performs underneath, so that you get a good understanding of all the changes needed.

Como ayuda significativamente a facilitar el proceso de desarrollo, recomiendo usarlo al menos para sus proyectos iniciales hasta que domine todas las cosas que hace y se sienta cómodo haciéndolo usted mismo. En este libro, cubriremos tanto el comando CLI como las acciones que realiza debajo, para que comprenda bien todos los cambios necesarios.

Installing the latest version (1.7.3 at the time of writing this book) is as simple as running the following command:

Instalar la última versión (1.7.3 en el momento de escribir este libro) es tan simple como ejecutar el siguiente comando:

```
npm install -g @angular/cli
```

TIP CONSEJO

If you are scratching your head at this newfangled naming convention for Angular packages, the new syntax is a feature of NPM called *scoped packages*. It allows packages to be grouped together within NPM under a single folder. You can read more [here](#).

Si se está rascando la cabeza ante esta novedosa convención de nomenclatura para paquetes Angular, la nueva sintaxis es una característica de NPM llamada paquetes con alcance. Permite agrupar paquetes dentro de NPM en una sola carpeta. Puede leer más aquí.

Once installed, you can confirm if it was successful by running the following command:

Una vez instalado, puede confirmar si fue exitoso ejecutando el siguiente comando:

```
ng --version
```

Getting the Codebase

Obtener el código base

All the examples from this book, along with the exercises and the final solution, are hosted as a Git repository. While it is not mandatory to download this, you can choose to do so if you want a reference or want to play around with the samples in this book. You

can do so by cloning the Git repository by running the following command:

Todos los ejemplos de este libro, junto con los ejercicios y la solución final, están alojados en un repositorio Git. Si bien no es obligatorio descargarlo, puedes hacerlo si quieres una referencia o quieres jugar con los ejemplos de este libro. Puede hacerlo clonando el repositorio de Git ejecutando el siguiente comando:

```
git clone https://github.com/shyamseshadri/angular-up-and-running.git
```

This will create a folder called *angular-up-and-running* in your current working directory with all the necessary examples. Within this directory you'll find subfolders containing the examples, organized by chapter.

Esto creará una carpeta llamada *angular-up-and-running* en su directorio de trabajo actual con todos los ejemplos necesarios. Dentro de este directorio encontrará subcarpetas que contienen los ejemplos, organizados por capítulo.

Conclusion

Conclusión

At this point, we are all set up with our development environment and are ready to start developing Angular applications. We have installed Node.js, TypeScript, as well as the Angular CLI and understand the need and use of each.

En este punto, ya estamos configurados con nuestro entorno de desarrollo y listos para comenzar a desarrollar aplicaciones Angular.

Hemos instalado Node.js, TypeScript y Angular CLI y entendemos la necesidad y el uso de cada uno.

In the next chapter, we will finally get our hands dirty building our first Angular application and understanding some of the basic terms and concepts of Angular.

En el próximo capítulo, finalmente nos ensuciaremos las manos construyendo nuestra primera aplicación Angular y comprendiendo algunos de los términos y conceptos básicos de Angular.

Chapter 2. Hello Angular

Capítulo 2. Hola Angular

In the previous chapter, we got a very quick overview of Angular and its features, as well as a step-by-step guide on how to set up our local environment for developing any Angular application. In this chapter, we will go through the various parts of an Angular application by creating a very simple application from scratch.

Through the use of this application, we will cover some of the basic terminologies and concepts like modules, components, data and event binding, and passing data to and from components.

En el capítulo anterior, obtuvimos una descripción general muy rápida de Angular y sus características, así como una guía paso a paso sobre cómo configurar nuestro entorno local para desarrollar cualquier aplicación Angular. En este capítulo, repasaremos las distintas partes de una aplicación Angular creando una aplicación muy simple desde cero. Mediante el uso de esta aplicación, cubriremos algunas de las terminologías y conceptos básicos como módulos, componentes, enlace de datos y eventos, y transferencia de datos hacia y desde componentes.

We will start with a very simple stock market application, which allows us to see a list of stocks, each with its own name, stock code, and price. During the course of this chapter, we will see how to package rendering a stock into an individual, reusable component, and how to work with Angular event and data binding.

Comenzaremos con una aplicación de bolsa muy sencilla, que nos permite ver una lista de acciones, cada una con su propio nombre,

código bursátil y precio. Durante el transcurso de este capítulo, veremos cómo empaquetar la representación de un stock en un componente individual y reutilizable, y cómo trabajar con eventos Angular y enlace de datos.

Starting Your First Angular Project

Comenzando su primer proyecto angular

As mentioned in the previous chapter, we will heavily rely on the Angular CLI to help us bootstrap and develop our application. I will assume that you have already followed the initial setup instructions in the previous chapter and have Node.js, TypeScript, and the Angular CLI installed in your development environment.

Como se mencionó en el capítulo anterior, dependeremos en gran medida de Angular CLI para ayudarnos a iniciar y desarrollar nuestra aplicación. Asumiré que ya siguió las instrucciones de configuración inicial del capítulo anterior y que tiene Node.js, TypeScript y Angular CLI instalados en su entorno de desarrollo.

Creating a new application is as simple as running the following command:

Crear una nueva aplicación es tan sencillo como ejecutar el siguiente comando:

```
ng new stock-market
```

When you run this command, it will automatically generate a skeleton application under the folder *stock-market* with a bunch of files, and install all the necessary dependencies for the Angular application to work. This might take a while, but eventually, you should see the following line in your terminal:

Cuando ejecuta este comando, generará automáticamente una aplicación esqueleto en la carpeta stock-market con un montón de archivos e instalará todas las dependencias necesarias para que la aplicación Angular funcione. Esto puede tardar un poco, pero eventualmente deberías ver la siguiente línea en tu terminal:

```
Project 'stock-market' successfully created.
```

Congratulations, you have just created your first Angular application!

¡Felicitaciones, acaba de crear su primera aplicación Angular!

TIP CONSEJO

While we created our first application with the vanilla Angular CLI command, the `ng new` command takes a few arguments that allow you to customize the application generated to your preference. These include:

Si bien creamos nuestra primera aplicación con el comando Vanilla Angular CLI, el comando `ng new` toma algunos argumentos que le permiten personalizar la aplicación generada según sus preferencias. Éstas incluyen:

- Whether you want to use vanilla CSS or SCSS or any other CSS framework (for example, `ng new --style=scss`)
Si desea utilizar CSS básico o SCSS o cualquier otro marco CSS (por ejemplo, `ng new --style=scss`)
- Whether you want to generate a routing module (for example, `ng new --routing`); we'll discuss this further in [Chapter 11](#).
Si desea generar un módulo de enrutamiento (por ejemplo, `ng new --routing`); Hablaremos de esto más a fondo en el Capítulo 11.
- Whether you want inline styles/templates
Si desea estilos/plantillas en línea
- Whether you want a common prefix to all components (for example, to prefix `acme` to all components, `ng new --prefix=acme`)
Si desea un prefijo común para todos los componentes (por ejemplo, prefijar `acme` a todos los componentes, `ng new --prefix=acme`)

And much more. It's worth exploring these options by running `ng help` once you are a bit more familiar with the Angular framework to decide if you have specific preferences one way or the other.

Y mucho más. Vale la pena explorar estas opciones ejecutando `ng help` una vez que esté un poco más familiarizado con el marco Angular para decidir si tiene preferencias específicas de una forma u otra.

Understanding the Angular CLI

Entendiendo la CLI angular

While we have just created our first Angular application, the Angular CLI does a bit more than just the initial skeleton creation. In fact, it is useful throughout the development process for a variety of tasks, including:

Si bien acabamos de crear nuestra primera aplicación Angular, Angular CLI hace un poco más que solo la creación inicial del esqueleto. De hecho, es útil durante todo el proceso de desarrollo para una variedad de tareas, que incluyen:

- Bootstrapping your application
Iniciando su aplicación
- Serving the application
Sirviendo la aplicación
- Running the tests (both unit and end-to-end)
Ejecución de las pruebas (tanto unitarias como de un extremo a otro)
- Creating a build for distribution
Creando una compilación para distribución
- Generating new components, services, routes and more for your application
Generando nuevos componentes, servicios, rutas y más para tu aplicación

Each of these corresponds to one or more Angular CLI commands, and we will cover each one as and when we need or encounter them, instead of trying to cover each command and its uses upfront.

Each command provides further flexibility with a variety of arguments and options, making the Angular CLI truly diverse and capable for a wide variety of uses.

Cada uno de estos corresponde a uno o más comandos de Angular CLI, y cubriremos cada uno cuando los necesitemos o los encontremos, en lugar de intentar cubrir cada comando y sus usos por adelantado. Cada comando proporciona mayor flexibilidad con una variedad de argumentos y opciones, lo que hace que Angular CLI sea verdaderamente diversa y capaz para una amplia variedad de usos.

Running the Application

Ejecutando la aplicación

Now that we have generated our application, the next part is to run it so that we can see our live running application in the browser. There are technically two ways to run it:

Ahora que hemos generado nuestra aplicación, la siguiente parte es ejecutarla para que podamos ver nuestra aplicación en vivo en el navegador. Técnicamente hay dos formas de ejecutarlo:

- Running it in development mode, where the Angular CLI compiles the changes as it happens and refreshes our UI
Ejecutarlo en modo de desarrollo, donde Angular CLI compila los cambios a medida que ocurren y actualiza nuestra interfaz de usuario.
- Running it in production mode, with an optimal compiled build, served via static files
Ejecutarlo en modo de producción, con una compilación compilada óptima, entregada a través de archivos estáticos.

For now, we will run it in development mode, which is as simple as running

Por ahora lo ejecutaremos en modo desarrollo, que es tan sencillo como ejecutar

```
ng serve
```

from the root folder of the generated project, which is the *stock-market* folder in this case. After a little bit of processing and compilation, you should see something like the following in your terminal:

desde la carpeta raíz del proyecto generado, que es la carpeta del mercado de valores en este caso. Despues de un poco de procesamiento y compilación, deberías ver algo como lo siguiente en tu terminal:

```
** NG Live Development Server is listening on localhost:4200,  
  open your browser on http://localhost:4200/ **  
Date: 2018-03-26T10:09:18.869Z  
Hash: 0b730a52f97909e2d43a  
Time: 11086ms  
chunk {inline} inline.bundle.js (inline) 3.85 kB [entry] [rendered]  
chunk {main} main.bundle.js (main) 17.9 kB [initial] [rendered]  
chunk {polyfills} polyfills.bundle.js (polyfills) 549 kB [initial] [rendered]  
chunk {styles} styles.bundle.js (styles) 41.5 kB [initial] [rendered]  
chunk {vendor} vendor.bundle.js (vendor) 7.42 MB [initial] [rendered]  
  
webpack: Compiled successfully.
```

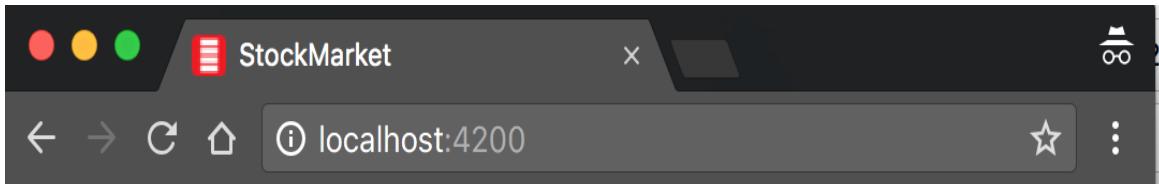
The preceding output is a snapshot of all the files that the Angular CLI generates in order for your Angular application to be served successfully. It includes the *main.bundle.js*, which is the transpiled code that is specific to your application, and the *vendor.bundle.js*, which includes all the third-party libraries and frameworks you depend on (including Angular). *styles.bundle.js* is a compilation of all the CSS styles that are needed for your application, while

polyfills.bundle.js includes all the polyfills needed for supporting some capabilities in older browsers (like advanced ECMAScript features not yet available in all browsers). Finally, *inline.bundle.js* is a tiny file with webpack utilities and loaders that is needed for bootstrapping the application.

El resultado anterior es una instantánea de todos los archivos que Angular CLI genera para que su aplicación Angular se entregue correctamente. Incluye *main.bundle.js*, que es el código transpilado específico de su aplicación, y *seller.bundle.js*, que incluye todas las bibliotecas y marcos de terceros de los que depende (incluido Angular). *estilos.bundle.js* es una compilación de todos los estilos CSS necesarios para su aplicación, mientras que *polyfills.bundle.js* incluye todos los polyfills necesarios para admitir algunas capacidades en navegadores más antiguos (como funciones avanzadas de ECMAScript que aún no están disponibles en todos los navegadores) . Finalmente, *inline.bundle.js* es un archivo pequeño con utilidades y cargadores de paquete web que se necesita para iniciar la aplicación.

`ng serve` starts a local development server on port 4200 for you to hit from your browser. Opening `http://localhost:4200` in your browser should result in you seeing the live running Angular application, which should look like [Figure 2-1](#).

`ng serve` inicia un servidor de desarrollo local en el puerto 4200 al que puede acceder desde su navegador. Al abrir `http://localhost:4200` en su navegador, debería ver la aplicación Angular ejecutándose en vivo, que debería verse como en la Figura 2-1.



Welcome to app!



Here are some links to help you start:

- [Tour of Heroes](#)
- [CLI Documentation](#)
- [Angular blog](#)

Figure 2-1. Hello Angular application in the browser

Figura 2-1. Hola aplicación Angular en el navegador.

TIP CONSEJO

You can actually leave the `ng serve` command running in the terminal, and continue making changes. If you have the application opened in your browser, it will automatically refresh each time you save your changes. This makes the development quick and iterative.

De hecho, puedes dejar el comando `ng serve` ejecutándose en la terminal y continuar realizando cambios. Si tiene la aplicación abierta en su navegador, se actualizará automáticamente cada vez que guarde los cambios. Esto hace que el desarrollo sea rápido e iterativo.

In the following section, we will go into a bit more detail about what exactly happened under the covers to see how the generated Angular application works and what the various pieces are.

En la siguiente sección, entraremos en un poco más de detalle sobre lo que sucedió exactamente bajo las sábanas para ver cómo funciona la aplicación Angular generada y cuáles son las distintas piezas.

Basics of an Angular Application

Conceptos básicos de una aplicación angular

At its core, any Angular application is still a Single-Page Application (SPA), and thus its loading is triggered by a main request to the server. When we open any URL in our browser, the very first request is made to our server (which is running within `ng serve` in this

case). This initial request is satisfied by an HTML page, which then loads the necessary JavaScript files to load both Angular as well as our application code and templates.

En esencia, cualquier aplicación Angular sigue siendo una aplicación de página única (SPA) y, por lo tanto, su carga se activa mediante una solicitud principal al servidor. Cuando abrimos cualquier URL en nuestro navegador, la primera solicitud se realiza a nuestro servidor (que se ejecuta dentro de `ng serve` en este caso). Esta solicitud inicial es satisfecha por una página HTML, que luego carga los archivos JavaScript necesarios para cargar tanto Angular como el código y las plantillas de nuestra aplicación.

One thing to note is that although we develop our Angular application in TypeScript, the web application works with transpiled JavaScript. The `ng serve` command is responsible for translating our TypeScript code into JavaScript for the browser to load.

Una cosa a tener en cuenta es que aunque desarrollamos nuestra aplicación Angular en TypeScript, la aplicación web funciona con JavaScript transpilado. El comando `ng serve` es responsable de traducir nuestro código TypeScript a JavaScript para que el navegador lo cargue.

If we look at the structure the Angular CLI has generated, it is something like this:

Si miramos la estructura que ha generado Angular CLI, es algo como esto:

```
stock-market
+---e2e
+---src
    +---app
        +---app.component.css
        +---app.component.html
        +---app.component.spec.ts
        +---app.component.ts
        +---app.module.ts
    +---assets
```

①
②

```
+----environments  
+----index.html  
+----main.ts  
+----angular-cli.json
```

③
④
⑤

① Root component

② Main module

③ Root HTML

④ Entry point

⑤ Configuration file for CLI angular

There are a few more files than listed here in the *stock-market* folder, but these are the major ones we are going to focus on in this chapter. In addition, there are unit tests, end-to-end (e2e) tests, the assets that support our application, configuration specific to various environments (dev, prod, etc.), and other general configuration that we will touch upon in Chapters 5, 10, and 12.

Hay algunos archivos más que los enumerados aquí en la carpeta del mercado de valores, pero estos son los principales en los que nos centraremos en este capítulo. Además, existen pruebas unitarias, pruebas de extremo a extremo (e2e), los activos que respaldan nuestra aplicación, configuraciones específicas para varios entornos (dev, prod, etc.) y otras configuraciones generales que abordaremos en los capítulos 5, 10 y 12.

Root HTML—index.html

HTML raíz: index.html

If you take a look at the *index.html* file, which is in the *src* folder, you will notice that it looks very clean and pristine, with no references to any scripts or dependencies:

Si echas un vistazo al archivo index.html, que se encuentra en la carpeta src, notarás que se ve muy limpio y prístino, sin referencias a ningún script o dependencia:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>StockMarket</title>
  <base href="/">

  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root></app-root> ①
</body>
</html>
```

① Root component for our Angular application

The only thing of note in the preceding code is the `<app-root>` element in the HTML, which is the marker for loading our application code.

Lo único que cabe destacar en el código anterior es el elemento `<app-root>` en el HTML, que es el marcador para cargar el código de nuestra aplicación.

What about the part that loads the core Angular scripts and our application code? That is inserted dynamically at runtime by the `ng serve` command, which combines all the vendor libraries, our application code, the styles, and inline templates each into individual bundles and injects them into `index.html` to be loaded as soon as the page renders in our browser.

¿Qué pasa con la parte que carga los scripts principales de Angular y el código de nuestra aplicación? Esto se inserta dinámicamente en tiempo de ejecución mediante el comando `ng serve`, que combina todas las bibliotecas de proveedores, el código de nuestra aplicación,

los estilos y las plantillas en línea, cada uno en paquetes individuales y los inyecta en index.html para cargarlos tan pronto como La página se representa en nuestro navegador.

The Entry Point—main.ts

El punto de entrada—main.ts

The second important part of our bootstrapping piece is the *main.ts* file. The *index.html* file is responsible for deciding which files are to be loaded. The *main.ts* file, on the other hand, identifies which Angular module (which we will talk a bit more about in the following section) is to be loaded when the application starts. It can also change application-level configuration (like turning off framework-level asserts and verifications using the `enableProdMode()` flag), which we will cover in [Chapter 12](#):

La segunda parte importante de nuestra pieza de arranque es el archivo main.ts. El archivo index.html es responsable de decidir qué archivos se cargarán. El archivo main.ts, por otro lado, identifica qué módulo Angular (del que hablaremos un poco más en la siguiente sección) se cargará cuando se inicie la aplicación. También puede cambiar la configuración a nivel de aplicación (como desactivar las afirmaciones y verificaciones a nivel de marco usando el indicador `enableProdMode()`), que cubriremos en el Capítulo 12:

```
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';
import { environment } from './environments/environment';

if (environment.production) {
  enableProdMode();
}

platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.log(err));
```

① Bootstrapping the main Application

Most of the code in the *main.ts* file is generic, and you will rarely have to touch or change this entry point file. Its main aim is to point the Angular framework at the core module of your application and let it trigger the rest of your application source code from that point.

La mayor parte del código en el archivo *main.ts* es genérico y rara vez tendrá que tocar o cambiar este archivo de punto de entrada. Su objetivo principal es apuntar el marco Angular al módulo central de su aplicación y permitir que active el resto del código fuente de su aplicación desde ese punto.

Main Module—**app.module.ts**

Módulo principal: **app.module.ts**

This is where your application-specific source code starts from. The application module file can be thought of as the core configuration of your application, from loading all the relevant and necessary dependencies, declaring which components will be used within your application, to marking which is the main entry point component of your application:

Aquí es donde comienza el código fuente específico de su aplicación. El archivo del módulo de la aplicación puede considerarse como la configuración central de su aplicación, desde cargar todas las dependencias relevantes y necesarias, declarar qué componentes se utilizarán dentro de su aplicación, hasta marcar cuál es el componente de punto de entrada principal de su aplicación:

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';
```

```
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

- ❶ NgModule TypeScript annotation to mark this class definition as an ~~Angular module~~.
Anotación de TypeScript para marcar esta definición de clase como un módulo Angular
- ❷ Declarations marking out which components and directives can ~~be used in the application~~.
Declaraciones que marcan cuáles componentes y directivas se pueden utilizar dentro de la aplicación.
- ❸ Importing other modules that provide functionality needed in the ~~application~~.
Importar otros módulos que proporcionen la funcionalidad necesaria en la aplicación.
- ❹ The component that provides functionality for the application.
El componente que proporciona funcionalidad para la aplicación.

TIP CONSEJO

This is our first time dealing with a TypeScript-specific feature, which are decorators (you can think of them as annotations). Decorators allow us to decorate classes with annotations and properties as well as meta-functionality.

Esta es la primera vez que tratamos con una característica específica de TypeScript, que son los decoradores (puedes considerarlos como anotaciones). Los decoradores nos permiten decorar clases con anotaciones y propiedades, así como metafuncionalidad.

Angular heavily leverages this TypeScript feature across the board, such as using decorators for modules, components, and more.

Angular aprovecha en gran medida esta característica de TypeScript en todos los ámbitos, como el uso de decoradores para módulos, componentes y más.

You can read more about TypeScript decorators in [the official documentation](#).

Puede leer más sobre los decoradores de TypeScript en la documentación oficial.

We will go over the details of each of these sections in the following chapters, but at its core:

Repasaremos los detalles de cada una de estas secciones en los siguientes capítulos, pero en su núcleo:

declarations

The declarations block defines all the components that are allowed to be used in the scope of the HTML within this module. Any component that you create must be declared before it can be used.

El bloque declarations define todos los componentes que pueden usarse en el alcance del HTML dentro de este módulo. Cualquier componente que cree debe declararse antes de poder usarse.

imports

You will not create each and every functionality used in the application, and the `imports` array allows you to import other Angular application and library modules and thus leverage the components, services, and other capabilities that have already been created in those modules.

No creará todas y cada una de las funciones utilizadas en la aplicación, y la matriz `imports` le permite importar otras aplicaciones y módulos de biblioteca de Angular y así aprovechar los componentes, servicios y otras capacidades que ya se han creado en esos módulos. .

bootstrap

The `bootstrap` array defines the component that acts as the entry point to your application. If the main component is not added here, your application will not kick-start, as Angular will not know what elements to look for in your `index.html`.

La matriz `bootstrap` define el componente que actúa como punto de entrada a su aplicación. Si el componente principal no se agrega aquí, su aplicación no se iniciará, ya que Angular no sabrá qué elementos buscar en su `index.html`.

You usually end up needing (if you are not using the CLI for any reason!) to modify this file if and only if you add new components, services, or add/integrate with new libraries and modules.

Por lo general, terminará necesitando (si no está utilizando la CLI por algún motivo!) modificar este archivo si y solo si agrega nuevos componentes, servicios o agrega/integra con nuevas bibliotecas y módulos.

Root Component—AppComponent

Componente raíz: AppComponent

We finally get to the actual Angular code that drives the functionality of the application, and in this case, it is the main (and only) component we have, the AppComponent. The code for it looks something like this:

Finalmente llegamos al código Angular real que impulsa la funcionalidad de la aplicación y, en este caso, es el componente principal (y único) que tenemos, el AppComponent. El código para ello se parece a esto:

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'app';
}
```

- ❶ The DOM selector that gets translated into an instance of this componentDOM que se traduce en una instancia de este componente.
- ❷ The HTML template backing this component—in this case, the URLtemplate HTML que respalda este componente; en este caso, la URL del mismo.
- ❸ Any component-specific styling, again pointing to a separate file Estilo que es específico de componente, nuevamente apuntando a un archivo separado en este caso
- ❹ The class definition that holds its own properties and functions.

A *component* in Angular is nothing but a TypeScript class, decorated with some attributes and metadata. The class encapsulates all the data and functionality of the component, while the decorator specifies how it translates into the HTML.

Un componente en Angular no es más que una clase TypeScript, decorada con algunos atributos y metadatos. La clase encapsula todos los datos y la funcionalidad del componente, mientras que el decorador especifica cómo se traduce al HTML.

The *app-selector* is a CSS selector that identifies how Angular finds this particular component in any HTML page. While we generally use element selectors (`app-root` in the preceding example, which translates to looking for `<app-root>` elements in the HTML), they can be any CSS selector, from a CSS class to an attribute as well.

El selector de aplicaciones es un selector de CSS que identifica cómo Angular encuentra este componente en particular en cualquier página HTML. Si bien generalmente usamos selectores de elementos (`app-root` en el ejemplo anterior, que se traduce en buscar elementos `<app-root>` en HTML), pueden ser cualquier selector de CSS, desde una clase CSS hasta un atributo también.

The `templateUrl` is the path to the HTML used to render this component. We can also use inline templates instead of specifying a `templateUrl` like we have done in the example. In this particular case, the template we are referring to is `app.component.html`.

`templateUrl` es la ruta al HTML utilizado para representar este componente. También podemos usar plantillas en línea en lugar de especificar un `templateUrl` como hemos hecho en el ejemplo. En este caso particular, la plantilla a la que nos referimos es `app.component.html`.

`styleUrls` is the styling counterpart to the template, encapsulating all the styles for this component. Angular ensures that the styles are encapsulated, so you don't have to worry about your CSS classes

from one component affecting another. Unlike `templateUrl`, `styleUrls` is an array.

`styleUrls` es la contraparte de estilo de la plantilla, que encapsula todos los estilos de este componente. Angular garantiza que los estilos estén encapsulados, por lo que no tiene que preocuparse de que sus clases CSS de un componente afecten a otro. A diferencia de `templateUrl`, `styleUrls` es una matriz.

The component class itself finally encapsulates all the functionality backing your component. It makes it easy to think of the responsibilities of the component class as twofold:

La clase de componente en sí finalmente encapsula toda la funcionalidad que respalda su componente. Hace que sea fácil pensar en las responsabilidades de la clase de componente como dos:

- Load and hold all the data necessary for rendering the component
Cargue y mantenga todos los datos necesarios para renderizar el componente.
- Handle and process any events that may arise from any element in the component
Manejar y procesar cualquier evento que pueda surgir de cualquier elemento del componente.

The data in the class will drive what can be displayed as part of the component. So let's take a look at what the template for this component looks like:

Los datos de la clase impulsarán lo que se puede mostrar como parte del componente. Entonces, echemos un vistazo a cómo se ve la plantilla para este componente:

```
<h1>  
  {{title}}  
</h1>
```

❶

❶ Data binding for the component

Our HTML is as simple as can be for the component. All it has is one element, which is data-bound to a field in our component class. The double-curly ({{ }}) syntax is an indication to Angular to replace the value between the braces with the value of the variable from the corresponding class.

Nuestro HTML es lo más simple posible para el componente. Todo lo que tiene es un elemento, que está vinculado a datos a un campo en nuestra clase de componente. La sintaxis de doble rizado ({{ }}) es una indicación a Angular para reemplazar el valor entre llaves con el valor de la variable de la clase correspondiente.

In this case, once the application loads, and the component is rendered, the {{title}} will be replaced with the text app works!. We will talk in more detail about data binding in "[Understanding Data Binding](#)".

En este caso, una vez que se carga la aplicación y se procesa el componente, {{title}} se reemplazará con el texto app works!. Hablaremos con más detalle sobre la vinculación de datos en "Comprepción de la vinculación de datos".

Creating a Component

Creando un componente

So far, we have dealt with the basic skeleton code that the Angular CLI has generated for us. Let's now look at adding new components, and what that entails. We will use the Angular CLI to generate a new

component, but look underneath the covers to see what steps it takes. We will then walk through some very basic common tasks we try to accomplish with components.

Hasta ahora, nos hemos ocupado del código esqueleto básico que Angular CLI ha generado para nosotros. Veamos ahora cómo agregar nuevos componentes y lo que eso implica. Usaremos Angular CLI para generar un nuevo componente, pero mire debajo de las cubiertas para ver qué pasos se siguen. Luego, analizaremos algunas tareas comunes muy básicas que intentamos realizar con componentes.

Steps in Creating New Components

Pasos para crear nuevos componentes

Using the Angular CLI, creating a new component is simply running a simple command. We will first try creating a stock widget, which displays the name of the stock, its stock code, the current price, and whether it has changed for the positive or negative.

Con Angular CLI, crear un nuevo componente es simplemente ejecutar un comando simple. Primero intentaremos crear un widget de acciones, que muestre el nombre de la acción, su código de acción, el precio actual y si ha cambiado para positivo o negativo.

We can simply create a new `stock-item` by running the following command from the main folder of the application:

Simplemente podemos crear un nuevo `stock-item` ejecutando el siguiente comando desde la carpeta principal de la aplicación:

```
ng generate component stock/stock-item
```

There are a few interesting things to note here:

Hay algunas cosas interesantes a tener en cuenta aquí:

- The Angular CLI has a command called `generate`, which can be used to generate components (like we did in the preceding example), and also to generate other Angular elements, such as interfaces, services, modules, and more.

La CLI de Angular tiene un comando llamado `generate`, que se puede usar para generar componentes (como hicimos en el ejemplo anterior) y también para generar otros elementos de Angular, como interfaces, servicios, módulos y más.

- With the target type, we also specify the name (and the folder) within which the component has to be generated. Here, we are telling the Angular CLI to generate a component called `stock-item` within a folder called `stock`. If we don't specify `stock`, it will create a component called `stock-item` in the `app` folder itself.

Con el tipo de destino, también especificamos el nombre (y la carpeta) dentro del cual se debe generar el componente. Aquí, le estamos diciendo a Angular CLI que genere un componente llamado `stock-item` dentro de una carpeta llamada `stock`. Si no especificamos `stock`, creará un componente llamado `stock-item` en la carpeta de la aplicación.

The command will generate all the relevant files for a new component, including:

El comando generará todos los archivos relevantes para un nuevo componente, incluyendo:

- The component definition (named `stock-item.component.ts`)
La definición del componente (llamada `stock-item.component.ts`)
- The corresponding template definition (named `stock-item.component.html`)

La definición de plantilla correspondiente (llamada `stock-item.component.html`)

- The styles for the component (in a file named `stock-item.component.css`)

Los estilos del componente (en un archivo llamado `stock-item.component.css`)

- The skeleton unit tests for the component (named `stock-item.component.spec.ts`)

La unidad de esqueleto prueba el componente (llamado `stock-item.component.spec.ts`)

In addition, it updated the original `app module` that we saw earlier so that our Angular application recognizes the new module.

Además, actualizó el `app module` original que vimos anteriormente para que nuestra aplicación Angular reconozca el nuevo módulo.

This is the recommended convention to follow whenever you are working with components:

Esta es la convención recomendada a seguir siempre que trabaje con componentes:

- The filename starts with the name of the item you are creating

El nombre del archivo comienza con el nombre del elemento que está creando.

- This is followed by the type of element it is (in this case, a component)

A esto le sigue el tipo de elemento que es (en este caso, un componente)

- Finally, we have the relevant extension

Finalmente, tenemos la extensión relevante.

This allows us to both group and easily identify relevant and related files in a simple manner.

Esto nos permite agrupar e identificar fácilmente archivos relevantes y relacionados de forma sencilla.

When you run the command, you should see something like this:

Cuando ejecute el comando, debería ver algo como esto:

```
create src/app/stock/stock-item/stock-item.component.css
create src/app/stock/stock-item/stock-item.component.html
create src/app/stock/stock-item/stock-item.component.spec.ts
create src/app/stock/stock-item/stock-item.component.ts
update src/app/app.module.ts
```

The source for the component, HTML, and the CSS remain pretty much barebones, so I won't repeat that here. What is important is how this new component that we create is hooked up and made available to our Angular application. Let's take a look at the modified *app.module.ts* file:

La fuente del componente, HTML y CSS siguen siendo bastante básicas, por lo que no las repetiré aquí. Lo importante es cómo este nuevo componente que creamos se conecta y se pone a disposición de nuestra aplicación Angular. Echemos un vistazo al archivo *app.module.ts* modificado:

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';
import { StockItemComponent } from './stock/stock-item/stock-item.component';
①

@NgModule({
  declarations: [
    AppComponent,
    StockItemComponent
  ],
  imports: [
    BrowserModule
  ]
})
```

```
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

- ❶ Importing the new component into the application
- ❷ Adding the new component to the declarations section

In the application module, we have to ensure that the new component is imported and added to the declarations array, before we can start using it in our Angular application.

En el módulo de la aplicación, debemos asegurarnos de que el nuevo componente se importe y agregue a la matriz declarations, antes de que podamos comenzar a usarlo en nuestra aplicación Angular.

Using Our New Component

Usando nuestro nuevo componente

Now that we have created a new component, let's see how we can use it in our application. We will now try to use this skeleton in the app component. First, take a look at the generated *stock-item.component.ts* file:

Ahora que hemos creado un nuevo componente, veamos cómo podemos usarlo en nuestra aplicación. Ahora intentaremos utilizar este esqueleto en el componente de la aplicación. Primero, eche un vistazo al archivo stock-item.component.ts generado:

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-stock-item',
  templateUrl: './stock-item.component.html',
  styleUrls: ['./stock-item.component.css']
```

❶

```
})
export class StockItemComponent implements OnInit {

  constructor() { }

  ngOnInit() {
  }

}
```

- ❶ The selector for using this component. Note it is prefixed with app, which is added by the Angular CLI by default unless ~~El selector para usar este componente~~ tiene el prefijo app, que Angular CLI agrega de forma predeterminada a menos que se especifique lo contrario.

The component has no data and does not provide any functionality at this point; it simply renders the template associated with it. The template at this point is also trivial, and just prints out a static message.

El componente no tiene datos y no proporciona ninguna funcionalidad en este momento; simplemente representa la plantilla asociada a él. La plantilla en este punto también es trivial y simplemente imprime un mensaje estático.

To use this component in our application, we can simply create an element that matches the selector defined anywhere inside our main app component. If we had more components and a deeper hierarchy of components, we could choose to use it in any of their templates as well. So let's replace most of the placeholder content in `app.component.html` with the following, so that we can render the `stock-item` component:

Para usar este componente en nuestra aplicación, simplemente podemos crear un elemento que coincida con el selector definido en cualquier lugar dentro de nuestro componente principal de la aplicación. Si tuviéramos más componentes y una jerarquía de componentes más profunda, podríamos optar por usarlo también en

cualquiera de sus plantillas. Entonces, reemplazamos la mayor parte del contenido del marcador de posición en `app.component.html` con lo siguiente, para que podamos representar el componente `stock-item`:

```
<div style="text-align:center">
  <h1>
    Welcome to {{ title }}!
  </h1>
  <app-stock-item></app-stock-item> ①
</div>
```

① Agregando nuestro componente `stock-item`

All it takes is adding the `<app-stock-item></app-stock-item>` to our `app.component.html` file to use our component. We simply create an element using the selector we defined in our component. Then when the application loads, Angular recognizes that the element refers to a component, and triggers the relevant code path.

Todo lo que se necesita es agregar `<app-stock-item></app-stock-item>` a nuestro archivo `app.component.html` para usar nuestro componente. Simplemente creamos un elemento usando el selector que definimos en nuestro componente. Luego, cuando se carga la aplicación, Angular reconoce que el elemento se refiere a un componente y activa la ruta del código relevante.

When you run this (or if your `ng serve` is still running), you should see both the original "app works" along with a new "stock-item works" in the UI.

Cuando ejecuta esto (o si su `ng serve` todavía se está ejecutando), debería ver el "app works" original junto con un nuevo "stock-item works" en la interfaz de usuario.

Understanding Data Binding

Comprender el enlace de datos

Next, let's focus on getting some data and figuring out how to display it as part of our component. What we are trying to build is a stock widget, which will take some stock information, and render it accordingly.

A continuación, centrémonos en obtener algunos datos y descubrir cómo mostrarlos como parte de nuestro componente. Lo que estamos intentando crear es un widget de acciones, que tomará información sobre las acciones y la mostrará en consecuencia.

Let's assume that we have a stock for a company named Test Stock Company, with a stock code of TSC. Its current price is \$85, while the previous price it traded at was \$80. In the widget, we want to show both the name and its code, as well as the current price, the percentage change since last time, and highlight the price and percentage change in green if it is an increment, or red if it is a decrement.

Supongamos que tenemos acciones de una empresa llamada Test Stock Company, con un código de acciones de TSC. Su precio actual es \$85, mientras que el precio anterior al que cotizaba era \$80. En el widget, queremos mostrar tanto el nombre como su código, así como el precio actual, el cambio porcentual desde la última vez, y resaltar el precio y el cambio porcentual en verde si es un incremento, o en rojo si es un decremento.

Let's walk through this step by step. First, we will make sure we can display the name and code in the widget (we will hardcode the information for now, and we will build up the example to get the data from a different source later).

Repasemos esto paso a paso. Primero, nos aseguraremos de poder mostrar el nombre y el código en el widget (codificaremos la

información por ahora y desarrollaremos el ejemplo para obtener los datos de una fuente diferente más adelante).

We would change our component code (the *stock-item.component.ts* file) as follows:

Cambiaríamos el código de nuestro componente (el archivo *stock-item.component.ts*) de la siguiente manera:

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-stock-item',
  templateUrl: './stock-item.component.html',
  styleUrls: ['./stock-item.component.css']
})
export class StockItemComponent implements OnInit { ①

  public name: string; ②
  public code: string;
  public price: number;
  public previousPrice: number;

  constructor() { }

  ngOnInit() { ③
    this.name = 'Test Stock Company';
    this.code = 'TSC'; ④
    this.price = 85;
    this.previousPrice = 80;
  }
}
```

- ① Implement OnInit interface from Angular, which gives us a hook que permite la ejecución del inicializador de Angular, lo que nos proporciona un enlace para saber cuándo se inicializa el componente.
- ② Definition of the various fields we will want to access from the Definición de los distintos campos a los que querremos acceder desde el HTML

- ③ OnInit function that is triggered and executed immediately as initialized
- ④ Initializing all the values for each of the fields

Angular gives us hooks into the lifecycle of a component to let us take certain actions when a component is initialized, when its view is rendered, when it is destroyed, and so on. We've extended our trivial component with a few notable things:

Angular nos brinda enlaces al ciclo de vida de un componente para permitirnos realizar ciertas acciones cuando se inicializa un componente, cuando se representa su vista, cuando se destruye, etc. Hemos ampliado nuestro componente trivial con algunas cosas notables:

OnInit

Angular's OnInit hook is executed after the component is created by the Angular framework, after all the data fields are initialized. It is generally recommended to do any initialization work of a component in the OnInit hook, so that it makes it easier to test the functionality of the rest of the component without necessarily triggering the initialization flow every time. We will cover the remaining lifecycle hooks in [Chapter 4](#).

El gancho OnInit de Angular se ejecuta después de que el marco Angular crea el componente, después de que se inicializan todos los campos de datos. Generalmente se recomienda realizar cualquier trabajo de inicialización de un componente en el enlace OnInit, de modo que sea más fácil probar la funcionalidad del resto del componente sin necesariamente activar el flujo de inicialización cada vez. Cubriremos los ganchos restantes del ciclo de vida en el Capítulo 4.

ngOnInit

When you want to hook on the initialization phase of a component, you need to implement the OnInit interface (as in

the example) and then implement the `ngOnInit` function in the component, which is where you write your initialization logic. We have initialized the basic information we need to render our stock widget in the `ngOnInit` function.

Cuando desee conectarse a la fase de inicialización de un componente, debe implementar la interfaz `OnInit` (como en el ejemplo) y luego implementar la función `ngOnInit` en el componente, que es donde escribe su Lógica de inicialización. Hemos inicializado la información básica que necesitamos para representar nuestro widget de acciones en la función `ngOnInit`.

Class member variables

Variables de miembros de clase

We have declared a few public variables as class instance variables. This information will be used to render our template.

Hemos declarado algunas variables públicas como variables de instancia de clase. Esta información se utilizará para representar nuestra plantilla.

Now, let's change the template (the `stock-item.component.html` file) to start rendering this information:

Ahora, cambiemos la plantilla (el archivo `stock-item.component.html`) para comenzar a representar esta información:

```
<div class="stock-container">
  <div class="name"><h3>{{name}} - <h4>({{code}})</h4></div>
  <div class="price">$ {{price}}</div>
</div>
```

and its corresponding CSS (the `stock-item.component.css` file), to make it look nice:

y su CSS correspondiente (el archivo `stock-item.component.css`), para que se vea bien:

```
.stock-container {  
  border: 1px solid black;  
  border-radius: 5px;  
  display: inline-block;  
  padding: 10px;  
}  
  
.stock-container .name h3, .stock-container .name h4 {  
  display: inline-block;  
}
```

NOTE

NOTA

Note that the CSS is purely from a visual perspective, and is not needed nor impacts our Angular application. You could skip it completely and still have a functional application.

Tenga en cuenta que el CSS es puramente visual y no es necesario ni afecta nuestra aplicación Angular. Podrías omitirlo por completo y aún tener una aplicación funcional.

Once we make these changes and refresh our application, we should see something like **Figure 2-2** in our browser.

Una vez que hagamos estos cambios y actualicemos nuestra aplicación, deberíamos ver algo como la Figura 2-2 en nuestro navegador.

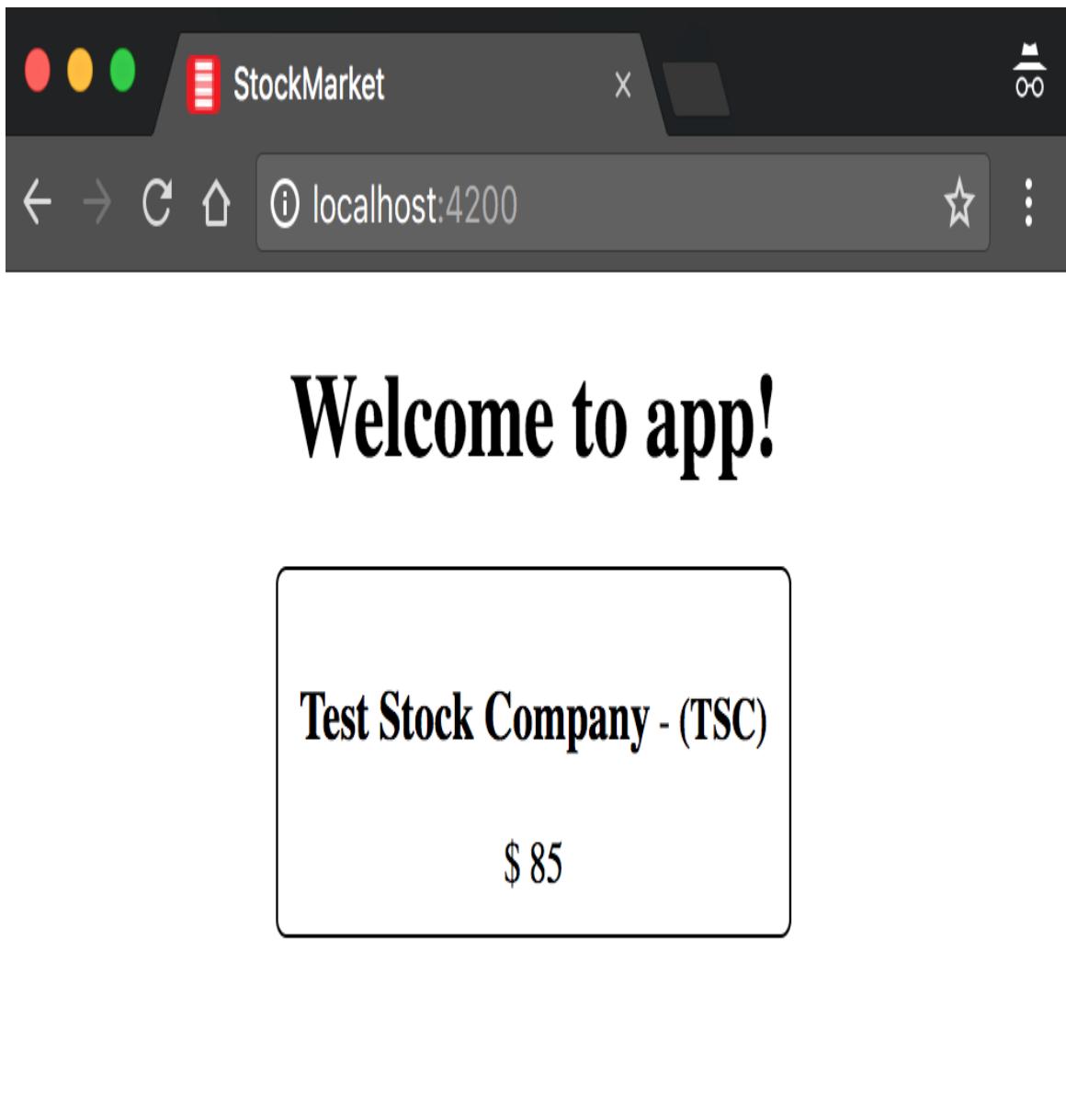


Figure 2-2. Angular app with stock component

Figura 2-2. Aplicación angular con componente stock.

We have just used one fundamental building block from Angular to render our data from our component into the HTML. We use the double-curly notation (`{{ }}`), which is also known as *interpolation*. Interpolation evaluates the expression between the curly braces as per the component backing it, and then renders the result as a string in place in the HTML. In this case, we render the name, code, and the price of the stock using interpolation. This picks up the

values of name, code, and price, and then replaces the double-curly expression with its value, thus rendering our UI.

Acabamos de utilizar un bloque de construcción fundamental de Angular para representar nuestros datos de nuestro componente en HTML. Usamos la notación de doble curvatura ({{ }}), que también se conoce como interpolación. La interpolación evalúa la expresión entre llaves según el componente que la respalda y luego representa el resultado como una cadena en el HTML. En este caso, representamos el nombre, el código y el precio de la acción mediante interpolación. Esto recoge los valores de nombre, código y precio, y luego reemplaza la expresión doble con su valor, representando así nuestra interfaz de usuario.

This is Angular's one-way data binding at work. One-way data binding simply refers to Angular's capability to automatically update the UI based on values in the component, and then keeping it updated as the value changes in the component. Without one-way binding, we would have to write code to take the value from our component, find the right element in HTML, and update its value. Then we would have to write listeners/watchers to keep track of when the value in the component changes, and then change the value in the HTML at that time. We can get rid of all of this extra code because of data binding.

Este es el enlace de datos unidireccional de Angular en funcionamiento. El enlace de datos unidireccional simplemente se refiere a la capacidad de Angular de actualizar automáticamente la interfaz de usuario en función de los valores del componente y luego mantenerla actualizada a medida que el valor cambia en el componente. Sin un enlace unidireccional, tendríamos que escribir código para tomar el valor de nuestro componente, encontrar el elemento correcto en HTML y actualizar su valor. Luego tendríamos que escribir oyentes/observadores para realizar un seguimiento de cuándo cambia el valor en el componente y luego cambiar el valor

en el HTML en ese momento. Podemos deshacernos de todo este código adicional gracias al enlace de datos.

In this particular case, we are binding to simple variables, but it is not necessarily restricted to simple variables. The expressions can be slightly more complex. For example, we could render the same UI by changing the binding expression as follows in *stock-item.component.html*:

En este caso particular, estamos vinculando variables simples, pero no necesariamente restringidos a variables simples. Las expresiones pueden ser un poco más complejas. Por ejemplo, podríamos representar la misma interfaz de usuario cambiando la expresión de enlace de la siguiente manera en stock-item.component.html:

```
<div class="stock-container">
  <div class="name">{{name + ' (' + code + ')'}}</div>
  <div class="price">$ {{price}}</div>
</div>
```

In this case, we replaced our multiple heading elements with a single div. The interpolation expression is now a combination of both the name and the code, with the code surrounded by parentheses. Angular will evaluate this like normal JavaScript, and return the value of it as a string to our UI.

En este caso, reemplazamos nuestros múltiples elementos de encabezado con un solo div. La expresión de interpolación ahora es una combinación del nombre y el código, con el código entre paréntesis. Angular evaluará esto como JavaScript normal y devolverá su valor como una cadena a nuestra interfaz de usuario.

Understanding Property Binding

Comprender la vinculación de propiedades

So far, we used interpolation to get data from our component code to the HTML. But Angular also provides a way to bind not just text, but also DOM element properties. This allows us to modify the content and the behavior of the HTML that is rendered in the browser.

Hasta ahora, utilizamos la interpolación para transferir datos del código de nuestro componente al HTML. Pero Angular también proporciona una forma de vincular no solo texto, sino también propiedades de elementos DOM. Esto nos permite modificar el contenido y el comportamiento del HTML que se representa en el navegador.

For example, let's try to modify our stock widget to highlight the price in red if the price is less than the previous price, and in green if it is equal to or more than the previous price. We can first change our component (the *stock-item.component.ts*) to precalculate if the difference is positive or negative like so:

Por ejemplo, intentemos modificar nuestro widget de acciones para resaltar el precio en rojo si el precio es menor que el precio anterior, y en verde si es igual o mayor que el precio anterior. Primero podemos cambiar nuestro componente (stock-item.component.ts) para precalcular si la diferencia es positiva o negativa, así:

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-stock-item',
  templateUrl: './stock-item.component.html',
  styleUrls: ['./stock-item.component.css']
})
export class StockItemComponent implements OnInit {

  public name: string;
```

```

public code: string;
public price: number;
public previousPrice: number;
public positiveChange: boolean;

constructor() { }

ngOnInit() {
  this.name = 'Test Stock Company';
  this.code = 'TSC';
  this.price = 85;
  this.previousPrice = 80;
  this.positiveChange = this.price >= this.previousPrice;
}
}

```

In this code, we added a new public variable called `positiveChange`, which is of type `boolean`, and then set the value based on comparing the current price with the previous price. This gives us a singular `boolean` value that we can use to decide whether to highlight the price in red or green.

En este código, agregamos una nueva variable pública llamada `positiveChange`, que es de tipo `boolean`, y luego configuramos el valor basándose en la comparación del precio actual con el precio anterior. Esto nos da un valor singular `boolean` que podemos usar para decidir si resaltar el precio en rojo o verde.

Next, let's add some classes in the `stock-item.component.css` file to allow for changing the color of the text:

A continuación, agreguemos algunas clases en el archivo `stock-item.component.css` para permitir cambiar el color del texto:

```

.stock-container {
  border: 1px solid black;
  border-radius: 5px;
  display: inline-block;
  padding: 10px;
}

.positive {

```

```
    color: green;
}

.negative {
  color: red;
}
```

We simply added two classes, `positive` and `negative`, which change the color of the text to green and red, respectively. Now let's tie this together to see how we can use this information and classes in our `stock-item.component.html` file:

Simplemente agregamos dos clases, `positive` y `negative`, que cambian el color del texto a verde y rojo, respectivamente. Ahora juntemos esto para ver cómo podemos usar esta información y clases en nuestro archivo `stock-item.component.html`:

```
<div class="stock-container">
  <div class="name">{{name + ' (' + code + ')'}}</div>
  <div class="price"
    [class]="positiveChange ? 'positive' : 'negative'">$ {{price}}</div>
</div>
```

We have added one new binding on the `price` div element, which reads as:

Hemos agregado un nuevo enlace en el elemento div `price`, que dice como:

```
[class]="positiveChange ? 'positive' : 'negative'"
```

This is the Angular syntax for property binding, which binds the value of the expression to the DOM property between the square brackets. The `[]` is the general syntax that can be used with any property on an element to bind one-way from the component to the UI.

Esta es la sintaxis angular para el enlace de propiedades, que vincula el valor de la expresión a la propiedad DOM entre corchetes.

[] es la sintaxis general que se puede usar con cualquier propiedad de un elemento para vincularse unidireccionalmente desde el componente a la interfaz de usuario.

In this particular case, we are telling Angular to bind to the class property of the DOM element to the value of the expression. Angular will evaluate it like a normal JavaScript expression, and assign the value (positive in this case) to the class property of the `div` element.

En este caso particular, le estamos diciendo a Angular que vincule la propiedad de clase del elemento DOM al valor de la expresión.

Angular lo evaluará como una expresión JavaScript normal y asignará el valor (positivo en este caso) a la propiedad de clase del elemento `div`.

WARNING ADVERTENCIA

When you bind to the class property like we did in the example, note that it overrides the existing value of the property. In our example, the "price" class is replaced with the class "positive", instead of appending to the existing value of the property. You can notice this for yourself if you inspect the rendered HTML in the browser. Be careful about this if you bind directly to the `class` property.

Cuando se vincula a la propiedad de clase como lo hicimos en el ejemplo, tenga en cuenta que anula el valor existente de la propiedad. En nuestro ejemplo, la clase "price" se reemplaza con la clase "positive", en lugar de agregarla al valor existente de la propiedad. Puedes notar esto por ti mismo si inspeccionas el HTML renderizado en el navegador. Tenga cuidado con esto si se vincula directamente a la propiedad `class`.

If the value of the variable `positiveChange` in the component changes, Angular will automatically re-evaluate the expression in the HTML and update it accordingly. Try changing the price so that there is a negative change and then refresh the UI to make sure it works.

Si el valor de la variable `positiveChange` en el componente cambia, Angular reevaluará automáticamente la expresión en el HTML y la

actualizará en consecuencia. Intente cambiar el precio para que haya un cambio negativo y luego actualice la interfaz de usuario para asegurarse de que funcione.

Notice that we have been explicitly referring to the data binding working with DOM properties, and not HTML attributes. The following sidebar goes into more detail on the difference between the two, and why it is important to know and understand as you work on Angular. But simplifying it, Angular data binding only works with DOM properties, and not with HTML attributes.

Tenga en cuenta que nos hemos estado refiriendo explícitamente al enlace de datos que trabaja con propiedades DOM y no con atributos HTML. La siguiente barra lateral proporciona más detalles sobre la diferencia entre los dos y por qué es importante conocerla y comprenderla mientras trabaja en Angular. Pero, simplificándolo, el enlace de datos de Angular solo funciona con propiedades DOM y no con atributos HTML.

HTML ATTRIBUTE VERSUS DOM PROPERTY: WHAT'S THE DIFFERENCE?

ATRIBUTO HTML VERSUS PROPIEDAD DOM: ¿CUÁL ES LA DIFERENCIA?

As mentioned, when we work with data binding in Angular, we aren't working with HTML attributes but rather DOM properties. Attributes are defined by HTML, while properties are defined by the Document Object Model. Though some attributes (like ID and class) directly map to DOM properties, others may exist on one side but not the other.

Como se mencionó, cuando trabajamos con enlace de datos en Angular, no trabajamos con atributos HTML sino con propiedades DOM. Los atributos están definidos por HTML, mientras que las propiedades están definidas por el modelo de objetos de documento. Aunque algunos atributos (como ID y clase) se asignan directamente a las propiedades DOM, otros pueden existir en un lado pero no en el otro.

But more importantly, the distinction between the two is that HTML attributes are generally used for initialization of a DOM element, but after that, they have no purpose or effect on the underlying element. Once the element is initialized, its behavior is controlled by the DOM properties from then on.

Pero lo más importante es que la distinción entre los dos es que los atributos HTML generalmente se usan para la inicialización de un elemento DOM, pero después de eso, no tienen ningún propósito ni efecto en el elemento subyacente. Una vez que se inicializa el elemento, su comportamiento es controlado por las propiedades DOM a partir de ese momento.

For example, consider the `input` HTML element. If we bootstrap our HTML with something like:

Por ejemplo, considere el elemento HTML `input`. Si arrancamos nuestro HTML con algo como:

```
<input type="text" value="foo"/>
```

this initializes an `input` DOM element, with the initial value of the DOM property `value` to be set to `foo`. Now let's assume we type something in the text box, say `bar`. At this point:

esto inicializa un elemento DOM `input`, con el valor inicial de la propiedad DOM `value` que se establecerá en `foo`. Ahora supongamos que escribimos algo en el cuadro de texto, digamos `bar`. En este punto:

- If we do `input.getAttribute('value')`, it would return `foo`, which was the attribute value we used to initialize the HTML.

Si hacemos `input.getAttribute('value')`, devolverá `foo`, que fue el valor del atributo que usamos para inicializar el HTML.

- If we do `input.value`, we will get the current value of the DOM property, which is `bar`.

Si hacemos `input.value`, obtendremos el valor actual de la propiedad DOM, que es `bar`.

That is, the attribute value is used to bootstrap and set the initial value of the HTML DOM element, but after that, it is the DOM property that drives the behavior. If you inspect the HTML, you will see that it is still the initial HTML we provided, and does not update either.

Es decir, el valor del atributo se utiliza para impulsar y establecer el valor inicial del elemento HTML DOM, pero después de eso, es la propiedad DOM la que impulsa el comportamiento. Si inspecciona el HTML, verá que sigue siendo el HTML inicial que proporcionamos y que tampoco se actualiza.

In Angular, we thus bind to the DOM property, and not to the HTML attributes. Whenever we think about one-way binding from the component to the UI, we should always keep this in mind!

En Angular, por lo tanto, nos vinculamos a la propiedad DOM y no a los atributos HTML. Siempre que pensamos en el enlace unidireccional del componente a la interfaz de usuario, siempre debemos tener esto en cuenta!

Just like we did for the `class` property, depending on the use case, we can actually bind to other HTML properties like the `src` property of an `img` tag, or the `disabled` property of `input` and `button`. We will cover this in more depth in the next chapter. We will also cover a simpler and more specific way of binding CSS classes in the next chapter as well.

Tal como hicimos con la propiedad `class`, dependiendo del caso de uso, podemos vincularnos a otras propiedades HTML como la propiedad `src` de una etiqueta `img` o la `disabled` propiedad de `input` y `button`. Cubriremos esto con más profundidad en el próximo capítulo. También cubriremos una forma más simple y específica de vincular clases CSS en el próximo capítulo.

Understanding Event Binding

Comprender el enlace de eventos

So far, we have worked on using the data in our component to both render values and change the look and feel of our component. In this section, we will start understanding how to handle user interactions, and work with events and event binding in Angular.

Hasta ahora, hemos trabajado en el uso de los datos de nuestro componente para representar valores y cambiar la apariencia de nuestro componente. En esta sección, comenzaremos a comprender

cómo manejar las interacciones del usuario y trabajar con eventos y enlaces de eventos en Angular.

Say we wanted to have a button that allows users to add the stock to their list of favorite stocks. Generally, with a button like this, when the user clicks it, we would want to make some server call and then process the result. So far, since we are working with very simple examples, let's just say we wanted to handle this click and get a hook to it in our component. Let's see how we might accomplish that.

Digamos que queremos tener un botón que permita a los usuarios agregar la acción a su lista de acciones favoritas. Generalmente, con un botón como este, cuando el usuario hace clic en él, queríamos realizar alguna llamada al servidor y luego procesar el resultado. Hasta ahora, dado que estamos trabajando con ejemplos muy simples, digamos que queríamos manejar este clic y conectarlo en nuestro componente. Veamos cómo podemos lograrlo.

First, we can change our component code in *stock-item.component.ts* to add a function `toggleFavorite`, which should be triggered each time the click happens from the UI:

Primero, podemos cambiar el código de nuestro componente en *stock-item.component.ts* para agregar una función `toggleFavorite`, que debería activarse cada vez que se haga clic desde la interfaz de usuario:

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-stock-item',
  templateUrl: './stock-item.component.html',
  styleUrls: ['./stock-item.component.css']
})
export class StockItemComponent implements OnInit {

  public name: string;
  public code: string;
  public price: number;
```

```

public previousPrice: number;
public positiveChange: boolean;
public favorite: boolean;

constructor() { }

ngOnInit() {
  this.name = 'Test Stock Company';
  this.code = 'TSC';
  this.price = 85;
  this.previousPrice = 80;
  this.positiveChange = this.price >= this.previousPrice;
  this.favorite = false;
}

toggleFavorite() {
  console.log('We are toggling the favorite state for this stock');
  this.favorite = !this.favorite;
}
}

```

We have added a new public boolean member variable called `favorite`, which is initialized with a `false` value. We then added a new function called `toggleFavorite()`, which simply flips the boolean value of `favorite`. We are also printing a log in the console to ensure this is getting triggered.

Hemos agregado una nueva variable miembro booleana pública llamada `favorite`, que se inicializa con un valor `false`. Luego agregamos una nueva función llamada `toggleFavorite()`, que simplemente invierte el valor booleano de `favorite`. También estamos imprimiendo un registro en la consola para asegurarnos de que esto se active.

Now, let's update the UI to use this concept of a `favorite` and also allow users to toggle the state:

Ahora, actualicemos la interfaz de usuario para usar este concepto de `favorite` y también permitamos a los usuarios alternar el estado:

```

<div class="stock-container">
  <div class="name">{{name + ' (' + code + ')'}}</div>
  <div class="price"
    [class]="positiveChange ? 'positive' : 'negative'">$ {{price}}</div>
  <button (click)="toggleFavorite()"
    [disabled]="favorite">Add to Favorite</button>
</div>

```

We have added a new button in the *stock-item.component.html* file to allow users to click and add the stock to their favorite set. We are using the data-binding concept from the previous section on the disabled property. Thus, we are disabling the button based on the boolean value favorite. If favorite is true, the button will be disabled, and if it is false, the button will be enabled. Thus, by default, the button is enabled.

Hemos agregado un nuevo botón en el archivo *stock-item.component.html* para permitir a los usuarios hacer clic y agregar el stock a su conjunto favorito. Estamos utilizando el concepto de enlace de datos de la sección anterior en la propiedad disabled. Por lo tanto, estamos deshabilitando el botón según el valor booleano favorite. Si favorite es true, el botón estará deshabilitado, y si es false, el botón estará habilitado. Por tanto, de forma predeterminada, el botón está habilitado.

The other major thing we have on the element is this fragment:

La otra cosa importante que tenemos sobre el elemento es este fragmento:

```
(click)="toggleFavorite()"
```

This syntax is called *event binding* in Angular. The left part of the equals symbol refers to the event we are binding to. In this case, it is the click event. Just like how the square-bracket notation refers to data flowing from the component to the UI, the parentheses notation refers to events. And the name between the parentheses is the name of the event we care about.

Esta sintaxis se llama enlace de eventos en Angular. La parte izquierda del símbolo igual se refiere al evento al que nos vinculamos. En este caso, es el evento `click`. Al igual que la notación entre corchetes se refiere a los datos que fluyen desde el componente a la interfaz de usuario, la notación entre paréntesis se refiere a eventos. Y el nombre entre paréntesis es el nombre del evento que nos interesa.

In this case, we are telling Angular that we are interested in the `click` event on this element. The right part of the equals symbol then refers to the template statement that Angular should execute whenever the event is triggered. In this case, we want it to execute the new function we created, `toggleFavorite`.

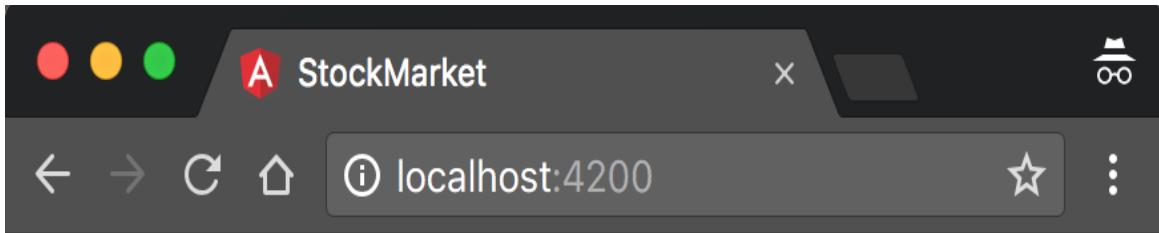
En este caso, le estamos diciendo a Angular que estamos interesados en el evento `click` en este elemento. La parte derecha del símbolo igual se refiere a la declaración de plantilla que Angular debe ejecutar cada vez que se activa el evento. En este caso, queremos que ejecute la nueva función que creamos, `toggleFavorite`.

When we run this application in our browser, we can see the new button. Clicking it would render something like [Figure 2-3](#).

Cuando ejecutamos esta aplicación en nuestro navegador, podemos ver el botón nuevo. Al hacer clic en él, se mostrará algo como la Figura 2-3.

Notice the other interesting thing, which is Angular data binding at play. When we click the button, our `toggleFavorite` function is executed. This flips the value of `favorite` from `false` to `true`. This in turn triggers the other Angular binding, which is the `disabled` property of the button, thus disabling the button after the first click. We don't have to do anything extra to get these benefits, which is the beauty of data binding.

Observe la otra cosa interesante, que es el enlace de datos angular en juego. Cuando hacemos clic en el botón, se ejecuta nuestra función `toggleFavorite`. Esto invierte el valor de `favorite` de `false` a `true`. Esto a su vez activa el otro enlace Angular, que es la propiedad `disabled` del botón, deshabilitando así el botón después del primer clic. No tenemos que hacer nada adicional para obtener estos beneficios, que es lo bueno del enlace de datos.



Welcome to Stock Market App!

Test Stock Company (TSC)
\$ 85

Add to Favorite

```
Elements  Console  Sources  Network  >  :  X
▶  top  ▾  Filter  All levels  ▾   Group similar  ⌂

Angular is      webpack-internal:///...e/esm5/core.js:3903
running in the development mode. Call enableProdMode() to
enable the production mode.

We are toggling the favorite      stock-item.component.ts:21
state for this stock
  ▶ MouseEvent {isTrusted: true, screenX: 206, screenY: 261,
    clientX: 206, clientY: 164, ...}
```

Figure 2-3. Handling events in an Angular app

Figura 2-3. Manejo de eventos en una aplicación Angular

There are times when we might also care about the actual event triggered. In those cases, Angular gives you access to the underlying DOM event by giving access to a special variable `$event`. You can access it or even pass it to your function as follows:

Hay ocasiones en las que también nos puede interesar el evento real desencadenado. En esos casos, Angular le brinda acceso al evento DOM subyacente al brindarle acceso a una variable especial `$event`. Puedes acceder a él o incluso pasarlo a tu función de la siguiente manera:

```
<div class="stock-container">
  <div class="name">{{name + ' (' + code + ')'}}</div>
  <div class="price">
    [class]="positiveChange ? 'positive' : 'negative'">$ {{price}}</div>
    <button (click)="toggleFavorite($event)"
      [disabled]="favorite">Add to Favorite</button>
  </div>
```

In the HTML, we simply add a reference to the variable `$event`, and pass it in as an argument to our `toggleFavorite` function. We can now refer to it in our component as follows:

En HTML, simplemente agregamos una referencia a la variable `$event` y la pasamos como argumento a nuestra función `toggleFavorite`. Ahora podemos referirnos a él en nuestro componente de la siguiente manera:

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-stock-item',
  templateUrl: './stock-item.component.html',
  styleUrls: ['./stock-item.component.css']
})
export class StockItemComponent implements OnInit {
```

```

public name: string;
public code: string;
public price: number;
public previousPrice: number;
public positiveChange: boolean;
public favorite: boolean;

constructor() { }

ngOnInit() {
  this.name = 'Test Stock Company';
  this.code = 'TSC';
  this.price = 85;
  this.previousPrice = 80;
  this.positiveChange = this.price >= this.previousPrice;
  this.favorite = false;
}

toggleFavorite(event) {
  console.log('We are toggling the favorite state for this stock', event);
  this.favorite = !this.favorite;
}
}

```

When you run the application, you will see that when you click the button, your console log now appends the actual MouseEvent that was triggered, in addition to our previous code.

Cuando ejecute la aplicación, verá que al hacer clic en el botón, el registro de su consola ahora agrega el MouseEvent real que se activó, además de nuestro código anterior.

In a similar manner, we can easily hook onto other standard DOM events that are triggered, like focus, blur, submit, and others like them.

De manera similar, podemos conectarnos fácilmente a otros eventos DOM estándar que se activan, como focus, blur, submit y otros similares.

WHY ANGULAR SHIFTED TO PROPERTY AND EVENT BINDING

POR QUÉ ANGULAR CAMBIÓ AL ENLACE DE PROPIEDAD Y EVENTO

Anyone who has worked on AngularJS would be wondering why the framework developers decided to do such a major breaking change when they created Angular. The binding syntax has changed drastically, as well as the directives and symbols used. In AngularJS, we had `ng-bind`, `ng-src`, and the like for binding from our controllers to the UI, and directives like `ng-click` and `ng-submit` for handling events.

Cualquiera que haya trabajado en AngularJS se preguntaría por qué los desarrolladores del marco decidieron hacer un cambio tan importante cuando crearon Angular. La sintaxis vinculante ha cambiado drásticamente, así como las directivas y símbolos utilizados. En AngularJS, teníamos `ng-bind`, `ng-src` y similares para vincular nuestros controladores a la interfaz de usuario, y directivas como `ng-click` y `ng-submit` para manejar eventos.

This meant that any time there was a new event or property that we wanted to bind to in AngularJS, we would end up writing a wrapper directive that would do the work of translating from AngularJS to the inner workings and vice versa.

Esto significaba que cada vez que había un nuevo evento o propiedad al que queríamos vincularnos en AngularJS, terminábamos escribiendo una directiva contenedora que haría el trabajo de traducir de AngularJS al funcionamiento interno y viceversa.

The other problem with the AngularJS syntax was that there was no clear differentiation between data flowing from our controller to the UI or from the UI to the controller. Both follow the same

syntax, which made understanding the HTML sometimes difficult, and required developers to understand each directive first.

El otro problema con la sintaxis de AngularJS fue que no había una diferenciación clara entre los datos que fluyen desde nuestro controlador a la interfaz de usuario o desde la interfaz de usuario al controlador. Ambos siguen la misma sintaxis, lo que a veces dificultaba la comprensión del HTML y requería que los desarrolladores comprendieran cada directiva primero.

In Angular, we instead rely on core DOM properties and events for binding. This means that if a property or event exists as per the HTML standards, we can bind to it. This also makes it very easy to work with web components that expose proper properties and events, as Angular works with them simply out of the box, without needing to write any additional code. This has also made obsolete the tons of AngularJS directives from the past, such as `ng-click`, `ng-submit`, and so on, and makes it easier for any web developer to quickly understand and work with Angular. You don't have to spend as much time learning Angular-specific knowledge.

En Angular, en cambio, confiamos en las propiedades y eventos centrales del DOM para la vinculación. Esto significa que si una propiedad o evento existe según los estándares HTML, podemos vincularnos a él. Esto también hace que sea muy fácil trabajar con componentes web que exponen propiedades y eventos adecuados, ya que Angular trabaja con ellos de forma inmediata, sin necesidad de escribir ningún código adicional. Esto también ha dejado obsoletas toneladas de directivas de AngularJS del pasado, como `ng-click`, `ng-submit`, etc., y facilita que cualquier desarrollador web comprenda y trabaje rápidamente con Angular. No es necesario que dedique tanto tiempo a aprender conocimientos específicos de Angular.

Furthermore, the square bracket and parentheses notation also makes it very obvious about the flow of data. Any time you see the square bracket notation, you can be assured that it is data flowing from the component into the HTML. Any time you see the parentheses notation, you are guaranteed that it refers to an event and flows from a user action to the component.

Además, la notación entre corchetes y paréntesis también hace que el flujo de datos sea muy obvio. Cada vez que vea la notación entre corchetes, puede estar seguro de que se trata de datos que fluyen desde el componente al HTML. Cada vez que vea la notación entre paréntesis, tiene la garantía de que se refiere a un evento y fluye desde una acción del usuario al componente.

Using Models for Cleaner Code

Uso de modelos para código más limpio

The last part of this chapter covers something that is more of a best practice, but it is worth adopting—especially as we aim to build large, maintainable web applications using Angular. We want to use encapsulation to ensure that our components don't work with lower-level abstractions and properties, like we did previously where the stock widget gets an individual name, price, etc. At the same time, we want to leverage TypeScript to make it easier to understand and reason about our application and its behavior. To this extent, we should ideally model our stock itself as a type in TypeScript, and leverage that instead.

La última parte de este capítulo cubre algo que es más bien una mejor práctica, pero que vale la pena adoptar, especialmente porque nuestro objetivo es crear aplicaciones web grandes y fáciles de mantener utilizando Angular. Queremos utilizar la encapsulación para garantizar que nuestros componentes no funcionen con

abstracciones y propiedades de nivel inferior, como lo hicimos anteriormente, donde el widget de stock obtiene un nombre, precio, etc. individual. Al mismo tiempo, queremos aprovechar TypeScript para facilitar la comprensión y el razonamiento sobre nuestra aplicación y su comportamiento. En este sentido, idealmente deberíamos modelar nuestro propio stock como un tipo en TypeScript y aprovecharlo en su lugar.

The way we would do it in TypeScript is to define an interface or a class with the definition for what belongs in a stock, and use that consistently throughout our application. In this case, since we might want additional logic in addition to just the values (calculating whether the price differential is positive or not, for example), we can use a class.

La forma en que lo haríamos en TypeScript es definir una interfaz o una clase con la definición de lo que pertenece a un stock y usarla de manera consistente en toda nuestra aplicación. En este caso, dado que es posible que queramos lógica adicional además de solo los valores (calcular si el diferencial de precios es positivo o no, por ejemplo), podemos usar una clase.

We can use the Angular CLI to quickly generate a skeleton class for us, by running:

Podemos usar Angular CLI para generar rápidamente una clase de esqueleto ejecutando:

```
ng generate class model/stock
```

This will generate an empty skeleton file called *stock.ts* in a folder called *model*. We can go ahead and change it as follows:

Esto generará un archivo de esqueleto vacío llamado *stock.ts* en una carpeta llamada *modelo*. Podemos seguir adelante y cambiarlo de la siguiente manera:

```

export class Stock {
    favorite: boolean = false;

    constructor(public name: string,
               public code: string,
               public price: number,
               public previousPrice: number) {}

    isPositiveChange(): boolean {
        return this.price >= this.previousPrice;
    }
}

```

This gives us a nice encapsulation while we work with stocks across our application. Note that we didn't actually define the variables `name`, `code`, and so on as properties of the class. This is because we are using TypeScript's syntactic sugar to automatically create the corresponding properties based on the constructor arguments by using the `public` keyword. To learn more about TypeScript classes, refer to the [official documentation](#). In short, we have created a class with five properties, four coming through the constructor and one autoinitialized. Let's see how we might use this now in our component:

Esto nos brinda una buena encapsulación mientras trabajamos con acciones en nuestra aplicación. Tenga en cuenta que en realidad no definimos las variables `name`, `code`, etc. como propiedades de la clase. Esto se debe a que estamos usando el azúcar sintáctico de TypeScript para crear automáticamente las propiedades correspondientes en función de los argumentos del constructor usando la palabra clave `public`. Para obtener más información sobre las clases de TypeScript, consulte la documentación oficial. En resumen, hemos creado una clase con cinco propiedades, cuatro provenientes del constructor y una autoinitializada. Veamos cómo podríamos usar esto ahora en nuestro componente:

```

import { Component, OnInit } from '@angular/core';

import { Stock } from '../model/stock';

```

```

@Component({
  selector: 'app-stock-item',
  templateUrl: './stock-item.component.html',
  styleUrls: ['./stock-item.component.css']
})
export class StockItemComponent implements OnInit {

  public stock: Stock;

  constructor() { }

  ngOnInit() {
    this.stock = new Stock('Test Stock Company', 'TSC', 85, 80);
  }

  toggleFavorite(event) {
    console.log('We are toggling the favorite state for this stock', event);
    this.stock.favorite = !this.stock.favorite;
  }
}

```

In *stock-item.component.ts*, we imported our new model definition at the top, and then replaced all the individual member variables with one variable of type Stock. This simplified the code in the component significantly, and encapsulated all the logic and underlying functionality within a proper TypeScript type. Now let's see how *stock-item.component.html* changes to accommodate this change:

En *stock-item.component.ts*, importamos nuestra nueva definición de modelo en la parte superior y luego reemplazamos todas las variables miembro individuales con una variable de tipo Stock. Esto simplificó significativamente el código del componente y encapsuló toda la lógica y la funcionalidad subyacente dentro de un tipo TypeScript adecuado. Ahora veamos cómo cambia *stock-item.component.html* para adaptarse a este cambio:

```

<div class="stock-container">
  <div class="name">{{stock.name + ' (' + stock.code + ')'}}</div>
  <div class="price">

```

```
[class]="stock.isPositiveChange() ? 'positive' : 'negative'">
  $ {{stock.price}}
</div>
<button (click)="toggleFavorite($event)"
  [disabled]="stock.favorite">Add to Favorite</button>
</div>
```

We have made a few changes in the HTML for our stock item. First, most of our references to the variable are now through the `stock` variable, instead of directly accessing the variables in the component. So `name` became `stock.name`, `code` became `stock.code`, and so on.

Hemos realizado algunos cambios en el HTML de nuestro artículo en `stock`. Primero, la mayoría de nuestras referencias a la variable ahora se realizan a través de la variable `stock`, en lugar de acceder directamente a las variables en el componente. Entonces `name` se convirtió en `stock.name`, `code` se convirtió en `stock.code`, y así sucesivamente.

Also, one more thing of note: our `class` property binding now refers to a function instead of a variable. This is acceptable, as a function is also a valid expression. Angular will just evaluate the function and use its return value to determine the final expression value.

Además, una cosa más a tener en cuenta: nuestro enlace de propiedad `class` ahora se refiere a una función en lugar de una variable. Esto es aceptable, ya que una función también es una expresión válida. Angular simplemente evaluará la función y usará su valor de retorno para determinar el valor de la expresión final.

Conclusion

Conclusión

In this chapter, we started building our very first Angular application. We learned how to bootstrap our Angular application, as well as understand the various pieces that the Angular skeleton application generates and their needs and uses. We then created our very first component, and looked at the steps involved in hooking it up to our application.

En este capítulo, comenzamos a construir nuestra primera aplicación Angular. Aprendimos cómo iniciar nuestra aplicación Angular, así como a comprender las diversas piezas que genera la aplicación de esqueleto Angular y sus necesidades y usos. Luego creamos nuestro primer componente y analizamos los pasos necesarios para conectarlo a nuestra aplicación.

Following that, we added some very basic data to our component, and then used that to understand how basic Angular data binding works, using both interpolation as well as property binding. We then looked at how event binding works, and handled user interactions through it. Finally, we encapsulated some information in a TypeScript class to make our code cleaner and modular.

Después de eso, agregamos algunos datos muy básicos a nuestro componente y luego los usamos para comprender cómo funciona el enlace de datos Angular básico, utilizando tanto la interpolación como el enlace de propiedades. Luego analizamos cómo funciona el enlace de eventos y manejamos las interacciones del usuario a través de él. Finalmente, encapsulamos cierta información en una clase TypeScript para que nuestro código sea más limpio y modular.

In the next chapter, we will go through some basic Angular directives that Angular provides out of the box and how it allows us to work with templates in an effective manner.

En el próximo capítulo, repasaremos algunas directivas básicas de Angular que Angular proporciona de manera inmediata y cómo nos permite trabajar con plantillas de manera efectiva.

Exercise

Ejercicio

Each chapter will have an optional exercise at the end to reinforce some of the concepts covered in the chapter, as well as give you some time to experiment and try your hand at Angular code. Over the course of the book, we will build on the same exercise and keep adding more features to it.

Cada capítulo tendrá un ejercicio opcional al final para reforzar algunos de los conceptos tratados en el capítulo, además de darle algo de tiempo para experimentar y probar el código Angular. A lo largo del libro, nos basaremos en el mismo ejercicio y seguiremos agregándole más funciones.

The finished source code for the exercise is available in the GitHub repository in each chapter's folder. You can refer to it in case you are stuck or want to compare the final solution. Of course, there are multiple ways to solve every problem, so the solution in the repository is one possible way. You may arrive at a slightly different solution in the course of your attempts.

El código fuente terminado para el ejercicio está disponible en el repositorio de GitHub en la carpeta de cada capítulo. Puede consultarla en caso de que esté atascado o desee comparar la solución final. Por supuesto, existen múltiples formas de resolver cada problema, por lo que la solución en el repositorio es una de las posibles. Es posible que llegue a una solución ligeramente diferente en el transcurso de sus intentos.

For the first exercise, try to accomplish the following:

Para el primer ejercicio, intente lograr lo siguiente:

1. Start a new project to build an ecommerce website.
Inicie un nuevo proyecto para crear un sitio web de comercio electrónico.
2. Create a component to display a single product.
Cree un componente para mostrar un solo producto.
3. The product component should display a name, price, and image for the product. You can initialize the component with some defaults for the same. Use any placeholder image you want.
El componente del producto debe mostrar un nombre, precio e imagen del producto. Puede inicializar el componente con algunos valores predeterminados para el mismo. Utilice cualquier imagen de marcador de posición que desee.
4. Highlight the entire element in a different color if the product is on sale. Whether the product is on sale can be an attribute of the product itself.
Resalte todo el elemento en un color diferente si el producto está en oferta. Que el producto esté en oferta puede ser un atributo del producto en sí.
5. Add buttons to increase and decrease the quantity of the product in the cart. The quantity in the cart should be visible in the UI. Disable the button if the quantity is already zero.
Añade botones para aumentar y disminuir la cantidad del producto en el carrito. La cantidad en el carrito debe ser visible en la interfaz de usuario. Desactive el botón si la cantidad ya es cero.

All of this can be accomplished using concepts covered in this chapter. You can check out the finished solution in [*chapter2/exercise/ecommerce*](#).

Todo esto se puede lograr utilizando los conceptos tratados en este capítulo. Puede consultar la solución terminada en el capítulo [*2/ejercicio/ecommerce*](#).

Chapter 3. Useful Built-In Angular Directives

Capítulo 3. Directivas angulares integradas útiles

In the previous chapter, we got started with our very first Angular application, and got a feel for using the Angular CLI to begin a new project and create components in it. We got a basic sense of how to use Angular's data and event binding mechanisms as well.

En el capítulo anterior, comenzamos con nuestra primera aplicación Angular y nos familiarizamos con el uso de Angular CLI para comenzar un nuevo proyecto y crear componentes en él. También tenemos una idea básica de cómo utilizar los mecanismos de enlace de eventos y datos de Angular.

In this chapter, we will first understand what directives are in Angular and how they are different from components. We will then cover some basic directives that Angular provides out of the box and the use cases where they're applicable. By the end of the chapter, you should be comfortable using most of the out-of-the-box directives that Angular provides and understand when and where to use them.

En este capítulo, primero comprenderemos qué directivas hay en Angular y en qué se diferencian de los componentes. Luego cubriremos algunas directivas básicas que Angular proporciona de forma inmediata y los casos de uso en los que son aplicables. Al final

del capítulo, debería sentirse cómodo utilizando la mayoría de las directivas listas para usar que proporciona Angular y comprender cuándo y dónde usarlas.

Directives and Components

Directivas y componentes

A directive in Angular allows you to attach some custom functionality to elements in your HTML. A component in Angular, like the one we built in the previous chapter, is a directive that provides both functionality and UI logic. It is fundamentally an element that encapsulates its behavior and rendering logic.

Una directiva en Angular le permite adjuntar algunas funciones personalizadas a elementos en su HTML. Un componente en Angular, como el que creamos en el capítulo anterior, es una directiva que proporciona tanto funcionalidad como lógica de interfaz de usuario. Es fundamentalmente un elemento que encapsula su comportamiento y lógica de representación.

Noncomponent directives, on the other hand, usually work on and modify existing elements. These can be further classified into two types:

Por otra parte, las directivas que no contienen componentes suelen trabajar y modificar elementos existentes. Estos se pueden clasificar además en dos tipos:

Attribute directives

Directivas de atributos

Attribute directives change the look and feel, or the behavior, of an existing element or component that it is applied on. NgClass

and `NgStyle`, which we will see later in this chapter, are examples of attribute directives.

Las directivas de atributos cambian la apariencia o el comportamiento de un elemento o componente existente al que se aplica. `NgClass` y `NgStyle`, que veremos más adelante en este capítulo, son ejemplos de directivas de atributos.

Structural directives

Directivas estructurales

Structural directives change the DOM layout by adding or removing elements from the view. `NgIf` and `NgFor` are examples of structural directives that we will see later in this chapter.

Las directivas estructurales cambian el diseño DOM agregando o eliminando elementos de la vista. `NgIf` y `NgFor` son ejemplos de directivas estructurales que veremos más adelante en este capítulo.

Built-In Attribute Directives

Directivas de atributos integradas

We will first explore attribute directives. There are two basic attribute directives that Angular provides out of the box, which are the `NgClass` and the `NgStyle` directives. Both of these are alternatives for the class and style bindings, of which we saw the class binding in the previous chapter.

Primero exploraremos las directivas de atributos. Hay dos directivas de atributos básicas que Angular proporciona de forma inmediata, que son las directivas `NgClass` y `NgStyle`. Ambas son alternativas para las vinculaciones de clase y estilo, de las cuales vimos la vinculación de clase en el capítulo anterior.

TIP CONSEJO

We generally refer to the directive with the name of its class, which is why we call the directive NgClass or NgIf. But the same directive, when used as an HTML attribute, is usually in camel-case, like ngClass or ngIf. Keep this in mind as we go along.

Generalmente nos referimos a la directiva con el nombre de su clase, por eso llamamos a la directiva NgClass o NgIf. Pero la misma directiva, cuando se usa como atributo HTML, generalmente está en formato camel, como ngClass o ngIf. Tenga esto en cuenta a medida que avanzamos.

NgClass

NgClase

The NgClass directive allows us to apply or remove multiple CSS classes simultaneously from an element in our HTML. Previously, we applied a single class to our element to highlight whether it was a positive or a negative change as follows:

La directiva NgClass nos permite aplicar o eliminar múltiples clases CSS simultáneamente de un elemento en nuestro HTML.

Anteriormente, aplicamos una sola clase a nuestro elemento para resaltar si fue un cambio positivo o negativo de la siguiente manera:

```
<div [class]="stock.isPositiveChange() ? 'positive' : 'negative'">
  $ {{stock.price}}
</div>
```

In this example, we simply look at a boolean value, and then decide whether to apply the class positive or negative based on that. But what if we had to apply multiple CSS classes? And they were all (or a lot of them) conditional? You would end up having to write code that does string generation based on these multiple conditions, so

that you could have one string that represents all the classes that need to be applied.

En este ejemplo, simplemente observamos un valor booleano y luego decidimos si aplicar la clase `positive` o `negative` en función de eso. Pero ¿y si tuviéramos que aplicar múltiples clases de CSS? ¿Y eran todos (o muchos de ellos) condicionales? Terminaría teniendo que escribir código que genere cadenas en función de estas múltiples condiciones, de modo que pueda tener una cadena que represente todas las clases que deben aplicarse.

This is cruft code that is not worth writing or maintaining. So for these kinds of situations, Angular provides the `NgClass` directive, which can take a JavaScript object as input. For each key in the object that has a truthy value, Angular will add that key (the key itself, not the value of the key!) as a class to the element. Similarly, each key in the object that has a falsy value will be removed as a class from that element.

Este es un código crudo que no vale la pena escribir ni mantener. Entonces, para este tipo de situaciones, Angular proporciona la directiva `NgClass`, que puede tomar un objeto JavaScript como entrada. Para cada clave en el objeto que tenga un valor verdadero, Angular agregará esa clave (la clave en sí, no el valor de la clave!) como una clase al elemento. De manera similar, cada clave del objeto que tenga un valor falso se eliminará como clase de ese elemento.

TRUTHY AND FALSEY IN JAVASCRIPT

VERDAD Y MENTIRA EN JAVASCRIPT

JavaScript allows us to use non-boolean values in conditional statements. Thus, instead of just `true` and `false`, a whole set of values are equivalent to true and false. In JavaScript, the following values are treated as `falsy`:

JavaScript nos permite utilizar valores distintos de boolean en declaraciones condicionales. Por lo tanto, en lugar de solo `true` y `false`, un conjunto completo de valores equivale a verdadero y falso. En JavaScript, los siguientes valores se tratan como `falsy`:

- `undefined`
`undefined`
- `null`
`null`
- `NaN`
`NaN`
- `0`
- `""` (any empty string)
`""` (cualquier cadena vacía)
- `false` (the boolean value)
`false` (el valor booleano)

Any other value is treated as `truthy`, including, but not limited to:

Cualquier otro valor se trata como `truthy`, incluido, entre otros:

- Any nonzero number
Cualquier número distinto de cero
- Any nonempty string
Cualquier cadena no vacía
- Any nonnull object or array
Cualquier objeto o matriz no nulo
- `true` (the boolean value)

`true` (el valor booleano)

Another way to easily remember is that any non-falsy value is truthy. We end up using these concepts quite often in conditionals in our applications.

Otra forma de recordar fácilmente es que cualquier valor que no sea falso es verdadero. Terminamos usando estos conceptos con bastante frecuencia en condicionales en nuestras aplicaciones.

Let's take an example to see this in action. Say we want to extend our example from before, where instead of just a positive and a negative class, we want to add another class that dictates whether it is a large or a small change. We want it to be a small change (denoted by the CSS class `small-change`) if the change percentage is less than 1%; otherwise it would be a large change (denoted by the CSS class `large-change`).

Tomemos un ejemplo para ver esto en acción. Digamos que queremos ampliar nuestro ejemplo anterior, donde en lugar de solo una clase positiva y una negativa, queremos agregar otra clase que dicte si se trata de un cambio grande o pequeño. Queremos que sea un pequeño cambio (indicado por la clase CSS `small-change`) si el porcentaje de cambio es inferior al 1%; de lo contrario, sería un cambio grande (indicado por la clase CSS `large-change`).

We will build on the example from the previous chapter, so if you don't have it, feel free to grab the final code we were working with from the [GitHub repository](#). The codebase is in `chapter2/stock-market`, which is the base for the code in this chapter.

Nos basaremos en el ejemplo del capítulo anterior, por lo que si no lo tiene, no dude en obtener el código final con el que estábamos trabajando del repositorio de GitHub. La base del código se encuentra en el capítulo 2/mercado de valores, que es la base del código de este capítulo.

First, we can add the new classes to the `src/app/stock/stock-item/stock-item.component.css` file:

Primero, podemos agregar las nuevas clases al archivo `src/app/stock/stock-item/stock-item.component.css`:

```
.stock-container {  
  border: 1px solid black;  
  border-radius: 5px;  
  display: inline-block;  
  padding: 10px;  
}  
  
.positive {  
  color: green;  
}  
  
.negative {  
  color: red;  
}  
  
.large-change {  
  font-size: 1.2em;  
}  
  
.small-change {  
  font-size: 0.8em;  
}
```

Next, we can change our component class to calculate and keep the JSON object ready with the classes to apply. We modify the `src/app/stock/stock-item/stock-item.component.ts` file to first calculate the difference between the current and the previous price, and then create an object that holds all the classes we need to apply:

A continuación, podemos cambiar la clase de nuestro componente para calcular y mantener listo el objeto JSON con las clases a aplicar. Modificamos el archivo `src/app/stock/stock-item/stock-item.component.ts` para calcular primero la diferencia entre el precio actual y el anterior, y luego creamos un objeto que contiene todas las clases que necesitamos aplicar:

```

import { Component, OnInit } from '@angular/core';

import { Stock } from '../../model/stock';

@Component({
  selector: 'app-stock-item',
  templateUrl: './stock-item.component.html',
  styleUrls: ['./stock-item.component.css']
})
export class StockItemComponent implements OnInit {

  public stock: Stock;
  public stockClasses;

  constructor() { }

  ngOnInit() {
    this.stock = new Stock('Test Stock Company', 'TSC', 85, 80);
    let diff = (this.stock.price / this.stock.previousPrice) - 1;
    let largeChange = Math.abs(diff) > 0.01;
    this.stockClasses = {
      "positive": this.stock.isPositiveChange(),
      "negative": !this.stock.isPositiveChange(),
      "large-change": largeChange,
      "small-change": !largeChange
    };
  }

  toggleFavorite(event) {
    console.log('We are toggling the favorite state for this stock', event);
    this.stock.favorite = !this.stock.favorite;
  }
}

```

In the component code, we created a `stockClasses` object with four keys: `positive`, `negative`, `large-change`, and `small-change`. Based on the current price and previous prices, each of these keys will have a value of `true` or `false`.

En el código del componente, creamos un objeto `stockClasses` con cuatro claves: `positive`, `negative`, `large-change` y `small-change`. Según el precio actual y los precios anteriores, cada una de estas claves tendrá un valor de `true` o `false`.

Now, let's see how we can use the NgClass directive to use this instead of the class binding we were previously using in the *src/app/stock/stock-item/stock-item.component.html* file:

Ahora, veamos cómo podemos usar la directiva NgClass para usar esto en lugar del enlace class que estábamos usando anteriormente en el componente *src/app/stock/stock-item/stock-item.component.html*:

```
<div class="stock-container">
  <div class="name">{{stock.name + ' (' + stock.code + ')'}}</div>
  <div class="price"
    [ngClass]="stockClasses">$ {{stock.price}}</div>
  <button (click)="toggleFavorite($event)"
    [disabled]="stock.favorite">Add to Favorite</button>
</div>
```

We primarily replaced the
Principalmente reemplazamos el

```
[class]="stock.isPositiveChange() ? 'positive' : 'negative'"
```

line with the following:

Línea con lo siguiente:

```
[ngClass]="stockClasses"
```

Now if you run this application in your browser (`ng serve`, in case you have forgotten), you can see the impact of your changes. The price is now shown in green, as well as in slightly larger font, since it has applied both the `positive` and the `large-change` classes. You can play around with the values of price and previous price in the component class to see the different combinations reflected.

Ahora, si ejecuta esta aplicación en su navegador (`ng serve`, en caso de que lo haya olvidado), podrá ver el impacto de sus cambios. El precio ahora se muestra en verde, así como en una fuente un poco más grande, ya que se han aplicado las clases `positive` y `large-`

change. Puedes jugar con los valores de precio y precio anterior en la clase de componente para ver reflejadas las diferentes combinaciones.

Another thing to note is that unlike before, where the `class` binding was overwriting our initial class from the element, the `NgClass` directive retains the classes on the element.

Otra cosa a tener en cuenta es que, a diferencia de antes, donde el enlace `class` sobrescribía nuestra clase inicial del elemento, la directiva `NgClass` conserva las clases del elemento.

The finished code for the preceding example is available in the `chapter3/ng-class` folder in the GitHub repository.

El código terminado para el ejemplo anterior está disponible en la carpeta `capítulo3/ng-class` en el repositorio de GitHub.

You should consider using the `NgClass` directive if you have the use case of having to apply various different CSS classes on an element conditionally. It makes it easy to reason and understand how and what classes are applied, and it also makes it easy to unit test the logic of selecting classes separate from the logic of applying classes to the elements.

Debería considerar usar la directiva `NgClass` si tiene el caso de uso de tener que aplicar varias clases CSS diferentes en un elemento de manera condicional. Facilita razonar y comprender cómo y qué clases se aplican, y también facilita la prueba unitaria de la lógica de selección de clases separada de la lógica de aplicación de clases a los elementos.

NgStyle

NgEstilo

The NgStyle directive is the lower-level equivalent of the NgClass directive. It operates in a manner similar to the NgClass in that it takes a JSON object and applies it based on the values of the keys. But the NgStyle directive works at a CSS style/properties level. The keys and values it expects are CSS properties and attributes rather than class names.

La directiva NgStyle es el equivalente de nivel inferior de la directiva NgClass. Funciona de manera similar a NgClass en el sentido de que toma un objeto JSON y lo aplica en función de los valores de las claves. Pero la directiva NgStyle funciona a nivel de estilo/propiedades CSS. Las claves y valores que espera son propiedades y atributos CSS en lugar de nombres de clases.

Considering that our example with NgClass was using simple CSS classes each affecting only one CSS property, let's see how we can translate the same example using the NgStyle directive instead.

First, we need to make a change to the *src/app/stock/stock-item/stock-item.component.ts* file to create the style object based on the stock properties:

Teniendo en cuenta que nuestro ejemplo con NgClass estaba usando clases CSS simples, cada una de las cuales afectaba solo a una propiedad CSS, veamos cómo podemos traducir el mismo ejemplo usando la directiva NgStyle. Primero, necesitamos hacer un cambio en el archivo *src/app/stock/stock-item/stock-item.component.ts* para crear el objeto de estilo basado en las propiedades del stock:

```
import { Component, OnInit } from '@angular/core';
import { Stock } from '../../model/stock';

@Component({
```

```

        selector: 'app-stock-item',
        templateUrl: './stock-item.component.html',
        styleUrls: ['./stock-item.component.css']
    })
export class StockItemComponent implements OnInit {

    public stock: Stock;
    public stockStyles;

    constructor() { }

    ngOnInit() {
        this.stock = new Stock('Test Stock Company', 'TSC', 85, 80);
        let diff = (this.stock.price / this.stock.previousPrice) - 1;
        let largeChange = Math.abs(diff) > 0.01;
        this.stockStyles = {
            "color": this.stock.isPositiveChange() ? "green" : "red",
            "font-size": largeChange ? "1.2em" : "0.8em"
        };
    }

    toggleFavorite(event) {
        console.log('We are toggling the favorite state for this stock', event);
        this.stock.favorite = !this.stock.favorite;
    }
}

```

Similar to the previous section, we have created a `stockStyles` object. In the initialization code, we have initialized the `stockStyles` object with the keys `color` and `font-size`. Its values are CSS attributes that are generated based on the `stock` properties. We can then use this `stockStyles` object as an input to the `NgStyle` directive for binding.

De manera similar a la sección anterior, hemos creado un objeto `stockStyles`. En el código de inicialización, hemos inicializado el objeto `stockStyles` con las claves `color` y `font-size`. Sus valores son atributos CSS que se generan en función de las propiedades del `stock`. Luego podemos usar este objeto `stockStyles` como entrada a la directiva `NgStyle` para vinculación.

We can now change our HTML to use this information by editing the `src/app/stock/stock-item/stock-item.component.html` file as follows:

Ahora podemos cambiar nuestro HTML para usar esta información editando el archivo `src/app/stock/stock-item/stock-item.component.html` de la siguiente manera:

```
<div class="stock-container">
  <div class="name">{{stock.name + ' (' + stock.code + ')'}}</div>
  <div class="price"
    [ngStyle]="stockStyles">$ {{stock.price}}</div>
  <button (click)="toggleFavorite($event)"
    [disabled]="stock.favorite">Add to Favorite</button>
</div>
```

We have added a binding for the `NgStyle` directive:

Hemos agregado un enlace para la directiva `NgStyle`:

```
[ngStyle]="stockStyles"
```

Angular will look at the keys and values of the `stockStyles` object and add those particular styles to the HTML element. You can again run the application and try changing the values of the price and previous price to see this in action.

Angular observará las claves y los valores del objeto `stockStyles` y agregará esos estilos particulares al elemento HTML. Puede ejecutar nuevamente la aplicación e intentar cambiar los valores del precio y el precio anterior para verlo en acción.

The finished code for the preceding example is available in the `chapter3/ng-style` folder in the GitHub repository.

El código terminado para el ejemplo anterior está disponible en la carpeta de estilo `capítulo3/ng` en el repositorio de GitHub.

It is generally preferable to use the `class` or `NgClass` bindings to change the look and feel of your application, but the `NgStyle` does give you another tool in your toolkit, in case you have the use case

of changing the CSS style of elements and cannot (for whatever reasons) use classes to perform the same.

Generalmente es preferible usar los enlaces de clase o NgClass para cambiar la apariencia de su aplicación, pero NgStyle le brinda otra herramienta en su kit de herramientas, en caso de que tenga el caso de uso de cambiar el estilo CSS de los elementos y no puede (por cualquier motivo) usar clases para realizar lo mismo.

Alternative Class and Style Binding Syntax

Sintaxis de enlace de clase y estilo alternativa

We covered using the `[class]` binding syntax in the previous chapter, as well as the NgClass alternative to dynamically add classes to our elements using Angular. There is a third alternative for both classes and styles, which is to use a singular version of the class and style binding that adds and removes one particular class/style, instead of the all-or-nothing approach of the `[class]` binding.

Cubrimos el uso de la sintaxis de enlace `[class]` en el capítulo anterior, así como la alternativa NgClass para agregar clases dinámicamente a nuestros elementos usando Angular. Existe una tercera alternativa tanto para clases como para estilos, que consiste en utilizar una versión singular del enlace de clase y estilo que agrega y elimina una clase/estilo en particular, en lugar del enfoque de todo o nada de `[class]` vinculante.

We can add or remove individual classes based on evaluating a truthy expression in Angular with the following syntax:

Podemos agregar o eliminar clases individuales basándonos en evaluar una expresión verdadera en Angular con la siguiente sintaxis:

```
[class.class-name]="expression"
```

We would replace `class-name` with the particular CSS class we want to apply/remove on our element, and replace `expression` in the syntax with a valid JavaScript expression that would either return a truthy or falsy value.

Reemplazaremos `class-name` con la clase CSS particular que queremos aplicar/eliminar en nuestro elemento, y reemplazaremos `expression` en la sintaxis con una expresión JavaScript válida que devolvería un valor verdadero o falso.

Let's modify our stock example to apply and remove the positive and negative CSS classes using this syntax instead. We don't have to make any changes in the component or the CSS, and only make the following changes to the `src/app/stock/stock-item/stock-item.component.html` file:

Modifiquemos nuestro ejemplo estándar para aplicar y eliminar las clases CSS positive y negative usando esta sintaxis. No tenemos que realizar ningún cambio en el componente ni en el CSS, y solo realizamos los siguientes cambios en el archivo `src/app/stock/stock-item/stock-item.component.html`:

```
<div class="stock-container">
  <div class="name">{{stock.name + ' (' + stock.code + ')'}}</div>
  <div class="price"
    [class.positive]="stock.isPositiveChange()"
    [class.negative]="!stock.isPositiveChange()">$ {{stock.price}}</div>
  <button (click)="toggleFavorite($event)"
    [disabled]="stock.favorite">Add to Favorite</button>
</div>
```

Notice the two new lines:

Observe las dos nuevas líneas:

```
[class.positive]="stock.isPositiveChange()"
[class.negative]="!stock.isPositiveChange()"
```

We are telling Angular to apply the CSS class `positive` if there is a positive change (based on the call to `stock.isPositiveChange()`), and remove it if this is falsy. Similarly, we are doing the negation of that for `negative`. This is a nice simple way to deal with CSS classes if you only have to add/remove one or two classes. Note that this also works if the class name has dashes and so on. For example, you could do:

Le estamos diciendo a Angular que aplique la clase CSS `positive` si hay un cambio positivo (basado en la llamada a `stock.isPositiveChange()`) y que la elimine si es falso. De manera similar, estamos haciendo la negación de eso para `negative`. Esta es una forma sencilla y agradable de manejar las clases de CSS si solo tiene que agregar/eliminar una o dos clases. Tenga en cuenta que esto también funciona si el nombre de la clase tiene guiones, etc. Por ejemplo, podrías hacer:

```
[class.large-change]="someExpressionHere"
```

and this would apply the class `large-change` if `someExpressionHere` evaluated to true. Also note that this keeps the original class from the element intact (the `price` class), which the basic `[class]` binding does not. That is yet another advantage of this syntax.

y esto aplicaría la clase `large-change` si `someExpressionHere` se evaluara como verdadero. También tenga en cuenta que esto mantiene intacta la clase original del elemento (la clase `price`), algo que no hace el enlace básico `[class]`. Esa es otra ventaja más de esta sintaxis.

It is, though, preferred to use the `NgClass` directive any time you have to deal with more than one or two classes, as it makes it more manageable and easier to test.

Sin embargo, es preferible utilizar la directiva `NgClass` cada vez que tenga que tratar con más de una o dos clases, ya que la hace más

manejable y fácil de probar.

The style binding is also something similar, and can be done like follows:

El enlace de estilo también es algo similar y se puede hacer de la siguiente manera:

```
[style.background-color]="stock.isPositiveChange() ? 'green' : 'red'"
```

You can read more about the style binding in the [official Angular documentation](#).

Puede leer más sobre el enlace de estilo en la documentación oficial de Angular.

Built-In Structural Directives

Directivas estructurales incorporadas

Structural directives, as discussed earlier, are responsible for changing the layout of the HTML by adding, removing, or modifying elements from the DOM. Just like other directives that are not components, structural directives are applied on a pre-existing element, and the directive then operates on the content of that element.

Las directivas estructurales, como se analizó anteriormente, son responsables de cambiar el diseño del HTML agregando, eliminando o modificando elementos del DOM. Al igual que otras directivas que no son componentes, las directivas estructurales se aplican sobre un elemento preexistente y luego la directiva opera sobre el contenido de ese elemento.

Structural directives in Angular follow a very particular syntax, which makes it easy to recognize when a directive is a structural directive

versus a normal one. All structural directives in Angular start with an asterisk (*), like:

Las directivas estructurales en Angular siguen una sintaxis muy particular, lo que hace que sea fácil reconocer cuándo una directiva es estructural o normal. Todas las directivas estructurales en Angular comienzan con un asterisco (*), como:

```
<div *ngIf="stock.favorite"></div>
```

Unlike the data- or event-binding syntaxes, there are no square brackets or parentheses. Just a plain `*ngIf` followed by the expression. Angular understands the expression and translates it into the final HTML. To understand this process a bit more, refer to the following sidebar.

A diferencia de las sintaxis de enlace de datos o eventos, no hay corchetes ni paréntesis. Simplemente un simple `*ngIf` seguido de la expresión. Angular comprende la expresión y la traduce al HTML final. Para comprender un poco más este proceso, consulte la siguiente barra lateral.

SYNTAX AND REASONING

SINTAXIS Y RAZONAMIENTO

You might be curious about the asterisk syntax, and rightfully so. The star syntax is syntactic sugar that is replaced underneath by Angular to a series of steps, resulting in the final view you get. In fact, unlike the rest of the directives where you can use data/event binding correctly with expressions, with the structural directives, what you instead use is a microsyntax, a mini-language of sorts that Angular uses to accomplish certain things.

Es posible que sienta curiosidad por la sintaxis de los asteriscos, y con razón. La sintaxis estrella es azúcar sintáctica que se reemplaza debajo por Angular en una serie de pasos, lo que da como resultado la vista final que obtiene. De hecho, a diferencia del resto de las directivas donde puede usar el enlace de datos/eventos correctamente con expresiones, con las directivas estructurales, lo que usa es una microsintaxis, una especie de minilenguaje que Angular usa para lograr ciertas cosas.

You can skip this section if you don't want to dive into the innards of how exactly Angular works with these expressions.

Puede omitir esta sección si no desea profundizar en cómo funciona exactamente Angular con estas expresiones.

Let's walk through a simple lifecycle of how Angular translates a simple expression into its final state. Say we started with:

Repasemos un ciclo de vida simple de cómo Angular traduce una expresión simple a su estado final. Digamos que comenzamos con:

```
<div *ngIf="stock.favorite">
  <button>Remove from favorite</button>
</div>
```

Angular will recognize the structural directive and transform it into a template directive, to create something along the lines of the following:

Angular reconocerá la directiva estructural y la transformará en una directiva de plantilla, para crear algo como lo siguiente:

```
<div template="ngIf stock.favorite">
  <button>Remove from favorite</button>
</div>
```

This template directive is using the microsyntax that Angular recognizes and then translates it into an Angular template that surrounds this particular element, to something like:

Esta directiva de plantilla utiliza la microsintaxis que Angular reconoce y luego la traduce a una plantilla de Angular que rodea este elemento en particular, a algo como:

```
<ng-template [ngIf]="stock.favorite">
  <div>
    <button>Remove from favorite</button>
  </div>
</ng-template>
```

Finally, based on the value of `stock.favorite`, the inner div would get rendered to the DOM, or removed from it. You can technically use any of these syntaxes to achieve the same effect, but it is recommended for consistency and ease of readability.

Finalmente, según el valor de `stock.favorite`, el div interno se representaría en el DOM o se eliminaría de él. Técnicamente, puede utilizar cualquiera de estas sintaxis para lograr el mismo efecto, pero se recomienda por motivos de coherencia y facilidad de lectura.

NgIf

ngif

We will first take a look at the trusty and often used structural directive NgIf. The NgIf directive allows you to conditionally hide or show elements in your UI. The syntax, as mentioned earlier, starts with an asterisk as it is a structural directive that can conditionally remove or add elements to our rendered HTML.

Primero echaremos un vistazo a la directiva estructural confiable y de uso frecuente NgIf. La directiva NgIf le permite ocultar o mostrar elementos en su interfaz de usuario de forma condicional. La sintaxis, como se mencionó anteriormente, comienza con un asterisco, ya que es una directiva estructural que puede eliminar o agregar elementos condicionalmente a nuestro HTML renderizado.

The NgIf uses one of the simplest microsyntaxes of all the structural directives, as it simply expects the expression provided to it to evaluate to a truthy value. This is the JavaScript concept of truthiness (as explained previously), so the boolean true value, a nonzero number, a nonempty string, and a nonnull object would all be treated as true. This also makes it convenient to have templates that show up if certain objects are present and nonnull.

El NgIf utiliza una de las microsintaxis más simples de todas las directivas estructurales, ya que simplemente espera que la expresión que se le proporciona se evalúe con un valor verdadero. Este es el concepto de veracidad de JavaScript (como se explicó anteriormente), por lo que el valor verdadero booleano, un número distinto de cero, una cadena no vacía y un objeto no nulo se tratarían como verdaderos. Esto también hace que sea conveniente tener plantillas que se muestren si ciertos objetos están presentes y no son nulos.

We will build on the examples from the previous chapter, so if you don't have it, feel free to grab the final code we were working with

from the [GitHub repository](#). The codebase is in *chapter2/stock-market*, which is the base for the code in this chapter.

Nos basaremos en los ejemplos del capítulo anterior, por lo que si no lo tiene, no dude en obtener el código final con el que estábamos trabajando del repositorio de GitHub. La base del código se encuentra en el capítulo 2/mercado de valores, que es la base del código de este capítulo.

Let's modify the Add to Favorite button such that instead of getting disabled, we hide the button if the stock is already favorited. We don't need to make any changes to the component or CSS code, but instead just to the *src/app/stock/stock-item/stock-item.component.html* file as follows:

Modifiquemos el botón Agregar a favoritos de modo que, en lugar de desactivarlo, ocultemos el botón si la acción ya está como favorita. No necesitamos realizar ningún cambio en el componente o código CSS, sino solo en el archivo *src/app/stock/stock-item/stock-item.component.html* de la siguiente manera:

```
<div class="stock-container">
  <div class="name">{{stock.name + ' (' + stock.code + ')'}}</div>
  <div class="price"
    [class]="stock.isPositiveChange() ? 'positive' : 'negative'">
    $ {{stock.price}}
  </div>
  <button (click)="toggleFavorite($event)"
    *ngIf="!stock.favorite">Add to Favorite</button>
</div>
```

On the button, we added the conditionality with `*ngIf="!stock.favorite"`. This tells Angular to add the element if the stock is not favorited, and remove it from the DOM if it is currently a favorite stock. Now when you load the page, you will see the button by default. Once you click the Add to Favorite button, the boolean will flip and the stock will now be favorited. Angular will automatically at this point hide the button from the UI.

En el button, agregamos la condicionalidad con `*ngIf="!stock.favorite"`. Esto le indica a Angular que agregue el elemento si la acción no es favorita y que lo elimine del DOM si actualmente es una acción favorita. Ahora, cuando cargue la página, verá el botón de forma predeterminada. Una vez que haga clic en el botón Agregar a favoritos, el valor booleano se invertirá y la acción ahora será favorita. En este punto, Angular ocultará automáticamente el botón de la interfaz de usuario.

If you inspect the HTML of the page, you will see that the element is actually removed from the rendered DOM.

Si inspecciona el HTML de la página, verá que el elemento en realidad se elimina del DOM renderizado.

WHY REMOVE VERSUS HIDE ELEMENT? ¿POR QUÉ ELIMINAR U OCULTAR UN ELEMENTO?

Why does *ngIf remove the element, instead of hiding it via CSS? The reason comes down to performance and the implications thereof. Angular continues to listen and watch for events on all elements in the DOM. Removing the element from the DOM is a good way to ensure that it reduces any impact on performance, especially when the element being removed is a resource-intensive component (think a graph, or an autoupdating widget). This does mean that it is slightly less efficient when the condition toggles (since it involves a DOM addition/removal), but in the grand context of things, it is more efficient.

¿Por qué *ngIf elimina el elemento en lugar de ocultarlo mediante CSS? La razón se reduce al rendimiento y sus implicaciones. Angular continúa escuchando y observando eventos en todos los elementos del DOM. Eliminar el elemento del DOM es una buena manera de garantizar que reduzca cualquier impacto en el rendimiento, especialmente cuando el elemento que se elimina es un componente que consume muchos recursos (piense en un gráfico o un widget de actualización automática). Esto significa que es un poco menos eficiente cuando la condición cambia (ya que implica una adición/eliminación de DOM), pero en el contexto general de las cosas, es más eficiente.

The NgIf directive, along with the NgFor, which we will cover in the following sections, are work-horse directives. They do a lot of the heavy lifting and are commonly used in most applications you will develop.

La directiva NgIf, junto con la NgFor, que cubriremos en las siguientes secciones, son directivas de caballo de batalla. Hacen

gran parte del trabajo pesado y se usan comúnmente en la mayoría de las aplicaciones que desarrollará.

NgFor

ngfor

While the NgIf directive is used for conditionally showing/hiding elements, the NgFor directive is used for creating multiple elements, usually one for each instance of some or the other object in an array. It is a common practice to have a template, and then create an instance of that template for each instance of our object.

Mientras que la directiva NgIf se usa para mostrar/ocultar elementos condicionalmente, la directiva NgFor se usa para crear múltiples elementos, generalmente uno para cada instancia de uno u otro objeto en una matriz. Es una práctica común tener una plantilla y luego crear una instancia de esa plantilla para cada instancia de nuestro objeto.

NGFOR OR NGFOROF NGFOR O NGFOROF

In this book, we use NgFor to refer to the *ngFor directive. But technically, the *ngFor directive uses the class called NgForOf under the hood. Thus, you might see people use NgFor and NgForOf interchangeably when referring to the *ngFor directive in Angular.

En este libro, usamos NgFor para referirnos a la directiva *ngFor. Pero técnicamente, la directiva *ngFor usa la clase llamada NgForOf bajo el capó. Por lo tanto, es posible que veas que las personas usan NgFor y NgForOf indistintamente cuando se refieren a la directiva *ngFor en Angular.

NgFor uses a special microsyntax that comes with a set of mandatory and optional segments. Let's modify our example from

Chapter 2 to now show a list of stocks, instead of one individual stock.

NgFor utiliza una microsyntax especial que viene con un conjunto de segmentos obligatorios y opcionales. Modifiquemos nuestro ejemplo del Capítulo 2 para mostrar ahora una lista de acciones, en lugar de una acción individual.

First, we will modify the `src/app/stock/stock-item.component.ts` file, just for the sake of demonstration, to have an array of stocks instead of one stock:

Primero, modificaremos el archivo `src/app/stock/stock-item.component.ts`, solo a modo de demostración, para tener una serie de acciones en lugar de una sola acción:

```
import { Component, OnInit } from '@angular/core';

import { Stock } from '../model/stock';

@Component({
  selector: 'app-stock-item',
  templateUrl: './stock-item.component.html',
  styleUrls: ['./stock-item.component.css']
})
export class StockItemComponent implements OnInit {

  public stocks: Array<Stock>;

  constructor() { }

  ngOnInit() {
    this.stocks = [
      new Stock('Test Stock Company', 'TSC', 85, 80),
      new Stock('Second Stock Company', 'SSC', 10, 20),
      new Stock('Last Stock Company', 'LSC', 876, 765)
    ];
  }

  toggleFavorite(event, index) {
    console.log('We are toggling the favorite state for stock', index, event);
    this.stocks[index].favorite = !this.stocks[index].favorite;
  }
}
```

Generally, we want to keep a stock-item for one particular stock only, but we will circumvent that for this particular example. We changed stock to be stocks, an array of stock objects. We then created some dummy stocks in our initialization. Finally, we changed the toggleFavorite to take in an index as a parameter, instead of working with the current stock.

Generalmente, queremos mantener un stock-item solo para una acción en particular, pero lo evitaremos para este ejemplo en particular. Cambiamos stock para que sea stocks, una matriz de objetos de stock. Luego creamos algunas acciones ficticias en nuestra inicialización. Finalmente, cambiamos el toggleFavorite para tomar un index como parámetro, en lugar de trabajar con el stock actual.

Next, let's see how we can now modify our HTML in *src/app/stock/stock-item.component.html* to leverage the power of the NgFor directive:

A continuación, veamos cómo podemos modificar nuestro HTML en *src/app/stock/stock-item.component.html* para aprovechar el poder de la directiva NgFor:

```
<div class="stock-container" *ngFor="let stock of stocks; index as i">
  <div class="name">{{stock.name + ' (' + stock.code + ')'}}</div>
  <div class="price"
    [class]="stock.isPositiveChange() ? 'positive' : 'negative'">
    $ {{stock.price}}
  </div>
  <button (click)="toggleFavorite($event, i)"
    [disabled]="stock.favorite">Add to Favorite</button>
</div>
```

We updated our parent container with the NgFor directive. Let's explore how we have used it in a bit more detail:

Actualizamos nuestro contenedor principal con la directiva NgFor. Exploraremos cómo lo hemos usado con un poco más de detalle:

```
*ngFor="let stock of stocks; index as i"
```

The first part of the microsyntax is basically our `for` loop. We create a template instance variable named `stock`, which will be available within the scope of the element created. This can be equated to a standard `for-each` loop, with the variable `stock` referring to each individual item in the array. The second part after the semicolon is creating another template instance variable `i`, which holds the current index value. We will talk about the other properties in just a bit.

La primera parte de la microsyntax es básicamente nuestro bucle `for`. Creamos una variable de instancia de plantilla llamada `stock`, que estará disponible dentro del alcance del elemento creado. Esto se puede equiparar a un bucle `for-each` estándar, con la variable `stock` haciendo referencia a cada elemento individual de la matriz. La segunda parte después del punto y coma crea otra variable de instancia de plantilla `i`, que contiene el valor del índice actual. Hablaremos de las otras propiedades en un momento.

With this statement, Angular will repeat the `stock-container` `div` element once for each item in the `stocks` array, thus creating three elements in our final rendered HTML. You should see something like [Figure 3-1](#) on your screen.

Con esta declaración, Angular repetirá el elemento `div stock-container` una vez para cada elemento de la matriz `stocks`, creando así tres elementos en nuestro HTML renderizado final. Debería ver algo como la Figura 3-1 en su pantalla.

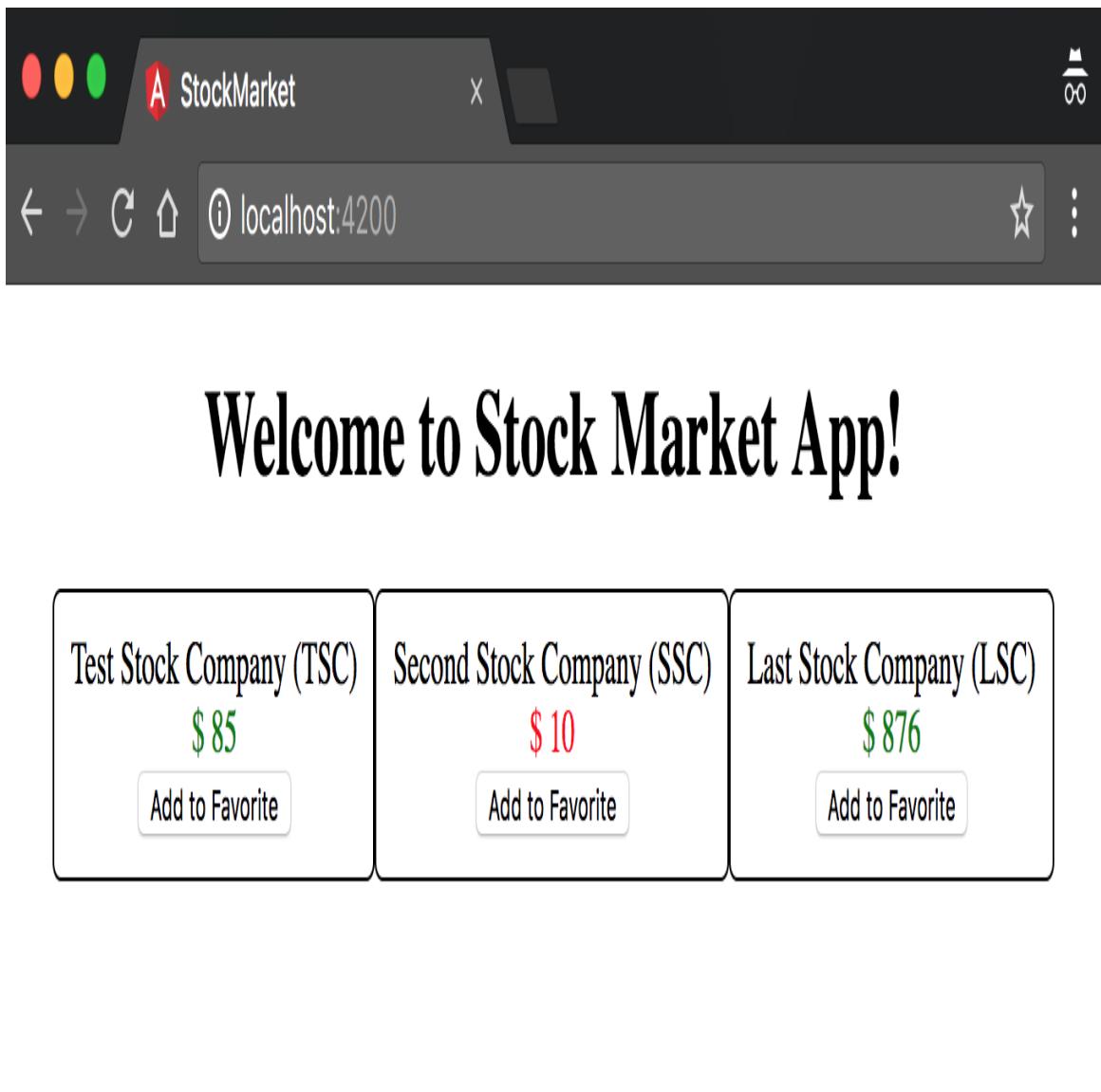


Figure 3-1. Angular app using *ngFor

Figura 3-1. Aplicación angular usando *ngFor

Angular encapsulates the `stock` and `i` variables such that each instance of the template has its own copy of them. Thus, when we refer to `stock.name` inside the element, Angular correctly picks up the relevant stock and displays its name. When we click the Add to Favorite button, the correct index is passed and only that particular button is disabled.

Angular encapsula las variables `stock` y `i` de modo que cada instancia de la plantilla tenga su propia copia de ellas. Por lo tanto,

cuando nos referimos a `stock.name` dentro del elemento, Angular selecciona correctamente la acción relevante y muestra su nombre. Cuando hacemos clic en el botón Agregar a favoritos, se pasa el índice correcto y solo se desactiva ese botón en particular.

Similar to `index`, there are other local values that are available within the context of an `NgFor` directive that you can assign to a local variable name (like we did with `i`) and bind them to different values:

De manera similar al índice, hay otros valores locales que están disponibles dentro del contexto de una directiva `NgFor` que puede asignar a un nombre de variable local (como hicimos con `i`) y vincularlos a diferentes valores:

`index`

This refers to the current element index

Esto se refiere al índice del elemento actual.

`even`

This is true when the item has an even index

Esto es true cuando el elemento tiene un índice par

`odd`

This is true when the item has an odd index

Este es true cuando el elemento tiene un índice impar

`first`

This is true when the item is the first item in the array

Esto es true cuando el elemento es el primer elemento de la matriz

`last`

This is true when the item is the last item in the array

Esto es true cuando el elemento es el último elemento de la matriz

You can then use these variables to bind to CSS classes, or display them in the UI, or run any other calculations you might want to. For instance, you might want to add the CSS class even-item to each even index item, which you can accomplish with a binding like:

Luego puede usar estas variables para vincularse a clases CSS, mostrarlas en la interfaz de usuario o ejecutar cualquier otro cálculo que desee. Por ejemplo, es posible que deseas agregar la clase CSS even-item a cada elemento de índice par, lo que puedes lograr con un enlace como:

```
[css.even-item]="isEven"
```

where even is assigned to the isEven variable in the NgFor microsyntax like:

donde even se asigna a la variable isEven en la microsintaxis NgFor como:

```
*ngFor="let stock of stocks; even as isEven; index as i"
```

The last major capability that the NgFor directive provides is a hook into how Angular recognizes elements and avoids re-creating element instances unnecessarily. By default, Angular uses object identity to track additions, removals, and modifications to an array. That is, as long as the object reference does not change, Angular will not create new elements for it, and reuse the old element reference. This is mostly for performance reasons, as element creation/removal are two of the more costly operations in the browser.

La última capacidad importante que proporciona la directiva NgFor es un enlace sobre cómo Angular reconoce elementos y evita recrear instancias de elementos innecesariamente. De forma

predeterminada, Angular usa la identidad del objeto para rastrear adiciones, eliminaciones y modificaciones a una matriz. Es decir, mientras la referencia del objeto no cambie, Angular no creará nuevos elementos para él y reutilizará la referencia del elemento anterior. Esto se debe principalmente a razones de rendimiento, ya que la creación/eliminación de elementos son dos de las operaciones más costosas en el navegador.

There are cases when the element reference might change, but you still want to continue using the same element. For example, when you fetch new data from the server, you don't want to blow away your list and re-create it unless the data has fundamentally changed. This is where the `trackBy` capability of the `NgFor` directive comes into play.

Hay casos en los que la referencia del elemento puede cambiar, pero aún deseas seguir usando el mismo elemento. Por ejemplo, cuando recupera datos nuevos del servidor, no desea borrar su lista y volver a crearla a menos que los datos hayan cambiado fundamentalmente. Aquí es donde entra en juego la capacidad `trackBy` de la directiva `NgFor`.

The `trackBy` option takes a function that has two arguments: an index and the item. If the `trackBy` function is provided to the `NgFor` directive, then it will use the return value of the function to decide how to identify individual elements. For example, in our particular case, we might want to use the stock's code as an identifier, instead of the object reference.

La opción `trackBy` toma una función que tiene dos argumentos: un índice y el elemento. Si la función `trackBy` se proporciona a la directiva `NgFor`, utilizará el valor de retorno de la función para decidir cómo identificar elementos individuales. Por ejemplo, en nuestro caso particular, es posible que queramos utilizar el código de la acción como identificador, en lugar de la referencia del objeto.

First, we will add the function used to track individual items in *src/app/stock/stock-item.component.ts*:

Primero, agregaremos la función utilizada para rastrear artículos individuales en *src/app/stock/stock-item.component.ts*:

```
import { Component, OnInit } from '@angular/core';

import { Stock } from '../../model/stock';

@Component({
  selector: 'app-stock-item',
  templateUrl: './stock-item.component.html',
  styleUrls: ['./stock-item.component.css']
})
export class StockItemComponent implements OnInit {

  public stocks: Array<Stock>;

  constructor() { }

  ngOnInit() {
    this.stocks = [
      new Stock('Test Stock Company', 'TSC', 85, 80),
      new Stock('Second Stock Company', 'SSC', 10, 20),
      new Stock('Last Stock Company', 'LSC', 876, 765)
    ];
  }

  toggleFavorite(event, index) {
    console.log('We are toggling the favorite state for stock', index, event);
    this.stocks[index].favorite = !this.stocks[index].favorite;
  }

  trackStockByCode(index, stock) {
    return stock.code;
  }
}
```

The last function (`trackStockByCode`) is the one we just added. We are taking both the index and the stock, and returning the `stock.code`. Angular will use this value to identify each element.

La última función (`trackStockByCode`) es la que acabamos de agregar. Tomamos tanto el índice como la acción y devolvemos el `stock.code`. Angular utilizará este valor para identificar cada elemento.

Next, we can change the HTML to pass this function to the `NgFor` directive by changing the `src/app/stock/stock-item/stock-item.component.html` file as follows:

A continuación, podemos cambiar el HTML para pasar esta función a la directiva `NgFor` cambiando el archivo `src/app/stock/stock-item/stock-item.component.html` de la siguiente manera:

```
<div class="stock-container"
  *ngFor="let stock of stocks; index as i; trackBy: trackStockByCode">
  <div class="name">{{stock.name + ' (' + stock.code + ')'}}</div>
  <div class="price">
    [class]="stock.isPositiveChange() ? 'positive' : 'negative'">
      $ {{stock.price}}
    </div>
    <button (click)="toggleFavorite($event, i)"
      [disabled]="stock.favorite">Add to Favorite</button>
  </div>
```

We just modified the `*ngFor` to pass in an additional attribute in the microsyntax, which is `trackBy: trackStockByCode`. This will ensure that Angular calls this function to figure out how to identify individual items, instead of using the object reference.

Acabamos de modificar `*ngFor` para pasar un atributo adicional en la microsintaxis, que es `trackBy: trackStockByCode`. Esto asegurará que Angular llame a esta función para descubrir cómo identificar elementos individuales, en lugar de usar la referencia del objeto.

This ensures that even if we reload all the stocks from the server (thus changing all the object references), Angular will still look at the stock code to decide whether or not to reuse the elements present in the DOM.

Esto garantiza que incluso si recargamos todos los stocks del servidor (cambiando así todas las referencias de objetos), Angular seguirá mirando el código stock para decidir si reutilizar o no los elementos presentes en el DOM.

TIP CONSEJO

Use the `trackBy` option in the `NgFor` directive whenever you know that you are going to reload the list (for example, loading it from an observable that makes a server call). It will definitely help in performance.

Utilice la opción `trackBy` en la directiva `NgFor` siempre que sepa que va a recargar la lista (por ejemplo, cargándola desde un observable que realiza una llamada al servidor). Definitivamente ayudará en el rendimiento.

NgSwitch

NgSwitch

The last built-in directive is actually a set of multiple directives. `NgSwitch` by itself is not a structural directive, but rather an attribute directive. You would use it with normal data-binding syntax with the square-bracket notation. It is the `NgSwitchCase` and the `NgSwitchDefault` directives that are, in fact, structural directives, as they would add or remove elements depending on the case.

La última directiva incorporada es en realidad un conjunto de directivas múltiples. `NgSwitch` por sí solo no es una directiva estructural, sino más bien una directiva de atributos. Lo usaría con la sintaxis normal de enlace de datos con la notación de corchetes. Son las directivas `NgSwitchCase` y `NgSwitchDefault` las que son, de hecho, directivas estructurales, ya que agregarían o eliminarían elementos según el caso.

Let's take a hypothetical example to see how this might look in action. Let's assume that instead of just stocks, we were dealing with all kinds of securities, like stocks, options, derivatives, and mutual funds. And we had built components to render each one of these. Now, given a security we want to render the right kind of component based on its type. This would be a great example of using an NgSwitch, which might look something like follows:

Tomemos un ejemplo hipotético para ver cómo se vería esto en acción. Supongamos que en lugar de sólo acciones, estuviéramos tratando con todo tipo de valores, como acciones, opciones, derivados y fondos mutuos. Y habíamos construido componentes para renderizar cada uno de estos. Ahora, dada una seguridad, queremos representar el tipo correcto de componente según su tipo. Este sería un gran ejemplo del uso de un NgSwitch, que podría verse así:

```
<div [ngSwitch]="security.type">
  <stock-item *ngSwitchCase="'stock'" [item]="security">
  </stock-item>
  <option-item *ngSwitchCase="'option'" [item]="security">
  </option-item>
  <derivative-item *ngSwitchCase="'derivative'" [item]="security">
  </derivative-item>
  <mutual-fund-item *ngSwitchCase="'mutual-fund'" [item]="security">
  </mutual-fund-item>
  <unknown-item *ngSwitchDefault [item]="security">
  </unknown-item>
</div>
```

A few interesting things of note in the example:

Algunas cosas interesantes a tener en cuenta en el ejemplo:

- We use normal data binding using square brackets with the ngSwitch directive, as we want it to understand and interpret the value underneath the expression. It is not a structural directive, as we mentioned previously.

Usamos enlace de datos normal usando corchetes con la directiva `ngSwitch`, ya que queremos que comprenda e interprete el valor debajo de la expresión. No es una directiva estructural, como hemos mencionado anteriormente.

- Each `*ngSwitchCase` is again taking an expression, and in this case, we are passing string constants like '`stock`', '`option`', and so on. If the value of `security.type` matches any one of these string constants, then that particular element will get rendered, and all other sibling elements will be removed.

Cada `*ngSwitchCase` vuelve a tomar una expresión y, en este caso, pasamos constantes de cadena como '`stock`', '`option`', etc. Si el valor de `security.type` coincide con cualquiera de estas constantes de cadena, entonces ese elemento en particular se representará y todos los demás elementos hermanos se eliminarán.

- If none of the sibling `*ngSwitchCase` statements matches, then the `*ngSwitchDefault` will get triggered and in the example, the `unknown-item` component will get rendered.

Si ninguna de las declaraciones del hermano `*ngSwitchCase` coincide, entonces se activará el `*ngSwitchDefault` y, en el ejemplo, se representará el componente `unknown-item`.

The NgSwitch family of directives is great when you have multiple elements/templates, of which one has to be rendered based on conditions.

La familia de directivas NgSwitch es excelente cuando tienes múltiples elementos/plantillas, de los cuales uno debe representarse según las condiciones.

Multiple Sibling Structural Directives

Directivas estructurales para múltiples hermanos

You might run into a case at some point where you want to run an *ngFor on a template, but only if some condition is met. Your instinctive reaction in that case might be to add both *ngFor and *ngIf on the same element. Angular will not let you.

Es posible que en algún momento te encuentres con un caso en el que quieras ejecutar un *ngFor en una plantilla, pero solo si se cumple alguna condición. Su reacción instintiva en ese caso podría ser agregar *ngFor y *ngIf en el mismo elemento. Angular no te dejará.

For example, consider the following code:

Por ejemplo, considere el siguiente código:

```
<div *ngFor="let stock of stocks" *ngIf="stock.active">
  <!-- Show stock details here if stock is active -->
</div>
```

Here, we want to run the *ngFor, and then decide if the stock is active before we display it. Now consider the following code:

Aquí queremos ejecutar *ngFor y luego decidir si la acción está activa antes de mostrarla. Ahora considere el siguiente código:

```
<div *ngFor="let stock of stocks" *ngIf="stocks.length > 2">
  <!-- Show stock details here if more than 2 stocks present -->
</div>
```

Both of these cases look very similar, but the intent and expectation is very different. In the first case, we expect the *ngFor to execute first followed by the *ngIf, and vice versa in the second case.

Ambos casos parecen muy similares, pero la intención y la expectativa son muy diferentes. En el primer caso, esperamos que

`*ngFor` se ejecute primero seguido de `*ngIf`, y viceversa en el segundo caso.

It is not immediately obvious which one between the two should run first. Rather than creating some innate order that one directive runs before the other, Angular simplifies it by not allowing it on the same element.

No es inmediatamente obvio cuál de los dos debería ejecutarse primero. En lugar de crear un orden innato en el que una directiva se ejecuta antes que la otra, Angular lo simplifica al no permitirla en el mismo elemento.

When you run into such a case, it is recommended to just use wrapping elements so that you can be explicit about the order in which these structural directives are executed.

Cuando se encuentre con un caso así, se recomienda utilizar simplemente elementos envolventes para poder ser explícito sobre el orden en el que se ejecutan estas directivas estructurales.

Conclusion

Conclusión

In this chapter, we covered what the built-in directives are, and then went over some common directives that you will need and use for your Angular application, such as `NgFor`, `NgIf`, and more. We looked at examples of how we could use them for various functionality and saw a little bit of some more complex use cases for `NgSwitch` as well as advanced usage of `NgFor`. Of course, Angular does allow you to extend and create your own directives, and you can read up on it [in the documentation](#).

En este capítulo, cubrimos cuáles son las directivas integradas y luego repasamos algunas directivas comunes que necesitará y

utilizará para su aplicación Angular, como NgFor, NgIf y más. Analizamos ejemplos de cómo podríamos usarlos para diversas funciones y vimos algunos casos de uso más complejos para NgSwitch, así como el uso avanzado de NgFor. Por supuesto, Angular le permite ampliar y crear sus propias directivas, y puede leerlo en la documentación.

In the next chapter, we will dive deep into Angular components and understand the various options we have for configuring them as well as the lifecycle of an Angular component.

En el próximo capítulo, profundizaremos en los componentes de Angular y comprenderemos las diversas opciones que tenemos para configurarlos, así como el ciclo de vida de un componente de Angular.

Exercise

Ejercicio

For our second exercise, try to change the exercise code from the previous chapter to perform the following:

Para nuestro segundo ejercicio, intente cambiar el código del ejercicio del capítulo anterior para realizar lo siguiente:

1. Move from using simple class binding to using either ngClass or the specific class binding from this chapter to highlight on-sale items. Have a combination of some on sale and some not on sale.

Pase del uso del enlace de clase simple al uso de ngClass o el enlace de clase específico de este capítulo para resaltar los artículos en oferta. Tenga una combinación de algunos en oferta y otros no en oferta.

- Instead of disabling the decrease quantity button when the quantity is zero, use `*ngIf` to show the button only if it can be clicked.

En lugar de deshabilitar el botón para reducir la cantidad cuando la cantidad es cero, use `*ngIf` para mostrar el botón solo si se puede hacer clic en él.

- Add a drop-down with quantity selection from 1 to 20 (generated via `*ngFor`). Don't worry about the action/update of data on selection of a quantity; we will get to that in one of the following chapters.

Agregue un menú desplegable con selección de cantidad del 1 al 20 (generado a través de `*ngFor`). No se preocupe por la acción/actualización de datos al seleccionar una cantidad; Llegaremos a eso en uno de los siguientes capítulos.

All of this can be accomplished using concepts covered in this chapter. You can check out the finished solution in [`chapter3/exercise/ecommerce`](#).

Todo esto se puede lograr utilizando los conceptos tratados en este capítulo. Puede consultar la solución terminada en el capítulo 3/ejercicio/ecommerce.

Chapter 4. Understanding and Using Angular Components

Capítulo 4. Comprensión y uso de componentes angulares

In the previous chapter, we did a deep dive into the built-in directives that Angular offers that allow us to perform common functionality like hiding and showing elements, repeating templates, and so on. We worked with directives like `ngIf` and `ngForOf` and got a feel for how and when to use them.

En el capítulo anterior, profundizamos en las directivas integradas que ofrece Angular y que nos permiten realizar funciones comunes como ocultar y mostrar elementos, repetir plantillas, etc. Trabajamos con directivas como `ngIf` y `ngForOf` y tuvimos una idea de cómo y cuándo usarlas.

In this chapter, we will go a bit deeper into components, those elements we have been creating to render the UI and let users interact with the applications we build. We will cover some of the more useful attributes you can specify when creating components, how to think about the lifecycle of the component and the various hooks that Angular gives you, and finally, cover how to pass data into and out of your custom components. By the end of the chapter, you should be able to perform most common tasks related to components while understanding what you are doing and why.

En este capítulo, profundizaremos un poco más en los componentes, aquellos elementos que hemos estado creando para representar la interfaz de usuario y permitir que los usuarios interactúen con las aplicaciones que creamos. Cubriremos algunos de los atributos más útiles que puede especificar al crear componentes, cómo pensar en el ciclo de vida del componente y los diversos enlaces que le brinda Angular y, finalmente, cubriremos cómo pasar datos dentro y fuera de sus componentes personalizados. Al final del capítulo, debería poder realizar las tareas más comunes relacionadas con los componentes y al mismo tiempo comprender lo que está haciendo y por qué.

Components—A Recap

Componentes: un resumen

In the previous chapter, we saw that Angular only has directives, and that directives are reused for multiple purposes. We dealt with attribute and structural directives, which allow us to change the behavior of an existing element or to change the structure of the template being rendered.

En el capítulo anterior, vimos que Angular solo tiene directivas y que las directivas se reutilizan para múltiples propósitos. Nos ocupamos de directivas estructurales y de atributos, que nos permiten cambiar el comportamiento de un elemento existente o cambiar la estructura de la plantilla que se está representando.

The third kind of directives are components, which we have been using pretty much from the first chapter. To some extent, you can consider an Angular application to be nothing but a tree of components. Each component in turn has some behavior and a template that gets rendered. This template can then continue to use

other components, thus forming a tree of components, which is the Angular application that gets rendered in the browser.

El tercer tipo de directivas son los componentes, que hemos estado usando prácticamente desde el primer capítulo. Hasta cierto punto, puedes considerar que una aplicación Angular no es más que un árbol de componentes. Cada componente, a su vez, tiene algún comportamiento y una plantilla que se representa. Luego, esta plantilla puede continuar usando otros componentes, formando así un árbol de componentes, que es la aplicación Angular que se representa en el navegador.

At its very simplest, a component is nothing but a class that encapsulates behavior (what data to load, what data to render, and how to respond to user interactions) and a template (how the data is rendered). But there are multiple ways to define that as well, along with other options, which we will cover in the following sections.

En su forma más simple, un componente no es más que una clase que encapsula el comportamiento (qué datos cargar, qué datos representar y cómo responder a las interacciones del usuario) y una plantilla (cómo se representan los datos). Pero también hay varias formas de definir eso, junto con otras opciones, que cubriremos en las siguientes secciones.

Defining a Component

Definición de un componente

We define a component using the TypeScript decorator `Component`. This allows us to annotate any class with some metadata that teaches Angular how the component works, what to render, and so on. Let's take a look again at the `stock-item` component we created

to see what a simple component would look like, and we will build up from there:

Definimos un componente usando el decorador TypeScript Component. Esto nos permite anotar cualquier clase con algunos metadatos que le enseñan a Angular cómo funciona el componente, qué renderizar, etc. Echemos un vistazo nuevamente al componente `stock-item` que creamos para ver cómo se vería un componente simple y lo desarrollaremos a partir de ahí:

```
@Component({
  selector: 'app-stock-item',
  templateUrl: './stock-item.component.html',
  styleUrls: ['./stock-item.component.css']
})
export class StockItemComponent implements OnInit {
  // Code omitted here for clarity
}
```

The very basic component only needs a selector (to tell Angular how to find instances of the component being used) and a template (that Angular has to render when it finds the element). All other attributes in the Component decorator are optional. In the preceding example, we have defined that the `StockItemComponent` is to be rendered whenever Angular encounters the `app-stock-item` selector, and to render the `stock-item.component.html` file when it encounters the element. Let's talk about the attributes of the decorator in a bit more detail.

El componente muy básico solo necesita un selector (para indicarle a Angular cómo encontrar instancias del componente que se está utilizando) y una plantilla (que Angular debe representar cuando encuentra el elemento). Todos los demás atributos del decorador Component son opcionales. En el ejemplo anterior, hemos definido que `StockItemComponent` se representará siempre que Angular encuentre el selector `app-stock-item` y que se represente el archivo `stock-item.component.html` cuando encuentre el elemento.

Hablemos de los atributos del decorador con un poco más de detalle.

Selector

Selector

The selector attribute, as we touched upon briefly in [Chapter 2](#), allows us to define how Angular identifies when the component is used in HTML. The selector takes a string value, which is the CSS selector Angular will use to identify the element. The recommended practice when we create new components is to use element selectors (like we did with `app-stock-item`), but technically you could use any other selector as well. For example, here are a few ways you could specify the selector attribute and how you would use it in the HTML:

El atributo selector, como mencionamos brevemente en el Capítulo 2, nos permite definir cómo Angular identifica cuando el componente se usa en HTML. El selector toma un valor de cadena, que es el selector CSS que Angular utilizará para identificar el elemento. La práctica recomendada cuando creamos nuevos componentes es usar selectores de elementos (como hicimos con `app-stock-item`), pero técnicamente también puedes usar cualquier otro selector. Por ejemplo, aquí hay algunas formas en las que puede especificar el atributo del selector y cómo lo usaría en HTML:

- selector: '`app-stock-item`' would result in the component being used as `<app-stock-item></app-stock-item>` in the HTML.

`selector: 'app-stock-item'` daría como resultado que el componente se usara como `<app-stock-item></app-stock-item>` en el HTML.

- selector: '.app-stock-item' would result in the component being used as a CSS class like `<div class="app-stock-item"></div>` in the HTML.
selector: '.app-stock-item' daría como resultado que el componente se use como una clase CSS como `<div class="app-stock-item"></div>` en HTML.
- selector: '[app-stock-item]' would result in the component being used as an attribute on an existing element like `<div app-stock-item></div>` in the HTML.
selector: '[app-stock-item]' daría como resultado que el componente se use como un atributo en un elemento existente como `<div app-stock-item></div>` en el HTML.

You can make the selector as simple or complex as you want, but as a rule of thumb, try to stick to simple element selectors unless you have a very strong reason not to.

Puede hacer que el selector sea tan simple o complejo como desee, pero como regla general, intente ceñirse a selectores de elementos simples a menos que tenga una razón muy importante para no hacerlo.

Template

Plantilla

We have been using `templateUrl` so far to define what the template to be used along with a component is. The path you pass to the `templateUrl` attribute is relative to the path of the component. In the previous case, we can either specify the `templateUrl` as:

Hemos estado usando `templateUrl` hasta ahora para definir cuál es la plantilla que se usará junto con un componente. La ruta que pasa

al atributo `templateUrl` es relativa a la ruta del componente. En el caso anterior, podemos especificar `templateUrl` como:

```
templateUrl: './stock.item.component.html'
```

or:

o:

```
templateUrl: 'stock.item.component.html'
```

and it would work. But if you try to specify an absolute URL or anything else, your compilation would break. One interesting thing to note is that unlike AngularJS (1.x), the application Angular builds does not load the template by URL at runtime. Instead, Angular precompiles a build and ensures that the template is inlined as part of the build process.

y funcionaría. Pero si intentas especificar una URL absoluta o cualquier otra cosa, tu compilación se interrumpirá. Una cosa interesante a tener en cuenta es que, a diferencia de AngularJS (1.x), la aplicación Angular builds no carga la plantilla por URL en tiempo de ejecución. En cambio, Angular precompila una compilación y garantiza que la plantilla esté integrada como parte del proceso de compilación.

Instead of `templateUrl`, we could also specify the template inline in the component, using the `template` option. This allows us to have the component contain all the information instead of splitting it across HTML and TypeScript code.

En lugar de `templateUrl`, también podríamos especificar la plantilla en línea en el componente, usando la opción `template`. Esto nos permite hacer que el componente contenga toda la información en lugar de dividirla en código HTML y TypeScript.

TIP CONSEJO

Only one of `template` and `templateUrl` can be specified in a component. You cannot use both, but at least one is essential.

Solo se puede especificar uno de `template` y `templateUrl` en un componente. No puedes usar ambos, pero al menos uno es esencial.

There is no impact on the final generated application as Angular compiles the code into a single bundle. The only reason you might want to split your template code into a separate file is to get nicer IDE features such as syntax completion and the like, which are specific to file extensions. Generally, you might want to keep your templates separate if they are over three or four lines or have any complexity.

No hay ningún impacto en la aplicación final generada ya que Angular compila el código en un solo paquete. La única razón por la que quizás quieras dividir el código de tu plantilla en un archivo separado es para obtener características IDE más agradables, como la finalización de sintaxis y similares, que son específicas de las extensiones de archivo. Generalmente, es posible que desees mantener tus plantillas separadas si tienen más de tres o cuatro líneas o si tienen alguna complejidad.

Let's see how our `stock-item` component might look with an inline template:

Veamos cómo se vería nuestro componente `stock-item` con una plantilla en línea:

```
import { Component, OnInit } from '@angular/core';

import { Stock } from '../model/stock';

@Component({
  selector: 'app-stock-item',
```

```

template: `
  <div class="stock-container">
    <div class="name">{{stock.name + ' (' + stock.code + ')'}}</div>
    <div class="price"
      [class]="stock.isPositiveChange() ? 'positive' : 'negative'"
      $ {{stock.price}}>
  </div>
  <button (click)="toggleFavorite($event)"
    *ngIf="!stock.favorite">Add to Favorite</button>
</div>
`,
styleUrls: ['./stock-item.component.css']
})
export class StockItemComponent implements OnInit {
  // Code omitted here for clarity
}

```

TIP CONSEJO

ECMAScript 2015 (and TypeScript) allows us to define multiline templates using the ` (backtick) symbol, instead of doing string concatenation across multiple lines using the + (plus) operator. We leverage this usually when we define inline templates.

ECMAScript 2015 (y TypeScript) nos permite definir plantillas multilínea usando el símbolo ` (comilla invertida), en lugar de realizar una concatenación de cadenas en varias líneas usando el operador + (más). Normalmente aprovechamos esto cuando definimos plantillas en línea.

You can find the completed code in the *chapter4/component-template* folder in the GitHub repository.

Puede encontrar el código completo en la carpeta *capítulo4/component-template* en el repositorio de GitHub.

All we have done is taken the template and moved it into the `template` attribute of the `Component` decorator. In this particular case though, because there are more than a few lines with some amount of work being done, I would recommend not moving it inline. Note that as a result of moving it to `template`, we have removed the previous `templateUrl` attribute.

Todo lo que hemos hecho es tomar la plantilla y moverla al atributo `template` del decorador `Component`. Sin embargo, en este caso particular, debido a que hay más de unas pocas líneas con cierta cantidad de trabajo por hacer, recomendaría no moverlo en línea. Tenga en cuenta que como resultado de moverlo a `template`, hemos eliminado el atributo `templateUrl` anterior.

Styles

Estilos

A given component can have multiple styles attached to it. This allows you to pull in component-specific CSS, as well as potentially any other common CSS that needs to be applied to it. Similar to templates, you can either inline your CSS using the `styles` attribute, or if there is a significant amount of CSS or you want to leverage your IDE, you can pull it out into a separate file and pull it into your component using the `styleUrls` attribute. Both of these take an array as an input.

Un componente determinado puede tener varios estilos asociados. Esto le permite incorporar CSS específico del componente, así como potencialmente cualquier otro CSS común que deba aplicarse. De manera similar a las plantillas, puede alinear su CSS usando el atributo `styles`, o si hay una cantidad significativa de CSS o desea aprovechar su IDE, puede extraerlo en un archivo separado y colocarlo en su componente utilizando el atributo `styleUrls`. Ambos toman una matriz como entrada.

One thing that Angular promotes out of the box is complete encapsulation and isolation of styles. That means by default, the styles you define and use in one component will not affect/impact any other parent or child component. This ensures that you can be confident that the CSS classes you define in any component will not

unknowingly affect anything else, unless you explicitly pull in the necessary styles.

Una cosa que Angular promueve de forma inmediata es la encapsulación completa y el aislamiento de estilos. Eso significa que, de forma predeterminada, los estilos que defina y utilice en un componente no afectarán ni impactarán a ningún otro componente principal o secundario. Esto garantiza que pueda estar seguro de que las clases CSS que defina en cualquier componente no afectarán a nada más sin saberlo, a menos que introduzca explícitamente los estilos necesarios.

Again, just like templates, Angular will not pull in these styles at runtime, but rather precompile and create a bundle with the necessary styles. Thus, the choice of using `styles` or `styleUrls` is a personal one, without any major impact at runtime.

Nuevamente, al igual que las plantillas, Angular no incorporará estos estilos en tiempo de ejecución, sino que precompilará y creará un paquete con los estilos necesarios. Por lo tanto, la elección de utilizar `styles` o `styleUrls` es personal, sin ningún impacto importante en tiempo de ejecución.

WARNING ADVERTENCIA

Do not use both `styles` and `styleUrls` together. Angular will end up picking one or the other and will lead to unexpected behavior.

No utilice `styles` y `styleUrls` juntos. Angular terminará eligiendo uno u otro y dará lugar a un comportamiento inesperado.

Let's quickly see how the component might look if we inlined the styles:

Veamos rápidamente cómo se vería el componente si alineáramos los estilos:

```
import { Component, OnInit } from '@angular/core';

import { Stock } from '../model/stock';

@Component({
  selector: 'app-stock-item',
  templateUrl: 'stock-item.component.html',
  styles: [
    '.stock-container {
      border: 1px solid black;
      border-radius: 5px;
      display: inline-block;
      padding: 10px;
    }

    .positive {
      color: green;
    }

    .negative {
      color: red;
    }
  ]
})
export class StockItemComponent implements OnInit {
  // Code omitted here for clarity
}
```

You can find the completed code in the *chapter4/component-style* folder in the GitHub repository.

Puede encontrar el código completo en la carpeta Chapter4/component-style en el repositorio de GitHub.

You can of course choose to pass in multiple style strings to the attribute. The decision between using `styles` and `styleUrls` is one of personal preference and has no impact on the final performance of the application.

Por supuesto, puede optar por pasar varias cadenas de estilo al atributo. La decisión entre usar `styles` y `styleUrls` es una preferencia personal y no tiene ningún impacto en el rendimiento final de la aplicación.

Style Encapsulation

Encapsulación de estilo

In the preceding section, we talked about how Angular encapsulates the styles to ensure that it doesn't contaminate any of your other components. In fact, you can actually tell Angular whether it needs to do this or not, or if the styles can be accessible globally. You can set this by using the `encapsulation` attribute on the `Component` decorator. The `encapsulation` attribute takes one of three values:

En la sección anterior, hablamos sobre cómo Angular encapsula los estilos para garantizar que no contamine ninguno de sus otros componentes. De hecho, puedes decirle a Angular si necesita hacer esto o no, o si los estilos pueden ser accesibles globalmente. Puede configurar esto utilizando el atributo `encapsulation` en el decorador `Component`. El atributo `encapsulation` toma uno de tres valores:

`ViewEncapsulation.Emulated`

This is the default, where Angular creates shimmed CSS to emulate the behavior that shadow DOMs and shadow roots provide.

Este es el valor predeterminado, donde Angular crea CSS modificado para emular el comportamiento que proporcionan los DOM de sombra y las raíces de sombra.

`ViewEncapsulation.Native`

This is the ideal, where Angular will use shadow roots. This will only work on browsers and platforms that natively support it.

Este es el ideal, donde Angular usará raíces de sombra. Esto sólo funcionará en navegadores y plataformas que lo admitan de forma nativa.

`ViewEncapsulation.None`

Uses global CSS, without any encapsulation.

Utiliza CSS global, sin ningún tipo de encapsulación.

WHAT IS THE SHADOW DOM? ¿QUÉ ES EL DOM EN LA SOMBRA?

HTML, CSS, and JavaScript have a default tendency to be global in the context of the current page. What this means is that an ID given to an element can easily clash with another element somewhere else on the page. Similarly, a CSS rule given to a button in one corner of the page might end up impacting another totally unrelated button.

HTML, CSS y JavaScript tienen una tendencia predeterminada a ser globales en el contexto de la página actual. Lo que esto significa es que una identificación proporcionada a un elemento puede chocar fácilmente con otro elemento en otro lugar de la página. De manera similar, una regla CSS aplicada a un botón en una esquina de la página podría terminar afectando a otro botón totalmente ajeno.

We end up having to come up with specific naming conventions, use CSS hacks like `!important`, and use many more techniques to work around this generally in our day-to-day development.

Terminamos teniendo que idear convenciones de nomenclatura específicas, utilizar trucos de CSS como `!important` y utilizar muchas más técnicas para solucionar este problema en general en nuestro desarrollo diario.

Shadow DOM fixes this by scoping HTML DOM and CSS. It provides the ability to have scoped styling to a component (thus preventing the styles from leaking out and affecting the rest of the application) and also the ability to isolate and make the DOM self-contained.

Shadow DOM soluciona este problema al determinar el alcance de HTML DOM y CSS. Proporciona la capacidad de aplicar un estilo a un componente (evitando así que los estilos se filtrean y afecten al resto de la aplicación) y también la capacidad de aislar y hacer que el DOM sea autónomo.

You can read up on it more in the documentation for [self-contained web components](#).

Puede leer más sobre esto en la documentación de componentes web autónomos.

The best way to see how this impacts our application is to make a slight change and see how our application behaves under different circumstances.

La mejor manera de ver cómo afecta esto a nuestra aplicación es hacer un ligero cambio y ver cómo se comporta nuestra aplicación en diferentes circunstancias.

First, let's add the following snippet of code to the *app.component.css* file. We are using the same base as the previous chapter, and the completed code is available in the *chapter4/component-style-encapsulation* folder:

Primero, agreguemos el siguiente fragmento de código al archivo *app.component.css*. Estamos utilizando la misma base que el capítulo anterior y el código completo está disponible en la carpeta *capítulo4/component-style-encapsulation*:

```
.name {  
  font-size: 50px;  
}
```

If we run the application right now, there is no impact on our application. Now, let's try changing the *encapsulation* property on the main *AppComponent*. We will change the component as follows:

Si ejecutamos la aplicación ahora mismo, no hay ningún impacto en nuestra aplicación. Ahora, intentemos cambiar la propiedad *encapsulation* en el *AppComponent* principal. Cambiaremos el componente de la siguiente manera:

```
import { Component, ViewEncapsulation } from '@angular/core';  
  
@Component({  
  selector: 'app-root',  
  templateUrl: './app.component.html',  
  styleUrls: ['./app.component.css'],  
  encapsulation: ViewEncapsulation.None  
})  
export class AppComponent {
```

```
    title = 'app works!';
}
```

We added the `encapsulation: ViewEncapsulation.None` line to our `Component` decorator (of course, after importing the `ViewEncapsulation` enum from Angular). Now if we refresh our application, you will see that the name of the stock has been blown up to 50px. This is because the styles applied on the `AppComponent` are not restricted to just the component but are now taking the global namespace. Thus, any element that adds the `name` class to itself will get this font-size applied to it.

Agregamos la línea `encapsulation: ViewEncapsulation.None` a nuestro decorador `Component` (por supuesto, después de importar la enumeración `ViewEncapsulation` desde Angular). Ahora, si actualizamos nuestra aplicación, verá que el nombre de la acción ha aumentado a 50 px. Esto se debe a que los estilos aplicados en `AppComponent` no se limitan solo al componente, sino que ahora ocupan el espacio de nombres global. Por lo tanto, a cualquier elemento que agregue la clase `name` a sí mismo se le aplicará este tamaño de fuente.

`ViewEncapsulation.None` is a good way of applying common styles to all child components, but definitely adds the risk of polluting the global CSS namespace and having unintentional effects.

`ViewEncapsulation.None` es una buena manera de aplicar estilos comunes a todos los componentes secundarios, pero definitivamente agrega el riesgo de contaminar el espacio de nombres CSS global y tener efectos no intencionales.

Others

Otros

There are a lot more attributes than what we covered on the Component decorator. We will briefly review a few of those here, and will reserve discussion of others for later chapters when they become more relevant. Here is a quick highlight of some of the other major attributes and their uses:

Hay muchos más atributos de los que cubrimos en el decorador Component. Revisaremos brevemente algunos de ellos aquí y reservaremos la discusión de otros para capítulos posteriores cuando se vuelvan más relevantes. Aquí hay un breve resumen de algunos de los otros atributos principales y sus usos:

Stripping white spaces

Eliminando espacios en blanco

Angular allows you to strip any unnecessary white spaces from your template (as defined by Angular, including more than one space, space between elements, etc.). This can help reduce the build size by compressing your HTML. You can set this feature (which is set to `false` by default) by using the `preserveWhitespaces` attribute on the component. You can read more about this feature in the [official documentation](#).

Angular le permite eliminar cualquier espacio en blanco innecesario de su plantilla (según lo define Angular, incluido más de un espacio, espacio entre elementos, etc.). Esto puede ayudar a reducir el tamaño de compilación comprimiendo su HTML. Puede configurar esta característica (que está configurada en `false` de forma predeterminada) utilizando el atributo `preserveWhitespaces` en el componente. Puede leer más sobre esta característica en la documentación oficial.

Animations

animaciones

Angular gives you multiple triggers to control and animate each part of the component and its lifecycle. To accomplish this, it provides its own DSL, which allows Angular to animate on state changes within the element.

Angular le brinda múltiples activadores para controlar y animar cada parte del componente y su ciclo de vida. Para lograr esto, proporciona su propio DSL, que permite a Angular animar los cambios de estado dentro del elemento.

Interpolation

Interpolación

There are times when the default Angular interpolation markers (the double-curly braces {{ and }}) interfere with integrating with other frameworks or technologies. For those scenarios, Angular allows you to override the interpolation identifiers at a component level by specifying the start and end delimiters. You can do so by using the `interpolation` attribute, which takes an array of two strings, the opening and closing markers for the interpolation. By default, they are ['{{', '}}'], but you override it by, say, providing `interpolation: ['<<', '>>']` to replace the interpolation symbols for just that component to << and >>.

Hay ocasiones en las que los marcadores de interpolación angular predeterminados (los dobles llaves {{ y }}) interfieren con la integración con otros marcos o tecnologías. Para esos escenarios, Angular le permite anular los identificadores de interpolación a nivel de componente especificando los delimitadores de inicio y fin. Puede hacerlo utilizando el atributo `interpolation`, que toma una matriz de dos cadenas, los marcadores de apertura y cierre para la interpolación. De forma predeterminada, son ['{{', '}}'], pero puede anularlos, por ejemplo, proporcionando `interpolation: ['<<', '>>']` para

reemplazar los símbolos de interpolación solo para ese componente por << y >>.

View providers

Ver proveedores

View providers allow you to define providers that inject classes/services into a component or any of its children. Usually, you won't need it, but if there are certain components where you want to override, or restrict the availability of a class or a service, you can specify an array of providers to a component using the `viewProviders` attribute. We will cover this in more detail in [Chapter 8](#).

Ver proveedores le permite definir proveedores que inyectan clases/servicios en un componente o cualquiera de sus hijos. Por lo general, no lo necesitará, pero si hay ciertos componentes que desea anular o restringir la disponibilidad de una clase o servicio, puede especificar una serie de proveedores para un componente usando el atributo `viewProviders`. Cubriremos esto con más detalle en el Capítulo 8.

Exporting the component

Exportando el componente

We have been working so far by using the component class's functions within the context of the template. But there are use cases (especially when we start dealing with directives and more complex components) for which we might want to allow the user of the component to call functions on the component from outside. A use case might be that we provide a carousel component, but want to provide functionality to allow the user of the component to control the next/previous functionality. In these cases, we can use the `exportAs` attribute of the Component decorator.

Hemos estado trabajando hasta ahora utilizando las funciones de la clase de componente dentro del contexto de la plantilla. Pero hay casos de uso (especialmente cuando empezamos a trabajar con directivas y componentes más complejos) para los que podríamos querer permitir que el usuario del componente llame a funciones en el componente desde fuera. Un caso de uso podría ser que proporcionemos un componente de carrusel, pero queramos proporcionar funcionalidad que permita al usuario del componente controlar la funcionalidad siguiente/anterior. En estos casos, podemos usar el atributo `exportAs` del decorador `Component`.

`changeDetection`

By default, Angular checks every binding in the UI to see if it needs to update any UI element whenever any value changes in our component. This is acceptable for most applications, but as our applications get larger in size and complexity, we might want control over how and when Angular updates the UI. Instead of Angular deciding when it needs to update the UI, we might want to be explicit and tell Angular when it needs to update the UI manually. To do this, we use the `changeDetection` attribute, where we can override the default value of `ChangeDetectionStrategy.Default` to `ChangeDetectionStrategy.OnPush`. This means that after the initial render, it will be up to us to let Angular know when the value changes. Angular will not check the component's bindings automatically. We will cover this in more detail later in the chapter.

De forma predeterminada, Angular verifica cada enlace en la interfaz de usuario para ver si necesita actualizar algún elemento de la interfaz de usuario cada vez que cambia algún valor en nuestro componente. Esto es aceptable para la mayoría de las aplicaciones, pero a medida que nuestras aplicaciones aumentan

en tamaño y complejidad, es posible que queramos controlar cómo y cuándo Angular actualiza la interfaz de usuario. En lugar de que Angular decida cuándo necesita actualizar la interfaz de usuario, es posible que queramos ser explícitos y decirle a Angular cuándo necesita actualizar la interfaz de usuario manualmente. Para hacer esto, usamos el atributo `changeDetection`, donde podemos anular el valor predeterminado de `ChangeDetectionStrategy.Default` a `ChangeDetectionStrategy.OnPush`. Esto significa que después del renderizado inicial, dependerá de nosotros informarle a Angular cuando cambie el valor. Angular no comprobará los enlaces del componente automáticamente. Cubriremos esto con más detalle más adelante en el capítulo.

There are a lot more attributes and features with regards to components that we don't cover in this chapter. You should take a look at the [official documentation for components](#) to get familiar with what else is possible, or dive deeper into the details.

Hay muchos más atributos y características con respecto a los componentes que no cubrimos en este capítulo. Debería echar un vistazo a la documentación oficial de los componentes para familiarizarse con qué más es posible o profundizar en los detalles.

Components and Modules

Componentes y módulos

Before we go into the details of the lifecycle of a component, let's quickly sidetrack into how components are linked to modules and what their relation is. In [Chapter 2](#), we saw how any time we created a new component, we had to include it in a module. If you create a new component, and do not add it to a module, Angular will

complain that you have components that are not part of any modules.

Antes de entrar en los detalles del ciclo de vida de un componente, analicemos rápidamente cómo se vinculan los componentes a los módulos y cuál es su relación. En el Capítulo 2, vimos cómo cada vez que creábamos un nuevo componente, teníamos que incluirlo en un módulo. Si crea un nuevo componente y no lo agrega a un módulo, Angular se quejará de que tiene componentes que no forman parte de ningún módulo.

For any component to be used within the context of a module, it has to be imported into your module declaration file and declared in the declarations array. This ensures that the component is visible to all other components within the module.

Para que cualquier componente se utilice dentro del contexto de un módulo, debe importarse al archivo de declaración de su módulo y declararse en la matriz declarations. Esto garantiza que el componente sea visible para todos los demás componentes dentro del módulo.

There are three specific attributes on the NgModule that directly impact components and their usage, which are important to know. While only declarations is important initially, once you start working with multiple modules, or if you are either creating or importing other modules, the other two attributes become essential:

Hay tres atributos específicos en NgModule que impactan directamente los componentes y su uso, y es importante conocerlos. Si bien inicialmente solo declarations es importante, una vez que comienza a trabajar con varios módulos, o si está creando o importando otros módulos, los otros dos atributos se vuelven esenciales:

declarations

The `declarations` attribute ensures that components and directives are available to use within the scope of the module. The Angular CLI will automatically add your component or directive to the module when you create a component through it. When you first start out building Angular applications, you might easily forget to add your newly created components to the `declarations` attribute, so keep track of that (if you are not using the Angular CLI, that is!) in order to avoid this common mistake.

El atributo `declarations` garantiza que los componentes y directivas estén disponibles para su uso dentro del alcance del módulo. Angular CLI agregará automáticamente su componente o directiva al módulo cuando cree un componente a través de él. Cuando comienza a crear aplicaciones Angular por primera vez, es posible que se olvide fácilmente de agregar los componentes recién creados al atributo `declarations`, así que haga un seguimiento de eso (si no está utilizando la CLI de Angular, claro está) para poder Evite este error común.

imports

The `imports` attribute allows you to specify modules that you want imported and accessible within your module. This is mostly as a way to pull in third-party modules to make the components and services available within your application. If you want to use a component from other modules, make sure you import the relevant modules into the module you have declared and where the component exists.

El atributo `imports` le permite especificar los módulos que desea importar y acceder a ellos dentro de su módulo. Esto es principalmente como una forma de incorporar módulos de terceros para que los componentes y servicios estén disponibles dentro de su aplicación. Si desea utilizar un componente de otros

módulos, asegúrese de importar los módulos relevantes al módulo que ha declarado y donde existe el componente.

exports

The `exports` attribute is relevant if you either have multiple modules or you need to create a library that will be used by other developers. Unless you export a component, it cannot be accessed or used outside of the direct module where the component is declared. As a general rule of thumb, if you will need to use the component in another module, make sure you export it.

El atributo `exports` es relevante si tiene varios módulos o necesita crear una biblioteca que utilizarán otros desarrolladores. A menos que exporte un componente, no se puede acceder a él ni utilizarlo fuera del módulo directo donde se declara el componente. Como regla general, si necesita utilizar el componente en otro módulo, asegúrese de exportarlo.

TIP CONSEJO

If you are facing issues using a component, where Angular fails to recognize a component or says it does not recognize an element, it most likely is due to misconfigured modules. Check, in order, the following:

Si tiene problemas al utilizar un componente, donde Angular no reconoce un componente o dice que no reconoce un elemento, lo más probable es que se deba a módulos mal configurados. Marque, en orden, lo siguiente:

- Whether the component is added as a declaration in the module.
Si el componente se agrega como una declaración en el módulo.
- In case it is not a component that you wrote, make sure that you have imported the module that provides/exports the component.
En caso de que no sea un componente que usted escribió, asegúrese de haber importado el módulo que proporciona/exporta el componente.
- If you created a new component that needs to be used in other components, make sure that you export the component in its module so that any application including the module will get access to your newly created component.
Si creó un nuevo componente que necesita usarse en otros componentes, asegúrese de exportar el componente en su módulo para que cualquier aplicación, incluido el módulo, tenga acceso a su componente recién creado.

Input and Output

Entrada y salida

One common use case when we start creating components is that we want to separate the content that a component uses from the component itself. A component is truly useful when it is reusable. One of the ways we can make a component reusable (rather than having default, hardcoded values inside it) is by passing in different inputs depending on the use case. Similarly, there might be cases

where we want hooks from a component when a certain activity happens within its context.

Un caso de uso común cuando empezamos a crear componentes es que queremos separar el contenido que utiliza un componente del propio componente. Un componente es realmente útil cuando es reutilizable. Una de las formas en que podemos hacer que un componente sea reutilizable (en lugar de tener valores predeterminados codificados en su interior) es pasando diferentes entradas según el caso de uso. De manera similar, puede haber casos en los que queramos enlaces de un componente cuando ocurre una determinada actividad dentro de su contexto.

Angular provides hooks to specify each of these through decorators, aptly named `Input` and `Output`. These, unlike the `Component` and `NgModule` decorators, apply at a class member variable level.

Angular proporciona ganchos para especificar cada uno de estos a través de decoradores, acertadamente llamados `Input` y `Output`. Estos, a diferencia de los decoradores `Component` y `NgModule`, se aplican a un nivel de variable miembro de clase.

Input

Aporte

When we add an `Input` decorator on a member variable, it automatically allows you to pass in values to the component for that particular input via Angular's data binding syntax.

Cuando agregamos un decorador `Input` en una variable miembro, automáticamente le permite pasar valores al componente para esa entrada en particular a través de la sintaxis de enlace de datos de Angular.

Let's see how we can extend our `stock-item` component from the previous chapter to allow us to pass in the `stock` object, rather than

hardcoding it within the component itself. The finished example is available in the GitHub repository in the *chapter4/component-input* folder. If you want to code along and don't have the previous code, you can use the *chapter3/ng-if* codebase as the starter to code along from.

Veamos cómo podemos extender nuestro componente `stock-item` del capítulo anterior para permitirnos pasar el objeto `stock`, en lugar de codificarlo dentro del propio componente. El ejemplo terminado está disponible en el repositorio de GitHub en la carpeta `capítulo4/component-input`. Si desea codificar y no tiene el código anterior, puede usar el código base del capítulo 3/`ng-if` como punto de partida para codificar.

We will first modify the `stock-item` component to mark the `stock` as an input to the component, but instead of initializing the `stock` object, we will mark it as an `Input` to the component. We do this by importing the decorator and using it for the `stock` variable. The code for the `stock-item.component.ts` file should look like the following:

Primero modificaremos el componente `stock-item` para marcar el `stock` como una entrada para el componente, pero en lugar de inicializar el objeto de `stock`, lo marcaremos como `Input` para el componente. Hacemos esto importando el decorador y usándolo para la variable `stock`. El código para el archivo `stock-item.component.ts` debería verse similar al siguiente:

```
import { Component, OnInit, Input } from '@angular/core';

import { Stock } from '../model/stock';

@Component({
  selector: 'app-stock-item',
  templateUrl: './stock-item.component.html',
  styleUrls: ['./stock-item.component.css']
})
export class StockItemComponent {

  @Input() public stock: Stock;
```

```

constructor() { }

toggleFavorite(event) {
  this.stock.favorite = !this.stock.favorite;
}
}

```

We have removed all instantiation logic from the `app-stock-item` component, and marked the `stock` variable as an input. This means that the initialization logic has been moved out, and the component is only responsible for receiving the value of the stock from the parent component and just rendering the data.

Eliminamos toda la lógica de creación de instancias del componente `app-stock-item` y marcamos la variable `stock` como entrada. Esto significa que la lógica de inicialización se ha eliminado y el componente solo es responsable de recibir el valor de las acciones del componente principal y simplemente representar los datos.

Next, let's take a look at the `AppComponent` and how we can change that to now pass in the data to the `StockItemComponent`:

A continuación, echemos un vistazo a `AppComponent` y cómo podemos cambiarlo para pasar ahora los datos a `StockItemComponent`:

```

import { Component, OnInit } from '@angular/core';
import { Stock } from 'app/model/stock';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit {
  title = 'Stock Market App';

  public stockObj: Stock;

  ngOnInit(): void {
    this.stockObj = new Stock('Test Stock Company', 'TSC', 85, 80);
  }
}

```

```
    }  
}
```

We just moved the initialization of the stock object from the StockItemComponent to the AppComponent. Finally, let's take a look at the template of the AppComponent to see how we can pass in the stock to the StockItemComponent:

Acabamos de mover la inicialización del objeto de stock del StockItemComponent al AppComponent. Finalmente, echemos un vistazo a la plantilla de AppComponent para ver cómo podemos pasar las acciones a StockItemComponent:

```
<h1>  
  {{title}}  
</h1>  
<app-stock-item [stock]="stockObj"></app-stock-item>
```

We use Angular's data binding to pass in the stock from the AppComponent to the StockItemComponent. The name of the attribute (stock) has to match the name of the variable in the component that has been marked as input. The attribute name is case sensitive, so make sure it matches exactly with the input variable name. The value that we pass to it is the reference of the object in the AppComponent class, which is stockObj.

Usamos el enlace de datos de Angular para pasar el stock del AppComponent al StockItemComponent. El nombre del atributo (stock) tiene que coincidir con el nombre de la variable en el componente que se ha marcado como entrada. El nombre del atributo distingue entre mayúsculas y minúsculas, así que asegúrese de que coincida exactamente con el nombre de la variable de entrada. El valor que le pasamos es la referencia del objeto en la clase AppComponent, que es stockObj.

HTML AND CASE-SENSITIVE ATTRIBUTES? ¿HTML Y ATRIBUTOS QUE DISTINGUEN ENTRE MAYÚSCULAS Y MINÚSCULAS?

You might wonder how this is even possible. Angular has its own HTML parser under the covers that parses the templates for Angular-specific syntax, and does not rely on the DOM API for some of these. This is why Angular attributes are and can be case-sensitive.

Quizás te preguntes cómo es posible esto. Angular tiene su propio analizador HTML oculto que analiza las plantillas para la sintaxis específica de Angular y no depende de la API DOM para algunas de ellas. Es por eso que los atributos angulares distinguen y pueden distinguir entre mayúsculas y minúsculas.

These inputs are data bound, so if you end up changing the value of the object in AppComponent, it will automatically be reflected in the child StockItemComponent.

Estas entradas están vinculadas a datos, por lo que si terminas cambiando el valor del objeto en AppComponent, se reflejará automáticamente en el objeto secundario StockItemComponent.

Output

Producción

Just like we can pass data into a component, we can also register and listen for events from a component. We use data binding to pass data in, and we use event binding syntax to register for events. We use the Output decorator to accomplish this.

Así como podemos pasar datos a un componente, también podemos registrar y escuchar eventos de un componente. Usamos enlace de datos para pasar datos y usamos sintaxis de enlace de eventos para registrar eventos. Usamos el decorador Output para lograr esto.

We register an EventEmiter as an output from any component. We can then trigger the event using the EventEmiter object, which will

allow any component bound to the event to get the notification and act accordingly.

Registraremos un `EventEmitter` como salida de cualquier componente. Luego podemos activar el evento usando el objeto `EventEmitter`, lo que permitirá que cualquier componente vinculado al evento reciba la notificación y actúe en consecuencia.

We can use the code from the previous example where we registered an `Input` decorator and continue on from there. Let's now extend the `StockComponent` to trigger an event when it is favorited, and move the data manipulation out from the component to its parent. This makes sense as well because the parent component is responsible for the data and should be the single source of truth. Thus, we will let the parent `AppComponent` register for the `toggleFavorite` event and change the state of the stock when the event is triggered.

Podemos usar el código del ejemplo anterior donde registramos un decorador `Input` y continuar desde allí. Ahora extendamos `StockComponent` para activar un evento cuando sea favorito y traslademos la manipulación de datos del componente a su parent. Esto también tiene sentido porque el componente principal es responsable de los datos y debe ser la única fuente de verdad. Por lo tanto, permitiremos que el parent `AppComponent` se registre para el evento `toggleFavorite` y cambie el estado de las acciones cuando se active el evento.

The finished code for this is in the `chapter4/component-output` folder.

El código terminado para esto se encuentra en la carpeta `capítulo4/componente-salida`.

Take a look at the `StockItemComponent` code in `src/app/stock/stock-item/stock-item.component.ts`:

Eche un vistazo al código StockItemComponent en src/app/stock/stock-item/stock-item.component.ts:

```
import { Component, OnInit, Input, Output, EventEmitter } from
'@angular/core';

import { Stock } from '../model/stock';

@Component({
  selector: 'app-stock-item',
  templateUrl: './stock-item.component.html',
  styleUrls: ['./stock-item.component.css']
})
export class StockItemComponent {

  @Input() public stock: Stock;
  @Output() private toggleFavorite: EventEmitter<Stock>;

  constructor() {
    this.toggleFavorite = new EventEmitter<Stock>();
  }

  onToggleFavorite(event) {
    this.toggleFavorite.emit(this.stock);
  }
}
```

A few important things to note:

Algunas cosas importantes a tener en cuenta:

- We imported the Output decorator as well as the EventEmitter from the Angular library.

Importamos el decorador Output así como el EventEmitter de la biblioteca Angular.

- We created a new class member called toggleFavorite of type EventEmitter, and renamed our method to onToggleFavorite. The EventEmitter can be typed for additional type safety.

Creamos un nuevo miembro de clase llamado toggleFavorite de tipo EventEmitter y cambiamos el nombre de nuestro

método a `onToggleFavorite`. El `EventEmitter` se puede escribir para mayor seguridad de tipos.

- We need to ensure that the `EventEmitter` instance is initialized, as it is *not* auto-initialized for us. Either do it inline or do it in the constructor as we did earlier.

Necesitamos asegurarnos de que la instancia `EventEmitter` esté inicializada, ya que no se inicializa automáticamente por nosotros. Hágalo en línea o hágalo en el constructor como hicimos antes.

- The `onToggleFavorite` now just calls a method on the `EventEmitter` to emit the entire stock object. This means that all listeners of the `toggleFavorite` event will get the current stock object as a parameter.

El `onToggleFavorite` ahora simplemente llama a un método en el `EventEmitter` para emitir el objeto de stock completo. Esto significa que todos los oyentes del evento `toggleFavorite` obtendrán el objeto de stock actual como parámetro.

We will also change `stock-item.component.html` to call the `onToggleFavorite` method instead of `toggleFavorite`. The HTML markup remains pretty much the same otherwise:

También cambiaremos `stock-item.component.html` para llamar al método `onToggleFavorite` en lugar de `toggleFavorite`. El marcado HTML sigue siendo prácticamente el mismo por lo demás:

```
<div class="stock-container">
  <div class="name">{{stock.name + ' (' + stock.code + ')'}}</div>
  <div class="price"
    [class]="stock.isPositiveChange() ? 'positive' : 'negative'">
    $ {{stock.price}}
  </div>
  <button (click)="onToggleFavorite($event)"
    *ngIf="!stock.favorite">Add to Favorite</button>
</div>
```

Next, we add a method to the AppComponent that should be triggered whenever the `onToggleFavorite` method is triggered, which we will add event binding on:

A continuación, agregamos un método a AppComponent que debe activarse cada vez que se activa el método `onToggleFavorite`, al cual agregaremos enlace de evento:

```
import { Component, OnInit } from '@angular/core';
import { Stock } from 'app/model/stock';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit {
  title = 'app works!';

  public stock: Stock;

  ngOnInit(): void {
    this.stock = new Stock('Test Stock Company', 'TSC', 85, 80);
  }

  onToggleFavorite(stock: Stock) {
    console.log('Favorite for stock ', stock, ' was triggered');
    this.stock.favorite = !this.stock.favorite;
  }
}
```

The only thing new is the `onToggleFavorite` method we have added, which takes a stock as an argument. In this particular case, we don't use the stock passed to it other than for logging, but you could base any decision/work on that. Note also that the name of the function is not relevant, and you could name it whatever you want.

Lo único nuevo es el método `onToggleFavorite` que hemos agregado, que toma una acción como argumento. En este caso particular, no utilizamos el stock que se le pasa más que para registrar, pero usted podría basar cualquier decisión/trabajo en eso.

Tenga en cuenta también que el nombre de la función no es relevante y puede nombrarla como desee.

Finally, let's tie it all together by subscribing to the new output from our StockComponent in the *app-component.html* file:

Finalmente, unamos todo suscribiéndonos al nuevo resultado de nuestro StockComponent en el archivo app-component.html:

```
<h1>
  {{title}}
</h1>
<app-stock-item [stock]="stock"
  (toggleFavorite)="onToggleFavorite($event)">
</app-stock-item>
```

We just added an event binding using Angular's event-binding syntax to the output declared in the `stock-item` component. Notice again that it is case sensitive and it has to exactly match what member variable we decorated with the `Output` decorator. Also, to get access to the value emitted by the component, we use the keyword `$event` as a parameter to the function. Without it, the function would still get triggered, but you would not get any arguments with it.

Acabamos de agregar un enlace de evento usando la sintaxis de enlace de eventos de Angular a la salida declarada en el componente `stock-item`. Observe nuevamente que distingue entre mayúsculas y minúsculas y tiene que coincidir exactamente con la variable miembro que decoramos con el decorador `Output`. Además, para obtener acceso al valor emitido por el componente, usamos la palabra clave `$event` como parámetro de la función. Sin él, la función aún se activaría, pero no obtendría ningún argumento.

With this, if you run the application (remember, `ng serve`), you should see the fully functional app, and when you click the Add to Favorite button, it should trigger the method in the AppComponent.

Con esto, si ejecuta la aplicación (recuerde, `ng serve`), debería ver la aplicación completamente funcional y, cuando haga clic en el botón Agregar a favoritos, debería activar el método en `AppComponent`.

Change Detection

Detección de cambios

We mentioned `changeDetection` as an attribute on the `Component` decorator. Now that we have seen how `Input` and `Output` decorators work, let's deep dive a little bit into how Angular performs its change detection at a component level.

Mencionamos `changeDetection` como atributo en el decorador `Component`. Ahora que hemos visto cómo funcionan los decoradores `Input` y `Output`, profundicemos un poco en cómo Angular realiza su detección de cambios a nivel de componente.

By default, Angular applies the `ChangeDetectionStrategy.Default` mechanism to the `changeDetection` attribute. This means that every time Angular notices an event (say, a server response or a user interaction), it will go through each component in the component tree, and check each of the bindings individually to see if any of the values have changed and need to be updated in the view.

De forma predeterminada, Angular aplica el mecanismo `ChangeDetectionStrategy.Default` al atributo `changeDetection`. Esto significa que cada vez que Angular detecta un evento (por ejemplo, una respuesta del servidor o una interacción del usuario), revisará cada componente en el árbol de componentes y verificará cada uno de los enlaces individualmente para ver si alguno de los valores ha cambiado y necesita para ser actualizado en la vista.

For a very large application, you will have lots of bindings on a given page. When a user takes any action, you as a developer might know

for sure that most of the page will not change. In such cases, you can actually give a hint to the Angular change detector to check or not check certain components as you see fit. For any given component, we can accomplish this by changing the `ChangeDetectionStrategy` from the default to `ChangeDetectionStrategy.OnPush`. What this tells Angular is that the bindings for this particular component will need to be checked only based on the `Input` to this component.

Para una aplicación muy grande, tendrá muchos enlaces en una página determinada. Cuando un usuario realiza alguna acción, usted, como desarrollador, puede estar seguro de que la mayor parte de la página no cambiará. En tales casos, puede darle una pista al detector de cambio angular para que verifique o no ciertos componentes como mejor le parezca. Para cualquier componente determinado, podemos lograr esto cambiando el `ChangeDetectionStrategy` del valor predeterminado a `ChangeDetectionStrategy.OnPush`. Lo que esto le dice a Angular es que los enlaces para este componente en particular deberán verificarse solo en función del `Input` de este componente.

Let's consider a few examples to see how this might play out. Say we have a component tree `A → B → C`. That is, we have a root component `A`, which uses a component `B` in its template, which in turn uses a component `C`. And let's say component `B` passes in a composite object `compositeObj` to component `C` as input. Maybe something like:

Consideremos algunos ejemplos para ver cómo podría funcionar esto. Digamos que tenemos un árbol de componentes `A → B → C`. Es decir, tenemos un componente raíz `A`, que usa un componente `B` en su plantilla, que a su vez usa un componente `C`. Y digamos que el componente `B` pasa en un objeto compuesto `compositeObj` al componente `C` como entrada. Quizás algo como:

```
<c [inputToC]="compositeObj"></c>
```

That is, `inputToC` is the input variable marked with the `Input` decorator in component C, and is passed the object `compositeObj` from component B. Now say we marked component C's `changeDetection` attribute as `ChangeDetectionStrategy.OnPush`. Here are the implications of that change:

Es decir, `inputToC` es la variable de entrada marcada con el decorador `Input` en el componente C, y se le pasa el objeto `compositeObj` del componente B. Ahora digamos que marcamos el `changeDetection` del componente C. atributo como `ChangeDetectionStrategy.OnPush`. Estas son las implicaciones de ese cambio:

- If component C has bindings to any attributes of `compositeObj`, they will work as usual (no change from default behavior).

Si el componente C tiene enlaces a algún atributo de `compositeObj`, funcionarán como de costumbre (sin cambios con respecto al comportamiento predeterminado).

- If component C makes any changes to any of the attributes of `compositeObj`, they will also be updated immediately (no change from default behavior).

Si el componente C realiza algún cambio en cualquiera de los atributos de `compositeObj`, también se actualizarán inmediatamente (sin cambios con respecto al comportamiento predeterminado).

- If the parent component B creates a new `compositeObj` or changes the reference of `compositeObj` (think new operator, or assign from a server response), then component C would recognize the change and update its bindings for the new value (no change from default behavior, but internal behavior changes on how Angular recognizes the change).

Si el componente principal B crea un nuevo `compositeObj` o cambia la referencia de `compositeObj` (piense en un nuevo operador o asigne desde una respuesta del servidor), entonces el componente C reconocería el cambio y actualizaría sus enlaces para el nuevo valor (sin cambios con respecto al comportamiento predeterminado, pero el comportamiento interno cambia sobre cómo Angular reconoce el cambio).

- If the parent component B changes any attribute on the `compositeObj` directly (as a response to a user action outside component B), then these changes would not be updated in component C (major change from the default behavior).

Si el componente principal B cambia cualquier atributo en `compositeObj` directamente (como respuesta a una acción del usuario fuera del componente B), entonces estos cambios no se actualizarán en el componente C (cambio importante con respecto al comportamiento predeterminado).

- If the parent component B changes any attribute on response to an event emitter from component C, and then changes any attribute on the `compositeObj` (without changing the reference), this would still work and the bindings would get updated. This is because the change originates from component C (no change from default behavior).

Si el componente principal B cambia cualquier atributo en respuesta a un emisor de eventos del componente C, y luego cambia cualquier atributo en `compositeObj` (sin cambiar la referencia), esto aún funcionaría y los enlaces se actualizarían. Esto se debe a que el cambio se origina en el componente C (no hay cambios con respecto al comportamiento predeterminado).

Angular provides ways for us to signal when to check the bindings from within the component as well, to have absolute control on Angular's data binding. We will cover these in "["Change Detection"](#)".

For now, it is good to understand the difference between the two change detection strategies that Angular provides.

Angular también nos proporciona formas de indicarnos cuándo verificar los enlaces desde dentro del componente, para tener control absoluto sobre el enlace de datos de Angular. Los cubriremos en "Detección de cambios". Por ahora, es bueno comprender la diferencia entre las dos estrategias de detección de cambios que ofrece Angular.

Let's now modify the example code to see this in action. First, modify the *stock-item.component.ts* file to change the ChangeDetectionStrategy in the child component:

Modifiquemos ahora el código de ejemplo para verlo en acción. Primero, modifique el archivo stock-item.component.ts para cambiar el ChangeDetectionStrategy en el componente secundario:

```
import { Component, OnInit, Input, Output } from '@angular/core';
import { EventEmitter, ChangeDetectionStrategy } from '@angular/core';

import { Stock } from '../../model/stock';

@Component({
  selector: 'app-stock-item',
  templateUrl: './stock-item.component.html',
  styleUrls: ['./stock-item.component.css'],
  changeDetection: ChangeDetectionStrategy.OnPush
})
export class StockItemComponent {

  @Input() public stock: Stock;
  @Output() private toggleFavorite: EventEmitter<Stock>;

  constructor() {
    this.toggleFavorite = new EventEmitter<Stock>();
  }

  onToggleFavorite(event) {
    this.toggleFavorite.emit(this.stock);
  }

  changeStockPrice() {
```

```
        this.stock.price += 5;
    }
}
```

In addition to changing the `ChangeDetectionStrategy`, we also added another function to `changeStockPrice()`. We will use these functions to demonstrate the behavior of the change detection in the context of our application.

Además de cambiar `ChangeDetectionStrategy`, también agregamos otra función a `changeStockPrice()`. Usaremos estas funciones para demostrar el comportamiento de la detección de cambios en el contexto de nuestra aplicación.

Next, let's quickly modify `stock-item.component.html` to allow us to trigger the new function. We will simply add a new button to trigger and change the stock price when the button is clicked:

A continuación, modifiquemos rápidamente `stock-item.component.html` para permitirnos activar la nueva función. Simplemente agregaremos un nuevo botón para activar y cambiar el precio de las acciones cuando se haga clic en el botón:

```
<div class="stock-container">
  <div class="name">{{stock.name + ' (' + stock.code + ')'}}</div>
  <div class="price"
    [class]="stock.isPositiveChange() ? 'positive' : 'negative'"
    $ {{stock.price}}>
  </div>
  <button (click)="onToggleFavorite($event)"
    *ngIf="!stock.favorite">Add to Favorite</button>
  <button (click)="changeStockPrice()">Change Price</button>
</div>
```

There is no change to the HTML of the template other than adding a new button to change the stock price. Similarly, let's quickly change the main `app.component.html` file to add another button to trigger the change of the price from the parent component (similar to component B in the earlier hypothetical example):

No hay ningún cambio en el HTML de la plantilla más que agregar un nuevo botón para cambiar el precio de las acciones. De manera similar, cambiemos rápidamente el archivo principal app.component.html para agregar otro botón que active el cambio del precio del componente principal (similar al componente B en el ejemplo hipotético anterior):

```
<h1>
  {{title}}
</h1>
<app-stock-item [stock]="stock"
  (toggleFavorite)="onToggleFavorite($event)">
</app-stock-item>
<button (click)="changeStockObject()">Change Stock</button>
<button (click)="changeStockPrice()">Change Price</button>
```

We have added two new buttons to this template: one that will change the reference of the stock object directly, and another that will modify the existing reference of the stock object to change the price from the parent AppComponent. Now finally, we can see how all of this is hooked up in the *app.component.ts* file:

Hemos agregado dos botones nuevos a esta plantilla: uno que cambiará la referencia del objeto de acciones directamente y otro que modificará la referencia existente del objeto de acciones para cambiar el precio del principal AppComponent. Ahora, finalmente, podemos ver cómo todo esto está conectado en el archivo app.component.ts:

```
import { Component, OnInit } from '@angular/core';
import { Stock } from 'app/model/stock';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit {
  title = 'app works!';
```

```

public stock: Stock;
private counter: number = 1;

ngOnInit(): void {
  this.stock = new Stock('Test Stock Company - ' + this.counter++,
    'TSC', 85, 80);
}

onToggleFavorite(stock: Stock) {
  // This will update the value in the stock item component
  // Because it is triggered as a result of an event
  // binding from the stock item component
  this.stock.favorite = !this.stock.favorite;
}

changeStockObject() {
  // This will update the value in the stock item component
  // Because we are creating a new reference for the stock input
  this.stock = new Stock('Test Stock Company - ' + this.counter++,
    'TSC', 85, 80);
}

changeStockPrice() {
  // This will not update the value in the stock item component
  // because it is changing the same reference and angular will
  // not check for it in the OnPush change detection strategy.
  this.stock.price += 10;
}
}

```

The `app.component.ts` file has seen the most changes. The preceding code is also well annotated with comments to explain the expected behavior when each of these functions are triggered. We have added two new methods: `changeStockObject()`, which creates a new instance of the stock object in the `AppComponent`, and `changeStockPrice()`, which modifies the prices of the stock object in the `AppComponent`. We have also added a counter just to keep track of how many times we create a new stock object, but that is not strictly necessary.

El archivo `app.component.ts` ha visto la mayor cantidad de cambios. El código anterior también está bien anotado con comentarios para explicar el comportamiento esperado cuando se activa cada una de

estas funciones. Hemos añadido dos nuevos métodos: `changeStockObject()`, que crea una nueva instancia del objeto `stock` en el `AppComponent`, y `changeStockPrice()`, que modifica los precios del `stock` objeto en el `AppComponent`. También hemos agregado un contador solo para realizar un seguimiento de cuántas veces creamos un nuevo objeto de stock, pero eso no es estrictamente necesario.

Now when you run this application, you should expect to see the following behavior:

Ahora, cuando ejecute esta aplicación, debería esperar ver el siguiente comportamiento:

- Clicking Add to Favorite within the `StockItemComponent` still works as expected.
Hacer clic en Agregar a favoritos dentro de `StockItemComponent` todavía funciona como se esperaba.
- Clicking Change Price within the `StockItemComponent` will increase the price of the stock by \$5 each time.
Al hacer clic en Cambiar precio dentro de `StockItemComponent`, el precio de la acción aumentará en \$5 cada vez.
- Clicking Change Stock outside the `StockItemComponent` will change the name of the stock with each click. (This is why we added the counter!)
Al hacer clic en Cambiar acción fuera de `StockItemComponent`, se cambiará el nombre de la acción con cada clic. (Es por eso que agregamos el contador!)
- Clicking Change Price outside the `StockItemComponent` will have no impact (even though the actual value of the stock will jump if you click Change Price inside after this). This shows that the model is getting updated, but Angular is not updating the view.

Hacer clic en Cambiar precio fuera de StockItemComponent no tendrá ningún impacto (aunque el valor real de la acción aumentará si hace clic en Cambiar precio dentro después de esto). Esto muestra que el modelo se está actualizando, pero Angular no actualiza la vista.

You should also change back the ChangeDetectionStrategy to default to see the difference in action.

También debes volver a cambiar el ChangeDetectionStrategy al valor predeterminado para ver la diferencia en acción.

Component Lifecycle

Ciclo de vida del componente

Components (and directives) in Angular have their own lifecycle, from creation, rendering, changing, to destruction. This lifecycle executes in preorder tree traversal order, from top to bottom. After Angular renders a component, it starts the lifecycle for each of its children, and so on until the entire application is rendered.

Los componentes (y directivas) en Angular tienen su propio ciclo de vida, desde la creación, la renderización, el cambio hasta la destrucción. Este ciclo de vida se ejecuta en orden transversal del árbol de preorden, de arriba a abajo. Después de que Angular renderiza un componente, inicia el ciclo de vida de cada uno de sus hijos, y así sucesivamente hasta que se renderiza toda la aplicación.

There are times when these lifecycle events are useful to us in developing our application, so Angular provides hooks into this lifecycle so that we can observe and react as necessary. [Figure 4-1](#) shows the lifecycle hooks of a component, in the order in which they are invoked.

Hay ocasiones en las que estos eventos del ciclo de vida nos resultan útiles a la hora de desarrollar nuestra aplicación, por lo que Angular proporciona enlaces a este ciclo de vida para que podamos observar y reaccionar según sea necesario. La Figura 4-1 muestra los enlaces del ciclo de vida de un componente, en el orden en que se invocan.

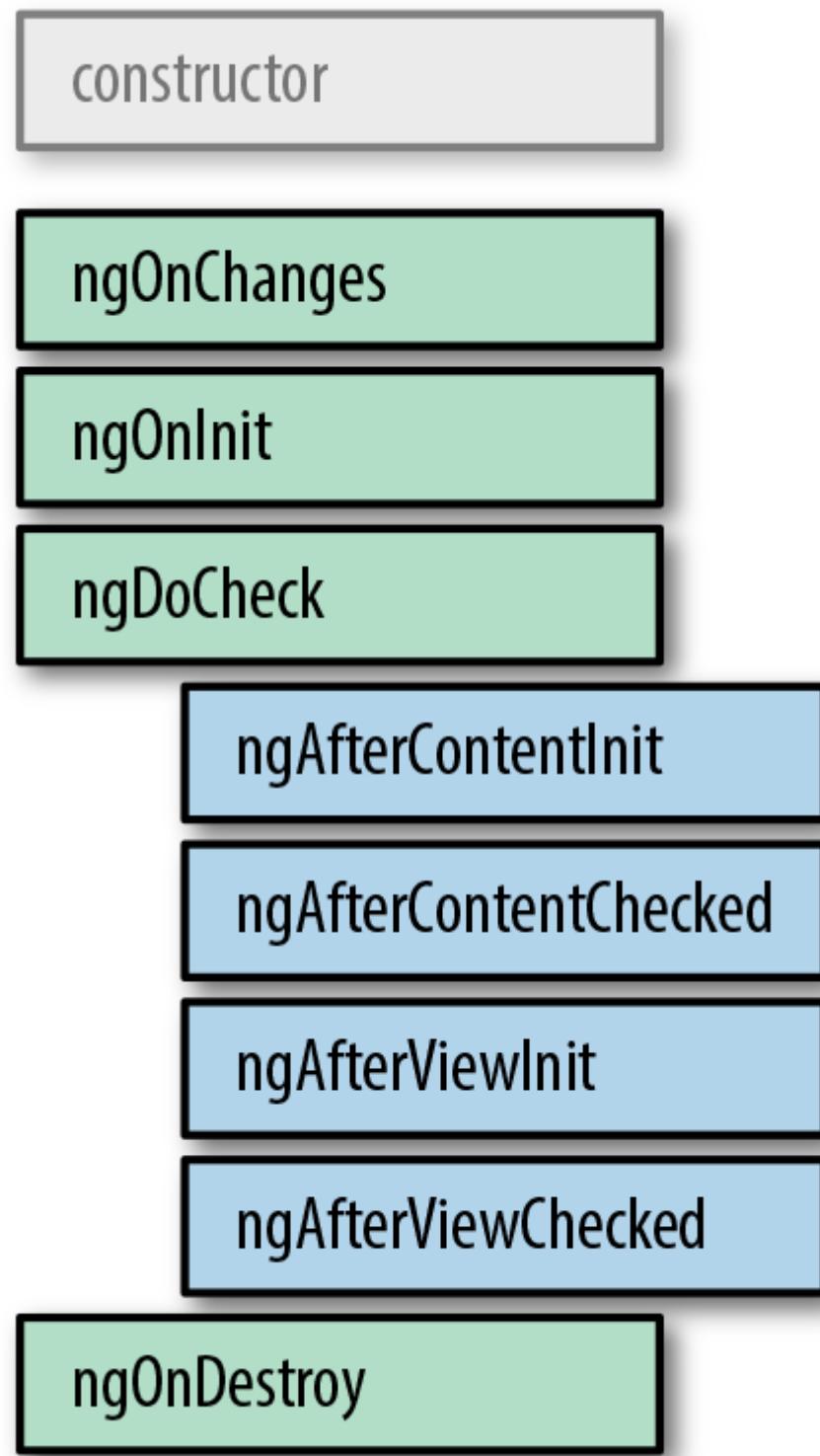


Figure 4-1. Angular component lifecycle hooks (original from <https://angular.io/guide/lifecycle-hooks>)

Figura 4-1. Ganchos del ciclo de vida de los componentes angulares (original de <https://angular.io/guide/lifecycle-hooks>)

Angular will first call the constructor for any component, and then the various steps mentioned earlier in order. Some of them, like the `OnInit` and `AfterContentInit` (basically, any lifecycle hook ending with `Init`) is called only once, when a component is initialized, while the others are called whenever any content changes. The `OnDestroy` hook is also called only once for a component.

Angular primero llamará al constructor de cualquier componente y luego realizará los distintos pasos mencionados anteriormente en orden. Algunos de ellos, como `OnInit` y `AfterContentInit` (básicamente, cualquier gancho de ciclo de vida que termina en `Init`) se llaman solo una vez, cuando se inicializa un componente, mientras que los otros se llaman cada vez que se invoca cualquier contenido. El gancho `OnDestroy` también se llama solo una vez para un componente.

Each of these lifecycle steps comes with an interface that should be implemented when a component cares about that particular lifecycle, and each interface provides a function starting with `ng` that needs to be implemented. For example, the `OnInit` lifecycle step needs a function called `ngOnInit` to be implemented in the component and so on.

Cada uno de estos pasos del ciclo de vida viene con una interfaz que debe implementarse cuando un componente se preocupa por ese ciclo de vida en particular, y cada interfaz proporciona una función que comienza con `ng` que debe implementarse. Por ejemplo, el paso del ciclo de vida `OnInit` necesita que se implemente una función llamada `ngOnInit` en el componente, y así sucesivamente.

We will walk through each of the lifecycle steps here, and then use one example to see this all in action and the ordering of lifecycle steps within a component and across components.

Aquí recorreremos cada uno de los pasos del ciclo de vida y luego usaremos un ejemplo para ver todo esto en acción y el orden de los

pasos del ciclo de vida dentro de un componente y entre componentes.

There is also one more concept to learn, which we will briefly touch upon in this chapter, and come back to later in more detail—the concept of `ViewChildren` and `ContentChildren`.

También hay un concepto más que aprender, que abordaremos brevemente en este capítulo y al que volveremos más adelante con más detalle: el concepto de `ViewChildren` y `ContentChildren`.

`ViewChildren` is any child component whose tags/selectors (mostly elements, as that is the recommendation for components) appear within the template of the component. So in our case, `app-stock-item` would be a `ViewChild` of the `AppComponent`.

`ViewChildren` es cualquier componente secundario cuyas etiquetas/selectores (principalmente elementos, ya que esa es la recomendación para los componentes) aparecen dentro de la plantilla del componente. Entonces, en nuestro caso, `app-stock-item` sería un `ViewChild` de `AppComponent`.

`ContentChildren` is any child component that gets projected into the view of the component, but is not directly included in the template within the component. Imagine something like a carousel, where the functionality is encapsulated in the component, but the view, which could be images or pages of a book, comes from the user of the component. Those are generally achieved through `ContentChildren`. We will cover this in more depth later in this chapter.

`ContentChildren` es cualquier componente secundario que se proyecta en la vista del componente, pero que no se incluye directamente en la plantilla dentro del componente. Imagine algo así como un carrusel, donde la funcionalidad está encapsulada en el componente, pero la vista, que podrían ser imágenes o páginas de un libro, proviene del usuario del componente. Generalmente se

logran a través de `ContentChildren`. Cubriremos esto con más profundidad más adelante en este capítulo.

Interfaces and Functions

Interfaces y funciones

Table 4-1 shows the interfaces and functions in the order in which they are called, along with specific details about the step if there is anything to note. Note that we are only covering component-specific lifecycle steps, and they are slightly different from a directive's lifecycle.

La Tabla 4-1 muestra las interfaces y funciones en el orden en que se llaman, junto con detalles específicos sobre el paso si hay algo que tener en cuenta. Tenga en cuenta que solo cubrimos los pasos del ciclo de vida específicos de los componentes y son ligeramente diferentes del ciclo de vida de una directiva.

Table 4-1. Angular lifecycle hooks and methods

Tabla 4-1. Métodos y ganchos del ciclo de vida angular

Int erf interface	Appl icense	Purpose Objetivo
ngOnChanges: () => void;	Componer las propiedades de entrada de una directiva.	ngOnChanges es llamado justo después del constructor para configurar y luego cada vez que cambian las propiedades de entrada de una directiva. Se llama antes del método ngOnInit.
ngOnInit(): void;	Componer las propiedades de entrada de una directiva.	This is your typical initialization hook, allowing you to do any one-time initialization specific to your component or directive. This is the ideal place to load data from the server and so on, rather than the constructor, both for separation of concerns as well as testability.
DoCheck(): void;	Comprobar si hay cambios en las propiedades de entrada de una directiva.	DoCheck es la forma de Angular de dar al componente una forma de comprobar si hay cambios en las propiedades de entrada que Angular no puede o debe detectar por su cuenta. Esto es una de las formas en las que podemos usar para notificar a Angular de un cambio en el componente, cuando lo overridemos.

Int	Appl		
erf	Meth	icab	
ace	od	le to	Purpose
Int	Méto	Aplic	Objetivo
erf	do	cabl	
az	e a		
DoC	ngDoC	direc	the default ChangeDetectionStrategy for a component from
hec	heck(tives	Default to OnPush.
k)	Com	DoCheck es la forma en que Angular le brinda al componente una
		pone	forma de verificar si hay enlaces o cambios que Angular no puede
		ntes	o no debe detectar por sí solo. Esta es una de las formas que
		y	podemos utilizar para notificar a Angular de un cambio en el
		direc	componente, cuando anulamos el ChangeDetectionStrategy
		tivas	predeterminado para un componente de Default a OnPush.
Aft			
erC			
ont	ngAft	Com	As mentioned, the AfterContentInit hook is triggered during
ent	erCon	pone	component projection cases, and only once during initialization of
Ini	tentI	nts	the component. If there is no projection, this is triggered
t	nit()	only	immediately.
Aft	ngAft	Solo	Como se mencionó, el enlace AfterContentInit se activa durante
erC	erCon	com	los casos de proyección del componente y solo una vez durante la
ont	tentI	pone	inicialización del componente. Si no hay proyección, ésta se activa
ent	nit()	ntes	inmediatamente.
Ini			
t			
Aft	ngAft	Com	AfterContentChecked is triggered each time Angular's change
erC	erCon	pone	detection cycle executes, and in case it is initialization, it is
ont	tentC	nts	triggered right after the AfterContentInit hook.
ent	hecke	only	
Che	d()	Solo	AfterContentChecked se activa cada vez que se ejecuta el ciclo de
cke	ngAft	com	detección de cambios de Angular y, en caso de que se trate de
d	erCon	pone	una inicialización, se activa justo después del enlace
Aft	tentC	ntes	AfterContentInit.
erC	hecke		
ont	d()		
ent			
Che			

Int	Appl
erf Meth	icab
ace od	le to Purpose
Int Méto	Aplic Objetivo
erf do	cabl
az	e a

cke

d

Aft

erV

iew

Ini

t

AfterViewInit is the complement to AfterContentInit, and is triggered after all the child components that are directly used in the template of the component are finished initializing and their views updated with bindings. This may not necessarily mean that

the views are rendered into the browser, but that Angular has finished updating its internal views to render as soon as possible. AfterViewInit is triggered only once during the load of the component.

Aft

erV

iew

Ini

t

Solo

AfterViewInit es el complemento de AfterContentInit y se activa después de que todos los componentes secundarios que se usan directamente en la plantilla del componente terminan de inicializarse y sus vistas se actualizan con enlaces. Es posible que esto no signifique necesariamente que las vistas se representen en el navegador, sino que Angular ha terminado de actualizar sus vistas internas para renderizarse lo antes posible. AfterViewInit se activa solo una vez durante la carga del componente.

Aft

erV

iew

Che

cke

d

ngAft

Com

pone

nts

only

AfterViewChecked is triggered each time after all the child components have been checked and updated. Again, a good way to think about both this and AfterContentChecked is like a depth-first tree traversal, in that it will execute only after all the children components' AfterViewChecked hooks have finished executing.

Aft

erV

iew

Che

cke

d

ngAft

Com

pone

nts

only

Solo

com

pone

ntes

AfterViewChecked se activa cada vez que se han verificado y actualizado todos los componentes secundarios. Nuevamente, una buena manera de pensar en esto y en AfterContentChecked es como un recorrido de árbol en profundidad, en el sentido de que se ejecutará solo después de que todos los ganchos AfterViewChecked de los componentes secundarios hayan terminado de ejecutarse.

OnD

est

roy

ngOnD

estro

y()

Com

pone

nts

The OnDestroy hook is called when a component is about to be destroyed and removed from the UI. It is a good place to do all cleanup, like unsubscribing any listeners you may have initialized

Int	Appl
erf Meth	icab
ace od	le to Purpose
Int Méto	Aplic Objetivo
erf do	cabl
az	e a

OnD ngOnD and and the like. It is generally good practice to clean up anything est estro direc that you have registered (timers, observables, etc.) as part of the roy y() tives component.

Com El gancho `OnDestroy` se llama cuando un componente está a pone punto de ser destruido y eliminado de la interfaz de usuario. Es ntes un buen lugar para realizar todas las tareas de limpieza, como y cancelar la suscripción de cualquier oyente que haya inicializado y direc cosas similares. En general, es una buena práctica limpiar todo lo tivas que haya registrado (temporizadores, observables, etc.) como parte del componente.

Let's try to add all these hooks to our existing application to see the order of execution in a real-world scenario. We will add all of these hooks to both our `AppComponent` and the `StockItemComponent`, with a simple `console.log` to just see when and how these functions are executed. We will use the base from the output example to build from, so in case you are not coding along, you can take the example from *chapter4/component-output* to build from there.

Intentemos agregar todos estos enlaces a nuestra aplicación existente para ver el orden de ejecución en un escenario del mundo real. Agregaremos todos estos enlaces tanto a nuestro `AppComponent` como al `StockItemComponent`, con un simple `console.log` para ver cuándo y cómo se ejecutan estas funciones. Usaremos la base del ejemplo de salida para construir, por lo que en caso de que no esté codificando, puede tomar el ejemplo del capítulo 4/component-output para construir desde allí.

The final finished example is also available in *chapter4/component-lifecycle*.

El ejemplo final terminado también está disponible en el capítulo 4/ciclo de vida del componente.

First, we can modify the `src/app/app.component.ts` file and add the hooks as follows:

Primero, podemos modificar el archivo `src/app/app.component.ts` y agregar los ganchos de la siguiente manera:

```
import { Component, SimpleChanges, OnInit, OnChanges, OnDestroy,
         DoCheck, AfterViewChecked, AfterViewInit, AfterContentChecked,
         AfterContentInit } from '@angular/core';
import { Stock } from 'app/model/stock';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit, OnChanges, OnDestroy,
                                  DoCheck, AfterContentChecked,
                                  AfterContentInit, AfterViewChecked,
                                  AfterViewInit {
  title = 'app works!';

  public stock: Stock;

  onToggleFavorite(stock: Stock) {
    console.log('Favorite for stock ', stock, ' was triggered');
    this.stock.favorite = !this.stock.favorite;
  }

  ngOnInit(): void {
    this.stock = new Stock('Test Stock Company', 'TSC', 85, 80);
    console.log('App Component - On Init');
  }

  ngAfterViewInit(): void {
    console.log('App Component - After View Init');
  }
  ngAfterViewChecked(): void {
    console.log('App Component - After View Checked');
  }
  ngAfterContentInit(): void {
    console.log('App Component - After Content Init');
  }
  ngAfterContentChecked(): void {
    console.log('App Component - After Content Checked');
  }
}
```

```

ngDoCheck(): void {
  console.log('App Component - Do Check');
}
ngOnDestroy(): void {
  console.log('App Component - On Destroy');
}
ngOnChanges(changes: SimpleChanges): void {
  console.log('App Component - On Changes - ', changes);
}
}

```

You can see that we have implemented the interfaces for OnInit, OnChanges, OnDestroy, DoCheck, AfterContentChecked, AfterContentInit, AfterViewChecked, AfterViewInit on the AppComponent class, and then went ahead and implemented the respective functions. Each of the methods simply prints out a log statement mentioning the component name and the trigger method name.

Puede ver que implementamos las interfaces para OnInit, OnChanges, OnDestroy, DoCheck, AfterContentChecked, AfterContentInit, AfterViewChecked, AfterViewInit en la clase AppComponent y luego continuamos e implementamos las funciones respectivas. Cada uno de los métodos simplemente imprime una declaración de registro que menciona el nombre del componente y el nombre del método desencadenante.

Similarly, we can do the same for the StockItemComponent:

De manera similar, podemos hacer lo mismo para StockItemComponent:

```

import { Component, SimpleChanges, OnInit,
         OnChanges, OnDestroy, DoCheck, AfterViewChecked,
         AfterViewInit, AfterContentChecked,
         AfterContentInit, Input,
         Output, EventEmitter } from '@angular/core';
import { Stock } from '../../model/stock';

@Component({
  selector: 'app-stock-item',

```

```

        templateUrl: './stock-item.component.html',
        styleUrls: ['./stock-item.component.css']
    })
export class StockItemComponent implements OnInit, OnChanges,
    OnDestroy, DoCheck,
    AfterContentChecked,
    AfterContentInit,
    AfterViewChecked,
    AfterViewInit {

    @Input() public stock: Stock;
    @Output() private toggleFavorite: EventEmitter<Stock>;

    constructor() {
        this.toggleFavorite = new EventEmitter<Stock>();
    }

    onToggleFavorite(event) {
        this.toggleFavorite.emit(this.stock);
    }

    ngOnInit(): void {
        console.log('Stock Item Component - On Init');
    }
    ngAfterViewInit(): void {
        console.log('Stock Item Component - After View Init');
    }
    ngAfterViewChecked(): void {
        console.log('Stock Item Component - After View Checked');
    }
    ngAfterContentInit(): void {
        console.log('Stock Item Component - After Content Init');
    }
    ngAfterContentChecked(): void {
        console.log('Stock Item Component - After Content Checked');
    }
    ngDoCheck(): void {
        console.log('Stock Item Component - Do Check');
    }
    ngOnDestroy(): void {
        console.log('Stock Item Component - On Destroy');
    }
    ngOnChanges(changes: SimpleChanges): void {
        console.log('Stock Item Component - On Changes - ', changes);
    }
}

```

We have done exactly the same thing we did on the AppComponent with the StockItemComponent. Now, we can run this application to see it in action.

Hemos hecho exactamente lo mismo que hicimos en el AppComponent con el StockItemComponent. Ahora podemos ejecutar esta aplicación para verla en acción.

When you run it, open the JavaScript console in the browser. You should see, in order of execution:

Cuando lo ejecutes, abre la consola de JavaScript en el navegador. Deberías ver, en orden de ejecución:

1. First, the AppComponent gets created. Then the following hooks are triggered on the AppComponent:

Primero, se crea el AppComponent. Luego se activan los siguientes enlaces en AppComponent:

- On Init
On Init
- Do Check
Do Check
- After Content Init
After Content Init
- After Content Checked
After Content Checked

The preceding two immediately execute because we don't have any content projection in our application so far.

Los dos anteriores se ejecutan inmediatamente porque hasta el momento no tenemos ninguna proyección de contenido en nuestra aplicación.

2. Next, the StockItemComponent OnChanges executes, with the input to the StockItemComponent being recognized as the change, followed by the hooks listed here within the StockItemComponent:

A continuación, se ejecuta StockItemComponent OnChanges, y la entrada a StockItemComponent se reconoce como el cambio, seguida de los enlaces enumerados aquí dentro de StockItemComponent:

- On Init
 - On Init
- Do Check
 - Do Check
- After Content Init
 - After Content Init
- After Content Checked
 - After Content Checked
- After View Init
 - After View Init
- After View Checked
 - After View Checked

3. Finally, there are no more subcomponents to traverse down on, so Angular steps back out to the parent AppComponent, and executes the following:

Finalmente, no hay más subcomponentes para recorrer, por lo que Angular regresa al elemento principal AppComponent y ejecuta lo siguiente:

- After View Init
 - After View Init
- After View Checked
 - After View Checked

This gives us a nice view of how and in which order Angular goes around initializing and the tree traversal it does under the covers. These hooks become very useful for certain trickier initialization logic, and are definitely essential for cleanup once your component is done and dusted, to avoid memory leaks.

Esto nos brinda una buena vista de cómo y en qué orden Angular realiza la inicialización y el recorrido del árbol que realiza bajo las sábanas. Estos ganchos se vuelven muy útiles para cierta lógica de inicialización más complicada y definitivamente son esenciales para la limpieza una vez que el componente está listo y desempolvado, para evitar pérdidas de memoria.

View Projection

Ver proyección

The last thing we will cover in this chapter is the concept of view projection. Projection is an important idea in Angular as it gives us more flexibility when we develop our components and again gives us another tool to make them truly reusable under different contexts.

Lo último que cubriremos en este capítulo es el concepto de proyección de vista. La proyección es una idea importante en Angular ya que nos brinda más flexibilidad cuando desarrollamos nuestros componentes y nuevamente nos brinda otra herramienta para hacerlos verdaderamente reutilizables en diferentes contextos.

Projection is useful when we want to build components but set some parts of the UI of the component to not be an innate part of it. For example, say we were building a component for a carousel. A carousel has a few simple capabilities: it is able to display an item, and allow us to navigate to the next/previous element. Your carousel component might also have other features like lazy loading, etc. But one thing that is not the purview of the carousel component is the content it displays. A user of the component might want to display an image, a page of a book, or any other random thing.

La proyección es útil cuando queremos construir componentes pero configurar algunas partes de la interfaz de usuario del componente para que no sean una parte innata del mismo. Por ejemplo, digamos que estamos construyendo un componente para un carrusel. Un carrusel tiene algunas capacidades simples: puede mostrar un elemento y permitirnos navegar al elemento siguiente/anterior. Su componente de carrusel también puede tener otras características como carga diferida, etc. Pero una cosa que no es competencia del componente de carrusel es el contenido que muestra. Es posible que un usuario del componente desee mostrar una imagen, una página de un libro o cualquier otro elemento aleatorio.

Thus, in these cases, the view would be controlled by the user of the component, and the functionality would be provided by the component itself. This is but one use case where we might want to use projection in our components.

Por lo tanto, en estos casos, la vista estaría controlada por el usuario del componente y la funcionalidad la proporcionaría el propio componente. Este es solo un caso de uso en el que es posible que deseemos utilizar la proyección en nuestros componentes.

Let's see how we might use content projection in our Angular application. We will use the base from the input example to build from, so in case you are not coding along, you can take the example from *chapter4/component-input* to build from there.

Veamos cómo podríamos usar la proyección de contenido en nuestra aplicación Angular. Usaremos la base del ejemplo de entrada para construir, por lo que en caso de que no esté codificando, puede tomar el ejemplo del capítulo 4/component-input para construir desde allí.

The final finished example is available in *chapter4/component-projection*.

El ejemplo final terminado está disponible en el capítulo 4/component-projection.

First, we will modify our StockItemComponent to allow for content projection. There is no code change in our component class; we only need to modify the *src/app/stock/stock-item/stock-item.component.html* file as follows:

Primero, modificaremos nuestro StockItemComponent para permitir la proyección de contenido. No hay ningún cambio de código en nuestra clase de componente; solo necesitamos modificar el archivo *src/app/stock/stock-item/stock-item.component.html* de la siguiente manera:

```
<div class="stock-container">
  <div class="name">{{stock.name + ' (' + stock.code + ')'}}</div>
  <div class="price"
    [class]="stock.isPositiveChange() ? 'positive' : 'negative'">
    $ {{stock.price}}
  </div>
  <ng-content></ng-content> ①
</div>
```

① The new element: the element for projection

We have simply removed the buttons we previously had, and are going to let the user of the component decide what buttons are to be shown. To allow for this, we have replaced the buttons with an *ng-content* element. There is no other change required in the component.

Simplemente hemos eliminado los botones que teníamos anteriormente y vamos a dejar que el usuario del componente decida qué botones se mostrarán. Para permitir esto, hemos reemplazado los botones con un elemento `ng-content`. No se requiere ningún otro cambio en el componente.

Next, we will make a change to the `AppComponent`, simply to add a method for testing purposes. Modify the `src/app/app.component.ts` file as follows:

A continuación, haremos un cambio en `AppComponent`, simplemente para agregar un método con fines de prueba. Modifique el archivo `src/app/app.component.ts` de la siguiente manera:

```
/** Imports and decorators skipped for brevity **/ 

export class AppComponent implements OnInit { 
  /** Constructor and OnInit skipped for brevity */ 

  testMethod() { 
    console.log('Test method in AppComponent triggered'); 
  } 
}
```

We have simply added a method that will log to the console when it is triggered. With this in place, now let's see how we can use our updated `StockItemComponent` and use the power of projection.

Modify the `app.component.html` file as follows:

Simplemente agregamos un método que se registrará en la consola cuando se active. Con esto implementado, ahora veamos cómo podemos usar nuestro `StockItemComponent` actualizado y usar el poder de la proyección. Modifique el archivo `app.component.html` de la siguiente manera:

```
<h1>
  {{title}}
</h1>
<app-stock-item [stock]="stockObj">
  <button (click)="testMethod()">With Button 1</button>
```

```
</app-stock-item>

<app-stock-item [stock]="stockObj">
  No buttons for you!!
</app-stock-item>
```

We have added two instances of the `app-stock-item` component in our HTML. And both of these now have some content inside them, as opposed to previously where these elements had no content. In one, we have a button that triggers the `testMethod` we added in the `AppComponent`, and the other simply has text content.

Hemos agregado dos instancias del componente `app-stock-item` en nuestro HTML. Y ambos ahora tienen algo de contenido dentro de ellos, a diferencia de antes donde estos elementos no tenían contenido. En uno, tenemos un botón que activa el `testMethod` que agregamos en el `AppComponent`, y el otro simplemente tiene contenido de texto.

When we run our Angular application and open it in the browser, we should see something like [Figure 4-2](#).

Cuando ejecutamos nuestra aplicación Angular y la abrimos en el navegador, deberíamos ver algo como la Figura 4-2.

Notice that the two stock item components on our browser, each with slightly different content, are based on what we provided. If you click the button in the first stock widget, you will see that the method in the `AppComponent` gets called and the `console.log` is triggered.

Tenga en cuenta que los dos componentes de artículos en stock en nuestro navegador, cada uno con contenido ligeramente diferente, se basan en lo que proporcionamos. Si hace clic en el botón en el primer widget de acciones, verá que se llama al método en `AppComponent` y se activa `console.log`.

Thus, users of the component now have the capability to change part of the UI of the component as they see fit. We can even access

functionality from the parent component as well, which makes it truly flexible. It is also possible to project multiple different sections and content into our child component. While the official Angular documentation is spare on this topic, there is a [great article](#) that can give you more insight on content projection.

Por lo tanto, los usuarios del componente ahora tienen la capacidad de cambiar parte de la interfaz de usuario del componente como mejor les parezca. Incluso podemos acceder a la funcionalidad desde el componente principal, lo que lo hace realmente flexible. También es posible proyectar múltiples secciones y contenidos diferentes en nuestro componente secundario. Si bien la documentación oficial de Angular es abundante sobre este tema, hay un excelente artículo que puede brindarle más información sobre la proyección de contenido.

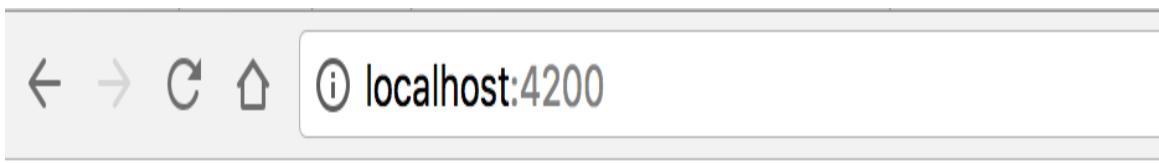


Figure 4-2. Angular app with view projection

Figura 4-2. Aplicación angular con proyección de vista.

Conclusion

Conclusión

In this chapter, we went into a lot more depth on components, and saw some of the more commonly used attributes when creating components. We took a detailed look at the Component decorator,

talking about attributes like `template` versus `templateUrl`, styles, and also covered at a high level how Angular's change detection works and how we can override it.

En este capítulo, profundizamos mucho más en los componentes y vimos algunos de los atributos más utilizados al crear componentes. Analizamos detalladamente el decorador `Component`, hablamos de atributos como `template` versus `templateUrl`, estilos y también cubrimos en un alto nivel cómo funciona la detección de cambios de Angular y cómo podemos anularla. .

We then covered the lifecycle of a component, as well as the hooks that Angular provides for us to hook on to and react to some of these lifecycle events. Finally, we covered projection in components and how we can make some truly powerful components that allow the user of the component to decide parts of the UI.

Luego cubrimos el ciclo de vida de un componente, así como los enlaces que Angular nos proporciona para conectarnos y reaccionar ante algunos de estos eventos del ciclo de vida. Finalmente, cubrimos la proyección en componentes y cómo podemos crear algunos componentes realmente poderosos que permitan al usuario del componente decidir partes de la interfaz de usuario.

In the next chapter, we will do a quick detour to understand unit testing of components, and see how we can test both the logic that drives the component as well as the view that gets rendered.

En el próximo capítulo, haremos un breve desvío para comprender las pruebas unitarias de componentes y veremos cómo podemos probar tanto la lógica que impulsa el componente como la vista que se representa.

Exercise

Ejercicio

For our third exercise, we can build on top of the previous exercise (*chapter3/exercise*) by including concepts from this chapter:

Para nuestro tercer ejercicio, podemos aprovechar el ejercicio anterior (capítulo 3/ejercicio) incluyendo conceptos de este capítulo:

1. Create a `ProductListComponent`. Initialize an array of products there, instead of initializing a single product in the `ProductComponent`. Change its template to use `NgFor` to create a `ProductItemComponent` for each product.

Crea un `ProductListComponent`. Inicialice una serie de productos allí, en lugar de inicializar un solo producto en `ProductComponent`. Cambie su plantilla para usar `NgFor` para crear un `ProductItemComponent` para cada producto.

2. Use inline templates and styles on the `ProductListComponent`. Generate it using the Angular CLI with that setting rather than generating it and changing it manually.

Utilice plantillas y estilos en línea en `ProductListComponent`. Generarlo usando Angular CLI con esa configuración en lugar de generarlo y cambiarlo manualmente.

3. Change the `ProductItemComponent` to take the product as an input.

Cambie el `ProductItemComponent` para tomar el producto como entrada.

4. Move the increment/decrement logic from the `ProductItem` to the `ProductListComponent`. Use an index or product ID to find the product and change its quantity.

Mueva la lógica de incremento/decremento de `ProductItem` a `ProductListComponent`. Utilice un índice o ID de producto para encontrar el producto y cambiar su cantidad.

5. Move the `ProductItemComponent` to be optimal and move from the default `ChangeDetectionStrategy` to an `OnPush` `ChangeDetectionStrategy`.

Mueva el `ProductItemComponent` para que sea óptimo y pase del `ChangeDetectionStrategy` predeterminado a un `OnPush` `ChangeDetectionStrategy`.

All of this can be accomplished using concepts covered in this chapter. You can check out the finished solution in [*chapter4/exercise/ecommerce*](#).

Todo esto se puede lograr utilizando los conceptos tratados en este capítulo. Puede consultar la solución terminada en el capítulo 4/ejercicio/ecommerce.

Chapter 5. Testing Angular Components

Capítulo 5. Prueba de componentes angulares

In the chapters so far, we focused on writing our Angular application, and dealt with how to use the Angular CLI, how to create components, some common built-in Angular directives, and the like.

En los capítulos hasta ahora, nos centramos en escribir nuestra aplicación Angular y abordamos cómo usar la CLI de Angular, cómo crear componentes, algunas directivas Angular integradas comunes y cosas por el estilo.

In this chapter, we will take a detour to learn how we can write unit tests for the components we have written so far. To do this, we will first get a sense of the unit testing setup for Angular, the various frameworks and libraries we use to accomplish this, and walk step by step through writing unit tests for a component.

En este capítulo, nos desviaremos para aprender cómo podemos escribir pruebas unitarias para los componentes que hemos escrito hasta ahora. Para hacer esto, primero tendremos una idea de la configuración de las pruebas unitarias para Angular, los diversos marcos y bibliotecas que usamos para lograr esto, y caminaremos paso a paso a través de la escritura de pruebas unitarias para un componente.

Why Unit Test?

¿Por qué realizar pruebas unitarias?

But before we go into all that, let's quickly talk about unit tests and why they are important. Unit tests are called that because they test an individual unit within your application. Given a very large application and its many pieces, it can become very tricky to test each and every flow through the application. This is why we break up the testing to check each individual component, such that we can be assured that when they are hooked up together, they will work correctly under all cases without necessarily testing each and every flow.

Pero antes de entrar en todo eso, hablemos rápidamente sobre las pruebas unitarias y por qué son importantes. Las pruebas unitarias se llaman así porque prueban una unidad individual dentro de su aplicación. Dada una aplicación muy grande y sus muchas piezas, puede resultar muy complicado probar todos y cada uno de los flujos a través de la aplicación. Es por eso que dividimos las pruebas para verificar cada componente individual, de modo que podamos estar seguros de que cuando estén conectados, funcionarán correctamente en todos los casos sin necesariamente probar todos y cada uno de los flujos.

This is easily demonstrated with a simple example. Assume we have a very simple application with three different parts. Each part itself might have five different flows through it, giving us a total of $5 * 5 * 5 = 125$ flows through the entire application. If we focused only on end-to-end or overall testing, we would have to test 125 flows (or somewhere approaching that number) to get a reasonable assurance on the quality of the application.

Esto se demuestra fácilmente con un ejemplo sencillo. Supongamos que tenemos una aplicación muy sencilla con tres partes

diferenciadas. Cada parte en sí puede tener cinco flujos diferentes, lo que nos da un total de $5 * 5 * 5 = 125$ flujos en toda la aplicación. Si nos centráramos únicamente en las pruebas de un extremo a otro o en general, tendríamos que probar 125 flujos (o algún lugar cercano a ese número) para obtener una garantía razonable sobre la calidad de la aplicación.

Now on the other hand, if we tested each part in isolation, assuming that the other parts were reasonably tested and functional, we would have to write 5 tests for each part, which gives us a total of about 15 tests. Add another 10 to 20 end-to-end tests to make sure the parts are hooked up correctly, and you can have reasonable (not 100% of course) confidence in the overall quality of your application.

Ahora, por otro lado, si probáramos cada parte de forma aislada, asumiendo que las otras partes fueron razonablemente probadas y funcionales, tendríamos que escribir 5 pruebas para cada parte, lo que nos da un total de aproximadamente 15 pruebas. Agregue otras 10 a 20 pruebas de extremo a extremo para asegurarse de que las piezas estén conectadas correctamente y podrá tener una confianza razonable (no 100%, por supuesto) en la calidad general de su aplicación.

That said, unit tests are not just for overall quality. There are a few other good reasons why we write unit tests, some being:

Dicho esto, las pruebas unitarias no son sólo para la calidad general. Hay algunas otras buenas razones por las que escribimos pruebas unitarias, algunas de las cuales son:

- It is an assertion that what you have written actually does what it is meant to. Without unit tests, there is no way to prove that your code performs correctly.

Es una afirmación de que lo que usted ha escrito realmente hace lo que debe hacer. Sin pruebas unitarias, no hay forma de demostrar que su código funciona correctamente.

- It guards your code against future breakages, aka regressions. You write your code today with certain assumptions. You or someone else tomorrow may not remember or know them and may unwittingly change something that breaks your underlying assumption. Unit tests are guards that make sure your code remains future-proof.

Protege su código contra futuras roturas, también conocidas como regresiones. Escribe tu código hoy con ciertas suposiciones. Es posible que mañana usted u otra persona no los recuerde o no los conozca y, sin darse cuenta, puede cambiar algo que rompa su suposición subyacente. Las pruebas unitarias son protecciones que garantizan que su código esté preparado para el futuro.

- Unit tests are great documentation for your code. Comments have an annoying tendency of becoming obsolete, as people tend to forget to update them. Unit tests, on the other hand, will break if you forget to update your tests as you make changes to your code.

Las pruebas unitarias son una excelente documentación para su código. Los comentarios tienen una molesta tendencia a volverse obsoletos, ya que la gente tiende a olvidarse de actualizarlos. Las pruebas unitarias, por otro lado, se romperán si olvida actualizar sus pruebas mientras realiza cambios en su código.

- Unit tests are a great view on how testable and modular your design is. If tests are hard to read or write, it usually is a signal that there might be design flaws or issues in the underlying code.

Las pruebas unitarias son una excelente vista de cuán comprobable y modular es su diseño. Si las pruebas son difíciles de leer o escribir, generalmente es una señal de que puede haber fallas de diseño o problemas en el código subyacente.

That said, people have different associations when they hear the term “unit test.” The prototypical definition of a unit test is a test of a component, with all dependencies completely mocked out. You are testing only the code you have written and nothing else.

Dicho esto, las personas tienen diferentes asociaciones cuando escuchan el término "prueba unitaria". La definición prototípica de prueba unitaria es una prueba de un componente, con todas las dependencias completamente eliminadas. Estás probando sólo el código que has escrito y nada más.

Of course, with frameworks like Angular, sometimes unit tests are useful, and other times you actually want a little bit of integration involved. You want to test how your component behaves rather than the class that defines the component. Angular gives you the capability to write both kinds of tests.

Por supuesto, con marcos como Angular, a veces las pruebas unitarias son útiles y otras veces realmente quieres un poco de integración. Quiere probar cómo se comporta su componente en lugar de la clase que define el componente. Angular le brinda la capacidad de escribir ambos tipos de pruebas.

Testing and Angular

Pruebas y angulares

Before we go deep into how we write tests for Angular components, let's take a look at the various frameworks and libraries we will use to write and run our Angular tests. Each of these can be used in other non-Angular projects as well, or can be swapped out for something comparable if you have a preference:

Antes de profundizar en cómo escribimos pruebas para componentes Angular, echemos un vistazo a los diversos marcos y bibliotecas que

usaremos para escribir y ejecutar nuestras pruebas Angular. Cada uno de estos también se puede usar en otros proyectos que no sean Angular, o se puede cambiar por algo comparable si lo prefiere:

Jasmine

Jazmín

Jasmine is a test framework that is oriented toward writing specifications rather than traditional unit tests. It is what is referred to as a behavior-driven development (BDD) framework. It is a standalone framework that can be used to write tests or specifications for any code, not just Angular. The major difference from traditional unit testing frameworks and something like Jasmine is that Jasmine is more oriented to reading like plain English, so instead of writing a test, you write a specification. A specification is a series of commands, and expectations on what should have happened as a result of these commands.

Jasmine es un marco de prueba que está orientado a escribir especificaciones en lugar de pruebas unitarias tradicionales. Es lo que se conoce como marco de desarrollo impulsado por el comportamiento (BDD). Es un marco independiente que se puede utilizar para escribir pruebas o especificaciones para cualquier código, no solo Angular. La principal diferencia con los marcos de pruebas unitarias tradicionales y algo como Jasmine es que Jasmine está más orientada a leer como en inglés simple, por lo que en lugar de escribir una prueba, escribe una especificación. Una especificación es una serie de comandos y expectativas sobre lo que debería haber sucedido como resultado de estos comandos.

Karma

Karma

If Jasmine is the test-writing framework, then Karma is the test-running framework. Karma's sole task is to take any kind of test,

and run it across a suite of real browsers and report the results back. It is highly tuned toward development workflow, as it is heavily oriented toward rapid execution and reporting. It is possible to have Karma run the tests every time you hit save, giving you real-time feedback on whether tests are still passing as you write your code.

Si Jasmine es el marco de redacción de pruebas, entonces Karma es el marco de ejecución de pruebas. La única tarea de Karma es realizar cualquier tipo de prueba, ejecutarla en un conjunto de navegadores reales e informar los resultados. Está muy orientado al flujo de trabajo de desarrollo, ya que está muy orientado a la ejecución y generación de informes rápidos. Es posible hacer que Karma ejecute las pruebas cada vez que presionas guardar, brindándote retroalimentación en tiempo real sobre si las pruebas aún se están pasando mientras escribes tu código.

Angular testing utilities

Utilidades de prueba angulares

Angular provides a suite of various functions and utilities that make testing Angular-specific functionality easier. These are common tasks that you might need to perform in any test, such as initializing modules, components, working with services and routes, and the like. We will touch upon the relevant ones as we work our way through Angular, but in case you want to see all the utility functions upfront, you can check out [the documentation](#).

Angular proporciona un conjunto de diversas funciones y utilidades que facilitan las pruebas de la funcionalidad específica de Angular. Estas son tareas comunes que es posible que deba realizar en cualquier prueba, como inicializar módulos, componentes, trabajar con servicios y rutas, etc. Tocaremos las relevantes a medida que avancemos en Angular, pero en caso de

que desee ver todas las funciones de la utilidad por adelantado, puede consultar la documentación.

Protractor

Transportador

This framework is not relevant with regards to this chapter and unit testing, but for the sake of completeness, we will quickly mention it. Protractor is a framework that is built to write and run end-to-end tests. While the tests we write in this chapter will instantiate various classes and test functionality, it is useful to also test from the perspective of an end user. This would involve opening the browser, clicking, and interacting with the application. Protractor supports this capability of running the real application and simulating actions and verifying behavior, thus completing the circle of testing.

Este marco no es relevante con respecto a este capítulo y las pruebas unitarias, pero en aras de la exhaustividad, lo mencionaremos rápidamente. Protractor es un marco diseñado para escribir y ejecutar pruebas de un extremo a otro. Si bien las pruebas que escribimos en este capítulo crearán instancias de varias clases y probarán funcionalidades, también es útil realizar pruebas desde la perspectiva de un usuario final. Esto implicaría abrir el navegador, hacer clic e interactuar con la aplicación. Protractor admite esta capacidad de ejecutar la aplicación real y simular acciones y verificar el comportamiento, completando así el círculo de pruebas.

The Test Setup

La configuración de la prueba

With that background, let's write our first unit test. Since we generated our applications so far using the Angular CLI, we have the basic infrastructure already set up for us. In fact, every time we generate a component using the Angular CLI, it also generates a skeleton spec for us to write our test code in.

Con esos antecedentes, escribamos nuestra primera prueba unitaria. Dado que hasta ahora generamos nuestras aplicaciones utilizando Angular CLI, ya tenemos la infraestructura básica configurada para nosotros. De hecho, cada vez que generamos un componente usando Angular CLI, también genera un esqueleto spec para que escribamos nuestro código de prueba.

For the purpose of understanding how the testing infrastructure is set up, we'll walk through the major files one by one. The entire finished code for this chapter, including all the tests, is available in the GitHub repository in the *chapter5/component-spec* folder.

Para comprender cómo está configurada la infraestructura de prueba, revisaremos los archivos principales uno por uno. El código completo para este capítulo, incluidas todas las pruebas, está disponible en el repositorio de GitHub en la carpeta capítulo5/component-spec.

Karma Config

Configuración de karma

The first file of interest is the configuration file for how Karma should find and execute files. The pregenerated *karma.conf.js* is in the main application folder, and looks like the following:

El primer archivo de interés es el archivo de configuración sobre cómo Karma debería encontrar y ejecutar archivos. El karma.conf.js pregenerado se encuentra en la carpeta principal de la aplicación y tiene un aspecto similar al siguiente:

```
// Karma configuration file, see link for more information
// https://karma-runner.github.io/1.0/config/configuration-file.html

module.exports = function (config) {
  config.set({
    basePath: '',
    frameworks: ['jasmine', '@angular/cli'],
    plugins: [
      require('karma-jasmine'),
      require('karma-chrome-launcher'),
      require('karma-jasmine-html-reporter'),
      require('karma-coverage-istanbul-reporter'),
      require('@angular/cli/plugins/karma')
    ],
    client: {
      clearContext: false // leave Jasmine Spec Runner output visible in browser
    },
    coverageIstanbulReporter: {
      reports: [ 'html', 'lcovonly' ],
      fixWebpackSourcePaths: true
    },
    angularCli: {
      environment: 'dev'
    },
    reporters: ['progress', 'kjhtml'],
    port: 9876,
    colors: true,
    LogLevel: config.LOG_INFO,
    autoWatch: true,
    browsers: ['Chrome'],
    singleRun: false
  });
};
```

The Karma configuration is responsible for identifying the various plug-ins needed for Karma to run (which includes an Angular CLI-specific plug-in), the files it needs to watch or execute, and then some Karma-specific configuration including coverage reporting

(coverageIstanbulReporter), which port it needs to run on (port), which browsers to run it on (browsers), whether it should rerun every time the file changes (autoWatch), and which level of logs it needs to capture.

La configuración de Karma es responsable de identificar los diversos complementos necesarios para que Karma se ejecute (que incluye un complemento específico de Angular CLI), los archivos que necesita monitorear o ejecutar y luego alguna configuración específica de Karma que incluye informes de cobertura (coverageIstanbulReporter), en qué puerto debe ejecutarse (port), en qué navegadores ejecutarlo (browsers), si debe volver a ejecutarse cada vez que el archivo cambie (autoWatch) y qué nivel de registros necesita capturar.

test.ts

prueba.ts

The *test.ts* file is the main entry point for our testing, and responsible for loading all our components, the related specifications, and the testing framework and utilities needed to run them:

El archivo *test.ts* es el punto de entrada principal para nuestras pruebas y es responsable de cargar todos nuestros componentes, las especificaciones relacionadas y el marco de pruebas y las utilidades necesarias para ejecutarlos:

```
// This file is required by karma.conf.js and loads recursively all the .spec
// and framework files

import 'zone.js/dist/zone-testing';
import { TestBed } from '@angular/core/testing';
import {
  BrowserDynamicTestingModule,
  platformBrowserDynamicTesting
} from '@angular/platform-browser-dynamic/testing';
```

```
declare const require: any;

// First, initialize the Angular testing environment.
getTestBed().initTestEnvironment(
  BrowserDynamicTestingModule,
  platformBrowserDynamicTesting()
);
// Then we find all the tests.
const context = require.context('./', true, /\.spec\.ts$/);
// And load the modules.
context.keys().map(context);
```

The *test.ts* file is basically responsible for loading a series of files for the testing framework, and then initializing the Angular testing environment. Then, it looks for specifications (files ending with *.spec.ts*) recursively in all the folders from the current directory (which is the *src* folder). It then loads all the relevant modules for them and starts executing Karma.

El archivo *test.ts* es básicamente responsable de cargar una serie de archivos para el marco de prueba y luego inicializar el entorno de prueba Angular. Luego, busca especificaciones (archivos que terminan en *.spec.ts*) de forma recursiva en todas las carpetas del directorio actual (que es la carpeta *src*). Luego carga todos los módulos relevantes para ellos y comienza a ejecutar Karma.

This file is the reason why we don't have to manually list all the specification files in *karma.conf.js*, as it recursively loads them.

Este archivo es la razón por la que no tenemos que enumerar manualmente todos los archivos de especificaciones en *karma.conf.js*, ya que los carga de forma recursiva.

Writing Unit Tests

Escritura de pruebas unitarias

With these two files in place, we can now focus on writing our unit test. To get familiar with Jasmine, we will first start with writing what we call an "isolated unit test."

Con estos dos archivos en su lugar, ahora podemos concentrarnos en escribir nuestra prueba unitaria. Para familiarizarnos con Jasmine, primero comenzaremos escribiendo lo que llamamos una "prueba unitaria aislada".

An Isolated Unit Test

Una prueba unitaria aislada

An isolated unit test in Angular terminology is a vanilla JavaScript unit test. This has nothing to do with Angular, but just instantiates classes and methods and executes them. This is sufficient for a large number of classes, as they will mostly be doing simple data manipulation and execution.

Una prueba unitaria aislada en terminología angular es una prueba unitaria de JavaScript básica. Esto no tiene nada que ver con Angular, solo crea instancias de clases y métodos y los ejecuta. Esto es suficiente para una gran cantidad de clases, ya que en su mayoría realizarán manipulación y ejecución de datos simples.

We will start with writing a very simple isolated unit test for the AppComponent using the example we were working on from [Chapter 4](#). The base codebase to start with can be found in the GitHub repository in the *chapter4/component-output* folder.

Comenzaremos escribiendo una prueba unitaria aislada muy simple para AppComponent usando el ejemplo en el que estábamos

trabajando en el Capítulo 4. El código base para comenzar se puede encontrar en el repositorio de GitHub en la carpeta capítulo4/component-output. .

The very first thing we will do is create (if it doesn't already exist) a file right beside *app.component.ts* in the *src/app* folder called *app.component.spec.ts*. If it already exists and you haven't deleted it yet, you can go ahead and clear the content, as we will write it from scratch so that we are aware of all the intricacies involved in the test.

Lo primero que haremos será crear (si aún no existe) un archivo justo al lado de *app.component.ts* en la carpeta *src/app* llamado *app.component.spec.ts*. Si ya existe y aún no lo ha eliminado, puede continuar y borrar el contenido, ya que lo escribiremos desde cero para que estemos al tanto de todas las complejidades involucradas en la prueba.

The first two tests that we will write will focus on *AppComponent* as a class, and focus on its initialization and how a stock's favorite state is toggled. In these tests, we will not focus on any Angular-specific functionality and see how to test *AppComponent* in isolation:

Las dos primeras pruebas que escribiremos se centrarán en *AppComponent* como clase y se centrarán en su inicialización y en cómo se alterna el estado favorito de una acción. En estas pruebas, no nos centraremos en ninguna funcionalidad específica de Angular y veremos cómo probar *AppComponent* de forma aislada:

```
import { AppComponent } from './app.component';      ①
import { Stock } from 'app/model/stock';

describe('AppComponent', () => {                  ②

  it('should have stock instantiated on ngInit', () => { ③
    const appComponent = new AppComponent();            ④
    expect(appComponent.stock).toBeUndefined();        ⑤
    appComponent.ngOnInit();
    expect(appComponent.stock).toEqual(
```

```

    new Stock('Test Stock Company', 'TSC', 85, 80));      ⑥
  });

it('should have toggle stock favorite', () => {
  const appComponent = new AppComponent();
  appComponent.ngOnInit();
  expect(appComponent.stock.favorite).toBeFalsy();
  appComponent.onToggleFavorite(new Stock('Test', 'TEST', 54, 55));
  expect(appComponent.stock.favorite).toBeTruthy();
  appComponent.onToggleFavorite(new Stock('Test', 'TEST', 54, 55));
  expect(appComponent.stock.favorite).toBeFalsy();
});
});

```

- ① Importando las **relevantes dependencias** para nuestras pruebas.
- ② Questa importación de `app.module.ts` es el principal `AppComponent`
- ③ La primera estructura, `describe`, es la que define el bloqie `it`
- ④ Instantiating the `AppComponent`
- ⑤ Una `expectation` o `assertion` o `statement` que matchea el `should be` comportamiento.
- ⑥ Nuestra `expectation` final establece lo que debe haberla acción

Our isolated unit tests read just like plain old JavaScript, with Jasmine syntax thrown in. We first import our relevant class and interfaces to be able to use them in the specification. Then we define our first describe block, which is Jasmine's way of encapsulating a set of tests as one suite. Describe blocks can be nested any number deep, which we will use in the feature to create separate describe blocks for Angular-aware tests versus isolated unit tests.

Nuestras pruebas unitarias aisladas se leen como el antiguo JavaScript, con la sintaxis Jasmine incluida. Primero importamos nuestra clase e interfaces relevantes para poder usarlas en la especificación. Luego definimos nuestro primer bloque de

descripción, que es la forma que tiene Jasmine de encapsular un conjunto de pruebas como un solo conjunto. Los bloques de descripción se pueden anidar con cualquier número de profundidad, lo que usaremos en la función para crear bloques de descripción separados para pruebas con reconocimiento de Angular frente a pruebas unitarias aisladas.

We then write our first test block, which uses Jasmine's `it` to define a specification block. Within this, we define and instantiate our `AppComponent` instance, and then write an expectation that, by default, the `stock` instance of the `AppComponent` is `undefined`. We then call the `ngOnInit` method of the `AppComponent` manually, which creates a `stock` instance for us. We then write another expectation to make sure this value is created as we expect it to be.

Luego escribimos nuestro primer bloque de prueba, que utiliza `it` de Jasmine para definir un bloque de especificación. Dentro de esto, definimos y creamos una instancia de nuestra instancia `AppComponent` y luego escribimos una expectativa de que, de forma predeterminada, la instancia `stock` de `AppComponent` sea `undefined`. Luego llamamos al método `ngOnInit` de `AppComponent` manualmente, lo que crea una instancia de `stock` para nosotros. Luego escribimos otra expectativa para asegurarnos de que este valor se cree como esperamos que sea.

Note that this behavior mirrors how the `AppComponent` behaves. When an instance of the `AppComponent` is created, we just have a definition for the `stock` object, but with no initial value. Thus, our initial expectation for the `stock` instance in the test is that it should be `undefined`. Then, we trigger the `ngOnInit` method, which ends up creating a `stock` instance with some values. We then assert in our test that the instance that gets created in the `AppComponent` actually has the values we want.

Tenga en cuenta que este comportamiento refleja cómo se comporta `AppComponent`. Cuando se crea una instancia de `AppComponent`, solo

tenemos una definición para el objeto stock, pero sin ningún valor inicial. Por lo tanto, nuestra expectativa inicial para la instancia stock en la prueba es que debería ser undefined. Luego, activamos el método `ngOnInit`, que termina creando una instancia stock con algunos valores. Luego afirmamos en nuestra prueba que la instancia que se crea en AppComponent en realidad tiene los valores que queremos.

TIP CONSEJO

Note that in an isolated unit test, Angular lifecycle methods are not called automatically, which is why we manually trigger `ngOnInit` ourselves in the test. This gives us the flexibility that we might want to test some other function and avoid the `ngOnInit`, which might make expensive or complex server calls.

Tenga en cuenta que en una prueba unitaria aislada, los métodos del ciclo de vida de Angular no se llaman automáticamente, por lo que activamos manualmente `ngOnInit` en la prueba. Esto nos da la flexibilidad de que podríamos querer probar alguna otra función y evitar el `ngOnInit`, que podría generar llamadas al servidor costosas o complejas.

Similarly, we write the second test that evaluates the `onToggleFavorite` method on the `AppComponent` class. We pass a random value to it, as the value is not used in the class except to log it. We use a different expectation, `toBeFalsy` and `toBeTruthy`, instead of the `toEqual`s we used in the previous test. These are all in-built matchers that Jasmine provides for us to use in our specifications. You can see the whole list of matchers that Jasmine provides out of the box in the [official documentation](#).

De manera similar, escribimos la segunda prueba que evalúa el método `onToggleFavorite` en la clase `AppComponent`. Le pasamos un valor aleatorio, ya que el valor no se usa en la clase excepto para registrarla. Usamos una expectativa diferente, `toBeFalsy` y `toBeTruthy`, en lugar del `toEqual`s que usamos en la prueba

anterior. Todos estos son comparadores integrados que Jasmine nos proporciona para usar en nuestras especificaciones. Puede ver la lista completa de coincidencias que Jasmine proporciona de forma inmediata en la documentación oficial.

Running the Tests

Ejecutando las pruebas

Running the actual tests, if you are using the Angular CLI as we are doing in this book, is actually very simple. Simply execute

Ejecutar las pruebas reales, si está utilizando Angular CLI como lo hacemos en este libro, es realmente muy simple. Simplemente ejecute

```
ng test
```

from the command line in the root folder to execute the tests. This will:

desde la línea de comando en la carpeta raíz para ejecutar las pruebas. Esta voluntad:

1. Pick up the configuration from the *karma.conf.js* Karma configuration file

Elija la configuración del archivo de configuración *karma.conf.js* Karma

2. Load all the relevant tests and files as per the *test.ts* file

Cargue todas las pruebas y archivos relevantes según el archivo *test.ts*

3. Capture the default browser (which is Chrome in our case)

Capture el navegador predeterminado (que es Chrome en nuestro caso)

4. Execute the tests and report the results in the terminal

Ejecutar las pruebas y reportar los resultados en la terminal.

5. Keep a watch on files to continue executing on change

Vigile los archivos para continuar ejecutando los cambios

You should see Karma capturing Chrome, and spawn up a browser that looks like **Figure 5-1**.

Deberías ver a Karma capturando Chrome y generar un navegador como el de la Figura 5-1.

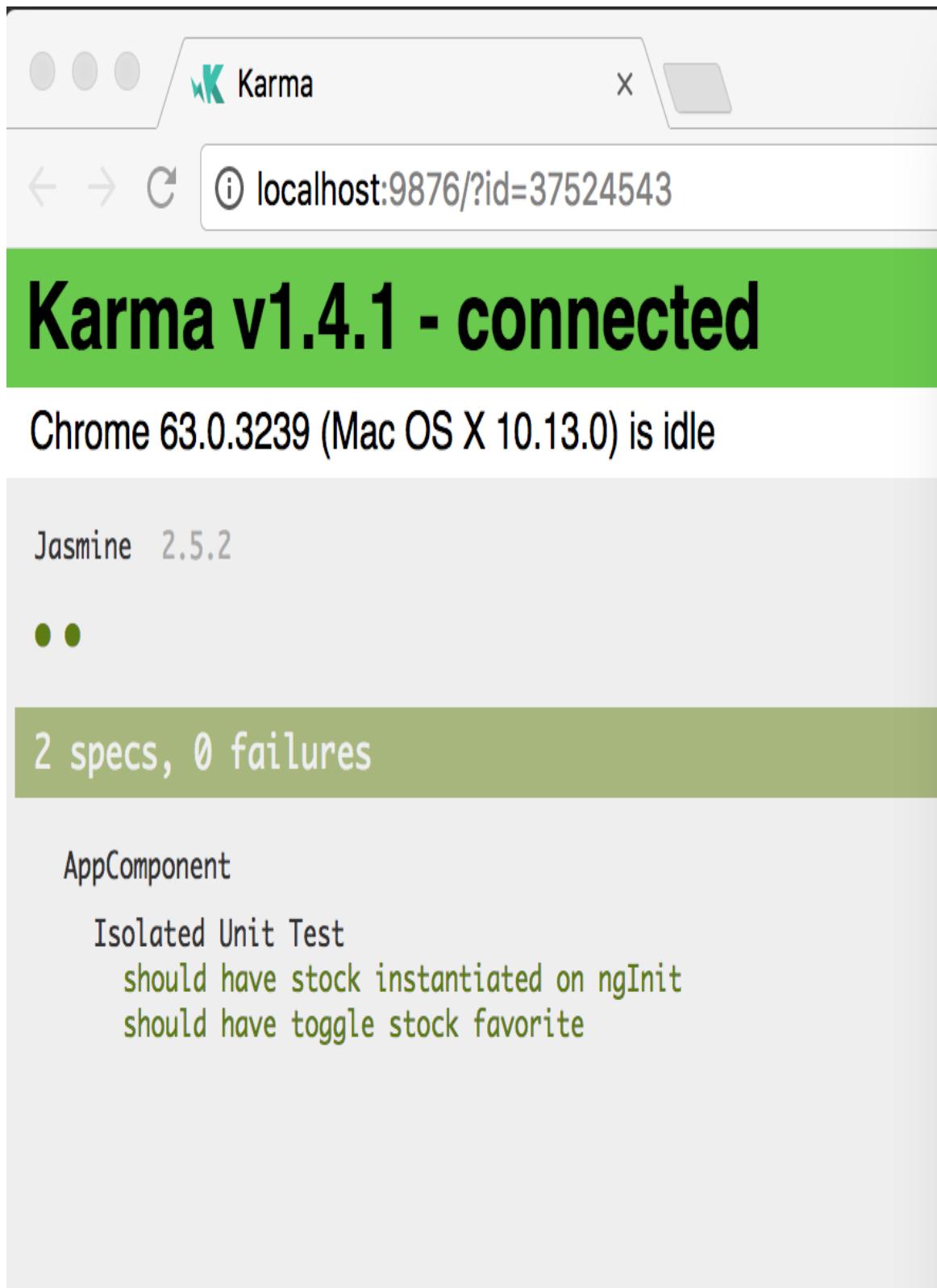


Figure 5-1. Angular tests running via Karma in Chrome

Figura 5-1. Pruebas angulares ejecutándose a través de Karma en Chrome

In the terminal, once you run the `ng test` command, you should see output similar to [Figure 5-2](#).

En la terminal, una vez que ejecute el comando `ng test`, debería ver un resultado similar a la Figura 5-2.

```
1. Google Chrome He
X Google Chrome He ⌘1

$ ng test
10% building modules 1/1 modules 0 active26 10 2017 12:10:47.740:WARN [karma]: No ca
ptured browser, open http://localhost:9876/
26 10 2017 12:10:47.754:INFO [karma]: Karma v1.4.1 server started at http://0.0.0.0:9
876/
26 10 2017 12:10:47.755:INFO [launcher]: Launching browser Chrome with unlimited conc
urrency
26 10 2017 12:10:47.763:INFO [launcher]: Starting browser Chrome
26 10 2017 12:10:56.139:WARN [karma]: No captured browser, open http://localhost:9876
/
26 10 2017 12:10:56.429:INFO [Chrome 63.0.3239 (Mac OS X 10.13.0)]: Connected on sock
et _J6aYQ9lcqd5nIqQAAA with id 37524543
Chrome 63.0.3239 (Mac OS X 10.13.0): Executed 0 of 2 SUCCESS (0 secs / 0 secs)
LOG: 'Favorite for stock ', Stock{name: 'Test', code: 'TEST', price: 54, previousPrice: 55
, favorite: false}, ' was triggered'
Chrome 63.0.3239 (Mac OS X 10.13.0): Executed 0 of 2 SUCCESS (0 secs / 0 secs)
LOG: 'Favorite for stock ', Stock{name: 'Test', code: 'TEST', price: 54, previousPrice: 55
, favorite: false}, ' was triggered'
LOG: 'Favorite for stock ', Stock{name: 'Test', code: 'TEST', price: 54, previousPrice: 55
, favorite: false}, ' was triggered'
Chrome 63.0.3239 (Mac OS X 10.13.0): Executed 0 of 2 SUCCESS (0 secs / 0 secs)
LOG: 'Favorite for stock ', Stock{name: 'Test', code: 'TEST', price: 54, previousPrice: 55
, favorite: false}, ' was triggered'
Chrome 63.0.3239 (Mac OS X 10.13.0): Executed 1 of 2 SUCCESS (0 secs / 0.004 sec
Chrome 63.0.3239 (Mac OS X 10.13.0): Executed 2 of 2 SUCCESS (0 secs / 0.005 sec
Chrome 63.0.3239 (Mac OS X 10.13.0): Executed 2 of 2 SUCCESS (0.009 secs / 0.005 secs)
```

Figure 5-2. Angular test output in the terminal

Figura 5-2. Salida de prueba angular en el terminal

Once this completes, you should see (in both your Karma captured browser and the terminal) a result of having run two tests with both passing successfully.

Una vez que esto se complete, debería ver (tanto en su navegador capturado Karma como en la terminal) el resultado de haber ejecutado dos pruebas y ambas pasaron con éxito.

Writing an Angular-Aware Unit Test

Escribir una prueba unitaria con reconocimiento angular

The next thing we want to learn is how to write a test that is Angular-aware and goes through the Angular lifecycle. In such a test, rather than just instantiating the component class, we want to instantiate a component just the way Angular does, including the HTML for the component.

Lo siguiente que queremos aprender es cómo escribir una prueba que sea compatible con Angular y pase por el ciclo de vida de Angular. En dicha prueba, en lugar de simplemente crear una instancia de la clase de componente, queremos crear una instancia de un componente tal como lo hace Angular, incluido el HTML del componente.

We will write a test for the `StockItemComponent`, in which we will make sure that given the right stock input, the template gets rendered with the correct bindings. Furthermore, we want to make sure the input and output bindings are connected and triggered correctly.

Escribiremos una prueba para `StockItemComponent`, en la que nos aseguraremos de que, dada la entrada de stock correcta, la plantilla

se represente con los enlaces correctos. Además, queremos asegurarnos de que los enlaces de entrada y salida estén conectados y activados correctamente.

Let's create a *stock-item.component.spec.ts* file as a sibling to *stock-item.component.ts*. Again, if this file already exists, replace it with the following contents:

Creamos un archivo stock-item.component.spec.ts como hermano de stock-item.component.ts. Nuevamente, si este archivo ya existe, reemplácelo con el siguiente contenido:

```
import { TestBed, async } from '@angular/core/testing';      ①

import { StockItemComponent } from './stock-item.component';
import { Stock } from '../../model/stock';
import { By } from '@angular/platform-browser';

describe('Stock Item Component', () => {

  let fixture, component;

  beforeEach(async(() => {                                ②
    TestBed.configureTestingModule({                                    ③
      declarations: [
        StockItemComponent
      ],
    }).compileComponents();                                     ④
  }));

  beforeEach(() => {                                      ⑤
    fixture = TestBed.createComponent(StockItemComponent);  ⑥
    component = fixture.componentInstance;                  ⑦
    component.stock = new Stock('Testing Stock', 'TS', 100, 200);
    fixture.detectChanges();                            ⑧
  });

  it('should create stock component and render stock data', () => {
    const nameEl = fixture.debugElement.query(By.css('.name')); ⑨
    expect(nameEl.nativeElement.textContent).toEqual('Testing Stock (TS)');
  ⑩
    const priceEl = fixture.debugElement.query(By.css('.price.negative'));
    expect(priceEl.nativeElement.textContent).toEqual('$ 100');
    const addToFavoriteBtnEl = fixture.debugElement.query(By.css('button'));
    expect(addToFavoriteBtnEl).toBeDefined();
  });
}
```

```

    });

    it('should trigger event emitter on add to favorite', () => {
      let selectedStock: Stock;
      component.toggleFavorite.subscribe((stock: Stock) => selectedStock =
        stock);
      const addToFavoriteBtnEl = fixture.debugElement.query(By.css('button'));

      expect(selectedStock).toBeUndefined();
      addToFavoriteBtnEl.triggerEventHandler('click', null);
      expect(selectedStock).toEqual(component.stock);
    });
  });
}

```

- ① Importando Angular utilities de las utilidades de prueba angular
- ② An async beforeEach, to ensure templates are loaded in the component. Un ejecuado asíncrono, para garantizar que las plantillas se carguen en los componentes
- ③ Using the Angular testing utilities to configure a module for testing. Usando las utilidades de prueba de Angular para configurar un módulo para pruebas
- ④ Compiling all the declared components for add to favorite su uso posterior.
- ⑤ A non-async beforeEach executed only after the previous one finishes. Un ejecuado no asíncrono ejecutado solo después de que finaliza el anterior
- ⑥ Creating a instance of the component under test. Creando una instancia del componente para la prueba
- ⑦ Getting the instance of the component by injecting the dispositivo de prueba
- ⑧ Manually triggering the Angular change detection to update template. Activando manualmente la detección de cambios angulares para actualizar plantillas
- ⑨ Getting a patched template. Obteniendo la plantilla parchada del componente compilado

~~⑩ Verifying that the element has the expected value~~

We have added a lot of code, and highlighted most of the important sections in the code as well. But let's walk through it step by step so that we are aware of the intricacies of using the Angular testing utilities to write tests for components:

Hemos agregado una gran cantidad de código y también resaltamos la mayoría de las secciones importantes del código. Pero veamos paso a paso para que conozcamos las complejidades del uso de las utilidades de prueba de Angular para escribir pruebas para componentes:

1. `@angular/core/testing` provides a set of functionality for testing in Angular. We use the `TestBed`, which is used to create modules and components while testing, and `async`, which is to allow the Jasmine framework to understand Angular's `async` behavior (like loading templates for a component, which in our case is from an external template file). The `async` utility function ensures that we don't start executing the test until these `async` tasks are finished. Notice how we call the `async` function with a function that does all the tasks which might be asynchronous, and then pass this result to the `beforeEach`.

`@angular/core/testing` proporciona un conjunto de funciones para realizar pruebas en Angular. Usamos `TestBed`, que se usa para crear módulos y componentes durante las pruebas, y `async`, que permite que el marco Jasmine comprenda el comportamiento asíncrono de Angular (como cargar plantillas para un componente, que en nuestro caso es de un archivo de plantilla externo). La función de utilidad `async` garantiza que no comencemos a ejecutar la prueba hasta que finalicen estas tareas asíncronas. Observe cómo llamamos a la función `async` con una función que realiza todas las tareas que pueden ser asíncronas y luego pasamos este resultado a `beforeEach`.

2. We use the TestBed to configure a module for our test. Rather than using an existing module, we create a new module with just our component. This gives us absolute control and can also ensure that we don't inadvertently depend on something else other than what we have defined. In this case, we declare the StockItemComponent, and then finally compile the component. This does the task of loading the component, loading all its related templates and styles, and then creating a compiled component for us to use later. This happens asynchronously in some cases, which is why we are wrapping it in an async block.

Usamos el TestBed para configurar un módulo para nuestra prueba. En lugar de utilizar un módulo existente, creamos un nuevo módulo solo con nuestro componente. Esto nos da control absoluto y también puede garantizar que no dependamos inadvertidamente de otra cosa que no sea lo que hemos definido. En este caso, declaramos StockItemComponent y finalmente compilamos el componente. Esto realiza la tarea de cargar el componente, cargar todas sus plantillas y estilos relacionados y luego crear un componente compilado para que lo usemos más adelante. Esto sucede de forma asíncrona en algunos casos, por lo que lo envolvemos en un bloque asíncrono.

3. In the non-async beforeEach, we create a fixture, which is an instance of the component along with its template and everything related to it. Unlike the previous isolated unit test, where we unit tested just the component class, the fixture is the combination of the template, the component class instance, and Angular's magic to combine the two.

En el beforeEach no asíncrono, creamos un dispositivo, que es una instancia del componente junto con su plantilla y todo lo relacionado con él. A diferencia de la prueba unitaria aislada anterior, donde probamos unitariamente solo la clase de componente, el dispositivo es la combinación de la plantilla, la

instancia de clase de componente y la magia de Angular para combinar los dos.

4. From the fixture instance, we can get a handle to the underlying component class instance by using the `componentInstance` variable on it. We can then manipulate the inputs and outputs directly from the component instance.

Desde la instancia `fixture`, podemos obtener un identificador de la instancia de clase del componente subyacente usando la variable `componentInstance` en ella. Luego podemos manipular las entradas y salidas directamente desde la instancia del componente.

5. In this test, we are not working with a higher-level component, so we cannot directly test the `Input` and `Output` bindings. But we can set these values by accessing them directly from the component instance.

En esta prueba, no estamos trabajando con un componente de nivel superior, por lo que no podemos probar directamente los enlaces `Input` y `Output`. Pero podemos establecer estos valores accediendo a ellos directamente desde la instancia del componente.

6. Finally, in the `beforeEach`, we trigger `fixture.detectChanges()`. This is a signal to Angular to trigger its change detection flow, which will look at the values in the component and update the bindings in the corresponding HTML. It is also the trigger to execute the `ngOnInit` for the component the very first time. Without this, the HTML for the component will not have any values. We trigger this after setting the `stock` value so that these values will be propagated to the HTML.

Finalmente, en `beforeEach`, activamos `fixture.detectChanges()`. Esta es una señal para que Angular active su flujo de detección de cambios, que observará los

valores en el componente y actualizará los enlaces en el HTML correspondiente. También es el desencadenante para ejecutar `ngOnInit` para el componente la primera vez. Sin esto, el HTML del componente no tendrá ningún valor. Activamos esto después de configurar el valor `stock` para que estos valores se propaguen al HTML.

7. In the actual tests, we can get access to individual elements from the generated component by using the `fixture.debugElement` and running CSS queries against it. This allows us to check whether the template has the correct bindings and values. Thus, we can avoid writing an end-to-end test for a lot of these basic checks and simply write Angular tests for them.

En las pruebas reales, podemos obtener acceso a elementos individuales del componente generado usando `fixture.debugElement` y ejecutando consultas CSS en él. Esto nos permite comprobar si la plantilla tiene los enlaces y valores correctos. Por lo tanto, podemos evitar escribir pruebas de un extremo a otro para muchas de estas comprobaciones básicas y simplemente escribir pruebas Angular para ellas.

8. In the second test, we can actually see that when we trigger a `click` event on the button element in the template, the corresponding function in the `StockItemComponent` class is triggered, and an event is emitted with the current stock value.

En la segunda prueba, podemos ver que cuando activamos un evento `click` en el elemento de botón en la plantilla, se activa la función correspondiente en la clase `StockItemComponent` y se emite un evento con el valor actual valor de las acciones.

Similarly, we can write a test for most components and test the interaction with the templates and get a good sense for most of the basic functionality and whether it is working as intended.

De manera similar, podemos escribir una prueba para la mayoría de los componentes y probar la interacción con las plantillas y tener una buena idea de la mayor parte de la funcionalidad básica y de si funciona según lo previsto.

Testing Component Interactions

Prueba de interacciones de componentes

The final thing that we'll need to check is whether the AppComponent and the StockItemComponent interact with each other correctly, and whether the stock value is passed from the AppComponent to the StockItemComponent as input correctly. We can also test these functionalities and more using the Angular testing utilities. Let's extend our tests for the AppComponent by adding another sub-suite as follows in the `app.component.spec.ts` file:

Lo último que necesitaremos verificar es si AppComponent y StockItemComponent interactúan entre sí correctamente y si el valor stock se pasa de AppComponent a el StockItemComponent como entrada correctamente. También podemos probar estas funcionalidades y más usando las utilidades de prueba de Angular. Ampliemos nuestras pruebas para AppComponent agregando otro subconjunto de la siguiente manera en el archivo `app.component.spec.ts`:

```
import { TestBed, async } from '@angular/core/testing';

import { AppComponent } from './app.component';
import { StockItemComponent } from 'app/stock/stock-item/stock-item.component';
import { Stock } from 'app/model/stock';
import { By } from '@angular/platform-browser';

describe('AppComponent', () => {

  describe('Simple, No Angular Unit Test', () => {
    /* Move all the previous test code into a
```

```

        child describe block
    */
});

describe('Angular-Aware test', () => {

    let fixture, component;

    beforeEach(async(() => {
        TestBed.configureTestingModule({
            declarations: [
                AppComponent,
                StockItemComponent,
            ],
        }).compileComponents();
    }));

    beforeEach(() => {
        fixture = TestBed.createComponent(AppComponent);
        component = fixture.componentInstance;
        fixture.detectChanges();
    });

    it('should load stock with default values', () => {
        const titleEl = fixture.debugElement.query(By.css('h1'));
        // Trim to avoid HTML whitespaces
        expect(titleEl.nativeElement.textContent.trim())
            .toEqual('Stock Market App');

        // Check for default stock values in template
        const nameEl = fixture.debugElement.query(By.css('.name'));
        expect(nameEl.nativeElement.textContent)
            .toEqual('Test Stock Company (TSC)');
        const priceEl = fixture.debugElement.query(By.css('.price.positive'));
        expect(priceEl.nativeElement.textContent).toEqual('$ 85');
        const addToFavoriteBtnEl = fixture.debugElement.query(By.css('button'));
        expect(addToFavoriteBtnEl).toBeDefined();
    });

    });
});

```

Most of this test will look very similar to the previous test we wrote for the StockItemComponent, but here are a few notable differences:

La mayor parte de esta prueba será muy similar a la prueba anterior que escribimos para StockItemComponent, pero aquí hay algunas diferencias notables:

- When we configure our testing module, this time we have to mention both the AppComponent, which is the component under test, as well as the StockItemComponent. This is because the AppComponent uses the StockItemComponent internally, and without declaring it, Angular would complain about an unknown element.

Cuando configuramos nuestro módulo de prueba, esta vez tenemos que mencionar tanto el AppComponent, que es el componente bajo prueba, como el StockItemComponent. Esto se debe a que AppComponent usa StockItemComponent internamente y, sin declararlo, Angular se quejaría de un elemento desconocido.

- No major changes to either of the `beforeEach` other than this. One thing to note (and it's worth playing around with and trying it out yourself) is that without the `fixture.detectChanges()` in the second `beforeEach`, none of the bindings would happen. You can test this by commenting it out and making sure the test fails.

No hay cambios importantes en ninguno de los `beforeEach` aparte de este. Una cosa a tener en cuenta (y vale la pena jugar y probarlo usted mismo) es que sin el `fixture.detectChanges()` en el segundo `beforeEach`, ninguna de las vinculaciones ocurriría. Puede probar esto comentándolo y asegurándose de que la prueba falle.

- Our test follows a similar pattern like before (in fact, it is pretty much a copy/paste of the previous test for StockItemComponent). The one thing to note is we are trimming the actual text contents retrieved from the DOM to account for

extra whitespaces in the HTML from how we have done our binding. This can be useful sometimes when the HTML is not an exact replica of your bound value.

Nuestra prueba sigue un patrón similar al anterior (de hecho, es más o menos una copia y pega de la prueba anterior para `StockItemComponent`). Lo único a tener en cuenta es que estamos recortando el contenido del texto real recuperado del DOM para tener en cuenta los espacios en blanco adicionales en el HTML debido a la forma en que realizamos nuestro enlace. Esto puede resultar útil a veces cuando el HTML no es una réplica exacta del valor enlazado.

Let's quickly add another test that ensures that the end-to-end flow is both ways. We will add a test that makes sure that clicking Add to Favorite updates both the model value as well as hiding the button in the template. The following code is just the test, ignoring the imports and everything else. This is a part of `app.component.spec.ts` like the previous specification:

Agreguemos rápidamente otra prueba que garantice que el flujo de un extremo a otro sea en ambos sentidos. Agregaremos una prueba que garantice que al hacer clic en Agregar a favoritos se actualice tanto el valor del modelo como que se oculte el botón en la plantilla. El siguiente código es sólo la prueba, ignorando las importaciones y todo lo demás. Esto es parte de `app.component.spec.ts` como la especificación anterior:

```
it('should toggle stock favorite correctly', () => {
  expect(component.stock.favorite).toBeFalsy();
  let addToFavoriteBtnEl = fixture.debugElement.query(By.css('button'));
  expect(addToFavoriteBtnEl).toBeDefined();
  addToFavoriteBtnEl.triggerEventHandler('click', null);

  fixture.detectChanges();
  expect(component.stock.favorite).toBeTruthy();
  addToFavoriteBtnEl = fixture.debugElement.query(By.css('button'));
  expect(addToFavoriteBtnEl).toBeNull();
});
```

First, we check the base default value to ensure that the stock is not favorited by default and that the Add to Favorite button is present. Post that, we trigger the `click` event on the button.

Primero, verificamos el valor predeterminado base para asegurarnos de que la acción no sea favorita de forma predeterminada y que el botón Agregar a favoritos esté presente. Publique eso, activamos el evento `click` en el botón.

At this point, Angular is supposed to kick in, emit an event from the `StockItemComponent` to the `AppComponent`, change the model value, trigger the change detection flow, and update the UI. But in our test, we need to tell Angular to trigger the change detection flow, and hence after we trigger the event, we manually call `fixture.detectChanges()`.

En este punto, se supone que Angular se activa, emite un evento desde el `StockItemComponent` al `AppComponent`, cambia el valor del modelo, activa el flujo de detección de cambios y actualiza la interfaz de usuario. Pero en nuestra prueba, necesitamos decirle a Angular que active el flujo de detección de cambios y, por lo tanto, después de activar el evento, llamamos manualmente a `fixture.detectChanges()`.

After this, we can write our assertions to ensure that the behavior matches our expectations.

Después de esto, podemos escribir nuestras afirmaciones para asegurarnos de que el comportamiento coincide con nuestras expectativas.

WARNING ADVERTENCIA

Forgetting to trigger `fixture.detectChanges()` is one of the most common mistakes when writing Angular tests. By default, it is manual and thus up to the developer to trigger it when events corresponding to user interactions or server responses happen.

Olvidarse de activar `fixture.detectChanges()` es uno de los errores más comunes al escribir pruebas de Angular. De forma predeterminada, es manual y, por lo tanto, corresponde al desarrollador activarlo cuando ocurren eventos correspondientes a interacciones del usuario o respuestas del servidor.

Debugging

Depuración

Oftentimes, there are cases when your unit test is not working as expected, or is slightly off from what you expect. In these cases, one common approach would be to add tons of `console.log` statements to try and figure out when and where things are going wrong.

[Figure 5-3](#) shows how you can see the test results in the Chrome browser captured by Karma.

A menudo, hay casos en los que la prueba unitaria no funciona como se esperaba o está ligeramente fuera de lo esperado. En estos casos, un enfoque común sería agregar toneladas de declaraciones `console.log` para intentar descubrir cuándo y dónde van mal las cosas. La Figura 5-3 muestra cómo puede ver los resultados de la prueba en el navegador Chrome capturado por Karma.

Karma

localhost:9876/?id=34990476

Karma v2.0.0 - connected

DEBUG

Chrome 67.0.3381 (Mac OS X 10.13.3) is idle

Jasmine 2.8.0 finished in 0.324s

• • • • •

6 specs, 0 failures raise exceptions

AppComponent

Isolated Unit Test

- should have stock instantiated on ngInit
- should have toggle stock favorite

Angular test

- should load stock with default values
- should toggle stock favorite correctly

Stock Item Component

- should create stock component and render stock data
- should trigger event emitter on add to favorite

Testing Stock (TS)

\$ 100

Add to Favorite

Figure 5-3. Debugging Karma tests using Chrome

Figura 5-3. Depuración de pruebas de Karma usando Chrome

Karma allows you to debug your tests and application code the same way you would debug in your normal browser. To debug your tests, simply:

Karma le permite depurar sus pruebas y el código de la aplicación de la misma manera que lo haría en su navegador normal. Para depurar sus pruebas, simplemente:

1. Open up the Karma Chrome browser window that Karma spawned on startup. It is the one with the green bar at the top, as shown in [Figure 5-3](#).

Abra la ventana del navegador Karma Chrome que Karma generó al inicio. Es el que tiene la barra verde en la parte superior, como se muestra en la Figura 5-3.

2. Click the DEBUG button on the top right of the Karma browser window. This will open up a new tab in debug mode for you to start debugging.

Haga clic en el botón DEPURAR en la parte superior derecha de la ventana del navegador Karma. Esto abrirá una nueva pestaña en modo de depuración para que pueda comenzar a depurar.

3. Open the Chrome developer tools (Command-Option-I on macOS, Ctrl-Shift-I on Windows) in this tab. Then open up the Sources tab of the developer tools.

Abra las herramientas de desarrollador de Chrome (Comando-Opción-I en macOS, Ctrl-Shift-I en Windows) en esta pestaña. Luego abra la pestaña Fuentes de las herramientas de desarrollador.

4. Select the file you want to debug from the Sources tab. Use Command-P on macOS and Ctrl-P on Windows to start typing

the name of the file and select it in case you can't find it.

Seleccione el archivo que desea depurar en la pestaña Fuentes. Utilice Comando-P en macOS y Ctrl-P en Windows para comenzar a escribir el nombre del archivo y seleccionarlo en caso de que no pueda encontrarlo.

5. Add your breakpoints where you need them by clicking the line number on the left of the source code. You can learn more about how to do this by referring to [the documentation](#).

Agregue sus puntos de interrupción donde los necesite haciendo clic en el número de línea a la izquierda del código fuente. Puede obtener más información sobre cómo hacer esto consultando la documentación.

6. Run the tests by refreshing the browser; your test should stop at your breakpoint.

Ejecute las pruebas actualizando el navegador; su prueba debe detenerse en su punto de interrupción.

Conclusion

Conclusión

In this chapter, we started digging into the testing sections of Angular. We saw how to use Karma and Jasmine to write simple, isolated unit tests for our component classes, without dealing with anything from Angular. We then saw how to use the Angular testing utilities to be able to test the component logic along with the Angular integration. We saw how to use the TestBed to test both individual components as well as cross-component interactions. Finally, we learned how to both run and debug these tests.

En este capítulo, comenzamos a profundizar en las secciones de prueba de Angular. Vimos cómo usar Karma y Jasmine para escribir pruebas unitarias simples y aisladas para nuestras clases de componentes, sin tener que lidiar con nada de Angular. Luego vimos cómo usar las utilidades de prueba de Angular para poder probar la lógica del componente junto con la integración de Angular. Vimos cómo utilizar TestBed para probar tanto componentes individuales como interacciones entre componentes. Finalmente, aprendimos cómo ejecutar y depurar estas pruebas.

In the next chapter, we will start digging into forms to understand how to capture data from users, and how to validate them and process them in a convenient manner.

En el próximo capítulo, comenzaremos a profundizar en los formularios para comprender cómo capturar datos de los usuarios y cómo validarlos y procesarlos de manera conveniente.

Exercise

Ejercicio

Take the finished exercise from the previous chapter (available in [chapter4/exercise](#)). Do the following:

Realice el ejercicio terminado del capítulo anterior (disponible en el capítulo 4/ejercicio). Haz lo siguiente:

1. Add isolated unit tests for the `ProductListComponent` that checks the `onQuantityChange` functionality.

Agregue pruebas unitarias aisladas para `ProductListComponent` que verifican la funcionalidad `onQuantityChange`.
2. Add three Angular tests for the `ProductItemComponent` that test the initial rendering, the `incrementInCart`, and the

`decrementInCart`.

Agregue tres pruebas angulares para `ProductItemComponent` que prueban la representación inicial, el `incrementInCart` y el `decrementInCart`.

3. Add an integrated Angular test for the `ProductListComponent` that checks the integration between the `ProductListComponent` and its children `ProductItemComponent`.

Agregue una prueba Angular integrada para `ProductListComponent` que verifique la integración entre `ProductListComponent` y sus hijos `ProductItemComponent`.

All of this can be accomplished using concepts covered in this chapter. You can check out the finished solution in `chapter5/exercise/ecommerce`.

Todo esto se puede lograr utilizando los conceptos tratados en este capítulo. Puede consultar la solución terminada en el capítulo 5/ejercicio/ecommerce.

Chapter 6. Working with Template-Driven Forms

Capítulo 6. Trabajar con formularios basados en plantillas

In the chapters so far, we have worked on a basic Angular application, with the scaffolding in place and working with very simple components and user interactions. We have learned how to create components and do basic data and event binding, and took a look at the capabilities and extensions that this enables.

En los capítulos hasta ahora, hemos trabajado en una aplicación Angular básica, con el andamiaje implementado y trabajando con componentes e interacciones de usuario muy simples. Hemos aprendido a crear componentes y a vincular datos y eventos básicos, y analizamos las capacidades y extensiones que esto permite.

In this chapter, we will focus purely on how to handle user input, primarily via the use of forms. Forms are the mainstay for many web applications, and are used for everything from logging in and registering to more complex use cases. Creating and using forms is not simply about the template for the form, but also the data binding (both from UI to the code, and vice versa), form state tracking, validation, and error handling. There are two primary mechanisms to work with forms in Angular, and we will explore the

template-driven approach in this chapter, followed by how we can create *reactive forms* in the following chapter.

En este capítulo, nos centraremos exclusivamente en cómo manejar la entrada del usuario, principalmente mediante el uso de formularios. Los formularios son el pilar de muchas aplicaciones web y se utilizan para todo, desde iniciar sesión y registrarse hasta casos de uso más complejos. La creación y el uso de formularios no se trata simplemente de la plantilla del formulario, sino también del enlace de datos (desde la interfaz de usuario al código y viceversa), el seguimiento del estado del formulario, la validación y el manejo de errores. Hay dos mecanismos principales para trabajar con formularios en Angular, y exploraremos el enfoque basado en plantillas en este capítulo, seguido de cómo podemos crear formularios reactivos en el siguiente capítulo.

Template-Driven Forms

Formularios basados en plantillas

Template-driven forms in Angular are an extension of how we have been creating and working with components so far. This approach is also reminiscent of how forms worked in AngularJS (1.x and below), as it uses a similar syntax and methodology. Anyone well-versed with that would have minimal problems adapting to this. In this section, we will create a simple form that allows us to add new stocks, and build our way up from there.

Los formularios basados en plantillas en Angular son una extensión de cómo hemos estado creando y trabajando con componentes hasta ahora. Este enfoque también recuerda cómo funcionaban los formularios en AngularJS (1.xy versiones anteriores), ya que utiliza una sintaxis y una metodología similares. Cualquiera que esté bien versado en eso tendría mínimos problemas para adaptarse a esto.

En esta sección, crearemos un formulario simple que nos permitirá agregar nuevas acciones y avanzar desde allí.

Template-driven forms, as the name suggests, start with the template, and use data binding to get the data to and from your components. It is template-first, and allows you to drive the logic of your application via your template.

Los formularios basados en plantillas, como su nombre indica, comienzan con la plantilla y utilizan el enlace de datos para obtener datos hacia y desde sus componentes. Es una plantilla primero y le permite controlar la lógica de su aplicación a través de su plantilla.

Setting Up Forms

Configurar formularios

Before we dig into the actual form and the template and how it would work, we need to establish some basic groundwork with Angular. At this point, we still don't know how to create multiple routes, so we will simply, for convenience, add a form on our main page itself.

Antes de profundizar en el formulario real, la plantilla y cómo funcionaría, debemos establecer algunas bases básicas con Angular. En este punto, todavía no sabemos cómo crear múltiples rutas, por lo que simplemente, para mayor comodidad, agregaremos un formulario en nuestra página principal.

We can use the same codebase from [Chapter 5](#) as our base to build this in case you are not coding along with the book. You can get that from the *chapter5/component-spec* folder in the GitHub repository.

Podemos usar el mismo código base del Capítulo 5 como nuestra base para construir esto en caso de que no esté codificando junto con el libro. Puede obtenerlo en la carpeta capítulo5/component-spec en el repositorio de GitHub.

The very first thing we will do is extend our `AppModule` to import `FormsModule` into our main `app.module.ts` file if we have not already done so. The `src/app/app.module.ts` file should look like this:

Lo primero que haremos será extender nuestro `AppModule` para importar `FormsModule` a nuestro archivo principal `app.module.ts` si aún no lo hemos hecho. El archivo `src/app/app.module.ts` debería verse así:

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';

import { AppComponent } from './app.component';
import { StockItemComponent } from './stock/stock-item/stock-item.component';

@NgModule({
  declarations: [
    AppComponent,
    StockItemComponent
  ],
  imports: [
    BrowserModule,
    FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

①

① The `FormsModule` is imported in the `imports` section of your `AppModule`

What this does is ensure that all the template-driven form features that are built into Angular are made available to your application. The reason form-specific logic and functionality is in a separate module is for performance and size, so that developers can decide whether or not they need form-specific features in their application.

Lo que esto hace es garantizar que todas las funciones de formulario basadas en plantillas integradas en Angular estén disponibles para su aplicación. La razón por la que la lógica y la funcionalidad

específicas del formulario se encuentran en un módulo separado es por el rendimiento y el tamaño, de modo que los desarrolladores puedan decidir si necesitan o no características específicas del formulario en su aplicación.

The `FormsModule` adds the capability of using `ngModel`, which allows for two-way data binding in Angular. But before we use it, let's explore what other ways we can accomplish this, so that we understand what `ngModel` does for us.

`FormsModule` agrega la capacidad de usar `ngModel`, lo que permite el enlace de datos bidireccional en Angular. Pero antes de usarlo, exploremos qué otras formas podemos lograr esto, para que entendamos qué hace `ngModel` por nosotros.

Alternative to ngModel—Event and Property Binding

Alternativa a ngModel: enlace de eventos y propiedades

At its core, what `ngModel` does for us is two-way data binding. That is, whenever a user enters any text in the UI, it binds that data back to the component. And whenever the value in our component changes (say as a response to a server call, or the initialization logic), then it updates that value in the UI.

En esencia, lo que `ngModel` hace por nosotros es el enlace de datos bidireccional. Es decir, cada vez que un usuario ingresa cualquier texto en la interfaz de usuario, vincula esos datos al componente. Y cada vez que el valor en nuestro componente cambia (por ejemplo, como respuesta a una llamada al servidor o la lógica de inicialización), actualiza ese valor en la interfaz de usuario.

If we break it down, the first one (the user entering a value in the UI) can be handled through event binding. We can listen for the

`input` event, grab the value from the event property `target`, and update the value in the component class.

Si lo desglosamos, el primero (el usuario que ingresa un valor en la interfaz de usuario) se puede manejar mediante enlace de eventos. Podemos escuchar el evento `input`, tomar el valor del objetivo de propiedad del evento y actualizar el valor en la clase de componente.

The second one similarly can be simply handled through data binding, in which we can bind the HTML element property `value` to the variable in our component.

El segundo de manera similar se puede manejar simplemente mediante el enlace de datos, en el que podemos vincular el valor de la propiedad del elemento HTML a la variable de nuestro componente.

Let's first create a new component, called `CreateStockComponent`. We can use the Angular CLI to create it by running:

Primero creamos un nuevo componente, llamado `CreateStockComponent`. Podemos usar Angular CLI para crearlo ejecutando:

```
ng g component stock/create-stock
```

This will create the skeleton component along with its test. Now, let's modify the `app/stock/create-stock/create-stock.component.ts` file as follows:

Esto creará el componente esqueleto junto con su prueba. Ahora, modifiquemos el archivo `app/stock/create-stock/create-stock.component.ts` de la siguiente manera:

```
import { Component, OnInit } from '@angular/core';
import { Stock } from 'app/model/stock';

@Component({
```

```

        selector: 'app-create-stock',
        templateUrl: './create-stock.component.html',
        styleUrls: ['./create-stock.component.css']
    })
export class CreateStockComponent {

    public stock: Stock;
    constructor() {
        this.stock = new Stock('test', '', 0, 0);
    }
}

```

We are using the generated template of the Angular CLI, but we have made two small changes. We added a `stock` member variable with visibility `public`, and then initialized it with some dummy values in the constructor. We have also cleaned up and removed the `ngOnInit` because it was unnecessary.

Estamos utilizando la plantilla generada de Angular CLI, pero hemos realizado dos pequeños cambios. Agregamos una variable miembro `stock` con visibilidad `public` y luego la inicializamos con algunos valores ficticios en el constructor. También limpiamos y eliminamos el `ngOnInit` porque era innecesario.

Next, let's take a look at the template for the `CreateStockComponent`, which is where all the magic is. We will edit the file `app/stock/create-stock/create-stock.component.html` as follows:

A continuación, echemos un vistazo a la plantilla de `CreateStockComponent`, que es donde está toda la magia. Editaremos el archivo `app/stock/create-stock/create-stock.component.html` de la siguiente manera:

```

<h2>Create Stock Form</h2>

<div class="form-group">
    <form>
        <div class="stock-name">
            <input type="text"
                placeholder="Stock Name"
                [value]="stock.name"

```

```

        (input)="stock.name=$event.target.value">
      </div>
    </form>
    <button (click)="stock.name='test'">Reset stock name</button>
  </div>

<h4>Stock Name is {{stock.name}}</h4>

```

We have added a header, followed by a simple form with one `input` element of type `text`. Finally, we have another header that is using interpolation to show the current value of `name` from the `stock` variable in the component class. Now let's take a look at the `input` element in more detail. Other than the `type` and `placeholder`, we have two bindings that we need to talk about:

Hemos agregado un encabezado, seguido de un formulario simple con un elemento `input` de tipo `text`. Finalmente, tenemos otro encabezado que usa interpolación para mostrar el valor actual de `name` de la variable `stock` en la clase de componente. Ahora echemos un vistazo al elemento `input` con más detalle. Además de `type` y `placeholder`, tenemos dos enlaces de los que debemos hablar:

- The `value` binding is telling Angular to update the `value` property of the `input` element using the `stock.name` field in the component class. If and when it changes, Angular will be responsible for updating the property as well.

El enlace `value` le dice a Angular que actualice la propiedad de valor del elemento `input` usando el campo `stock.name` en la clase de componente. Si cambia, Angular también será responsable de actualizar la propiedad.

- The `input` event binding is instructing Angular to update the `value` of `stock.name` with the value from the event. The `$event` in this case is the underlying DOM `InputEvent`, through which we access the target and from it, the changed value.

El enlace del evento `input` le indica a Angular que actualice el `value` de `stock.name` con el valor del evento. El `$event` en este caso es el DOM subyacente `InputEvent`, a través del cual accedemos al objetivo y desde él, al valor cambiado.

Finally, we have a button that on `click` resets the value of `stock.name` to 'test'.

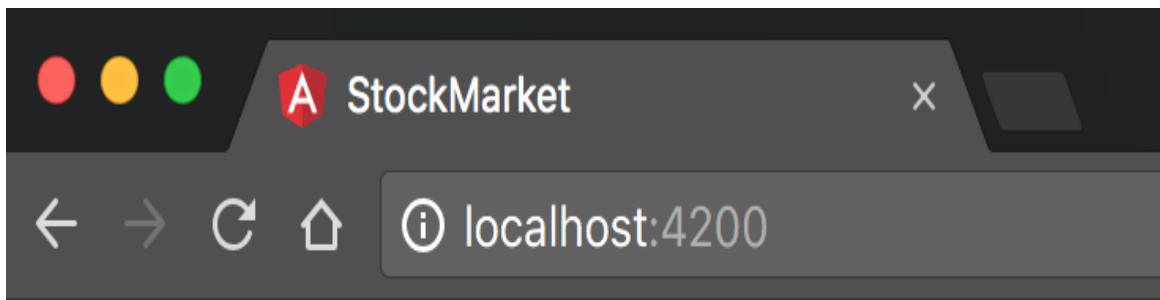
Finalmente, tenemos un botón que en `click` restablece el valor de `stock.name` a 'test'.

When you run this, you should see something like [Figure 6-1](#) on your screen.

Cuando ejecute esto, debería ver algo como la Figura 6-1 en su pantalla.

To really understand how this works, I would recommend removing one binding and keeping just the other, and then reversing it.

Para entender realmente cómo funciona esto, recomendaría quitar una fijación y conservar sólo la otra, y luego invertirla.



Stock Market App

Test Stock Company (TSC)

\$ 85

Add to Favorite

Create Stock Form

Reset stock name

Stock Name is test stock 2

Figure 6-1. Angular template-driven form in action

Figura 6-1. Formulario angular basado en plantilla en acción

When you remove just the `value` binding and run the application (`ng serve`, in case you forgot!), you will see that when you type into the text box, the value in the UI gets updated as you type. But when you click the Reset button, then while the header gets updated, the value in the text box does not. Thus, the component gets updated when the event happens, but because we don't have the Component → UI binding, the UI doesn't update if the component gets updated under the covers.

Cuando eliminas solo el enlace `value` y ejecutas la aplicación (`ng serve`, en caso de que lo hayas olvidado), verás que cuando escribes en el cuadro de texto, el valor en la interfaz de usuario se actualiza a medida que tipo. Pero cuando hace clic en el botón Restablecer, mientras el encabezado se actualiza, el valor en el cuadro de texto no. Por lo tanto, el componente se actualiza cuando ocurre el evento, pero debido a que no tenemos el enlace Componente → UI, la UI no se actualiza si el componente se actualiza en secreto.

Similarly, we can turn off the `input` event binding. In this case, regardless of what we type in the text box, the underlying model does not get updated.

De manera similar, podemos desactivar el enlace del evento `input`. En este caso, independientemente de lo que escribamos en el cuadro de texto, el modelo subyacente no se actualiza.

The combination of the two bindings is what gives us the look and feel of a two-way data binding.

La combinación de los dos enlaces es lo que nos da la apariencia de un enlace de datos bidireccional.

ngModel

ngModelo

Of course, who among us remembers exactly which property is used by each form field? Who can remember what are the various events and where the values would be available? To simplify and abstract this particular information away for ease of use, Angular has the `ngModel` directive.

Por supuesto, ¿quién de nosotros recuerda exactamente qué propiedad utiliza cada campo del formulario? ¿Quién puede recordar cuáles son los distintos eventos y dónde estarían disponibles los valores? Para simplificar y abstraer esta información particular para facilitar su uso, Angular tiene la directiva `ngModel`.

The `ngModel` directive and its special syntax abstracts away the internals of each and every input type from us developers, making it easier for us to quickly develop form-based applications. Let's see how the code gets modified when we use `ngModel` instead of the `input` and `value` binding.

La directiva `ngModel` y su sintaxis especial abstrae los aspectos internos de todos y cada uno de los tipos de entrada de nosotros, los desarrolladores, lo que nos facilita el desarrollo rápido de aplicaciones basadas en formularios. Veamos cómo se modifica el código cuando usamos `ngModel` en lugar del enlace `input` y `value`.

We will only modify the `src/app/stock/create-stock/create-stock.component.html` file as follows; the rest of the code remains the same:

Solo modificaremos el archivo `src/app/stock/create-stock/create-stock.component.html` de la siguiente manera; el resto del código sigue siendo el mismo:

<`h2>Create Stock Form</h2>`

```

<div class="form-group">
  <form>
    <div class="stock-name">
      <input type="text"
        placeholder="Stock Name"
        name="stockName"
        [ngModel]="stock.name"
        (ngModelChange)="stock.name=$event">
    </div>
  </form>
  <button (click)="stock.name='test'">Reset stock name</button>
</div>

<h4>Stock Name is {{stock.name}}</h4>

```

Most of the HTML remains the same except for the following changes:

La mayor parte del HTML sigue siendo el mismo excepto por los siguientes cambios:

- First, we added a name field to the input form element. This is necessary for the ngModel directive to work. If you remove this, you will see errors in the console.

Primero, agregamos un campo de nombre al elemento del formulario de entrada. Esto es necesario para que funcione la directiva ngModel. Si elimina esto, verá errores en la consola.

- We added two bindings. The first one is ngModel data binding. This does the work of the value binding we had previously, but abstracting out which property underneath needs to be bound. It points to the component member variable that it takes the value from.

Agregamos dos encuadernaciones. El primero es el enlace de datos ngModel. Esto hace el trabajo del enlace value que teníamos anteriormente, pero abstrae qué propiedad debajo debe vincularse. Apunta a la variable miembro del componente de la que toma el valor.

- The second binding we added is the `ngModelChange` event binding. In this, we update the underlying component member variable (`stock.name`) with the value of the `$event`, which is the changed value of the text field.

El segundo enlace que agregamos es el enlace de evento `ngModelChange`. En esto, actualizamos la variable miembro del componente subyacente (`stock.name`) con el valor de `$event`, que es el valor modificado del campo de texto.

There is a simpler version of this, which is what we use in most normal cases, which is the `[(ngModel)]` banana-in-a-box syntax, as it is called. This encapsulates both of these statements into a single expression, like so:

Hay una versión más simple de esto, que es la que usamos en la mayoría de los casos normales, que es la sintaxis `[(ngModel)]` banana-in-a-box, como se la llama. Esto encapsula ambas declaraciones en una sola expresión, así:

```
<h2>Create Stock Form</h2>

<div class="form-group">
  <form>
    <div class="stock-name">
      <input type="text"
        placeholder="Stock Name"
        name="stockName"
        [(ngModel)]="stock.name">
    </div>
  </form>
  <button (click)="stock.name='test'">Reset stock name</button>
</div>

<h4>Stock Name is {{stock.name}}</h4>
```

WHY IS IT CALLED BANANA-IN-A-BOX? ¿POR QUÉ SE LLAMA PLÁTANO EN CAJA?

When used together, a common confusion while using the `ngModel` directive could be the order of the types of parentheses, whether it is `[()]` or `([])`. This is why the Angular team came up with a nickname to make it easy to remember. The `()` looks like a banana (yes, it's a stretch, but roll with it!), and it is enclosed in a box `[]`. Hence, the banana-in-a-box, which is `[()]`.

Cuando se usan juntos, una confusión común al usar la directiva `ngModel` podría ser el orden de los tipos de paréntesis, ya sea `[()]` o `([])`. Es por eso que al equipo de Angular se le ocurrió un apodo para que sea fácil de recordar. El `()` parece un plátano (sí, es exagerado, ipero sigue adelante!) y está encerrado en una caja `[]`. De ahí el plátano en caja, que es `[()]`.

In this, we replaced the two individual `ngModel` and `ngModelChange` bindings with the single banana-in-the-box `[(ngModel)]`. The end result will still be the same, as you should see the text value change as you type, as well as the value in the text box get reset when you press the Reset button.

En esto, reemplazamos los dos enlaces individuales `ngModel` y `ngModelChange` con el único plátano en la caja `[(ngModel)]`. El resultado final seguirá siendo el mismo, ya que debería ver que el valor del texto cambia a medida que escribe, así como el valor en el cuadro de texto se restablece cuando presiona el botón Restablecer.

WHEN TO USE THE EXPANDED NGMODEL VARIANT CUÁNDO UTILIZAR LA VARIANTE NGMODEL AMPLIADA

Given that we accomplished the same thing via both the expanded and the collapsed version of the `ngModel` syntax, is there a need for the expanded version at all?

Dado que logramos lo mismo a través de la versión expandida y contraída de la sintaxis `ngModel`, ¿existe alguna necesidad de la versión expandida?

The combined `ngModel` syntax only has the capability to set the data-bound property. If you need to do something more complicated (say convert the text into upper-case before setting the model variable), or set it in a different field itself (a calculated value maybe?), or do multiple things, then you might want to consider the expanded syntax. For all other needs, the banana-in-the-box combined syntax works great!

La sintaxis combinada `ngModel` solo tiene la capacidad de establecer la propiedad enlazada a datos. Si necesita hacer algo más complicado (por ejemplo, convertir el texto a mayúsculas antes de configurar la variable del modelo), o configurarlo en un campo diferente (¿tal vez un valor calculado?), o hacer varias cosas, entonces es posible que desee Considerar la sintaxis ampliada. Para todas las demás necesidades, la sintaxis combinada de banana-in-the-box funciona muy bien!

A Complete Form

Un formulario completo

Now that we have seen a basic form field, let's extend this to a complete form that has different types of controls that are bound to our component along with handling submission of the form. We will continue building on the previous example, the code for which can be found in the `chapter6/simple-ng-model` folder in case you don't have the codebase.

Ahora que hemos visto un campo de formulario básico, ampliémoslo a un formulario completo que tiene diferentes tipos de controles vinculados a nuestro componente junto con el manejo del envío del formulario. Continuaremos basándonos en el ejemplo anterior, cuyo

código se puede encontrar en la carpeta capítulo6/simple-ng-model en caso de que no tenga el código base.

Let's extend our example now to allow users to enter in the other information about a stock, including its code, price, and the exchange it is listed on. In addition, we will have a confirmation checkbox that needs to be checked before the form can be submitted. Finally, we will see how to handle the actual submission event.

Ampliemos nuestro ejemplo ahora para permitir a los usuarios ingresar otra información sobre una acción, incluido su código, precio y la bolsa en la que cotiza. Además, tendremos una casilla de verificación de confirmación que deberá marcarse antes de poder enviar el formulario. Finalmente, veremos cómo manejar el evento de envío real.

First, we'll add an additional field to our stock model (*src/app/model/stock.ts*), which we haven't changed since the initial version, as follows:

Primero, agregaremos un campo adicional a nuestro modelo de stock (*src/app/model/stock.ts*), que no hemos cambiado desde la versión inicial, de la siguiente manera:

```
export class Stock {
  favorite = false;

  constructor(public name: string,
             public code: string,
             public price: number,
             public previousPrice: number,
             public exchange: string) {}

  isPositiveChange(): boolean {
    return this.price >= this.previousPrice;
  }
}
```

Next, let's extend *app/model/create-stock/create-stock.component.ts*:

A continuación, ampliamos *app/model/create-stock/create-stock.component.ts*:

```
import { Component, OnInit } from '@angular/core';
import { Stock } from 'app/model/stock';

@Component({
  selector: 'app-create-stock',
  templateUrl: './create-stock.component.html',
  styleUrls: ['./create-stock.component.css']
})
export class CreateStockComponent {

  public stock: Stock;
  public confirmed = false;
  constructor() {
    this.stock = new Stock('test', '', 0, 0, 'NASDAQ');
  }

  setStockPrice(price) {
    this.stock.price = price;
    this.stock.previousPrice = price;
  }

  createStock() {
    console.log('Creating stock ', this.stock);
  }
}
```

We have added a few new pieces here, in particular:

Hemos agregado algunas piezas nuevas aquí, en particular:

- We added to the stock initialization (an argument 'NASDAQ' in this case).

Agregamos a la inicialización del stock (un argumento 'NASDAQ' en este caso).

- We added `confirmed`, a boolean member variable, to the component class, with a default value of `false`.

Agregamos `confirmed`, una variable miembro boolean, a la clase de componente, con un valor predeterminado de `false`.

- We created a new function `setStockPrice`, which takes a price and then sets both the current and previous price for the stock.

Creamos una nueva función `setStockPrice`, que toma un precio y luego establece el precio actual y el anterior para `stock`.

- Finally, we have a new `createStock` method, which simply logs out the current stock variable to the console.

Finalmente, tenemos un nuevo método `createStock`, que simplemente cierra la sesión de la variable de stock actual en la consola.

Now let's see how these are used as we create and hook up the finished template for the form. We will change `src/app/stock/create-stock/create-stock.component.html` as follows :

Ahora veamos cómo se usan mientras creamos y conectamos la plantilla terminada para el formulario. Cambiaremos `src/app/stock/create-stock/create-stock.component.html` de la siguiente manera:

```
<h2>Create Stock Form</h2>

<div class="form-group">
  <form (ngSubmit)="createStock()"> ①
    <div class="stock-name">
      <input type="text" placeholder="Stock Name"
             name="stockName" [(ngModel)]="stock.name">
    </div>
    <div class="stock-code">
      <input type="text" placeholder="Stock Code"
             name="stockCode" [(ngModel)]="stock.code">
    </div>
    <div class="stock-price">
      <input type="number" placeholder="Stock Price"
             name="stockPrice" [ngModel]="stock.price" ②
             (ngModelChange)="setStockPrice($event)">
    </div>
  </form>
</div>
```

```

<div class="stock-exchange">
  <div>
    <input type="radio" name="stockExchange" ③
      [(ngModel)]="stock.exchange" value="NYSE">NYSE
  </div>
  <div>
    <input type="radio" name="stockExchange"
      [(ngModel)]="stock.exchange" value="NASDAQ">NASDAQ
  </div>
  <div>
    <input type="radio" name="stockExchange"
      [(ngModel)]="stock.exchange" value="OTHER">OTHER
  </div>
</div>
<div class="stock-confirm">
  <input type="checkbox" name="stockConfirm" ④
    [(ngModel)]="confirmed">
    I confirm that the information provided above is accurate!
</div>
<button [disabled]="!confirmed" type="submit">Create</button> ⑤
</form>
</div>

<h4>Stock Name is {{stock | json}}</h4>

```

- ① Manejando el envío de los datos mediante el evento ngSubmit
- ② Expanded version of ngModel to handle setting both current and previous price
- ③ Manejando los cambios en la configuración de ngModel
- ④ Manejando la marca de verificación de ngModel
- ⑤ Desabilitando el formulario si la caja de verificación no está marcada

In the template, we have added a whole new set of form fields, from input boxes for stock code and price, to radio buttons to select the

exchange and a checkbox to confirm if the data is correct. Let's go through this step by step:

En la plantilla, hemos agregado un conjunto completamente nuevo de campos de formulario, desde cuadros de entrada para el código de acciones y el precio, hasta botones de opción para seleccionar el intercambio y una casilla de verificación para confirmar si los datos son correctos. Repasemos esto paso a paso:

1. For the stock code, it remains similar to the stock name which we had. Nothing different other than, of course, the target variable.

Para el código de acciones, sigue siendo similar al nombre de acciones que teníamos. Nada diferente aparte, por supuesto, de la variable objetivo.

2. For the stock price, we are using the expanded version of the `ngModel` syntax. This is because while we want the value in the text box to come from `stock.price`, when the user sets it, we want both the `price` and the `previousPrice` to be set through the method `setStockPrice`.

Para el precio de las acciones, utilizamos la versión ampliada de la sintaxis `ngModel`. Esto se debe a que si bien queremos que el valor en el cuadro de texto provenga de `stock.price`, cuando el usuario lo establece, queremos que tanto el `price` como el `previousPrice` se establezcan mediante el método `setStockPrice`.

3. Next, we have a set of radio buttons that we are using to set the exchange. Each radio button has the same name (which is the standard HTML way of creating a radio group), and are bound to the same model variable (`stock.exchange`) using `ngModel`. The value for each radio button is what defines what the value in the `stock.exchange` variable is, and similarly, the

value in the `stock.exchange` variable defines which of the radio buttons are selected.

A continuación, tenemos un conjunto de botones de opción que estamos usando para configurar el intercambio. Cada botón de opción tiene el mismo nombre (que es la forma HTML estándar de crear un grupo de opción) y está vinculado a la misma variable de modelo (`stock.exchange`) usando `ngModel`. El valor de cada botón de opción es lo que define cuál es el valor en la variable `stock.exchange` y, de manera similar, el valor en la variable `stock.exchange` define cuál de los botones de opción está seleccionado.

4. We then have a checkbox, which is bound to the variable `confirmed` on the component class. Since it is a checkbox, toggling the checkbox on and off sets the value of the variable `confirmed` to `true` and `false`, respectively.

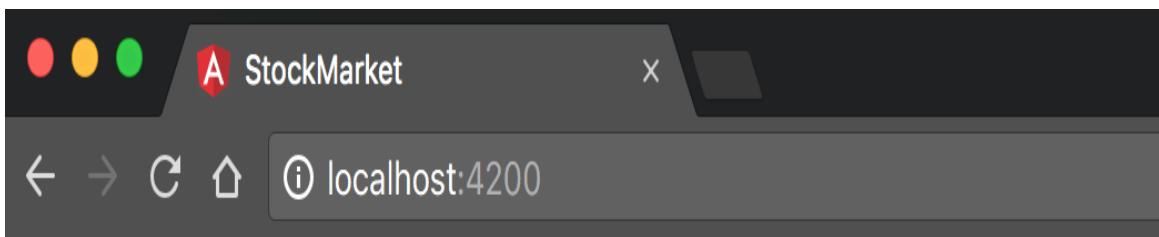
Luego tenemos una casilla de verificación, que está vinculada a la variable `confirmed` en la clase de componente. Dado que es una casilla de verificación, al activarla y desactivarla se establece el valor de la variable `confirmed` en `true` y `false`, respectivamente.

5. Finally, we have a button of type `submit`. This is only enabled when the `confirmed` boolean is set to `true`. On clicking this, it triggers a form `submit`, which is then intercepted by our `ngSubmit` event handler at the form level. This will then trigger the `createStock` method on our component class.

Finalmente, tenemos un button de tipo `submit`. Esto solo se habilita cuando el valor booleano `confirmed` está establecido en `true`. Al hacer clic aquí, se activa un formulario `submit`, que luego es interceptado por nuestro controlador de eventos `ngSubmit` en el nivel del formulario. Esto activará el método `createStock` en nuestra clase de componente.

Once you run this, you should see something like **Figure 6-2** on your screen.

Una vez que ejecute esto, debería ver algo como la Figura 6-2 en su pantalla.



Stock Market App

Test Stock Company (TSC)

\$ 85

Add to Favorite

Create Stock Form

I confirm that the information provided above is accurate!

Stock Name is { "name": "test Stock", "code": "TTT", "price": 244, "previousPrice": 244, "exchange": "NASDAQ", "favorite": false }

Data has been confirmed: true

Figure 6-2. Angular form with data binding

Figura 6-2. Forma angular con enlace de datos.

We thus have a very simple form that handles different kinds of form elements and binds the value to the component, allowing us to get to the user input once he submits the form. To be fair, we haven't handled any errors or requirements on the form, which we will do shortly in the next section.

Por lo tanto, tenemos un formulario muy simple que maneja diferentes tipos de elementos de formulario y vincula el valor al componente, lo que nos permite acceder a la entrada del usuario una vez que envía el formulario. Para ser justos, no hemos solucionado ningún error ni requisito en el formulario, lo cual haremos en breve en la siguiente sección.

WHAT ABOUT SELECT BOXES? ¿QUÉ PASA CON LAS CASILLAS SELECT?

One type of input we didn't cover in the preceding example was a `select` dropdown/combo box. But we could have used it instead of the radio buttons, if we felt like it. How would our example have changed? We currently have the following HTML template code for displaying our radio buttons:

Un tipo de entrada que no cubrimos en el ejemplo anterior fue un cuadro desplegable/combinado `select`. Pero podríamos haberlo usado en lugar de los botones de radio, si hubiéramos querido. ¿Cómo habría cambiado nuestro ejemplo? Actualmente tenemos el siguiente código de plantilla HTML para mostrar nuestros botones de opción:

```
<input type="radio" name="stockExchange"
       [(ngModel)]="stock.exchange" value="NYSE">NYSE
<input type="radio" name="stockExchange"
       [(ngModel)]="stock.exchange" value="NASDAQ">NASDAQ
<input type="radio" name="stockExchange"
       [(ngModel)]="stock.exchange" value="OTHER">OTHER
```

We can replace this with a simple select HTML template as follows:

Podemos reemplazar esto con una plantilla HTML de selección simple de la siguiente manera:

```
<select name="stockExchange" [(ngModel)]="stock.exchange">
  <option value="NYSE">NYSE</option>
  <option value="NASDAQ">NASDAQ</option>
  <option value="OTHER">OTHER</option>
</select>
```

Again, very similar to the other form elements, as long as we have a `name` and the `ngModel` directive, Angular internally handles the data binding for us. We could very easily replace the individual `option` tags with a single `option` tag and use `ngFor` to programmatically generate a list of options as well. In those cases, assuming our component has the following code in it:

Nuevamente, muy similar a los otros elementos del formulario, siempre que tengamos una directiva `name` y `ngModel`, Angular maneja internamente el enlace de datos por nosotros. Podríamos reemplazar muy fácilmente las etiquetas `option` individuales con una sola etiqueta `option` y usar `ngFor` para generar también una lista de opciones mediante programación. En esos casos, suponiendo que nuestro componente tenga el siguiente código:

```
public exchanges = ['NYSE', 'NASDAQ', 'OTHER'];
```

Then we could replace our template code to be as simple as:

Entonces podríamos reemplazar nuestro código de plantilla para que sea tan simple como:

```
<select name="stockExchange" [(ngModel)]="stock.exchange">
  <option *ngFor="let exchange of exchanges"
    [ngValue]="exchange">{{exchange}}</option>
</select>
```

Note the *ngFor loop, and the usage of ngValue to ensure that the current value is used instead of a hardcoded value.

Tenga en cuenta el bucle *ngFor y el uso de ngValue para garantizar que se utilice el valor actual en lugar de un valor codificado.

Control State

Estado de control

Angular form validation for template-driven forms relies and extends the [native form validation from HTML](#). Thus, you can simply use most of the constraints that you already know and love out of the box, and they should work directly and cleanly with an Angular form. That said, Angular does the work of integrating these control states and validations with its own internal model (whether it is ngModel or ngForm), and it is up to us to use this internal model to then show the right kind of message to the user.

La validación de formularios angulares para formularios basados en plantillas se basa y amplía la validación de formularios nativos de HTML. Por lo tanto, puede simplemente usar la mayoría de las restricciones que ya conoce y ama de inmediato, y deberían funcionar directa y limpiamente con una forma angular. Dicho esto, Angular hace el trabajo de integrar estos estados de control y validaciones con su propio modelo interno (ya sea ngModel o

`ngForm`), y depende de nosotros usar este modelo interno para luego mostrar el tipo correcto de mensaje para el usuario.

There are two aspects to this:

Hay dos aspectos de esto:

- The state, which allows us to peek into the state of the form control, on whether the user has visited it, whether the user has changed it, and finally whether it is in a valid state.

El estado, que nos permite echar un vistazo al estado del control del formulario, si el usuario lo ha visitado, si el usuario lo ha cambiado y finalmente si está en un estado válido.

- The validity, which tells us whether a form control is valid or not, and if it is not valid, the underlying reason (or reasons) for which the form element is invalid.

La validez, que nos dice si un control de formulario es válido o no, y si no es válido, la razón (o razones) subyacentes por las cuales el elemento del formulario no es válido.

Let's first see how state is made available to us and how we can use it. The `ngModel` directive changes and adds CSS classes to the element it is on, based on the user's interaction with it. There are three primary modes of interaction that it tracks, and two CSS classes per mode of interaction associated with it. They are:

Primero veamos cómo el estado está disponible para nosotros y cómo podemos usarlo. La directiva `ngModel` cambia y agrega clases CSS al elemento en el que se encuentra, según la interacción del usuario con él. Hay tres modos principales de interacción que rastrea y dos clases de CSS por modo de interacción asociados con él. Ellos son:

Control state Estado de control	CSS class if True Clase CSS si es verdadero	CSS class if False Clase CSS si es falso
Visited	ng-touched	ng-untouched
Visitado	ng-touched	ng-untouched
Changed	ng-dirty	ng-pristine
Cambió	ng-dirty	ng-pristine
Valid	ng-valid	ng-invalid
Válido	ng-valid	ng-invalid

In particular, these states allow you to present different experiences or views to users under various scenarios. We can use the code from the previous section as a base to work off, in case you are not coding along. The starting code can be found in *chapter6/template-driven/full-form*, which holds the completed code from the previous section.

En particular, estos estados le permiten presentar diferentes experiencias o puntos de vista a los usuarios en diversos escenarios. Podemos usar el código de la sección anterior como base para trabajar, en caso de que no esté codificando. El código inicial se puede encontrar en el capítulo 6/template-driven/full-form, que contiene el código completo de la sección anterior.

Now, to use these control state classes, we actually don't need to make any component class code changes. We only need to tweak the CSS a bit and then leverage that in the HTML template for the component.

Ahora, para usar estas clases de estado de control, en realidad no necesitamos realizar ningún cambio en el código de clase de componente. Sólo necesitamos modificar un poco el CSS y luego aprovecharlo en la plantilla HTML para el componente.

Let's first add the following CSS class definitions to the *src/app/stock/create-stock/create-stock.component.css* file:

Primero agreguemos las siguientes definiciones de clases CSS al archivo *src/app/stock/create-stock/create-stock.component.css*:

```
.stock-name .ng-valid,  
.stock-code .ng-pristine,  
.stock-price .ng-untouched {  
  background-color: green;  
}  
  
.stock-name .ng-invalid,  
.stock-code .ng-dirty,  
.stock-price .ng-touched {  
  background-color: pink;  
}
```

Now, we will make minor tweaks (which are more for highlighting different elements rather than any functional changes) to the template HTML in *src/app/stock/create-stock/create-stock.component.html*:

Ahora, haremos ajustes menores (que son más para resaltar diferentes elementos que para cambios funcionales) a la plantilla HTML en *src/app/stock/create-stock/create-stock.component.html*:

```
<h2>Create Stock Form</h2>  
  
<div class="form-group">  
  <form (ngSubmit)="createStock()">  
    <div>  
      The following element changes from green to red  
      when it is invalid  
    </div>  
    <div class="stock-name">  
      <input type="text"  
        placeholder="Stock Name"  
        required  
        name="stockName"  
        [(ngModel)]="stock.name">  
    </div>  
    <div>
```

①

```

The following element changes from green to red
when it has been modified
</div>
<div class="stock-code">
  <input type="text"
    placeholder="Stock Code"
    name="stockCode"
    [(ngModel)]="stock.code">
</div>
<div>
  The following element changes from green to red
  when it is visited by the user, regardless of change
  </div>
<div class="stock-price">
  <input type="number"
    placeholder="Stock Price"
    name="stockPrice"
    [ngModel]="stock.price"
    (ngModelChange)="setStockPrice($event)">
</div>
<div class="stock-exchange">
  <div>
    <select name="stockExchange" [(ngModel)]="stock.exchange">
      <option *ngFor="let exchange of exchanges"
        [ngValue]="exchange">{{exchange}}</option>
    </select>
  </div>
</div>
<div class="stock-confirm">
  <input type="checkbox"
    name="stockConfirm"
    [(ngModel)]="confirmed">
  I confirm that the information provided above is accurate!
</div>
<button [disabled]="!confirmed"
  type="submit">Create</button>
</form>
</div>

<h4>Stock Name is {{stock | json}}</h4>
<h4>Data has been confirmed: {{confirmed}}</h4>

```

① Adding a required attribute to the stock field in the acción

The only change we did to our component HTML is to make the stock name a required field. Angular's Form Module is responsible for

reading it and applying the form control state classes accordingly, without any other work on our side.

El único cambio que hicimos en nuestro componente HTML es convertir el nombre de la acción en un campo obligatorio. El módulo de formulario de Angular es responsable de leerlo y aplicar las clases de estado de control del formulario en consecuencia, sin ningún otro trabajo de nuestra parte.

With these two changes, we now have an application that:

Con estos dos cambios ahora tenemos una aplicación que:

- For the stock name, when it is valid (the class `ng-valid`), we make the background color of the text box green, and when it is invalid (the class `ng-invalid`), we change the background color of the text box to be pink.

Para el nombre de la acción, cuando es válido (la clase `ng-valid`), hacemos que el color de fondo del cuadro de texto sea verde, y cuando no es válido (la clase `ng-invalid`), cambiamos el color de fondo. del cuadro de texto sea rosa.

- For the stock code, when the user has not made any changes to the field (the class `ng-pristine`), we make the background color of the text box green, and whenever the user makes any change (the class `ng-dirty`), regardless of whether he reverts the change or not, we change the background color of the text box to pink.

Para el código de stock, cuando el usuario no ha realizado ningún cambio en el campo (la clase `ng-pristine`), hacemos que el color de fondo del cuadro de texto sea verde, y cada vez que el usuario realiza algún cambio (la clase `ng-dirty`), independientemente de si revierte el cambio o no, cambiamos el color de fondo del cuadro de texto a rosa.

- Finally, for the stock price, the background color remains green as long as the field is not interacted with (the class `ng-untouched`), and changes to pink as soon as the user interacts and then leaves it (the class `ng-touched`). Note that the color remains green while the user is interacting/typing on it, and only changes once he moves away from it.

Finalmente, para el precio de las acciones, el color de fondo permanece verde mientras no se interactúe con el campo (la clase `ng-untouched`), y cambia a rosa tan pronto como el usuario interactúa y luego lo abandona (la clase `ng-touched`). Tenga en cuenta que el color permanece verde mientras el usuario interactúa/escribe en él, y solo cambia una vez que se aleja de él.

You can test it, of course, by:

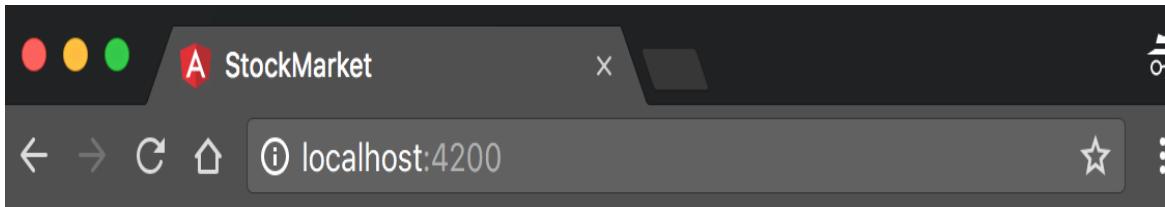
Puedes probarlo, por supuesto, mediante:

1. Removing the default name of the stock (the background color should toggle from green to pink when there is no text in the field)
Eliminar el nombre predeterminado de la acción (el color de fondo debe cambiar de verde a rosa cuando no hay texto en el campo)
2. Typing any character in the stock code field (the background color should toggle from green to pink when you do)
Escribir cualquier carácter en el campo del código de acciones (el color de fondo debe cambiar de verde a rosa cuando lo hagas)
3. Focusing and navigating away from the stock price field (the background color should toggle from green to pink when you do)

Centrarse y alejarse del campo del precio de las acciones (el color de fondo debe cambiar de verde a rosa cuando lo haga)

The end result if you do these actions in order is a screen that looks something like the image in [Figure 6-3](#).

El resultado final, si realiza estas acciones en orden, es una pantalla que se parece a la imagen de la Figura 6-3.



Stock Market App

Test Stock Company (TSC)

\$ 85

Add to Favorite

Create Stock Form

The following element changes from green to red when it is invalid

Stock Name

The following element changes from green to red when it has been modified

d

The following element changes from green to red when it is visited by the user, regardless of change

0

NASDAQ ↴

I confirm that the information provided above is accurate!

Create

Stock Name is { "name": "", "code": "d", "price": 0, "previousPrice": 0, "exchange": "NASDAQ", "favorite": false }

Data has been confirmed: false

Figure 6-3. Angular form with control state bindings

Figura 6-3. Forma angular con enlaces de estado de control.

Control Validity

Validez del control

Next, let's take a look at how we can leverage the HTML form validation to create nice, meaningful errors for our users. We won't get into custom validations just yet in this chapter, but will cover how to use multiple validations on the same elements, and how to display error messages for the same.

A continuación, echemos un vistazo a cómo podemos aprovechar la validación del formulario HTML para crear errores agradables y significativos para nuestros usuarios. No entraremos en validaciones personalizadas todavía en este capítulo, pero cubriremos cómo usar múltiples validaciones en los mismos elementos y cómo mostrar mensajes de error para los mismos.

Internally, Angular has its own set of validators. These mirror the HTML Form validators that are then used to drive a similar behavior in the context of an Angular application. Once you add any validators to your form elements, Angular will take care of running them every time any of the form control changes. This would then be reflected at each control level, as well as at an aggregate level at the form.

Internamente, Angular tiene su propio conjunto de validadores. Estos reflejan los validadores de formulario HTML que luego se utilizan para impulsar un comportamiento similar en el contexto de una aplicación Angular. Una vez que agregue validadores a los elementos de su formulario, Angular se encargará de ejecutarlos cada vez que cambie alguno de los controles del formulario. Esto luego se reflejaría en cada nivel de control, así como a nivel agregado en el formulario.

Before we dig into this, you can take a look at the following sidebar to get a brief understanding of template reference variables, which are a major workhorse in how we execute and work with form control validity.

Antes de profundizar en esto, puede echar un vistazo a la siguiente barra lateral para obtener una breve comprensión de las variables de referencia de la plantilla, que son un caballo de batalla importante en la forma en que ejecutamos y trabajamos con la validez del control de formulario.

TEMPLATE REFERENCE VARIABLES

VARIABLES DE REFERENCIA DE PLANTILLA

A template reference variable in Angular allows us to get a temporary handle on a DOM element, component, or directive directly in the template. It is denoted by a standard syntax in the HTML, which is a prefix of `#`. For example, in the following HTML:

Una variable de referencia de plantilla en Angular nos permite obtener un identificador temporal de un elemento, componente o directiva DOM directamente en la plantilla. Se indica mediante una sintaxis estándar en HTML, que es el prefijo `#`. Por ejemplo, en el siguiente HTML:

```
<input type="text" #myStockField name="stockName">
```

The `#myStockField` is a template reference variable that gives us a reference to the input form field. We can then use this as a variable in any Angular expression, or directly access its value through `myStockField.value` and pass it as an argument to a function.

El `#myStockField` es una variable de referencia de plantilla que nos da una referencia al campo del formulario de entrada. Luego podemos usar esto como una variable en cualquier expresión angular, o acceder directamente a su valor a través de `myStockField.value` y pasarlo como argumento a una función.

In addition to the DOM elements, it can also be used to reference the class/value underlying a directive, which is how we use it in conjunction with forms and form fields.

Además de los elementos DOM, también se puede usar para hacer referencia a la clase/valor subyacente de una directiva, que es como lo usamos junto con formularios y campos de formulario.

By default, when we don't pass it any value, it will always refer to the HTML DOM element.

Por defecto, cuando no le pasamos ningún valor, siempre hará referencia al elemento HTML DOM.

Once you have looked through the sidebar note, we'll dig into how we can accomplish simple error handling via our template-driven forms. In case you are not coding along, you can use the codebase from the previous section (which is located in *chapter6/template-driven/control-state*) as a base.

Una vez que haya leído la nota de la barra lateral, profundizaremos en cómo podemos lograr un manejo simple de errores a través de nuestros formularios basados en plantillas. En caso de que no esté codificando, puede usar el código base de la sección anterior (que se encuentra en el capítulo 6/template-driven/control-state) como base.

We will undo some of the changes from the previous section and now focus purely on form validation. First, let's change our CSS to reflect all invalid form controls with a pink background, by editing the *src/app/stock/create-stock/create-stock.component.css* file as follows:

Desharemos algunos de los cambios de la sección anterior y ahora nos centraremos únicamente en la validación del formulario.

Primero, cambiemos nuestro CSS para reflejar todos los controles de formulario no válidos con un fondo rosa, editando el archivo *src/app/stock/create-stock/create-stock.component.css* de la siguiente manera:

```
.stock-name .ng-valid,  
.stock-code .ng-valid,  
.stock-price .ng-valid {  
  background-color: green;  
}  
  
.stock-name .ng-invalid,  
.stock-code .ng-invalid,
```

```
.stock-price .ng-invalid {  
  background-color: pink;  
}
```

Next, we will make minor changes to the component class (*src/app/stock/create-stock/create-stock.component.ts*), simply to log out different values under certain conditions as follows:

A continuación, haremos cambios menores en la clase de componente (*src/app/stock/create-stock/create-stock.component.ts*), simplemente para cerrar sesión en diferentes valores bajo ciertas condiciones, de la siguiente manera:

```
import { Component, OnInit } from '@angular/core';  
import { Stock } from 'app/model/stock';  
  
@Component({  
  selector: 'app-create-stock',  
  templateUrl: './create-stock.component.html',  
  styleUrls: ['./create-stock.component.css']  
})  
export class CreateStockComponent {  
  
  public stock: Stock;  
  public confirmed = false;  
  public exchanges = ['NYSE', 'NASDAQ', 'OTHER'];  
  constructor() {  
    this.stock = new Stock('', '', 0, 0, 'NASDAQ');  
  }  
  
  setStockPrice(price) {  
    this.stock.price = price;  
    this.stock.previousPrice = price;  
  }  
  
  createStock(stockForm) {  
    console.log('Stock form', stockForm);  
    if (stockForm.valid) {  
      console.log('Creating stock ', this.stock);  
    } else {  
      console.error('Stock form is in an invalid state');  
    }  
  }  
}
```

We have made minor changes only to the `createStock` method:

Hemos realizado cambios menores solo en el método `createStock`:

- We have now started taking the `stockForm` as an argument to the function. This is the `ngForm` object representing the form we have in our template, including all its controls and states. We are also logging this out to the web console.

Ahora hemos comenzado a tomar `stockForm` como argumento para la función. Este es el objeto `ngForm` que representa el formulario que tenemos en nuestra plantilla, incluidos todos sus controles y estados. También cerraremos sesión en la consola web.

- We are checking if the form is valid using this passed-in object, and then only proceeding to create the stock (for what it's worth, logging it out at this point).

Estamos verificando si el formulario es válido usando este objeto pasado y luego solo procedemos a crear el stock (por si sirve de algo, desconéctelo en este momento).

- We have also changed the constructor to initialize the stock with an empty name, unlike before.

También cambiamos el constructor para inicializar el stock con un nombre vacío, a diferencia de antes.

Next, let's see the changes made to the template in `src/app/stock/create-stock/create-stock.component.html`:

A continuación, veamos los cambios realizados en la plantilla en `src/app/stock/create-stock/create-stock.component.html`:

```
<h2>Create Stock Form</h2>

<div class="form-group">
  <form (ngSubmit)="createStock(stockForm)" #stockForm="ngForm">
    <div class="stock-name">
```

```

<input type="text"
       placeholder="Stock Name"
       required
       name="stockName"
       #stockName="ngModel"
       [(ngModel)]="stock.name"> ②
</div>
<div *ngIf="stockName.errors && stockName.errors.required"> ③
    Stock Name is Mandatory
</div>
<div class="stock-code">
    <input type="text"
           placeholder="Stock Code"
           required
           minlength="2"
           name="stockCode"
           #stockCode="ngModel"
           [(ngModel)]="stock.code"> ④
</div>
<div *ngIf="stockCode.dirty && stockCode.invalid"> ⑤
    <div *ngIf="stockCode.errors.required"> ⑥
        Stock Code is Mandatory
    </div>
    <div *ngIf="stockCode.errors.minlength">
        Stock Code must be atleast of length 2
    </div>
</div>
<div class="stock-price">
    <input type="number"
           placeholder="Stock Price"
           name="stockPrice"
           required
           #stockPrice="ngModel"
           [ngModel]="stock.price"
           (ngModelChange)="setStockPrice($event)"> ⑦
</div>
<div *ngIf="stockPrice.dirty && stockPrice.invalid">
    <div *ngIf="stockPrice.errors.required">
        Stock Price is Mandatory
    </div>
</div>
<div class="stock-exchange">
    <div>
        <select name="stockExchange" [(ngModel)]="stock.exchange">
            <option *ngFor="let exchange of exchanges"
                   [ngValue]="exchange">{{exchange}}</option>
        </select>
    </div>
</div>

```

```

</div>
<div class="stock-confirm">
  <input type="checkbox"
    name="stockConfirm"
    required
    [(ngModel)]="confirmed">
    I confirm that the information provided above is accurate!
  </div>
  <button type="submit">Create</button>
</form>
</div>

<h4>Stock Name is {{stock | json}}</h4>
<h4>Data has been confirmed: {{confirmed}}</h4>

```

- ➊ Template reference variable stockForm to work at the form model
Variable de referencia de plantilla stockForm para trabajar en el nivel del modelo de formulario
- ➋ Template reference variable stockName to expose the name
Variable de referencia de plantilla stockName para exponer el nombre ngModel
- ➌ Check on the template reference variable for presence errors and presence
Verifique la variable de referencia de la plantilla para detectar errores y presencia.
- ➍ Template reference variable stockCode to expose the code
Variable de referencia de plantilla stockCode para exponer el código ngModel
- ➎ Check on the stockCode template reference variable for dirty and
Verifique la variable de referencia de la plantilla stockCode para ver si hay controles de formulario sucios o no válidos
- ➏ Check on the stockCode for errors
Verifique el stockCode para ver si hay errores
- ➐ Template reference variable stockPrice to expose the code
Variable de referencia de plantilla stockPrice para exponer el código ngModel

Most of the template remains the same, but there are a few items worth calling out:

La mayor parte de la plantilla sigue siendo la misma, pero hay algunos elementos que vale la pena destacar:

- We added a template reference variable at the form level, and at each control level. The form-level template reference variable (`stockForm`) gets the `NgForm` model object bound to it, which allows us to check on things like form and control validity and values through it.

Agregamos una variable de referencia de plantilla en el nivel de formulario y en cada nivel de control. La variable de referencia de plantilla a nivel de formulario (`stockForm`) obtiene el objeto de modelo `NgForm` vinculado a ella, lo que nos permite verificar cosas como la forma y controlar la validez y los valores a través de ella.

- We added template reference variables (`stockName`, `stockPrice`, `stockCode`) on each of the text boxes, and assigned the `NgModel` model object to it. This allows us to check the form field for all the control states that we were previously using through CSS classes (dirty/pristine, valid/invalid, and touched/untouched), in addition to errors.

Agregamos variables de referencia de plantilla (`stockName`, `stockPrice`, `stockCode`) en cada uno de los cuadros de texto y le asignamos el objeto modelo `NgModel`. Esto nos permite verificar el campo del formulario para todos los estados de control que estábamos usando anteriormente a través de clases CSS (sucio/prístino, válido/inválido y tocado/sin tocar), además de los errores.

- For the first form field (stock name), we added a div to show the error message that it is required, if the error exists.

Para el primer campo del formulario (nombre del stock), agregamos un div para mostrar el mensaje de error que es obligatorio, si el error existe.

- For the second and third field messages, we enclosed the error message within another div, which first checks if the form field is dirty and invalid before looking for a particular error message.

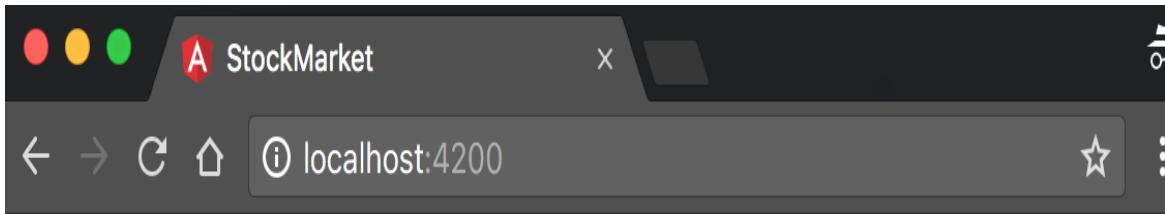
Para los mensajes de segundo y tercer campo, incluimos el mensaje de error dentro de otro div, que primero verifica si el campo del formulario está sucio y no es válido antes de buscar un mensaje de error en particular.

- On form submit we pass the `stockForm` template reference variable, which points to the form model, to the `createStock` method. This is another capability of template reference variables: you can pass them in as arguments to your component class.

Al enviar el formulario, pasamos la variable de referencia de la plantilla `stockForm`, que apunta al modelo del formulario, al método `createStock`. Esta es otra capacidad de las variables de referencia de plantilla: puede pasarlas como argumentos a su clase de componente.

This completed code is available in the *chapter6/template-driven/control-validity* folder. When you run this application, you should see something like **Figure 6-4**.

Este código completo está disponible en la carpeta `capítulo6/template-driven/control-validity`. Cuando ejecute esta aplicación, debería ver algo como la Figura 6-4.



Stock Market App

Test Stock Company (TSC)

\$ 85

Add to Favorite

Create Stock Form

Stock Name

Stock Name is Mandatory

s

Stock Code must be atleast of length 2

Stock Price

Stock Price is Mandatory

NASDAQ

I confirm that the information provided above is accurate!

Create

```
Stock Name is { "name": "", "code": "s", "price": null, "previousPrice": null, "exchange": "NASDAQ", "favorite": false }
```

Data has been confirmed: false

Figure 6-4. Angular form with form validity checks

Figura 6-4. Formulario angular con comprobaciones de validez del formulario.

A few interesting things worth noting:

Algunas cosas interesantes que vale la pena señalar:

- Notice that the error message for the stock name is displayed by default, but the error message that the price and code are required is only displayed after we touch the field. This is the advantage of wrapping the error message under the control state (dirty and invalid). Otherwise, the field is invalid by default (because it is empty).

Observe que el mensaje de error para el nombre de la acción se muestra de forma predeterminada, pero el mensaje de error que indica que el precio y el código son obligatorios solo se muestra después de tocar el campo. Esta es la ventaja de envolver el mensaje de error bajo el estado de control (sucio e inválido). De lo contrario, el campo no es válido de forma predeterminada (porque está vacío).

- Most of the default validators (`required`, `minlength`, `maxlength`) will add a corresponding entry on the errors field on the template reference variable, which you can use to display a relevant message.

La mayoría de los validadores predeterminados (`required`, `minlength`, `maxlength`) agregarán una entrada correspondiente en el campo de errores en la variable de referencia de la plantilla, que puede usar para mostrar un mensaje relevante.

- Note that for the stock price, the `minlength` and `required` validators are not shown simultaneously. This is handled by the Angular built-in validators, but you need to be aware and handle multiple validators having errors simultaneously, and decide and show messages conditionally.

Tenga en cuenta que para el precio de las acciones, los validadores `minlength` y `required` no se muestran simultáneamente. Esto lo manejan los validadores integrados de Angular, pero debe tener en cuenta y manejar varios validadores que tienen errores simultáneamente, y decidir y mostrar mensajes de forma condicional.

- Finally, we pass the `stockForm` template reference variable to the `createStock` method in the component class. This actually gives us access to the individual controls, as well as the value of the form model. We can use this (or similar template reference variables) to, for example, start showing error messages only after the first submit, instead of, say, as soon as the user types.

Finalmente, pasamos la variable de referencia de plantilla `stockForm` al método `createStock` en la clase de componente. En realidad, esto nos da acceso a los controles individuales, así como al valor del modelo de formulario. Podemos usar esto (o variables de referencia de plantilla similares) para, por ejemplo, comenzar a mostrar mensajes de error solo después del primer envío, en lugar de, por ejemplo, tan pronto como el usuario escribe.

Thus, between the template reference variables and the validators, you get utmost control in how and when you want to show your validation messages. You can choose to do it completely in the template, or choose to have them in the template, but decide when and how to show them from your component class, or go to the extreme of creating your validation messages and drive it completely from your component class. It would be up to you which of the approaches you choose, as Angular gives you the tools and the complete flexibility to choose.

Por lo tanto, entre las variables de referencia de la plantilla y los validadores, usted obtiene el máximo control sobre cómo y cuándo desea mostrar sus mensajes de validación. Puede elegir hacerlo

completamente en la plantilla, o elegir tenerlos en la plantilla, pero decidir cuándo y cómo mostrarlos desde su clase de componente, o llegar al extremo de crear sus mensajes de validación y manejarlos completamente desde su componente. Dependrá de usted cuál de los enfoques elija, ya que Angular le brinda las herramientas y la total flexibilidad para elegir.

Working with FormGroups

Trabajar con grupos de formularios

Before we finish with this chapter, we will quickly introduce another method of working with template-driven forms and the `ngModel` directive. So far, we have been declaring a member variable in our component, and then using `ngModel` to bind to it. We can instead let the form model drive the entire form, and copy over or use the values from it once the form has been submitted.

Antes de terminar con este capítulo, presentaremos rápidamente otro método para trabajar con formularios basados en plantillas y la directiva `ngModel`. Hasta ahora, hemos estado declarando una variable miembro en nuestro componente y luego usando `ngModel` para vincularla. En su lugar, podemos dejar que el modelo de formulario controle todo el formulario y copiar o usar sus valores una vez que se haya enviado el formulario.

We can use the same codebase from the previous section (*chapter6/template-driven/control-validity*) as the base to work off of.

Podemos usar la misma base de código de la sección anterior (capítulo 6/basado en plantilla/validez de control) como base para trabajar.

Let's first modify the `CreateStockComponent` class with minor changes to take the model value from the form instead of relying on it being updated via data binding:

Primero modifiquemos la clase `CreateStockComponent` con cambios menores para tomar el valor del modelo del formulario en lugar de depender de que se actualice mediante el enlace de datos:

```
import { Component, OnInit } from '@angular/core';
import { Stock } from 'app/model/stock';

@Component({
  selector: 'app-create-stock',
  templateUrl: './create-stock.component.html',
  styleUrls: ['./create-stock.component.css']
})
export class CreateStockComponent {

  public stock: Stock;
  public confirmed = false;
  public exchanges = ['NYSE', 'NASDAQ', 'OTHER'];
  constructor() {
    this.stock = new Stock('', '', 0, 0, 'NASDAQ');
  }

  createStock(stockForm) {
    console.log('Stock form', stockForm.value);
    if (stockForm.valid) {
      this.stock = stockForm.value.stock;
      console.log('Creating stock ', this.stock);
    } else {
      console.error('Stock form is in an invalid state');
    }
  }
}
```

We have changed the `createStock` method, to copy over the `stock` object from the form `value` field. We have also dropped the `setStockPrice` method as we won't be using it. Now, let's take a look at the changes needed in the template to be able to support this:

Hemos cambiado el método `createStock`, para copiar el objeto `stock` del campo del formulario `value`. También eliminamos el método `setStockPrice` ya que no lo usaremos. Ahora, echemos un vistazo a los cambios necesarios en la plantilla para poder admitir esto:

```
<h2>Create Stock Form</h2>

<div class="form-group">
  <form (ngSubmit)="createStock(stockForm)" #stockForm="ngForm" >
    <div ngModelGroup="stock">
      <div class="stock-name">
        <input type="text"
          placeholder="Stock Name"
          required
          name="name"
          ngModel>
      </div>
      <div class="stock-code">
        <input type="text"
          placeholder="Stock Code"
          required
          minlength="2"
          name="code"
          ngModel>
      </div>
      <div class="stock-price">
        <input type="number"
          placeholder="Stock Price"
          name="price"
          required
          ngModel>
      </div>
      <div class="stock-exchange">
        <div>
          <select name="exchange" ngModel>
            <option *ngFor="let exchange of exchanges"
              [ngValue]="exchange">{{exchange}}</option>
          </select>
        </div>
      </div>
      <button type="submit">Create</button>
    </form>
  </div>
```

```
<h4>Stock Name is {{stock | json}}</h4>
<h4>Data has been confirmed: {{confirmed}}</h4>
```

We have temporarily removed all validators, so as to not distract us from the core change. The major changes are as follows:

Hemos eliminado temporalmente todos los validadores para no distraernos del cambio principal. Los principales cambios son los siguientes:

- We have removed the banana-in-a-box syntax from all the ngModel bindings, and just kept it as an attribute. When we use ngModel like this, Angular uses the name field on the form element as the model name and creates a model object corresponding to it on the form.

Hemos eliminado la sintaxis banana-in-a-box de todos los enlaces ngModel y simplemente la mantuvimos como un atributo. Cuando usamos ngModel de esta manera, Angular usa el campo de nombre en el elemento del formulario como nombre del modelo y crea un objeto de modelo correspondiente en el formulario.

- We have surrounded the form fields with another div, and used an Angular directive called ngModelGroup on it, providing it a name (stock in this case). What this does is group the form elements, thus creating the name, price, code, and exchange fields as models under the common name stock. This is visible in the component when we access this entire set of values through form.value.stock.

Hemos rodeado los campos del formulario con otro div y utilizamos una directiva angular llamada ngModelGroup en él, proporcionándole un nombre (stock en este caso). Lo que esto hace es agrupar los elementos del formulario, creando así los campos de nombre, precio, código e intercambio como modelos

bajo el nombre común stock. Esto es visible en el componente cuando accedemos a todo este conjunto de valores a través de `form.value.stock`.

We can similarly create multiple form groups and use `ngModel` directly, and then finally copy over the entire values to a common field (or not copy it over at all) in our component on form submit. This is another way we can use `ngModel` and template-driven forms in our applications. The finished code for this is available in the `chapter6/template-driven/form-groups/` folder in the GitHub repository.

De manera similar, podemos crear múltiples grupos de formularios y usar `ngModel` directamente, y luego, finalmente, copiar todos los valores a un campo común (o no copiarlos en absoluto) en nuestro componente al enviar el formulario. Esta es otra forma en que podemos usar `ngModel` y formularios basados en plantillas en nuestras aplicaciones. El código terminado para esto está disponible en la carpeta `capítulo6/template-driven/form-groups/` en el repositorio de GitHub.

Conclusion

Conclusión

In this chapter, we started exploring how to handle user input through the use of forms. In particular, we took a deep dive into how to create and work with template-driven forms, and leverage `ngModel` for two-way data binding. We further saw what control states Angular provides out of the box and how to leverage that as well as show and deal with validations and error messages.

En este capítulo, comenzamos a explorar cómo manejar la entrada del usuario mediante el uso de formularios. En particular,

profundizamos en cómo crear y trabajar con formularios basados en plantillas y aprovechar `ngModel` para el enlace de datos bidireccional. Además, vimos qué estados de control proporciona Angular de forma inmediata y cómo aprovecharlos, así como mostrar y manejar validaciones y mensajes de error.

In the next chapter, we will look at a different way of approaching this: reactive forms. We will discuss how to use reactive forms but will first review the ways they differ from template-driven forms and why you might choose one over the other.

En el próximo capítulo veremos una forma diferente de abordar esto: las formas reactivas. Analizaremos cómo utilizar formularios reactivos, pero primero revisaremos en qué se diferencian de los formularios basados en plantillas y por qué podría elegir uno sobre el otro.

Exercise

Ejercicio

Take the finished exercise from the previous chapter (available in `chapter5/exercise/ecommerce`). Do the following:

Realice el ejercicio terminado del capítulo anterior (disponible en el capítulo 5/ejercicio/comercio electrónico). Haz lo siguiente:

1. Create a new component that allows us to add new products.
Crear un nuevo componente que nos permita agregar nuevos productos.
2. Create a form that takes in the product name, price, image URL, and whether or not it is on sale. Try to use the form groups approach rather than two-way binding through `ngModel`.

Cree un formulario que incluya el nombre del producto, el precio, la URL de la imagen y si está en oferta o no. Intente utilizar el enfoque de grupos de formularios en lugar del enlace bidireccional a través de `ngModel`.

3. Make all the fields required, and see if you can add a basic Regex pattern validation for the image URL.

Haga que todos los campos sean obligatorios y vea si puede agregar una validación de patrón Regex básica para la URL de la imagen.

4. Display relevant error messages, but only after the user either edits the field or after the first submit.

Muestra mensajes de error relevantes, pero solo después de que el usuario edite el campo o después del primer envío.

5. Copy over the form and print it to the console after successful submission.

Copie el formulario e imprímalo en la consola después de enviarlo correctamente.

All of this can be accomplished using concepts covered in this chapter. You can check out the finished solution in [*chapter6/exercise/ecommerce*](#).

Todo esto se puede lograr utilizando los conceptos tratados en este capítulo. Puede consultar la solución terminada en el capítulo 6/ejercicio/ecommerce.

Chapter 7. Working with Reactive Forms

Capítulo 7. Trabajar con formas reactivas

In the previous chapter, we started working on our first form-based Angular application. To do this, we explored template-driven forms and how we might build and use them within the context of an Angular application. We saw how to perform data binding, work with different form elements, and also perform validity and show relevant error messages.

En el capítulo anterior, comenzamos a trabajar en nuestra primera aplicación Angular basada en formularios. Para hacer esto, exploramos formularios basados en plantillas y cómo podríamos crearlos y usarlos dentro del contexto de una aplicación Angular. Vimos cómo realizar el enlace de datos, trabajar con diferentes elementos de formulario y también realizar la validez y mostrar mensajes de error relevantes.

In this chapter, we will focus on doing the exact same set of things, though this time, we will use a reactive approach. As mentioned in the previous chapter, Angular allows us to build forms using two approaches: template-driven and reactive. Both these approaches are part of the core `@angular/forms` library, but are part of two different modules, `FormsModule` and `ReactiveFormsModule`, respectively.

En este capítulo, nos centraremos en hacer exactamente el mismo conjunto de cosas, aunque esta vez utilizaremos un enfoque reactivo. Como se mencionó en el capítulo anterior, Angular nos permite crear formularios utilizando dos enfoques: basado en plantillas y reactivo. Ambos enfoques son parte de la biblioteca principal `@angular/forms`, pero son parte de dos módulos diferentes, `FormsModule` y `ReactiveFormsModule`, respectivamente.

Reactive Forms

Formas reactivas

To understand and get into building forms in a reactive manner, it is important to understand what reactive programming is. Reactive programming, to overly simplify it, is the concept of writing a program in a way that it fundamentally deals with and acts on asynchronous data streams. While most programs (especially web apps) do this, reactive programming does this by providing an amazing toolbox of utilities and functions to combine, filter, and merge these various streams and act on them, which is where it gets real fun real fast.

Para comprender y comenzar a crear formularios de manera reactiva, es importante comprender qué es la programación reactiva. La programación reactiva, para simplificarla demasiado, es el concepto de escribir un programa de una manera que fundamentalmente trate y actúe sobre flujos de datos asíncronos. Si bien la mayoría de los programas (especialmente las aplicaciones web) hacen esto, la programación reactiva lo hace proporcionando una sorprendente caja de herramientas de utilidades y funciones para combinar, filtrar y fusionar estas diversas corrientes y actuar en consecuencia, que es donde se vuelve realmente divertido muy rápidamente.

Unlike template-driven forms in Angular, with reactive forms, you define the entire tree of Angular form control objects in your component code, and then bind them to native form control elements in your template. Because the component has access to the form controls as well as the backing data model, it can push data model changes into the form control and vice versa, thus reacting to changes either way.

A diferencia de los formularios basados en plantillas en Angular, con los formularios reactivos, usted define el árbol completo de objetos de control de formulario Angular en el código de su componente y luego los vincula a los elementos de control de formulario nativos en su plantilla. Debido a que el componente tiene acceso a los controles del formulario, así como al modelo de datos de respaldo, puede enviar cambios del modelo de datos al control del formulario y viceversa, reaccionando así a los cambios de cualquier manera.

Understanding the Differences

Comprender las diferencias

Now, with two approaches in front of us (even though we haven't looked at a single line of code of the reactive form approach), the question naturally arises: which one is better? The answer, as you would expect, is that neither is truly "better" than the other. Both have their own advantages and disadvantages.

Ahora, con dos enfoques frente a nosotros (aunque no hemos visto ni una sola línea de código del enfoque de forma reactiva), surge naturalmente la pregunta: ¿cuál es mejor? La respuesta, como era de esperar, es que ninguno es realmente "mejor" que el otro. Ambos tienen sus propias ventajas y desventajas.

When we create forms using the template-driven approach, we declare the form controls in the template, and add directives to it

(like `ngModel`). Then Angular is responsible for creating the form controls through the use of directives.

Cuando creamos formularios utilizando el enfoque basado en plantillas, declaramos los controles del formulario en la plantilla y le agregamos directivas (como `ngModel`). Luego, Angular es responsable de crear los controles del formulario mediante el uso de directivas.

That said, template-driven forms are nice and declarative, and easy to understand. Angular is responsible for the data model sync and pushes data to the model and reads and updates values in the UI via directives like `ngModel`. This also usually means less code in the component class.

Dicho esto, los formularios basados en plantillas son agradables, declarativos y fáciles de entender. Angular es responsable de la sincronización del modelo de datos, envía datos al modelo y lee y actualiza valores en la interfaz de usuario a través de directivas como `ngModel`. Esto también suele significar menos código en la clase de componente.

Reactive forms, on the other hand, are synchronous, and you as a developer have absolute control over how and when the data is synced from the UI to the model and vice versa. Because you create the entire form control tree in the component, you have access to it immediately and don't have to deal with Angular's asynchronous life-cycle. While we haven't encountered this ourselves in any of the examples so far, you will run into this if you try to update the form controls from your component class on initialization, as they might not be immediately available. You can read up on this in the [official Angular docs](#). It also fits in better if the rest of your application is following a reactive style of programming.

Los formularios reactivos, por otro lado, son sincrónicos y usted, como desarrollador, tiene control absoluto sobre cómo y cuándo se sincronizan los datos desde la interfaz de usuario con el modelo y

viceversa. Debido a que crea todo el árbol de control de formulario en el componente, tiene acceso a él de inmediato y no tiene que lidiar con el ciclo de vida asincrónico de Angular. Si bien no hemos encontrado esto en ninguno de los ejemplos hasta ahora, se encontrará con esto si intenta actualizar los controles de formulario de su clase de componente durante la inicialización, ya que es posible que no estén disponibles de inmediato. Puede leer sobre esto en los documentos oficiales de Angular. También encaja mejor si el resto de su aplicación sigue un estilo de programación reactivo.

Using Reactive Forms

Usando formas reactivas

Now that we have briefly compared reactive versus template-driven forms and reviewed the pros and cons of each, let's look at building a reactive form. We will do this step by step, starting with the building block of reactive forms, `Form Controls`, and work our way upwards to the other various components like `Form Groups` and `Form Builders`.

Ahora que hemos comparado brevemente los formularios reactivos con los basados en plantillas y hemos revisado los pros y los contras de cada uno, veamos cómo crear un formulario reactivo. Haremos esto paso a paso, comenzando con el bloque de construcción de formularios reactivos, `Form Controls`, y avanzaremos hacia los otros componentes como `Form Groups` y `Form Builder`.

Form Controls

Controles de formulario

The core of any reactive form is the `FormControl`, which directly represents an individual form element in the template. Thus, any reactive form is nothing but a set of grouped `FormControls`. It is at the `FormControl` level that we also assign initial values and validators (both sync and async). Thus, everything that we did in the template with template-driven forms now happens at a `FormControl` level in the TypeScript code.

El núcleo de cualquier formulario reactivo es `FormControl`, que representa directamente un elemento de formulario individual en la plantilla. Por lo tanto, cualquier forma reactiva no es más que un conjunto de `FormControl` agrupados. Es en el nivel `FormControl` donde también asignamos valores iniciales y validadores (tanto sincronizados como asíncronos). Por lo tanto, todo lo que hicimos en la plantilla con formularios basados en plantilla ahora sucede en un nivel `FormControl` en el código TypeScript.

We can use the same codebase from [Chapter 6](#) as our base to build this in case you are not coding along with the book. You can get that from the `chapter6/template-driven/simple-ng-model` folder in the GitHub repository.

Podemos usar el mismo código base del Capítulo 6 como base para construir esto en caso de que no esté codificando junto con el libro. Puede obtenerlo en la carpeta `capítulo6/template-driven/simple-ng-model` en el repositorio de GitHub.

The very first thing we will do is import `ReactiveFormsModule` into our main `app.module.ts` file. For this example, we can remove the old `FormsModule` from it. The `src/app/app.module.ts` file should look like this:

Lo primero que haremos será importar ReactiveFormsModule a nuestro archivo principal app.module.ts. Para este ejemplo, podemos eliminar el antiguo FormsModule. El archivo src/app/app.module.ts debería verse así:

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { ReactiveFormsModule } from '@angular/forms'; ①

import { AppComponent } from './app.component';
import { StockItemComponent } from './stock/stock-item/stock-item.component';
import { CreateStockComponent }
  from './stock/create-stock/create-stock.component';

@NgModule({
  declarations: [
    AppComponent,
    StockItemComponent,
    CreateStockComponent
  ],
  imports: [
    BrowserModule,
    ReactiveFormsModule, ②
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

① Import the ReactiveFormsModule

② Add the ReactiveFormsModule to the NgModule's imports section

Now given this base setup, which enables reactive form features in our application, let's see how to create a simple form that allows us to take and work with a name. We will first modify the template for our CreateStockComponent by changing the *src/app/stock/create-stock/create-stock.component.html* file as follows:

Ahora, dada esta configuración básica, que habilita funciones de formulario reactivo en nuestra aplicación, veamos cómo crear un formulario simple que nos permita tomar y trabajar con un nombre.

Primero modificaremos la plantilla para nuestro CreateStockComponent cambiando el archivo src/app/stock/create-stock/create-stock.component.html de la siguiente manera:

```
<h2>Create Stock Form</h2>

<div class="form-group">
  <div class="stock-name">
    <input type="text"
      placeholder="Stock Name"
      name="stockName"
      [FormControl]="nameControl"> ①
  </div>
  <button (click)="onSubmit()">Submit</button>
</div>

<p>Form Control value: {{ nameControl.value | json }}</p> ②
<p>Form Control status: {{ nameControl.status | json }}</p>
```

① Usando una forma con el binding FormControl en lugar de ngModel

② Accediendo al valor actual del campo de formulario

The way we created this form is very different than the template-driven approach outlined in the previous chapter. Instead of using ngModel, we are binding the form element to nameControl. We can then derive the current value of the form control via this field, whether it is the value (via nameControl.value) or its status (via nameControl.status, which is always valid for this simple element). Finally, we have a simple button that triggers the onSubmit() method in the component.

La forma en que creamos este formulario es muy diferente al enfoque basado en plantillas descrito en el capítulo anterior. En lugar de usar ngModel, vinculamos el elemento del formulario a nameControl. Luego podemos derivar el valor actual del control de formulario a través de este campo, ya sea el valor (a través de nameControl.value) o su estado (a través de nameControl.status,

que siempre es válido para este elemento simple). Finalmente, tenemos un botón simple que activa el método `onSubmit()` en el componente.

Next, let's take a look at the `src/app/component/stock/create-stock/create-stock.component.ts` file to see what changes we have to make to it to support this form:

A continuación, echemos un vistazo al archivo `src/app/component/stock/create-stock/create-stock.component.ts` para ver qué cambios debemos realizarle para admitir este formulario:

```
import { Component, OnInit } from '@angular/core';
import { FormControl } from '@angular/forms';

@Component({
  selector: 'app-create-stock',
  templateUrl: './create-stock.component.html',
  styleUrls: ['./create-stock.component.css']
})
export class CreateStockComponent {

  public nameControl = new FormControl();
  constructor() {}

  onSubmit() {
    console.log('Name Control Value', this.nameControl.value);
  }
}
```

We have removed all references of the Stock model from this. Instead, we import and create an instance of `FormControl` called `nameControl`. This is the variable we bound to in the template. Then, on the `onSubmit()` call, we simply print the current value of the `nameControl` control. Again, note that unlike traditional non-MVC frameworks, at no point is the control reaching out into the view to get the current value of the element. We rely on the `FormControl` to provide a representative view of the input element, and keep it up to date.

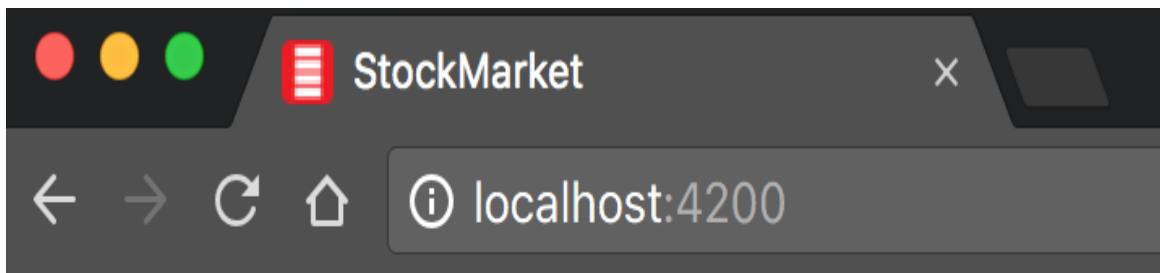
Hemos eliminado todas las referencias del modelo Stock de esto. En su lugar, importamos y creamos una instancia de `FormControl` llamada `nameControl`. Esta es la variable que vinculamos en la plantilla. Luego, en la llamada `onSubmit()`, simplemente imprimimos el `value` actual del control `nameControl`. Nuevamente, tenga en cuenta que, a diferencia de los marcos tradicionales que no son MVC, en ningún momento el control se extiende hacia la vista para obtener el valor actual del elemento. Confiamos en `FormControl` para proporcionar una vista representativa del elemento de entrada y mantenerlo actualizado.

We have used the default `FormControl` constructor, but it can also take the initial value along with a list of validators (both sync and async) as arguments. We will see how to add validators in a bit more detail in "[Form Groups](#)".

Hemos utilizado el constructor predeterminado `FormControl`, pero también puede tomar el valor inicial junto con una lista de validadores (tanto sincronizados como asíncronos) como argumentos. Veremos cómo agregar validadores con un poco más de detalle en "[Grupos de formularios](#)".

When we run this application, we should see something like [Figure 7-1](#), with a simple form field below our stock widgets. If you type into it, you should see the form field below it.

Cuando ejecutamos esta aplicación, deberíamos ver algo como la Figura 7-1, con un campo de formulario simple debajo de nuestros widgets estándar. Si escribe en él, debería ver el campo de formulario debajo.



Stock Market App

Test Stock Company (TSC)

\$ 85

Add to Favorite

Create Stock Form

Form Control value: null

Form Control status: "VALID"

Figure 7-1. Simple Angular reactive form

Figura 7-1. Forma reactiva angular simple

Summing up, it is helpful to think of `FormControl` when we need to track the state and value of any individual form element, like an input box or a checkbox. In the next section, we will see how to build a more complete form using `FormControl` and something called `FormGroup`.

En resumen, es útil pensar en `FormControl` cuando necesitamos rastrear el estado y el valor de cualquier elemento de formulario individual, como un cuadro de entrada o una casilla de verificación. En la siguiente sección, veremos cómo crear un formulario más completo usando `FormControl` y algo llamado `FormGroup`.

The finished example is available in the *chapter7/form-control* folder in the GitHub repository.

El ejemplo terminado está disponible en la carpeta *capítulo7/form-control* en el repositorio de GitHub.

Form Groups

Formar grupos

Usually, when we build any form, it rarely has just one element. We typically have a set of different fields and elements we want to track under one common form or heading. In these cases, the `FormGroup` is useful as a way to group relevant form fields under one group. This gives us the convenience of whether we want to track the form controls individually, or as a group. For example, we can get the entire form value, or check whether the form as a whole is valid (as a result of individual elements and their state).

Por lo general, cuando construimos cualquier formulario, rara vez tiene un solo elemento. Normalmente tenemos un conjunto de campos y elementos diferentes que queremos rastrear bajo un formulario o encabezado común. En estos casos, `FormGroup` es útil

como una forma de agrupar campos de formulario relevantes en un grupo. Esto nos da la conveniencia de si queremos rastrear los controles del formulario individualmente o como grupo. Por ejemplo, podemos obtener el valor completo del formulario o comprobar si el formulario en su conjunto es válido (como resultado de los elementos individuales y su estado).

Let's see how we can extend the example from the previous section to create a comprehensive form for creating a stock using `FormControl` and `FormGroup` instances.

Veamos cómo podemos ampliar el ejemplo de la sección anterior para crear un formulario completo para crear una acción usando las instancias `FormControl` y `FormGroup`.

First, we will see how to modify the `src/app/stock/create-stock/create-stock.component.html` template to ask for all the relevant fields of a stock from the user:

Primero, veremos cómo modificar la plantilla `src/app/stock/create-stock/create-stock.component.html` para solicitar al usuario todos los campos relevantes de una acción:

```
<h2>Create Stock Form</h2>

<div class="form-group">
  <form [formGroup]="stockForm" (ngSubmit)="onSubmit()"> ❶
    <div class="stock-name">
      <input type="text"
        placeholder="Stock Name"
        name="stockName"
        formControlName="name"> ❷
    </div>
    <div class="stock-code">
      <input type="text"
        placeholder="Stock Code"
        formControlName="code">
    </div>
    <div class="stock-price">
      <input type="number"
        placeholder="Stock Price"
        formControlName="price">
    </div>
  </form>
</div>
```

```

    </div>
    <button type="submit">Submit</button>
  </form>
</div>

<p>Form Control value: {{ stockForm.value | json }}</p> ③
<p>Form Control status: {{ stockForm.status | json }}</p>

```

- ① Ahora nos vinculamos con un formGroup instead of a FormControl
- ② Once we use a FormGroup, we use formControlName inside the grupo que usamos un FormGroup, usamos formControlName dentro del grupo
- ③ Change over to printing the form group value instead of the Elemento imprimir el valor del grupo de formulario en lugar del elemento

The major change from the previous example is that we switched over from binding to FormControl to a FormGroup. We do this at the form level. Now within it, for each form element, we mention a formControlName. Each of these will bind to an individual element within the FormGroup. Finally, we display the current value and status of the form similar to how we did for the FormControl.

El cambio más importante con respecto al ejemplo anterior es que cambiamos del enlace a FormControl a FormGroup. Hacemos esto en el nivel form. Ahora dentro de él, para cada elemento del formulario, mencionamos un formControlName. Cada uno de estos se vinculará a un elemento individual dentro de FormGroup. Finalmente, mostramos el value y el status actuales del formulario de forma similar a como lo hicimos para el FormControl.

Also, for easier readability, we have switched from calling the control nameControl, and instead just call it name, code, and price in the component.

Además, para facilitar la lectura, hemos pasado de llamar al control `nameControl` y, en su lugar, simplemente lo llamamos `name`, `code` y `price` en el componente.

Next, let's take a look at how we change the `src/app/stock/create-stock/create-stock.component.ts` component class to get this example to work:

A continuación, veamos cómo cambiamos la clase de componente `src/app/stock/create-stock/create-stock.component.ts` para que este ejemplo funcione:

```
import { Component, OnInit } from '@angular/core';
import { FormControl, FormGroup, Validators } from '@angular/forms';

@Component({
  selector: 'app-create-stock',
  templateUrl: './create-stock.component.html',
  styleUrls: ['./create-stock.component.css']
})
export class CreateStockComponent {

  public stockForm: FormGroup = new FormGroup({
    name: new FormControl(null, Validators.required),
    code: new FormControl(null, [Validators.required,
      Validators.minLength(2)]),
    price: new FormControl(0, [Validators.required, Validators.min(0)])
  });
  constructor() {}

  onSubmit() {
    console.log('Stock Form Value', this.stockForm.value);
  }
}
```

In the component, we now instantiate and expose a `FormGroup` instance, named `stockForm`. This is what we had bound to at the form level in the template. `FormGroup` allows us to instantiate multiple named controls within it, and we have done so to instantiate a form control for the `name`, `code`, and `price`. This time, we are also using the constructor of the `FormControl` to the

maximum extent possible, by adding a default value and validators as required.

En el componente, ahora creamos y exponemos una instancia `FormGroup`, denominada `stockForm`. Esto es a lo que nos habíamos vinculado en el nivel del formulario en la plantilla. `FormGroup` nos permite crear instancias de múltiples controles con nombre dentro de él, y lo hemos hecho para crear una instancia de un control de formulario para el nombre, código y precio. Esta vez, también usamos el constructor de `FormControl` en la máxima medida posible, agregando un valor predeterminado y validadores según sea necesario.

The first argument to the `FormControl` constructor is the default value of the form control. Here, we initialize the default values of the two form controls to be `null` and `0`, respectively.

El primer argumento del constructor `FormControl` es el valor predeterminado del control de formulario. Aquí, inicializamos los valores predeterminados de los dos controles de formulario para que sean `null` y `0`, respectivamente.

The second argument to the `FormControl` constructor can either be a single `Validator`, or an array of `Validators`. There are a set of built-in validators to ensure that the `FormControl` is `require`, or has a minimum value. These validators can either be synchronous (like the ones we have used), or can also be asynchronous (for example, to check whether a username is available in the server). You can check out the built-in validators in the [official Angular docs](#).

El segundo argumento del constructor `FormControl` puede ser un único `Validator` o una matriz de `Validator`. Hay un conjunto de validadores integrados para garantizar que `FormControl` sea `require` o tenga un valor mínimo. Estos validadores pueden ser síncronos (como los que hemos utilizado) o también pueden ser asíncronos (por ejemplo, para comprobar si un nombre de usuario está

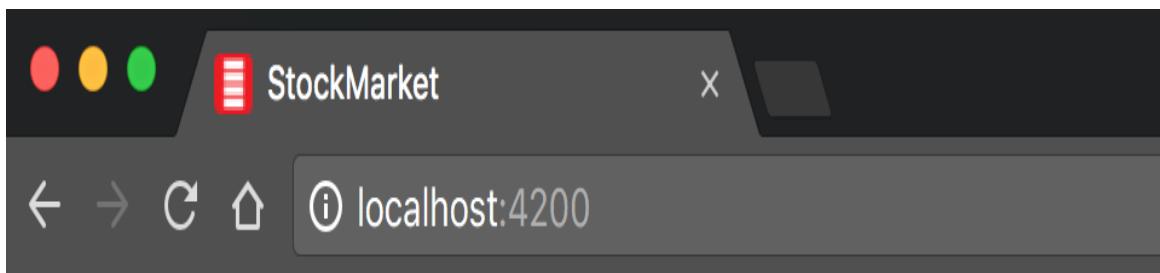
disponible en el servidor). Puede consultar los validadores integrados en los documentos oficiales de Angular.

When the form is submitted (using `ngSubmit` event binding), we then print the entire form group value to the console.

Cuando se envía el formulario (usando el enlace de evento `ngSubmit`), imprimimos el valor completo del grupo de formulario en la consola.

When we run this, and type in some values in the form, we should see the form react and print the current value of the form as well as its validity status in the UI. It would look something like [Figure 7-2](#).

Cuando ejecutamos esto y escribimos algunos valores en el formulario, deberíamos ver que el formulario reacciona e imprime el valor actual del formulario, así como su estado de validez en la interfaz de usuario. Se parecería a la Figura 7-2.



Stock Market App

Test Stock Company (TSC)
\$ 85
[Add to Favorite](#)

Create Stock Form

Name
SSS
Stock Price
<input type="button" value="Submit"/>

Form Control value: { "name": "Name", "code": "SSS", "price": null }

Form Control status: "INVALID"

Figure 7-2. Angular reactive form with form control state

Figura 7-2. Forma reactiva angular con estado de control de forma.

The finished example is available in the `chapter7/form-groups` folder in the GitHub repository.

El ejemplo terminado está disponible en la carpeta `capítulo7/form-groups` en el repositorio de GitHub.

Form Builders

Constructores de formularios

Now, while the `FormGroup` gives us the flexibility to build complex, nested forms (and by the way, you can absolutely nest further form groups within a form group!), its syntax is slightly verbose. And that's why, to replace it, we have a `FormBuilder` in Angular, which makes it slightly nicer to build these rich forms in a cleaner manner.

Ahora, si bien `FormGroup` nos brinda la flexibilidad de crear formularios anidados complejos (y, por cierto, puedes anidar absolutamente más grupos de formularios dentro de un grupo de formularios!), su sintaxis es un poco detallada. Y es por eso que, para reemplazarlo, tenemos un `FormBuilder` en Angular, lo que hace que sea un poco mejor construir estos formularios enriquecidos de una manera más limpia.

The nice thing about the `FormBuilder` is that we don't have to change or even touch our template. The `FormBuilder` fundamentally is syntactic sugar to allow us to quickly create `FormGroup` and `FormControl` elements without manually calling `new` for each one. The reactive form still relies on those elements under the covers for its functioning, and `FormBuilder` does not do away with them.

Lo bueno de `FormBuilder` es que no tenemos que cambiar ni siquiera tocar nuestra plantilla. El `FormBuilder` fundamentalmente es

azúcar sintáctico que nos permite crear rápidamente elementos FormGroup y FormControl sin llamar manualmente a new para cada uno. La forma reactiva todavía depende de esos elementos ocultos para su funcionamiento y FormBuilder no los elimina.

Let's see how we can switch our CreateStockComponent to use FormBuilder. We will change the *src/app/stock/create-stock/create-stock.component.ts* file as follows:

Veamos cómo podemos cambiar nuestro CreateStockComponent para usar FormBuilder. Cambiaremos el archivo *src/app/stock/create-stock/create-stock.component.ts* de la siguiente manera:

```
import { Component, OnInit } from '@angular/core';
import { FormControl, FormGroup } from '@angular/forms';
import { Validators, FormBuilder } from '@angular/forms'; ①

@Component({
  selector: 'app-create-stock',
  templateUrl: './create-stock.component.html',
  styleUrls: ['./create-stock.component.css']
})
export class CreateStockComponent {

  public stockForm: FormGroup; ②
  constructor(private fb: FormBuilder) { ③
    this.createForm();
  }

  createForm() {
    this.stockForm = this.fb.group({ ④
      name: [null, Validators.required], ⑤
      code: [null, [Validators.required, Validators.minLength(2)]],
      price: [0, [Validators.required, Validators.min(0)]]
    });
  }

  onSubmit() {
    console.log('Stock Form Value', this.stockForm.value);
  }
} ①
```

~~Import `FormBuilder` from `com.google.common.forms`~~

- ② ~~We do not initialize the `FormGroup` instance directly anymore~~
- ③ ~~Inject an instance of the `FormBuilder` into the constructor~~
- ④ ~~Create a `FormGroup` using the injected `FormBuilder` instance~~
- ⑤ Initialize the name control with an initial null value and a required ~~Validator~~ el control de nombre con un valor nulo inicial y un validador requerido

The major change from before is how we initialize the `stockForm` `FormGroup` instance. Instead of initializing it in line, with the `FormBuilder`, we first inject an instance of the `FormBuilder` into our constructor. Then, in the constructor itself, we use the `group` method on the `FormBuilder` instance to then create the various form controls.

El cambio principal con respecto a antes es cómo inicializamos la instancia `stockForm` `FormGroup`. En lugar de inicializarlo en línea, con `FormBuilder`, primero inyectamos una instancia de `FormBuilder` en nuestro constructor. Luego, en el constructor mismo, usamos el método `group` en la instancia `FormBuilder` para luego crear los distintos controles de formulario.

Even for creating the form controls, we simply use the `FormBuilder` syntactic sugar. For example, if we just wanted to initialize a text field called `name` with an empty string as the initial value, we could simply have a key called `name`, and pass an empty string as the value for it (that is, `name: ''`). In the preceding code, we actually want to both initialize it with a default value as well as add some validators, so instead of passing it just a value, we pass it an array.

Incluso para crear los controles de formulario, simplemente usamos el azúcar sintáctico `FormBuilder`. Por ejemplo, si solo quisiéramos inicializar un campo de texto llamado `name` con una cadena vacía

como valor inicial, simplemente podríamos tener una clave llamada `name` y pasarle una cadena vacía como valor. (es decir, `name: ''`). En el código anterior, en realidad queremos inicializarlo con un valor predeterminado y agregar algunos validadores, por lo que en lugar de pasarle solo un valor, le pasamos una matriz.

The first value in the array is the default value of the form control (null in the case of `name` and `code`, 0 in the case of `price`). The second value is then either a single validator (like in `name`), or an array of validators (like in `code` and `price`).

El primer valor de la matriz es el valor predeterminado del control de formulario (nulo en el caso de `name` y `code`, 0 en el caso de `price`). El segundo valor es entonces un validador único (como en `name`) o una serie de validadores (como en `code` y `price`).

Fundamentally, nothing else changes. The result of doing this is we get an application that behaves exactly like the version from the previous section, but we are able to hook it up in a more concise and less verbose manner. For any form with more than a few elements, it almost always makes sense to use the `FormBuilder` rather than the `FormGroup` method, as it reduces both the code and makes it a lot more concise and readable.

Básicamente, nada más cambia. El resultado de hacer esto es que obtenemos una aplicación que se comporta exactamente igual a la versión de la sección anterior, pero podemos conectarla de una manera más concisa y menos detallada. Para cualquier formulario con más de unos pocos elementos, casi siempre tiene sentido usar el método `FormBuilder` en lugar del `FormGroup`, ya que reduce el código y lo hace mucho más conciso y legible.

The finished example is available in the `chapter7/form-builder` folder in the GitHub repository.

El ejemplo terminado está disponible en la carpeta `capítulo7/form-builder` en el repositorio de GitHub.

Form Data

Datos del formulario

So far, we have hand-waved over handling the form data. We have simply used the `FormControl` or `FormGroup` and accessed the value from it. In this section, we will go into the data and form model and how reactive forms allow us to deal with it as well as control and form state (like valid, invalid, etc.).

Hasta ahora, hemos dejado de lado el manejo de los datos del formulario. Simplemente hemos usado `FormControl` o `FormGroup` y accedimos al valor desde allí. En esta sección, analizaremos el modelo de datos y formularios y cómo los formularios reactivos nos permiten manejarlos, así como el control y el estado del formulario (como válido, no válido, etc.).

Control State, Validity, and Error Messages

Estado de control, validez y mensajes de error

Before we go deep into how the form model is structured, and how that corresponds to our data model in the component, we will first cover the simpler aspects, which is dealing with the control's state and validity. Dealing with form control state and validity is quite similar to how we handle it with template-driven forms, in that the base control states and validity are the same. What changes is the method of accessing these properties.

Antes de profundizar en cómo está estructurado el modelo de formulario y cómo se corresponde con nuestro modelo de datos en el componente, primero cubriremos los aspectos más simples, que tienen que ver con el estado y la validez del control. Tratar el estado de control de formulario y la validez es bastante similar a cómo lo manejamos con formularios basados en plantillas, en el sentido de

que los estados de control base y la validez son los mismos. Lo que cambia es el método de acceso a estas propiedades.

Let's now add error messages to our form that we have been building so far, so that we can show respective error messages along each field. That said, we only want to show these error messages if the user has interacted with a field, and not before. So by default, when the page opens, we don't want to show any error messages.

Ahora agreguemos mensajes de error a nuestro formulario que hemos estado creando hasta ahora, para que podamos mostrar los respectivos mensajes de error en cada campo. Dicho esto, sólo queremos mostrar estos mensajes de error si el usuario ha interactuado con un campo, y no antes. Entonces, de forma predeterminada, cuando se abre la página, no queremos mostrar ningún mensaje de error.

We will build on the example from the previous section, so in case you are not coding along, you can copy the codebase from *chapter7/form-builder*. Before we start going into the code, here's a quick refresher on the basic Angular control states (repeated from [Chapter 6](#)):

Nos basaremos en el ejemplo de la sección anterior, por lo que, en caso de que no esté codificando, puede copiar el código base del capítulo 7/form-builder. Antes de comenzar a entrar en el código, aquí hay un repaso rápido sobre los estados básicos del control Angular (repetido desde el Capítulo 6):

Control state Estado de control	CSS class if True Clase CSS si es verdadero	CSS class if False Clase CSS si es falso
Visited	ng-touched	ng-untouched
Visitado	ng-touched	ng-untouched
Changed	ng-dirty	ng-pristine
Cambió	ng-dirty	ng-pristine
Valid	ng-valid	ng-invalid
Válido	ng-valid	ng-invalid

We could use these to highlight and show errors and state of the form, like we did before. But for this section, we will focus on showing only condition error messages, and cleanly handling cases with multiple validators.

Podríamos usarlos para resaltar y mostrar errores y el estado del formulario, como hicimos antes. Pero en esta sección, nos centraremos en mostrar solo mensajes de error de condición y en manejar limpiamente casos con múltiples validadores.

Now, let's see how to edit the template to start showing relevant error messages in the form, while using the reactive form approach. We will edit the *src/app/stock/create-stock/create-stock.component.html* file as follows:

Ahora, veamos cómo editar la plantilla para comenzar a mostrar mensajes de error relevantes en el formulario, mientras utilizamos el enfoque de formulario reactivo. Editaremos el archivo *src/app/stock/create-stock/create-stock.component.html* de la siguiente manera:

```
<h2>Create Stock Form</h2>
```

```
<div class="form-group">
```

```

<form [formGroup]="stockForm" (ngSubmit)="onSubmit()">
  <div class="stock-name">
    <input type="text"
      placeholder="Stock Name"
      name="stockName"
      formControlName="name">
    <div *ngIf="stockForm.get('name').invalid &&
      ( stockForm.get('name').dirty ||
        stockForm.get('name').touched )" style="color: red; font-size: small; margin-top: 5px;">
      Name is required
    </div>
  </div>
  <div class="stock-code">
    <input type="text"
      placeholder="Stock Code"
      name="stockCode"
      formControlName="code">
    <div *ngIf="stockForm.get('code').invalid &&
      ( stockForm.get('code').dirty ||
        stockForm.get('code').touched )" style="color: red; font-size: small; margin-top: 5px;">
      Stock Code is required
    </div>
    <div *ngIf="stockForm.get('code').errors.required" style="color: red; font-size: small; margin-top: 5px;">
      Stock Code must be at least 2 characters
    </div>
  </div>
  <div class="stock-price">
    <input type="number"
      placeholder="Stock Price"
      name="stockPrice"
      formControlName="price">
    <div *ngIf="stockForm.get('price').invalid &&
      ( stockForm.get('price').dirty ||
        stockForm.get('price').touched )" style="color: red; font-size: small; margin-top: 5px;">
      Stock Price is required
    </div>
    <div *ngIf="stockForm.get('price').errors.min" style="color: red; font-size: small; margin-top: 5px;">
      Stock Price must be positive
    </div>
  </div>
  <button type="submit">Submit</button>
</form>
</div>

<p>Form Control value: {{ stockForm.value | json }}</p>
<p>Form Control status: {{ stockForm.status | json }}</p>

```

- ① Accessing an individual control element's validity through the ~~Acceso a la~~ ~~verifica la~~ ~~valididad de un elemento de control individual a través del grupo de formularios~~
- ② ~~Comprobación individual de la validación para el elemento de formulario~~

While the base form remains the same from the previous example, we have modified it now to show conditional error messages. There are a few noteworthy things happening in this template, so let's walk through it step by step:

Si bien el formulario base sigue siendo el mismo del ejemplo anterior, ahora lo hemos modificado para mostrar mensajes de error condicionales. En esta plantilla suceden algunas cosas notables, así que veámoslas paso a paso:

- For each form element, we have added a `div` element beneath it to show conditional error messages.
- Para cada elemento del formulario, hemos agregado un elemento `div` debajo para mostrar mensajes de error condicionales.
- For each element, we first get the individual form element by calling `stockForm.get()` with the name of the individual form control that we provided while instantiating the `FormGroup` in the component class.

Para cada elemento, primero obtenemos el elemento de formulario individual llamando a `stockForm.get()` con el nombre del control de formulario individual que proporcionamos al crear una instancia de `FormGroup` en la clase de componente.

- With each `FormControl`, we can then check for various properties like whether the form element is touched or not (that is, whether the user has accessed the element), whether the

form element has been modified or not (dirty or pristine), and whether it is valid or not.

Con cada FormControl, podemos verificar varias propiedades, como si el elemento del formulario se tocó o no (es decir, si el usuario accedió al elemento), si el elemento del formulario se modificó o no (sucio o prístino).), y si es válido o no.

- For our example, we are relying on these properties to ensure that we display the error message only when the form element is both invalid and the user has interacted with it by either modifying it (dirty if modified, pristine otherwise) or at least accessing it (touched if accessed, untouched otherwise).

Para nuestro ejemplo, confiamos en estas propiedades para garantizar que mostramos el mensaje de error solo cuando el elemento del formulario no es válido y el usuario ha interactuado con él modificándolo (dirty si se modifica, pristine en caso contrario) o al menos acceder a él (touched si se accede, untouched en caso contrario).

- For form fields with more than one validator (primarily, the stock code and price), we further look at the errors property on the form control. This field allows us to check what kind of error is causing the form field to be invalid, and thus show the respective error message.

Para los campos de formulario con más de un validador (principalmente, el código de acciones y el precio), analizamos más a fondo la propiedad errors en el control del formulario. Este campo nos permite comprobar qué tipo de error está provocando que el campo del formulario no sea válido, y así mostrar el mensaje de error respectivo.

TIP CONSEJO

Instead of repeating `stockForm.get('price')` every time, you might want to create simple getters in the component class like so:

En lugar de repetir `stockForm.get('price')` cada vez, es posible que desees crear captadores simples en la clase de componente, así:

```
@Component({
  selector: 'app-create-stock',
  templateUrl: './create-stock.component.html',
  styleUrls: ['./create-stock.component.css']
})
export class CreateStockComponent {
  /* Skipping irrelevant code here */

  get name() { return this.stockForm.get('name'); }

  get price() { return this.stockForm.get('price'); }

  get code() { return this.stockForm.get('code'); }
}
```

Now in your HTML, you can simply refer to `name.invalid` instead of `stockForm.get('name').invalid` and so on.

Ahora en su HTML, puede simplemente hacer referencia a `name.invalid` en lugar de `stockForm.get('name').invalid` y así sucesivamente.

In this way, we can interact with the state of the form controls and provide the correct user experience to the users of our web application.

De esta manera, podemos interactuar con el estado de los controles del formulario y brindar la experiencia de usuario correcta a los usuarios de nuestra aplicación web.

The finished example is available in the `chapter7/control-state-validity` folder in the GitHub repository.

El ejemplo terminado está disponible en la carpeta capítulo7/control-state-validity en el repositorio de GitHub.

Form and Data Model

Modelo de formulario y datos

Now we will start digging into accessing and working with the data driving the form, and the interaction between the form and the data model in our component. We simplified this so far in the preceding examples, by simply accessing the value from the `FormGroup` or the `FormControl`. This is also what we log in both the template using the `json` pipe, as well as in the component when we click the Submit button.

Ahora comenzaremos a profundizar en el acceso y el trabajo con los datos que impulsan el formulario y la interacción entre el formulario y el modelo de datos en nuestro componente. Simplificamos esto hasta ahora en los ejemplos anteriores, simplemente accediendo al `value` desde el `FormGroup` o el `FormControl`. Esto también es lo que iniciamos sesión tanto en la plantilla usando la tubería `json` como en el componente cuando hacemos clic en el botón Enviar.

Let's use an example to demonstrate how we work with the form and data model, and how the two interact. First, let's change our template slightly from the previous example (in case you want the completed code, you can copy it from the `chapter7/control-state-validity` folder) to provide for a few more actions.

Usemos un ejemplo para demostrar cómo trabajamos con el formulario y el modelo de datos, y cómo interactúan ambos. Primero, cambiemos ligeramente nuestra plantilla con respecto al ejemplo anterior (en caso de que desee el código completo, puede copiarlo de la carpeta `capítulo7/control-state-validity`) para permitir algunas acciones más.

We will edit the template for the CreateStockComponent by changing `src/app/stock/create-stock/create-stock.component.html` as follows:

Editaremos la plantilla para CreateStockComponent cambiando `src/app/stock/create-stock/create-stock.component.html` de la siguiente manera:

```
<h2>Create Stock Form</h2>

<div class="form-group">
  <form [formGroup]="stockForm" (ngSubmit)="onSubmit()">

    <!-- Repeated code from before, omitted for brevity -->

    <button type="submit">Submit</button>
    <button type="button"
      (click)="resetForm()">
      Reset
    </button>
    <button type="button"
      (click)="loadStockFromServer()">
      Simulate Stock Load from Server
    </button>
    <button type="button"
      (click)="patchStockForm()">
      Patch Stock Form
    </button>
  </form>
</div>

<p>Form Control value: {{ stockForm.value | json }}</p>
<p>Form Control status: {{ stockForm.status | json }}</p>
```

Most of the template has not changed, but we have added three new buttons at the end of the form. All three of them call out to a method in the component class, which we will see in just a bit. But the three buttons fundamentally perform the following two actions:

La mayor parte de la plantilla no ha cambiado, pero hemos agregado tres botones nuevos al final del formulario. Los tres llaman a un método en la clase de componente, que veremos en un momento.

Pero los tres botones realizan fundamentalmente las dos acciones siguientes:

1. Reset the form to its original state

Restablecer el formulario a su estado original

2. Simulate loading a stock from the server

Simular la carga de un stock desde el servidor.

And to perform the latter, we show two methods by which we can accomplish that with our reactive form.

Y para realizar esto último, mostramos dos métodos mediante los cuales podemos lograrlo con nuestra forma reactiva.

WARNING ADVERTENCIA

Watch out in case you forget the type on the button element. Depending on the browser, it can take various defaults. For example, Chrome on Mac will assume the type as submit if omitted, causing a form submit even while triggering its event handler.

Tenga cuidado en caso de que olvide el type en el elemento button. Dependiendo del navegador, puede adoptar varios valores predeterminados. Por ejemplo, Chrome en Mac asumirá type como submit si se omite, lo que provocará el envío de un formulario incluso mientras se activa su controlador de eventos.

Now, let's move to the `CreateStockComponent` class, which is where most of the activity and changes happen. We will edit the `src/app/stock/create-stock/create-stock.component.ts` file as follows:

Ahora, pasemos a la clase `CreateStockComponent`, que es donde ocurre la mayor parte de la actividad y los cambios. Editaremos el archivo `src/app/stock/create-stock/create-stock.component.ts` de la siguiente manera:

```

/** NO CHANGE IN IMPORTS **/


let counter = 1;

/** NO CHANGE IN COMPONENT DECORATOR **/
export class CreateStockComponent {

    private stock: Stock;          ①
    public stockForm: FormGroup;
    constructor(private fb: FormBuilder) {
        this.createForm();
        this.stock = new Stock('Test ' + counter++, 'TST', 20, 10);  ②
    }

    createForm() {
        this.stockForm = this.fb.group({
            name: [null, Validators.required],
            code: [null, [Validators.required, Validators.minLength(2)]],
            price: [0, [Validators.required, Validators.min(0)]]
        });
    }

    loadStockFromServer() {
        this.stock = new Stock('Test ' + counter++, 'TST', 20, 10);
        let stockFormModel = Object.assign({}, this.stock);
        delete stockFormModel.previousPrice;
        delete stockFormModel.favorite;
        this.stockForm.setValue(stockFormModel);  ③
    }

    patchStockForm() {
        this.stock = new Stock(`Test ${counter++}`, 'TST', 20, 10);
        this.stockForm.patchValue(this.stock);  ④
    }

    resetForm() {
        this.stockForm.reset();  ⑤
    }

    onSubmit() {
        this.stock = Object.assign({}, this.stockForm.value);
        console.log('Saving stock', this.stock);
    }
}

```

①

We have introduced a stock model object, in addition to the form models introducido un objeto modelo stock, además del modelo de formulario.

- ② Instantiating our stock model with some default value algún valor predeterminado
- ③ Setting up the entire form model for it to work correctly dentro de la vista modelo de datos stock
- ④ Patching the form model with additional fields no disponibles
- ⑤ Restoring the form data to its initial state el estado inicial

While it seems like we have added a lot of code, it is actually not that much. Let's walk through the component step by step to understand what has changed and what it does:

Si bien parece que hemos agregado mucho código, en realidad no es tanto. Repasemos el componente paso a paso para comprender qué ha cambiado y qué hace:

1. You can ignore the counter; it is there simply to make sure that something (the name) changes every time we click a button. It is not tied into any of the Angular functionality otherwise.

Puedes ignorar el counter; está ahí simplemente para asegurarse de que algo (el nombre) cambie cada vez que hacemos clic en un botón. De lo contrario, no está vinculado a ninguna de las funciones de Angular.

2. We create a stock model object, which is a pure data model object. This is not tied to the Angular form models or anything we have been dealing with so far. It parallels the form model, but has a few extra fields that we don't ask for in the form. We instantiate this with a default value in the constructor (but it is not used or tied to the template until this point).

Creamos un objeto modelo `stock`, que es un objeto de modelo de datos puro. Esto no está vinculado a los modelos de forma angular ni a nada con lo que hayamos estado tratando hasta ahora. Es paralelo al modelo de formulario, pero tiene algunos campos adicionales que no solicitamos en el formulario.

Creamos una instancia de esto con un valor predeterminado en el constructor (pero no se usa ni se vincula a la plantilla hasta este momento).

3. We leave the `createForm` method untouched.

Dejamos el método `createForm` intacto.

4. We create a new method `loadStockFromServer`, to simulate us fetching the stock details from the server. While it is synchronous (unlike a real HTTP server call), it shows us how to take a value and push it to the UI's form model.

Creamos un nuevo método `loadStockFromServer`, para simular que obtenemos los detalles del stock del servidor. Si bien es sincrónico (a diferencia de una llamada real al servidor HTTP), nos muestra cómo tomar un valor y enviarlo al modelo de formulario de la interfaz de usuario.

5. We use the `setValue` method on the `stockForm FormGroup` instance. This method takes a JSON model object that matches the form model exactly. That means for the `setValue` to work in this case, it needs an object with a `name`, `code`, and `price` key. It should not have more or fewer keys than this, as it would throw an error in this case. Based on the object, the form's model object values would get updated and these values would be visible in the form in the UI. This is also the reason why we delete all other keys from the model object before calling `setValue`.

Usamos el método `setValue` en la instancia `stockForm FormGroup`. Este método toma un objeto de modelo JSON que

coincide exactamente con el modelo de formulario. Eso significa que para que `setValue` funcione en este caso, necesita un objeto con una clave `name`, `code` y `price`. No debería tener más o menos claves que esta, ya que arrojaría un error en este caso. Según el objeto, los valores del objeto modelo del formulario se actualizarián y estos valores serían visibles en el formulario en la interfaz de usuario. Esta es también la razón por la que eliminamos todas las demás claves del objeto modelo antes de llamar a `setValue`.

6. Thus, triggering the `loadStockFromServer` method would end up updating the form with the name, code, and price from the newly created `stock` instance.

Por lo tanto, activar el método `loadStockFromServer` terminaría actualizando el formulario con el nombre, código y precio de la instancia `stock` recién creada.

7. The second method, `patchStockForm`, uses another method on the `stockForm FormGroup` instance called `patchValue`. This is a more forgiving method that takes the fields it has available, and updates the form with them. It will ignore extra fields even if it has fewer fields.

El segundo método, `patchStockForm`, utiliza otro método en la instancia `stockForm FormGroup` llamada `patchValue`. Este es un método más indulgente que toma los campos que tiene disponibles y actualiza el formulario con ellos. Ignorará campos adicionales incluso si tiene menos campos.

8. Triggering the `patchStockForm` instance would also have the same effect as triggering the previous button. What is more interesting (and left as an exercise to the reader) is to delete the code from the `stock` object and try patching. In the case of `setValue`, it will fail and throw an exception, while `patchValue`

would happily set the other fields (`name`, `price`) and leave the code untouched in the form.

Activar la instancia `patchStockForm` también tendría el mismo efecto que activar el botón anterior. Lo que es más interesante (y se deja como ejercicio para el lector) es eliminar el código del objeto estándar e intentar parchearlo. En el caso de `setValue`, fallará y generará una excepción, mientras que `patchValue` felizmente configuraría los otros campos (`name`, `price`) y dejaría el código intacto en el forma.

9. The final method is the `resetForm`, which simply resets the form to its initial state.

El método final es el `resetForm`, que simplemente restablece el formulario a su estado inicial.

One final thing to note is that we have changed the `onSubmit` method slightly as well. While in this case our form model mirrors the data model, it is good practice to not directly assign the form model to our data model, but rather make a copy of it. In this case, since it is a simple object, a simple `Object.assign` or the spread operator works, but for slightly more complicated models, it might be necessary to do a deep copy.

Una última cosa a tener en cuenta es que también hemos cambiado ligeramente el método `onSubmit`. Si bien en este caso nuestro modelo de formulario refleja el modelo de datos, es una buena práctica no asignar directamente el modelo de formulario a nuestro modelo de datos, sino hacer una copia del mismo. En este caso, dado que es un objeto simple, un simple `Object.assign` o el operador de extensión funcionan, pero para modelos un poco más complicados, puede que sea necesario hacer una copia profunda.

The finished example is available in the `chapter7/form-model` folder in the GitHub repository.

El ejemplo terminado está disponible en la carpeta capítulo7/form-model en el repositorio de GitHub.

ANOTHER ADVANTAGE OF REACTIVE FORMS

OTRA VENTAJA DE LAS FORMAS REACTIVAS

Another slightly subtle advantage of using reactive forms over template-driven forms is that it forces developers to have a separation between what the user sees and interacts with (what we call the form model), and the persisted data model that drives our application. This is quite common in most applications, where the presented view is different from what the underlying data model is. Reactive forms make that distinction clear, while also keeping you cognizant of the data flow and giving you control over when and what flows from UI to the component and vice versa.

Otra ventaja ligeramente sutil de usar formularios reactivos sobre formularios basados en plantillas es que obliga a los desarrolladores a tener una separación entre lo que el usuario ve y con lo que interactúa (lo que llamamos el modelo de formulario) y el modelo de datos persistentes que impulsa nuestra aplicación. Esto es bastante común en la mayoría de las aplicaciones, donde la vista presentada es diferente del modelo de datos subyacente. Los formularios reactivos aclaran esa distinción, al mismo tiempo que lo mantienen consciente del flujo de datos y le brindan control sobre cuándo y qué fluye desde la interfaz de usuario al componente y viceversa.

FormArrays

matrices de formularios

For the purpose of demonstrating the last thing related to reactive forms, we are going to extend our stock market example in a slightly tangential manner. Let's suppose that for each stock, we wanted to capture and highlight the key people related to the company, as well as their titles. A company may have none, one, or many such people associated with it.

Con el fin de demostrar lo último relacionado con las formas reactivas, vamos a ampliar nuestro ejemplo del mercado de valores de forma ligeramente tangencial. Supongamos que para cada acción quisiéramos capturar y resaltar a las personas clave relacionadas con la empresa, así como sus títulos. Una empresa puede tener ninguna, una o muchas personas asociadas a ella.

This will allow us to see how we can handle forms where we need to capture multiple values as well as handle nested form elements cleanly. We will take the codebase from the previous section and modify it to support the following:

Esto nos permitirá ver cómo podemos manejar formularios en los que necesitamos capturar múltiples valores, así como manejar limpiamente elementos de formulario anidados. Tomaremos el código base de la sección anterior y lo modificaremos para admitir lo siguiente:

1. Add a new model to represent one or many notable persons under a stock

Agregar un nuevo modelo para representar a una o varias personas notables bajo una acción

2. Add a button in the UI to add a new notable person for a stock

Agregue un botón en la interfaz de usuario para agregar una nueva persona notable para una acción

3. Add a button to remove an added notable person from the stock

Agregue un botón para eliminar a una persona notable agregada del stock

4. Support basic validation on each notable person added

Admite validación básica de cada persona notable agregada

Let's walk through, step by step, how to accomplish this. In case you are not coding along with the examples, you can copy the codebase from *chapter7/form-model* and work from there.

Veamos, paso a paso, cómo lograr esto. En caso de que no esté codificando junto con los ejemplos, puede copiar el código base del capítulo 7/form-model y trabajar desde allí.

We will first update the model to understand this new concept of a person. Ideally, we would do this in a model file of its own, but here, we took a shortcut for readability and making it easier to understand. We added our model to the *src/app/model/stock.ts* file as follows:

Primero actualizaremos el modelo para comprender este nuevo concepto de persona. Idealmente, haríamos esto en un archivo modelo propio, pero aquí tomamos un atajo para facilitar la lectura y la comprensión. Agregamos nuestro modelo al archivo *src/app/model/stock.ts* de la siguiente manera:

```
export class Stock {
    favorite = false;
    notablePeople: Person[];

    constructor(public name: string,
                public code: string,
                public price: number,
                public previousPrice: number) {
        this.notablePeople = [];
    }
}
```

```

    }

    isPositiveChange(): boolean {
      return this.price >= this.previousPrice;
    }
  }

export class Person {
  name: string;
  title: string;
}

```

We added a new class `Person` with a `name` and a `title`, and then added it as a child to the `Stock` class, with the name `notablePeople`. In the constructor for `Stock` class, we initialized it to an empty array.

Agregamos una nueva clase `Person` con un `name` y un `title`, y luego la agregamos como hija a la clase `Stock`, con el nombre `notablePeople`. En el constructor de la clase `Stock`, lo inicializamos en una matriz vacía.

Now we move over to our `CreateStockComponent` class. First, let's walk through the changes to the component class, which is located in `src/app/stock/create-stock/create-stock.component.ts`:

Ahora pasamos a nuestra clase `CreateStockComponent`. Primero, veamos los cambios en la clase de componente, que se encuentra en `src/app/stock/create-stock/create-stock.component.ts`:

```

/*
Omitted for brevity, no change in imports
*/
export class CreateStockComponent {

  private stock: Stock;
  public stockForm: FormGroup;
  constructor(private fb: FormBuilder) {
    this.createForm();
  }

  createForm() {
    this.stockForm = this.fb.group({
      name: [null, Validators.required],
    });
  }
}

```

```

    code: [null, [Validators.required, Validators.minLength(2)]],
    price: [0, [Validators.required, Validators.min(0)]],
    notablePeople: this.fb.array([])           ①
  );
}

get notablePeople(): FormArray {            ②
  return this.stockForm.get('notablePeople') as FormArray;
}

addNotablePerson() {                      ③
  this.notablePeople.push(this.fb.group({
    name: ['', Validators.required],
    title: ['', Validators.required]
  )));
}

removeNotablePerson(index: number) {        ④
  this.notablePeople.removeAt(index);
}

resetForm() {
  this.stockForm.reset();
}

onSubmit() {
  this.stock = Object.assign({}, this.stockForm.value);
  console.log('Saving stock', this.stock);
}
}

```

- ① Initialize notablePeople as a FormArray instance
- ② Getter to make it easier to access the underlying FormArray from the template. Facilitar el acceso al FormArray subyacente desde la plantilla
- ③ Add a new FormGroup instance to the FormArray
- ④ Remove a particular FormGroup instance from the FormArray

There are a few notable things in the code for the component class. Primarily, we have removed all things related to the model, including

loading and patching the stock model object from the class. Instead, we have added:

Hay algunas cosas notables en el código de la clase de componente. Principalmente, hemos eliminado todo lo relacionado con el modelo, incluida la carga y la aplicación de parches al objeto del modelo original de la clase. En su lugar, hemos agregado:

- `notablePeople` to the main `FormGroup`. Note that `notablePeople` is a `FormArray` instance with an initial value that is empty. In case we have to populate it with existing values, we would pass it to the constructor.

`notablePeople` al `FormGroup` principal. Tenga en cuenta que `notablePeople` es una instancia de `FormArray` con un valor inicial que está vacío. En caso de que tengamos que completarlo con valores existentes, se lo pasaríamos al constructor.

- We have created a simple getter for `notablePeople`, which goes deep into the `stockForm` `FormGroup` instance and returns the `notablePeople` `FormArray` instance. This is more for the template to prevent us from writing `this.stockForm.get('notablePeople')` each time.

Hemos creado un captador simple para `notablePeople`, que profundiza en la instancia `stockForm` `FormGroup` y devuelve la instancia `notablePeople` `FormArray`. Esto es más para que la plantilla nos impida escribir `this.stockForm.get('notablePeople')` cada vez.

- Since we can have zero to many notable people per stock, we need a method to allow us to add as many notable people as we want. This is what the `addNotablePerson()` method does. Note that each instance of notable person in the actual form is represented by a `FormGroup`. So each time we want to add a new notable person, we add a `FormGroup` instance with a required name and title.

Dado que podemos tener de cero a muchas personas notables por acción, necesitamos un método que nos permita agregar tantas personas notables como queramos. Esto es lo que hace el método `addNotablePerson()`. Tenga en cuenta que cada instancia de persona notable en la forma real está representada por un `FormGroup`. Entonces, cada vez que queremos agregar una nueva persona notable, agregamos una instancia `FormGroup` con un `name` y un `title` obligatorios.

- Similarly, we want to be able to remove any notable person that we have added, which is what the `removeNotablePerson()` method does. It takes an index and just removes that particular index from the `FormArray` instance.

De manera similar, queremos poder eliminar cualquier persona notable que hayamos agregado, que es lo que hace el método `removeNotablePerson()`. Toma un índice y simplemente elimina ese índice particular de la instancia `FormArray`.

Next, we will add some simple CSS that will allow us to separate out each individual person by adding the following to `src/app/stock/create-stock/create-stock.component.css`:

A continuación, agregaremos algo de CSS simple que nos permitirá separar a cada persona individual agregando lo siguiente a `src/app/stock/create-stock/create-stock.component.css`:

```
.notable-people {
  border: 1px solid black;
  padding: 10px;
  margin: 5px;
}
```

Finally, let's now look at how the template changes to hook all of this up. We will modify `src/app/stock/create-stock/create-stock.component.html` as follows:

Finalmente, veamos ahora cómo cambia la plantilla para conectar todo esto. Modificaremos src/app/stock/create-stock/create-stock.component.html de la siguiente manera:

```
<h2>Create Stock Form</h2>

<div class="form-group">
  <form [formGroup]="stockForm" (ngSubmit)="onSubmit()">
    <!-- No change until the end of price form element -->
    <!-- Omitted for brevity -->
    <div formArrayName="notablePeople">
      <div *ngFor="let person of notablePeople.controls; let i = index"
          [formGroupName]="i"
          class="notable-people">
        <div>
          Person {{i + 1}}
        </div>
        <div>
          <input type="text"
            placeholder="Person Name"
            formControlName="name">
        </div>
        <div>
          <input type="text"
            placeholder="Person Title"
            formControlName="title">
        </div>
        <button type="button"
          (click)="removeNotablePerson(i)">
          Remove Person
        </button>
      </div>
    </div>
    <button type="button"
      (click)="addNotablePerson()">
      Add Notable Person
    </button>
    <button type="submit">Submit</button>
    <button type="button"
      (click)="resetForm()">
      Reset
    </button>
  </form>
</div>

<p>Form Control value: {{ stockForm.value | json }}</p>
```

```
<p>Form Control status: {{ stockForm.status | json }}</p>
<p>Stock Value: {{stock | json}}</p>
```

There are a few things to note on how we hooked together the `FormArray` instance we created in our component to the template for the component:

Hay algunas cosas a tener en cuenta sobre cómo conectamos la instancia `FormArray` que creamos en nuestro componente a la plantilla del componente:

- Instead of using `formControlName`, we use `formGroupName` on the enclosing `div` element. This is the element that will contain zero to many forms, one for each notable person.

En lugar de usar `formControlName`, usamos `formGroupName` en el elemento `div` adjunto. Este es el elemento que contendrá de cero a muchos formularios, uno para cada persona notable.

- We then have a `div` element that is repeated once for each entry in the `FormArray` instance, which we access through `notablePeople.controls`. The `notablePeople` accesses the getter that we created in the component.

Luego tenemos un elemento `div` que se repite una vez para cada entrada en la instancia `FormArray`, al que accedemos a través de `notablePeople.controls`. El `notablePeople` accede al captador que creamos en el componente.

- We also expose the current index of the `*ngFor` via the variable `i`.

También exponemos el índice actual de `*ngFor` a través de la variable `i`.

- We then connect the `FormGroup` that is each element in the `FormArray` via the `formGroupName` binding, binding it to each individual index in the array.

Luego conectamos el FormGroup que es cada elemento del FormArray a través del enlace formGroupName, vinculándolo a cada índice individual de la matriz.

- This allows us to then use formControlName individually for the name and title like we have done so far. This ensures that the name and title are bound to that particular FormGroup instance denoted by the index in the FormArray.

Esto nos permite usar formControlName individualmente para name y title como lo hemos hecho hasta ahora. Esto garantiza que name y title estén vinculados a esa instancia particular de FormGroup indicada por el índice en FormArray.

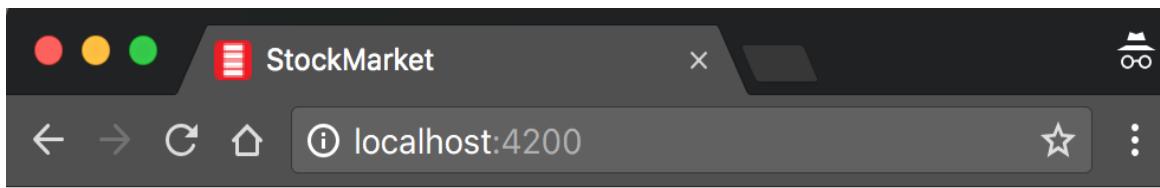
- Finally, we have the Remove Person button within each *ngFor instance, which calls the removeNotablePerson() method, and a global Add Person button, which calls the addNotablePerson() method.

Finalmente, tenemos el botón Eliminar persona dentro de cada instancia *ngFor, que llama al método removeNotablePerson(), y un botón global Agregar persona, que llama al método addNotablePerson().

When you run this, you should now see a new button, Add Notable Person. Clicking this should show new form elements to enter a person's details. You can click this multiple times to add more people, and you can click Remove on any individual person to remove them. Your working application should look something like [Figure 7-3](#).

Cuando ejecute esto, ahora debería ver un nuevo botón, Agregar persona notable. Al hacer clic aquí, se deberían mostrar nuevos elementos del formulario para ingresar los detalles de una persona. Puede hacer clic aquí varias veces para agregar más personas y puede hacer clic en Eliminar en cualquier persona individual para

eliminarla. Su aplicación funcional debería verse similar a la Figura 7-3.



Stock Market App

Test Stock Company (TSC)
\$ 85

Add to Favorite

Create Stock Form

Test
TTT
244

Person 1

First One
Alpha

Remove Person

Person 2

Second
Omega

Remove Person

Add Notable Person Submit Reset

Form Control value: { "name": "Test", "code": "TTT", "price": 244, "notablePeople": [{ "name": "First One", "title": "Alpha" }, { "name": "Second", "title": "Omega" }] }

Form Control status: "VALID"

Figure 7-3. Angular form using FormArray

Figura 7-3. Forma angular usando FormArray

Notice that the form value (printed as JSON) also holds all the person data that you enter. Thus, when the form is submitted, you can capture this data and transform it to the necessary data model before transmitting it.

Observe que el valor del formulario (impreso como JSON) también contiene todos los datos personales que ingresa. Por lo tanto, cuando se envía el formulario, puede capturar estos datos y transformarlos al modelo de datos necesario antes de transmitirlos.

The finished example is available in the *chapter7/form-arrays* folder in the GitHub repository.

El ejemplo terminado está disponible en la carpeta *capítulo7/form-arrays* en el repositorio de GitHub.

Conclusion

Conclusión

In this chapter, we explored how to create reactive forms as an alternative to using template-driven forms. We reviewed the building blocks of these forms and saw how to use elements like `FormControl`, `FormGroup`, and even `FormArray` to build complex user experiences. We also briefly touched upon the major differences between template-driven forms and reactive forms, and how reactive forms facilitate and highlight the difference between having a form model (or the presentation model), which is different from our underlying data model.

En este capítulo, exploramos cómo crear formularios reactivos como una alternativa al uso de formularios basados en plantillas.

Revisamos los componentes básicos de estos formularios y vimos cómo utilizar elementos como `FormControl`, `FormGroup` e incluso `FormArray` para crear experiencias de usuario complejas. También abordamos brevemente las principales diferencias entre los formularios basados en plantillas y los formularios reactivos, y cómo los formularios reactivos facilitan y resaltan la diferencia entre tener un modelo de formulario (o el modelo de presentación), que es diferente de nuestro modelo de datos subyacente.

In the next chapter, we will start exploring Angular services, what they are, and when and how to create them. We will also touch upon Angular's dependency injection framework and how to deal with asynchronous behavior using observables.

En el próximo capítulo, comenzaremos a explorar los servicios Angular, qué son, cuándo y cómo crearlos. También abordaremos el marco de inyección de dependencia de Angular y cómo lidiar con el comportamiento asincrónico utilizando observables.

Exercise

Ejercicio

Take the finished exercise from [Chapter 5](#) (available in `chapter5/exercise/ecommerce`). We will repeat the same exercise that we accomplished using template-driven forms, by performing the following:

Realice el ejercicio terminado del Capítulo 5 (disponible en el capítulo 5/ejercicio/comercio electrónico). Repetiremos el mismo ejercicio que realizamos usando formularios basados en plantillas, realizando lo siguiente:

1. Create a new component that allows us to add new products.

Crear un nuevo componente que nos permita agregar nuevos productos.

2. Create a form that takes in the product name, price, image URL, and whether it is on sale or not. Create proper FormGroup to encapsulate this form. Use FormBuilder ideally.

Cree un formulario que incluya el nombre del producto, el precio, la URL de la imagen y si está en oferta o no. Cree el FormGroup adecuado para encapsular este formulario. Utilice FormBuilder idealmente.

3. Make all the fields except the On Sale checkbox required. Set the minimum valid price as 1.

Haga que todos los campos, excepto la casilla de verificación En oferta, sean obligatorios. Establezca el precio mínimo válido en 1.

4. Add a basic Regex pattern validation for the image URL.

Agregue una validación de patrón Regex básica para la URL de la imagen.

5. Display relevant error messages, but only after the user either modifies a field or after the first submit.

Muestra mensajes de error relevantes, pero solo después de que el usuario modifique un campo o después del primer envío.

6. Copy over the form and print it to the console after successful submission.

Copie el formulario e imprímalo en la consola después de enviarlo correctamente.

All of this can be accomplished using concepts covered in this chapter. You can check out the finished solution in [*chapter7/exercise/ecommerce*](#).

Todo esto se puede lograr utilizando los conceptos tratados en este capítulo. Puede consultar la solución terminada en el capítulo 7/ejercicio/ecommerce.

Chapter 8. Angular Services

Capítulo 8. Servicios angulares

In the previous two chapters, we worked on creating form-based Angular applications. To this extent, we approached it in two ways: the traditional template-driven approach and then the reactive form-based approach. With both, we looked at the building blocks and then built live working forms with validation and proper messaging.

En los dos capítulos anteriores, trabajamos en la creación de aplicaciones Angular basadas en formularios. Hasta este punto, lo abordamos de dos maneras: el enfoque tradicional basado en plantillas y luego el enfoque reactivo basado en formularios. Con ambos, analizamos los componentes básicos y luego creamos formularios de trabajo en vivo con validación y mensajes adecuados.

In this chapter, we will move beyond the UI for a bit, and start exploring Angular services, which are the workhorses of any application. We will take a step back, understand what Angular services are, and how to create one. Then we will dig into the Angular dependency injection system and understand how to use and leverage it, before going off and building our own Angular services.

En este capítulo, iremos un poco más allá de la interfaz de usuario y comenzaremos a explorar los servicios Angular, que son los caballos de batalla de cualquier aplicación. Daremos un paso atrás, entenderemos qué son los servicios Angular y cómo crear uno.

Luego, profundizaremos en el sistema de inyección de dependencia de Angular y comprenderemos cómo usarlo y aprovecharlo, antes de comenzar a construir nuestros propios servicios de Angular.

What Are Angular Services?

¿Qué son los servicios angulares?

We have mostly worked with Angular components so far. Components, to quickly recap, are responsible for deciding what data to display and how to render and display it in the UI. We bind our data from the components to our UI and bind our events from the UI to methods in the components to allow and handle user interactions. That is, components in Angular are our presentation layer, and should be involved in and focus on the presentation aspects of data.

Hasta ahora hemos trabajado principalmente con componentes angulares. Los componentes, para resumir rápidamente, son responsables de decidir qué datos mostrar y cómo renderizarlos y mostrarlos en la interfaz de usuario. Vinculamos nuestros datos de los componentes a nuestra interfaz de usuario y vinculamos nuestros eventos de la interfaz de usuario a métodos en los componentes para permitir y manejar las interacciones del usuario. Es decir, los componentes en Angular son nuestra capa de presentación y deben participar y centrarse en los aspectos de presentación de los datos.

But if components are our presentation layer, it begs the question of what should be responsible for the actual data fetching and common business logic in an Angular application. This is where Angular services come in. Angular services are that layer that is common across your application, that can be reused across various components. Generally, you would create and use Angular services when:

Pero si los componentes son nuestra capa de presentación, surge la pregunta de quién debería ser responsable de la recuperación de datos real y la lógica empresarial común en una aplicación Angular. Aquí es donde entran los servicios Angular. Los servicios Angular son esa capa que es común en su aplicación y que se puede reutilizar en varios componentes. Generalmente, crearía y utilizaría servicios Angular cuando:

- You need to retrieve data from or send data to your server. This may or may not involve any processing of the data while it is being transferred.

Necesita recuperar datos o enviarlos a su servidor. Esto puede implicar o no algún tratamiento de los datos durante su transferencia.

- You need to encapsulate application logic that is not specific to any one component, or logic that can be reused across components.

Debe encapsular la lógica de la aplicación que no sea específica de ningún componente o la lógica que pueda reutilizarse en todos los componentes.

- You need to share data across components, especially across components that may or may not know about each other. Services by default are singletons across your application, which allows you to store state and access them across various components.

Necesita compartir datos entre componentes, especialmente entre componentes que pueden conocerse o no entre sí. Los servicios de forma predeterminada son únicos en su aplicación, lo que le permite almacenar el estado y acceder a ellos a través de varios componentes.

Another simple way to think about Angular services is that it is the layer to abstract the “how” away from the component, so that the

component can just focus on the “what,” and let the service decide the how.

Otra forma sencilla de pensar en los servicios Angular es que es la capa que abstrae el “cómo” del componente, de modo que el componente pueda centrarse simplemente en el “qué” y dejar que el servicio decida el cómo.

Creating Our Own Angular Service

Creando nuestro propio servicio angular

Instead of talking in abstract, let’s dig into some actual code so that we can more fully understand the concept of services. What we will do is build on the example we have worked on so far, and extend it using Angular services. We will try to accomplish the following things in our stock market application:

En lugar de hablar en abstracto, profundicemos en el código real para que podamos comprender mejor el concepto de servicios. Lo que haremos será basarnos en el ejemplo en el que hemos trabajado hasta ahora y ampliarlo utilizando los servicios Angular. Intentaremos lograr lo siguiente en nuestra aplicación del mercado de valores:

- Fetch the list of stocks to show from a service, instead of hardcoding it in the component.

Obtenga la lista de acciones para mostrar desde un servicio, en lugar de codificarla en el componente.
- When we create a stock, we will send it to the service.

Cuando creamos un stock, lo enviaremos al servicio.
- When we create a stock, we want it to show up in our list of stocks.

Cuando creamos una acción, queremos que aparezca en nuestra lista de acciones.

To do this, if we want to continue using components, we would have to come up with a way for our `CreateStockComponent` to communicate with our `StockItemComponent`, and keep the data in sync between the two. This would mean our components would have to know each other, and where in the hierarchy they and all other components fit.

Para hacer esto, si queremos continuar usando componentes, tendríamos que encontrar una forma para que nuestro `CreateStockComponent` se comunique con nuestro `StockItemComponent` y mantener los datos sincronizados entre los dos. Esto significaría que nuestros componentes tendrían que conocerse entre sí y saber en qué parte de la jerarquía encajan ellos y todos los demás componentes.

First, we will add a `StockListComponent` that fetches a list of stocks, and displays them using the `StockItemComponent`. This component will use the `StockService` that we will build to act as a layer behind both the components. It will be responsible for figuring out how and where to fetch the list of stocks from, and how to create a stock. In this chapter, we will continue with it being a list on the client, but we will see how to build it in a way that each component does not need to worry about the details. Later, when we learn how to work with an HTTP server, we will see how easy it is to switch once we have a service layer in between.

Primero, agregaremos un `StockListComponent` que obtiene una lista de acciones y las muestra usando `StockItemComponent`. Este componente utilizará el `StockService` que construiremos para actuar como una capa detrás de ambos componentes. Será responsable de descubrir cómo y dónde obtener la lista de acciones y cómo crear una acción. En este capítulo, continuaremos siendo una lista en el cliente, pero veremos cómo construirla de manera que cada

componente no tenga que preocuparse por los detalles. Más adelante, cuando aprendamos a trabajar con un servidor HTTP, veremos lo fácil que es cambiar una vez que tenemos una capa de servicio en el medio.

Digging into the Example

Profundizando en el ejemplo

For this example, you can use the code from *chapter6/template-driven/control-validity* as the base to code from. We will modify the example to achieve what we have just outlined.

Para este ejemplo, puede utilizar el código del capítulo 6/template-driven/control-validity como base para codificar. Modificaremos el ejemplo para lograr lo que acabamos de esbozar.

First, let's tackle the easy stuff. The StockItemComponent class does not need any change, but we will change the template for it slightly so that it displays both an Add to Favorite and a Remove from Favorite button depending on the state of the stock. We can modify *src/app/stock/stock-item/stock-item.component.html* as follows:

Primero, abordemos las cosas fáciles. La clase StockItemComponent no necesita ningún cambio, pero cambiaremos ligeramente la plantilla para que muestre los botones Agregar a favoritos y Quitar de favoritos dependiendo del estado del stock. Podemos modificar *src/app/stock/stock-item/stock-item.component.html* de la siguiente manera:

```
<div class="stock-container">
  <div class="name">{{stock.name + ' (' + stock.code + ')'}}</div>
  <div class="exchange">{{stock.exchange}}</div>
  <div class="price"
    [class.positive]="stock.isPositiveChange()"
    [class.negative]="!stock.isPositiveChange()">$ {{stock.price}}</div>
  <button (click)="onToggleFavorite($event)"
    *ngIf="!stock.favorite">Add to Favorite</button>
```

```

<button (click)="onToggleFavorite($event)"
         *ngIf="stock.favorite">Remove from Favorite</button>
</div>

```

We have just added a div to show the exchange of the stock, and a button to show Remove from Favorite if the stock is already favorited.

Acabamos de agregar un div para mostrar el exchange de la acción y un botón para mostrar Eliminar de favorito si la acción ya está como favorita.

Next, let's create the skeleton for the StockListComponent before we try creating and integrating our StockService. We can generate the skeleton by executing:

A continuación, creemos el esqueleto para StockListComponent antes de intentar crear e integrar nuestro StockService. Podemos generar el esqueleto ejecutando:

```
ng g component stock/stock-list
```

which we can then modify to our purpose. We will modify the `src/app/stock/stock-list/stock-list.component.ts` file as follows:

que luego podemos modificar según nuestro propósito.
Modificaremos el archivo `src/app/stock/stock-list/stock-list.component.ts` de la siguiente manera:

```

import { Component, OnInit } from '@angular/core';
import { Stock } from 'app/model/stock';

@Component({
  selector: 'app-stock-list',
  templateUrl: './stock-list.component.html',
  styleUrls: ['./stock-list.component.css']
})
export class StockListComponent implements OnInit {

  public stocks: Stock[];
  constructor() { }
}

```

```

ngOnInit() {
  this.stocks = [
    new Stock('Test Stock Company', 'TSC', 85, 80, 'NASDAQ'),
    new Stock('Second Stock Company', 'SSC', 10, 20, 'NSE'),
    new Stock('Last Stock Company', 'LSC', 876, 765, 'NYSE')
  ];
}

onToggleFavorite(stock: Stock) {
  console.log('Favorite for stock ', stock, ' was triggered');
  stock.favorite = !stock.favorite;
}
}

```

There are a few things of note here, but nothing fundamentally new or path-breaking so far:

Hay algunas cosas a destacar aquí, pero hasta ahora nada fundamentalmente nuevo o innovador:

- We have declared an array of stocks at the class level, and initialized it with some default values in the `ngOnInit` block.

Hemos declarado una matriz de acciones a nivel de clase y la inicializamos con algunos valores predeterminados en el bloque `ngOnInit`.

- We have a function `onToggleFavorite` that logs the stock and toggles its favorite state.

Tenemos una función `onToggleFavorite` que registra la acción y alterna su estado favorito.

Let's now look at its corresponding template, in `src/app/stock/stock-list/stock-list.component.html`:

Veamos ahora su plantilla correspondiente, en `src/app/stock/stock-list/stock-list.component.html`:

```

<app-stock-item *ngFor="let stock of stocks" [stock]="stock"
  (toggleFavorite)="onToggleFavorite($event)">

```

```
</app-stock-item>
```

In the template, we simply loop over all the stocks and display an instance of the StockItemComponent for each one. We ask the StockItemComponent to trigger the `onToggleFavorite` whenever someone clicks the Add to Favorite or Remove from Favorite button within the stock item.

En la plantilla, simplemente recorremos todas las acciones y mostramos una instancia de StockItemComponent para cada una. Le pedimos al StockItemComponent que active el `onToggleFavorite` cada vez que alguien haga clic en el botón Agregar a favoritos o Eliminar de favoritos dentro del artículo en stock.

TIP CONSEJO

Since we generated the component using the Angular CLI, we can avoid the step of having to register the newly created component in the Angular module. Otherwise, there would be one more step here of adding the newly created component in the `app.module.ts` file in the `declarations` section.

Dado que generamos el componente usando Angular CLI, podemos evitar el paso de tener que registrar el componente recién creado en el módulo Angular. De lo contrario, habría un paso más aquí para agregar el componente recién creado en el archivo `app.module.ts` en la sección `declarations`.

The AppComponent can now be simplified further. The component class in `src/app/app.component.ts` can be modified to:

El AppComponent ahora se puede simplificar aún más. La clase de componente en `src/app/app.component.ts` se puede modificar para:

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
```

```

        styleUrls: ['./app.component.css']
    })
export class AppComponent implements OnInit {
    title = 'Stock Market App';

    ngOnInit(): void {
    }
}

```

The corresponding template in *src/app/app.component.html* can be modified to:

La plantilla correspondiente en *src/app/app.component.html* se puede modificar para:

```

<h1>
    {{title}}
</h1>
<app-stock-list></app-stock-list>
<app-create-stock></app-create-stock>

```

Now finally, let's look at what it takes to build a very simple and trivial StockService. We can again generate the base skeleton for the service using the Angular CLI as follows:

Ahora, finalmente, veamos lo que se necesita para construir un StockService muy simple y trivial. Podemos nuevamente generar el esqueleto base para el servicio usando Angular CLI de la siguiente manera:

```
ng g service services/stock
```

This will generate two files, a skeleton *stock-service.ts* and a dummy test for it in *stock-service.spec.ts*. We will ignore the latter for now but will return to it in [Chapter 10](#), as part of our discussion of unit testing of services. The generated skeleton in *src/app/service/stock.service.ts* should look something like this:

Esto generará dos archivos, un esqueleto *stock-service.ts* y una prueba ficticia en *stock-service.spec.ts*. Ignoraremos esto último por

ahora, pero volveremos a él en el Capítulo 10, como parte de nuestra discusión sobre las pruebas unitarias de servicios. El esqueleto generado en `src/app/service/stock.service.ts` debería verse así:

```
import { Injectable } from '@angular/core';

@Injectable()
export class StockService {

  constructor() { }

}
```

The skeleton is literally just an empty shell class, with one decorator of note, which is `Injectable`. The `Injectable` decorator has no current value for us, but is a recommended decorator whenever you are working with services, as it is a hint to the Angular dependency injection system that the service you are working on might have other dependencies. With the `Injectable` decorator, Angular will take care of injecting them into our service.

El esqueleto es literalmente solo una clase de shell vacía, con un decorador destacado, que es `Injectable`. El decorador `Injectable` no tiene ningún valor actual para nosotros, pero es un decorador recomendado siempre que trabaje con servicios, ya que es una pista para el sistema de inyección de dependencia Angular de que el servicio en el que está trabajando podría tener otras dependencias. Con el decorador `Injectable`, Angular se encargará de inyectarlos en nuestro servicio.

We will leave the decorator untouched, keeping with the best practices. And pretty soon, as early as the next chapter, we will need it anyway.

Dejaremos al decorador intacto, siguiendo las mejores prácticas. Y muy pronto, ya en el próximo capítulo, lo necesitaremos de todos modos.

Now, let's get to the crux of the work, which is the data that the StockService is to provide. This is where services really shine. Components generally will defer and ask a service for data (or a section of the data). It is up to the service to decide how and where to fetch the data from, whether it is from a web service via HTTP calls, a local storage or cache, or even return mock data, as we will in just a bit. Later, if we want to change the source, we can do it in one place without touching any of the components, as long as our API signature remains the same.

Ahora, vayamos al meollo del trabajo, que son los datos que debe proporcionar StockService. Aquí es donde realmente brillan los servicios. Los componentes generalmente aplazarán y solicitarán datos a un servicio (o una sección de los datos). Depende del servicio decidir cómo y dónde obtener los datos, ya sea de un servicio web a través de llamadas HTTP, un almacenamiento local o caché, o incluso devolver datos simulados, como lo haremos en un momento. Posteriormente, si queremos cambiar la fuente, podemos hacerlo en un solo lugar sin tocar ninguno de los componentes, siempre y cuando nuestra firma API siga siendo la misma.

Let's define our StockService to continue returning mock data. We will edit the `src/app/services/stock.service.ts` file as follows:

Definamos nuestro StockService para continuar devolviendo datos simulados. Editaremos el archivo `src/app/services/stock.service.ts` de la siguiente manera:

```
import { Injectable } from '@angular/core';
import { Stock } from 'app/model/stock';

@Injectable()
export class StockService {

  private stocks: Stock[];
  constructor() {
    this.stocks = [
      new Stock('Test Stock Company', 'TSC', 85, 80, 'NASDAQ'),
      new Stock('Second Stock Company', 'SSC', 10, 20, 'NSE'),
    ]
  }

  getStocks(): Stock[] {
    return this.stocks;
  }
}
```

```

        new Stock('Last Stock Company', 'LSC', 876, 765, 'NYSE')
    ];
}

getStocks() : Stock[] {
    return this.stocks;
}

createStock(stock: Stock) {
    let foundStock = this.stocks.find(each => each.code === stock.code);
    if (foundStock) {
        return false;
    }
    this.stocks.push(stock);
    return true;
}

toggleFavorite(stock: Stock) {
    let foundStock = this.stocks.find(each => each.code === stock.code);
    foundStock.favorite = !foundStock.favorite;
}
}

```

While it seems like we have added a lot of code, if we dig into it, there is nothing that is Angular specific in the code we added. Most of the added code is business functionality that we have defined and enforced via the `StockService`. Let's quickly walk through the major functionality we have introduced in the service:

Si bien parece que hemos agregado mucho código, si profundizamos en él, no hay nada que sea específico de Angular en el código que agregamos. La mayor parte del código agregado es funcionalidad comercial que hemos definido y aplicado a través de `StockService`. Repasemos rápidamente las funciones principales que hemos introducido en el servicio:

- We have moved our initialization of the mock list of stocks to the constructor of the `StockService`, initializing it with some dummy values to provide an initial state for our UI when it is rendered.

Hemos movido nuestra inicialización de la lista simulada de acciones al constructor de StockService, inicializándola con algunos valores ficticios para proporcionar un estado inicial para nuestra interfaz de usuario cuando se represente.

- We have defined a very simple `getStocks()` method that simply returns the current list of stocks.

Hemos definido un método `getStocks()` muy simple que simplemente devuelve la lista actual de acciones.

- The `createStock` method does little more than simply adding the stock to our list of stocks. It first checks if the stock already exists (using the `code` on the stock to check for uniqueness), exiting early if it does. If not found, it adds the passed-in stock to our list of stocks.

El método `createStock` hace poco más que simplemente agregar la acción a nuestra lista de acciones. Primero verifica si el stock ya existe (usando el `code` en el stock para verificar la unicidad), y sale temprano si es así. Si no se encuentra, agrega las acciones pasadas a nuestra lista de acciones.

- Finally, we have a `toggleFavorite`, which simply finds the passed-in stock in our array and then toggles the state of the `favorite` key on it.

Finalmente, tenemos un `toggleFavorite`, que simplemente encuentra el stock pasado en nuestra matriz y luego alterna el estado de la clave `favorite` en él.

Now that we have defined our service, let's see what it takes to be able to use it in our components. Before we can start injecting it into our components, we need to define how this service will be provided and at what level. We can define this at the `StockListComponent` level, the `AppComponent` level, or the `AppModule` level. We will see

what the difference is in a bit, but in the meantime, let's define it at the module level.

Ahora que hemos definido nuestro servicio, veamos qué se necesita para poder usarlo en nuestros componentes. Antes de que podamos comenzar a inyectarlo en nuestros componentes, debemos definir cómo se brindará este servicio y a qué nivel. Podemos definir esto en el nivel `StockListComponent`, el nivel `AppComponent` o el nivel `AppModule`. Veremos cuál es la diferencia en un momento, pero mientras tanto, definámosla a nivel de módulo.

Let's edit the `src/app/app.module.ts` file to define the provider as follows:

Editemos el archivo `src/app/app.module.ts` para definir el proveedor de la siguiente manera:

```
/** Imports same as before, skipped for brevity */
import { StockService } from 'app/services/stock.service';

@NgModule({
  declarations: [
    AppComponent,
    StockItemComponent,
    CreateStockComponent,
    StockListComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpModule
  ],
  providers: [
    StockService
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

①

① Registering the provider for `StockService`

We have just made a small addition to the `NgModule` decorator on the `AppModule`. We have registered an array of providers, in which the `StockService` is the one and only service right now. The `providers` array in the Angular module tells Angular to create a singleton instance of the service, and make it available for any class or component that asks for it. When we register it at the module level, it means that any component within the module that asks for the service will get the exact same instance injected into it.

Acabamos de hacer una pequeña adición al decorador `NgModule` en el `AppModule`. Hemos registrado una matriz de `providers`, en la que `StockService` es el único servicio en este momento. La matriz `providers` en el módulo Angular le dice a Angular que cree una instancia única del servicio y la ponga a disposición de cualquier clase o componente que lo solicite. Cuando lo registramos a nivel de módulo, significa que a cualquier componente dentro del módulo que solicite el servicio se le injectará exactamente la misma instancia.

TIP CONSEJO

We could have skipped the step of manually adding the service to the module by asking the Angular CLI to also perform this. The Angular CLI doesn't know at which level the service is supposed to operate, so it skips the service registration. If we wanted it to register it at the app module level, we could have executed it as:

Podríamos haber omitido el paso de agregar manualmente el servicio al módulo pidiéndole a Angular CLI que también lo realice. Angular CLI no sabe en qué nivel se supone que debe operar el servicio, por lo que omite el registro del servicio. Si quisiéramos que lo registrara a nivel de módulo de la aplicación, podríamos haberlo ejecutado como:

```
ng g service services/stock --module=app
```

This would have both generated the service and also registered the provider in the `AppModule`.

Esto habría generado el servicio y también habría registrado al proveedor en `AppModule`.

At this point, we are ready to start using the service, so first we will use it in the newly created `StockListComponent`. Let's change the `src/app/stock/stock-list/stock-list.component.ts` file as follows:

En este punto, estamos listos para comenzar a usar el servicio, por lo que primero lo usaremos en el `StockListComponent` recién creado. Cambiemos el archivo `src/app/stock/stock-list/stock-list.component.ts` de la siguiente manera:

```
import { Component, OnInit } from '@angular/core';
import { StockService } from 'app/services/stock.service';
import { Stock } from 'app/model/stock';

@Component({
  selector: 'app-stock-list',
  templateUrl: './stock-list.component.html',
  styleUrls: ['./stock-list.component.css']
})
export class StockListComponent implements OnInit {
```

```

public stocks: Stock[];
constructor(private stockService: StockService) { } ①

ngOnInit() {
  this.stocks = this.stockService.getStocks(); ②
}

onToggleFavorite(stock: Stock) {
  console.log('Favorite for stock ', stock, ' was triggered');
  this.stockService.toggleFavorite(stock); ③
}
}

```

- ① Injected the StockService into the component
- ② Used the StockService to get the list of stocks
- ③ Used the StockService to toggle the favorite status on a stock

This is our first instance of injecting and using a service, so let's walk through it step by step:

Esta es nuestra primera instancia de inyección y uso de un servicio, así que veámoslo paso a paso:

1. We can inject any service we want into our component simply by listing it in our constructor. In this case, we have declared a private instance of StockService with the name stockService. The name itself doesn't matter; Angular uses the type definition to figure out what service to inject. For all we care, we could even call the instance of the service xyz (not that you should!), and it would still get injected correctly.

Podemos注入 cualquier servicio que queramos en nuestro componente simplemente incluyéndolo en nuestro constructor. En este caso, hemos declarado una instancia privada de StockService con el nombre stockService. El nombre en sí no importa; Angular usa la definición de tipo para determinar qué servicio注入. Por lo que a nosotros respecta, incluso

podríamos llamar a la instancia del servicio xyz (no es que debas hacerlo!), y aún así se inyectaría correctamente.

2. We simply call the methods we want on our service through our instance (like `stockService.getStocks()` or `stockService.toggleFavorite()`) at the correct time. We initialize our list of stocks, and pass through the toggle call to the service. Note that we need to access the service through an instance variable and cannot access it directly (that is, we need to call `this.stockService`, and cannot directly use `stockService`).

Simplemente llamamos a los métodos que queremos en nuestro servicio a través de nuestra instancia (como `stockService.getStocks()` o `stockService.toggleFavorite()`) en el momento correcto. Inicializamos nuestra lista de acciones y pasamos a través de la llamada de alternancia al servicio. Tenga en cuenta que necesitamos acceder al servicio a través de una variable de instancia y no podemos acceder a él directamente (es decir, debemos llamar a `this.stockService` y no podemos usar `stockService` directamente).

While we don't need to make any changes to the corresponding template, here is what the template for the StockListComponent looks like in case you don't remember:

Si bien no necesitamos realizar ningún cambio en la plantilla correspondiente, así es como se ve la plantilla para StockListComponent en caso de que no lo recuerdes:

```
<app-stock-item *ngFor="let stock of stocks" [stock]="stock"
  (toggleFavorite)="onToggleFavorite($event)">
</app-stock-item>
```

TIP CONSEJO

We just used one of TypeScript's features in the preceding example to both declare a parameter as well as a property simultaneously. By adding the `private` or `public` keyword in front of a constructor argument, we can make it a member property of the class with the same name.

Acabamos de utilizar una de las funciones de TypeScript en el ejemplo anterior para declarar un parámetro y una propiedad simultáneamente. Al agregar la palabra clave `private` o `public` delante de un argumento del constructor, podemos convertirlo en una propiedad miembro de la clase con el mismo nombre.

With this, we need to make no more changes in our template. If we run the application at this point, we should see it running with a list of three stocks displayed at the top.

Con esto, no necesitamos hacer más cambios en nuestra plantilla. Si ejecutamos la aplicación en este punto, deberíamos verla ejecutándose con una lista de tres acciones en la parte superior.

Let's continue with the minor changes to the `CreateStockComponent` to finish the service integration. First, we'll add a simple `message` to the top of the `CreateStockComponent` template, to show a message to the user if the stock was created successfully or if there was any error in creation. We will edit `src/app/stock/create-stock/create-stock.component.html` as follows:

Continuaremos con los cambios menores en `CreateStockComponent` para finalizar la integración del servicio. Primero, agregaremos un simple `message` en la parte superior de la plantilla `CreateStockComponent`, para mostrar un mensaje al usuario si el stock se creó correctamente o si hubo algún error en la creación. Editaremos `src/app/stock/create-stock/create-stock.component.html` de la siguiente manera:

```

<h2>Create Stock Form</h2>

<div *ngIf="message">{{message}}</div>      ①
<div class="form-group">
  <!-- Rest of the form omitted for brevity -->
  <!-- No change from the base code -->
</div>

<h4>Stock Name is {{stock | json}}</h4>

```

① Display a message if it exists

The highlighted line is the only change in this file, which is simply adding a `div` that shows the `message` class variable if it has a value.

La línea resaltada es el único cambio en este archivo, que simplemente agrega un `div` que muestra la variable de clase `message` si tiene un valor.

Now, we can change the component in `src/app/stock/create-stock/create-stock.component.ts` to integrate with the `StockService`:

Ahora, podemos cambiar el componente en `src/app/stock/create-stock/create-stock.component.ts` para integrarlo con `StockService`:

```

import { Component, OnInit } from '@angular/core';
import { Stock } from 'app/model/stock';
import { StockService } from 'app/services/stock.service';

@Component({
  selector: 'app-create-stock',
  templateUrl: './create-stock.component.html',
  styleUrls: ['./create-stock.component.css']
})
export class CreateStockComponent {

  public stock: Stock;
  public confirmed = false;
  public message = null;          ①
  public exchanges = ['NYSE', 'NASDAQ', 'OTHER'];
  constructor(private stockService: StockService) { ②
    this.stock = new Stock('', '', 0, 0, 'NASDAQ');
  }
}

```

```

setStockPrice(price) {
  this.stock.price = price;
  this.stock.previousPrice = price;
}

createStock(stockForm) {
  if (stockForm.valid) {
    let created = this.stockService.createStock(this.stock); ③
    if (created) { ④
      this.message = 'Successfully created stock with stock code: '
        + this.stock.code;
      this.stock = new Stock('', '', 0, 0, 'NASDAQ');
    } else {
      this.message = 'Stock with stock code: ' + this.stock.code
        + ' already exists';
    }
  } else {
    console.error('Stock form is in an invalid state');
  }
}
}

```

- ① Agregue un campo de mensaje para mostrar mensajes de éxito y error
- ② Inyecte el servicio de acciones en la componente
- ③ Llame al servicio `createStock` cuando el formulario sea enviado
- ④ Defina un campo de mensaje para el escenario de creación exitosa de stock

The CreateStockComponent has been changed slightly to integrate with the StockService as follows:

El CreateStockComponent se ha modificado ligeramente para integrarse con el StockService de la siguiente manera:

- We created a message field to show useful messages to the user on successful creation of stock as well as errors while creating it.

Creamos un campo `message` para mostrar mensajes útiles al usuario sobre la creación exitosa de stock, así como errores al crearlo.

- We injected the `StockService` and then called it when the form is submitted by the user.

Inyectamos el `StockService` y luego lo llamamos cuando el usuario envía el formulario.

- We used the return value of the `StockService.createStock()` call to decide what message to show to the user in the UI.

Usamos el valor de retorno de la llamada `StockService.createStock()` para decidir qué mensaje mostrarle al usuario en la interfaz de usuario.

While we are at it, we can remove some of the CSS that makes the form hard to read from `src/app/stock/create-stock/create-stock.component.css`, and just empty the file.

Mientras estamos en esto, podemos eliminar parte del CSS que dificulta la lectura del formulario desde `src/app/stock/create-stock/create-stock.component.css` y simplemente vaciar el archivo.

Now when we run this application, it should still look the same as before. The difference is that when you fill the create stock form and click Create, it should clear the form, and you should see the stock you just entered like in [Figure 8-1](#).

Ahora, cuando ejecutemos esta aplicación, debería verse igual que antes. La diferencia es que cuando completa el formulario de creación de stock y hace clic en Crear, el formulario debería borrarse y debería ver el stock que acaba de ingresar, como en la Figura 8-1.

Now of course, one of the first things you might ask is "How did adding the stock to the list of stocks in the service make it appear in the `StockListComponent` magically?" It is a very valid question, and the answer to that is JavaScript! We return the reference to the

stock array in the `getStocks()` method, which is what the `StockListComponent` assigns to its member variable. Thus, any addition in the service automatically makes a change in the component that is holding on to the reference of the array.

Ahora, por supuesto, una de las primeras cosas que podría preguntarse es "¿Cómo es posible que al agregar la acción a la lista de acciones en el servicio aparezca mágicamente en `StockListComponent`?" Es una pregunta muy válida y la respuesta es JavaScript. Devolvemos la referencia a la matriz de valores en el método `getStocks()`, que es lo que `StockListComponent` asigna a su variable miembro. Por lo tanto, cualquier adición en el servicio realiza automáticamente un cambio en el componente que mantiene la referencia de la matriz.

This is still hacky in some sense, as we wouldn't want to rely on having the same reference to update the values, but we will fix this shortly. Also, because we are using mock data that is instantiated within the service when it is initialized, you will lose any new stocks you have created when you refresh the page. This is because the service is reloaded and reinitialized when we refresh the page, as we don't have a persistent store.

Esto sigue siendo complicado en cierto sentido, ya que no queremos depender de tener la misma referencia para actualizar los valores, pero lo solucionaremos en breve. Además, debido a que utilizamos datos simulados que se crean instancias dentro del servicio cuando se inicializa, perderá todas las existencias nuevas que haya creado cuando actualice la página. Esto se debe a que el servicio se recarga y reinicializa cuando actualizamos la página, ya que no tenemos un almacén persistente.

So just to recap, this is what we accomplished:

Para resumir, esto es lo que logramos:

- We created a `StockListComponent` to display a list of stocks.

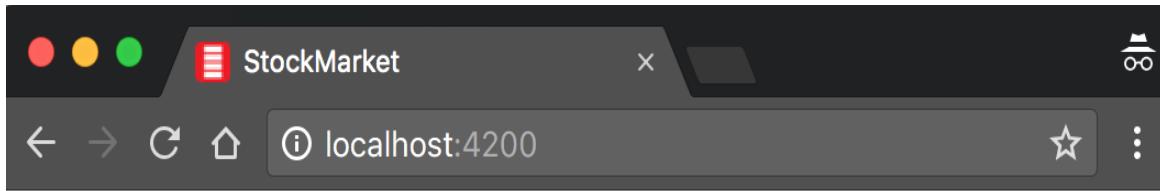
Creamos un StockListComponent para mostrar una lista de acciones.

- We created a StockService that acts as the layer behind all the components, providing APIs to fetch a list of stocks as well as to create new stocks. Currently, it is working off of mock data.

Creamos un StockService que actúa como capa detrás de todos los componentes, proporcionando API para obtener una lista de acciones y crear nuevas acciones. Actualmente, está trabajando con datos simulados.

- We made slight updates to all the components to integrate with this service.

Realizamos ligeras actualizaciones a todos los componentes para integrarlos con este servicio.



Stock Market App

Test Stock Company (TSC) NASDAQ \$ 85 <button>Add to Favorite</button>	Second Stock Company (SSC) NSE \$ 10 <button>Add to Favorite</button>
Last Stock Company (LSC) NYSE \$ 876 <button>Add to Favorite</button>	My New Stock (MNS) NASDAQ \$ 244 <button>Add to Favorite</button>

Create Stock Form

Successfully created stock with stock code: MNS

Stock Name is Mandatory

Stock Code is Mandatory

I confirm that the information provided above is accurate!

Stock Name is { "name": "", "code": "", "price": 0, "previousPrice": 0, "exchange": "NASDAQ", "favorite": false }

Data has been confirmed: true

Figure 8-1. Angular service sharing data between components

Figura 8-1. Servicio angular que comparte datos entre componentes.

We have dealt with a very basic and simple Angular service to show what an Angular service is, how it is created, and how it is hooked up and used within the context of an Angular application.

Nos hemos ocupado de un servicio Angular muy básico y simple para mostrar qué es un servicio Angular, cómo se crea y cómo se conecta y utiliza dentro del contexto de una aplicación Angular.

The completed version of this code is available in *chapter8/simple-service* in the GitHub repository.

La versión completa de este código está disponible en el capítulo 8/simple-service en el repositorio de GitHub.

An Introduction to Dependency Injection

Una introducción a la inyección de dependencia

Before we dive deeper into services and other related topics, let's take a step back to understand dependency injection, especially in context to Angular.

Antes de profundizar en los servicios y otros temas relacionados, retrocedamos un paso para comprender la inyección de dependencia, especialmente en el contexto de Angular.

Dependency injection started in static languages that are more common in server-side programming. In simple terms, dependency injection is the idea that any class or function should ask for its dependencies, rather than instantiating them themselves. Something else (usually called an injector) would be responsible for figuring out what is needed and how to instantiate it.

La inyección de dependencia comenzó en lenguajes estáticos que son más comunes en la programación del lado del servidor. En

términos simples, la inyección de dependencia es la idea de que cualquier clase o función debería solicitar sus dependencias, en lugar de crear una instancia de ellas por sí misma. Algo más (generalmente llamado inyector) sería responsable de descubrir qué se necesita y cómo crear una instancia.

Dependency injection has huge benefits when we practice it in our applications, as it allows us to create modular, reusable pieces while allowing us to test components and modules easily. Let's take a simple example to demonstrate how dependency injection can make your code more modular, easier to change, and testable:

La inyección de dependencia tiene enormes beneficios cuando la practicamos en nuestras aplicaciones, ya que nos permite crear piezas modulares y reutilizables al mismo tiempo que nos permite probar componentes y módulos fácilmente. Tomemos un ejemplo simple para demostrar cómo la inyección de dependencia puede hacer que su código sea más modular, más fácil de cambiar y comprobable:

```
class MyDummyService {  
  
    getMyData() {  
        let httpService = new HttpService();  
        return httpService.get('my/api');  
    }  
}  
  
class MyDIService {  
  
    constructor(private httpService: HttpService) {}  
  
    getMyData() {  
        return this.httpService.get('my/api');  
    }  
}
```

In this example, there is actually not much to differentiate between MyDummyService and MyDIService, except for the fact that one instantiates an HttpService before using it, while the other asks for

an instance of it in the constructor. But this small change allows for many things, including:

En este ejemplo, en realidad no hay mucho que diferenciar entre `MyDummyService` y `MyDIService`, excepto por el hecho de que uno crea una instancia de un `HttpService` antes de usarlo, mientras que el otro solicita una instancia del mismo. en el constructor. Pero este pequeño cambio permite muchas cosas, entre ellas:

- It makes it more obvious what is necessary for each service to actually execute, rather than finding out at the time of execution.

Hace que sea más obvio qué es necesario para que cada servicio se ejecute realmente, en lugar de averiguarlo en el momento de la ejecución.

- In our example, instantiating the `HttpService` was trivial, but it might not be in certain cases. In such a case, every user of `HttpService` will need to know exactly how to create it and configure it, before using it.

En nuestro ejemplo, crear una instancia de `HttpService` fue trivial, pero puede que no lo sea en ciertos casos. En tal caso, cada usuario de `HttpService` necesitará saber exactamente cómo crearlo y configurarlo antes de usarlo.

- In our test, we might not want to make actual HTTP calls. There, we can replace and instantiate `MyDIService` with a fake `HttpService` that does not make real calls, while there is nothing we can do for `MyDummyService`.

En nuestra prueba, es posible que no queramos realizar llamadas HTTP reales. Allí, podemos reemplazar y crear una instancia de `MyDIService` con un `HttpService` falso que no realiza llamadas reales, mientras que no hay nada que podamos hacer por `MyDummyService`.

There are many more advantages of dependency injection in general, and the [official Angular documentation has a great article](#) that covers this in depth if you want to read further on this.

Hay muchas más ventajas de la inyección de dependencia en general, y la documentación oficial de Angular tiene un excelente artículo que cubre esto en profundidad si desea leer más sobre esto.

Angular and Dependency Injection

Inyección angular y de dependencia

In the previous section, we covered dependency injection in general. In this section, we will dig deeper into how Angular has set up its dependency injection system, and the major things developers should be aware of. We will not go into each and every detail in this section, but cover the general aspects we expect most developers to encounter in their day-to-day work.

En la sección anterior, cubrimos la inyección de dependencia en general. En esta sección, profundizaremos en cómo Angular ha configurado su sistema de inyección de dependencia y los aspectos más importantes que los desarrolladores deben tener en cuenta. No entraremos en todos y cada uno de los detalles en esta sección, pero cubriremos los aspectos generales que esperamos que la mayoría de los desarrolladores encuentren en su trabajo diario.

We have seen Angular's dependency injection at work already, with the very first service that we created. For very simple asks, it is enough to think about Angular's dependency injection service as a very simple key-value store, with the ability of any component or class to ask for a key when they are getting initialized. In reality, it is much more complex than a simple key-value store. We will see how to leverage this dependency injection system in our unit tests in [Chapter 10](#).

Ya hemos visto la inyección de dependencia de Angular en funcionamiento, con el primer servicio que creamos. Para preguntas muy simples, basta con pensar en el servicio de inyección de dependencia de Angular como un almacén clave-valor muy simple, con la capacidad de cualquier componente o clase de solicitar una clave cuando se inicializa. En realidad, es mucho más complejo que un simple almacén clave-valor. Veremos cómo aprovechar este sistema de inyección de dependencia en nuestras pruebas unitarias en el Capítulo 10.

Every service that we create needs to be registered as a provider with an injector. Then any other class can ask for the service and the injector will be responsible for providing it. However, as briefly mentioned in the previous section, Angular doesn't have just one injector—instead, it has a whole hierarchy of injectors.

Cada servicio que creamos debe estar registrado como proveedor con un inyector. Luego cualquier otra clase puede solicitar el servicio y el inyector será el responsable de brindarlo. Sin embargo, como se mencionó brevemente en la sección anterior, Angular no tiene un solo inyector, sino que tiene toda una jerarquía de inyectores.

At its root, at the application level, Angular has the root module and the root injector. When we created our service, and registered it in the providers section of the NgModule, the service was in fact registered with the root injector. This would mean that the service is a singleton for the entire application, and that any class or component in the application can ask for the service and would be handed the very same instance of the service.

En su raíz, a nivel de aplicación, Angular tiene el módulo raíz y el inyector raíz. Cuando creamos nuestro servicio y lo registramos en la sección providers de NgModule, el servicio de hecho se registró con el inyector raíz. Esto significaría que el servicio es un singleton para toda la aplicación y que cualquier clase o componente de la

aplicación puede solicitar el servicio y se le entregará la misma instancia del servicio.

Let's now understand Angular's hierarchical dependency injection system by taking our example and modifying it to make the dependency injection system apparent. We can use the base code from the previous section in case you are not coding along, which is available in *chapter8/simple-service*. We will continue from there.

Ahora comprendamos el sistema de inyección de dependencia jerárquica de Angular tomando nuestro ejemplo y modificándolo para que el sistema de inyección de dependencia sea evidente. Podemos usar el código base de la sección anterior en caso de que no esté codificando, que está disponible en el capítulo 8/simple-service. Continuaremos desde allí.

Before we get into the code, let's cover what exactly will we be trying to accomplish. We will:

Antes de entrar en el código, cubramos qué es exactamente lo que intentaremos lograr. Lo haremos:

- Add a new service called `MessageService`. This will be used to display messages at various points in the UI, and also as a mechanism to communicate across services.

Agregue un nuevo servicio llamado `MessageService`. Esto se utilizará para mostrar mensajes en varios puntos de la interfaz de usuario y también como un mecanismo para comunicarse entre servicios.

- Introduce and use the `MessageService` in both the `CreateStockComponent` as well as the `AppComponent`.

Introduzca y utilice el `MessageService` tanto en el `CreateStockComponent` como en el `AppComponent`.

First, let's create the `MessageService`. This time, we will use the Angular CLI to create it completely for us, including registering it

with its provider in the `AppModule`. Just execute:

Primero, creamos el `MessageService`. Esta vez, usaremos Angular CLI para crearlo completamente para nosotros, incluido registrarlo con su proveedor en `AppModule`. Simplemente ejecuta:

```
ng g service services/message --module=app
```

WARNING ADVERTENCIA

Make sure you don't miss the `--module=app` argument to the command. In case you do, just open the `app.module.ts` file and manually add the `MessageService` to the list of providers in the `NgModule`.

Asegúrese de no perderse el argumento `--module=app` del comando. En caso de que lo haga, simplemente abra el archivo `app.module.ts` y agregue manualmente el `MessageService` a la lista de providers en el `NgModule`.

This will create a skeleton `MessageService` in the `services` folder, and register it with the `providers` section of the `AppModule`. Let's now update the `src/app/services/message.service.ts` file as follows:

Esto creará un esqueleto `MessageService` en la carpeta de servicios y lo registrará en la sección `providers` de `AppModule`. Ahora actualicemos el archivo `src/app/services/message.service.ts` de la siguiente manera:

```
import { Injectable } from '@angular/core';

@Injectable()
export class MessageService {

  public message: string = null;

  constructor() { }
}
```

MessageService is nothing but a very simple container to hold a message string. It is publicly available, so any class or component can simply reach out and access or change it.

MessageService no es más que un contenedor muy simple para contener una cadena message. Está disponible públicamente, por lo que cualquier clase o componente puede simplemente acceder a él o cambiarlo.

Now, let's modify the AppComponent to use this service and display the current message in the UI. Modify the *src/app/app.component.ts* file as follows:

Ahora, modifiquemos el AppComponent para usar este servicio y mostrar el message actual en la interfaz de usuario. Modifique el archivo *src/app/app.component.ts* de la siguiente manera:

```
import { Component, OnInit } from '@angular/core';
import { MessageService } from 'app/services/message.service';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit {
  title = 'app works!';

  constructor(public messageService: MessageService) {}

  ngOnInit(): void {
    this.messageService.message = 'Hello Message Service!';
  }
}
```

We simply injected our new MessageService into the AppComponent, and while initializing set it to some default value. Notice that we made the default access for the messageService public, so that we can use it from the template. Let's now look at the template in *src/app/app.component.html*:

Simplemente注入我们的新MessageService到AppComponent中，在初始化时，我们可以在某个预定义的值中配置它。请记住，我们为messageService public化了，以便可以从模板中使用它。现在让我们看看src/app/app.component.html：

```
<h1>
  {{title}}
</h1>
<h3>App level: {{messageService.message}}</h3>
<app-stock-list></app-stock-list>
<app-create-stock></app-create-stock>
```

We just added one line, which is an h3 element that shows the current value of the message variable in the MessageService that is injected into our AppComponent.

Acabamos de agregar una línea, que es un elemento h3 que muestra el valor actual de la variable message en el MessageService que se injecta en nuestro AppComponent.

Next, we'll modify the CreateStockComponent to use the same service. Modify the *src/app/stock/create-stock/create-stock.component.ts* file as follows:

A continuación, modificaremos el CreateStockComponent para utilizar el mismo servicio. Modifique el archivo *src/app/stock/create-stock/create-stock.component.ts* de la siguiente manera:

```
import { Component, OnInit } from '@angular/core';
import { Stock } from 'app/model/stock';
import { StockService } from 'app/services/stock.service';
import { MessageService } from 'app/services/message.service';

@Component({
  selector: 'app-create-stock',
  templateUrl: './create-stock.component.html',
  styleUrls: ['./create-stock.component.css'],
  providers: [MessageService]
```

```

})
export class CreateStockComponent {

  public stock: Stock;
  public confirmed = false;
  public exchanges = ['NYSE', 'NASDAQ', 'OTHER'];
  constructor(private stockService: StockService,
              public messageService: MessageService) { ❶
    this.stock = new Stock('', '', 0, 0, 'NASDAQ');
  }

  setStockPrice(price) {
    this.stock.price = price;
    this.stock.previousPrice = price;
  }

  createStock(stockForm) {
    if (stockForm.valid) {
      let created = this.stockService.createStock(this.stock);
      if (created) { ❷
        this.messageService.message =
          'Successfully created stock with stock code: ' +
          this.stock.code;
        this.stock = new Stock('', '', 0, 0, 'NASDAQ');
      } else {
        this.messageService.message = 'Stock with stock code: ' +
          this.stock.code + ' already exists';
      }
    } else {
      console.error('Stock form is in an invalid state');
    }
  }
}

```

❶ Injected `MessageService` into the constructor

❷ Used `MessageService` both times for creation.

Most of the code remains unchanged from before. We just injected our new service into the class, and then used it instead using `console.log`. We set the `message` variable in the `MessageService` when the stock is created successfully or when there is an error.

La mayor parte del código permanece sin cambios respecto al anterior. Simplemente inyectamos nuestro nuevo servicio en la clase y luego lo usamos usando `console.log`. Configuramos la variable `message` en `MessageService` cuando el stock se crea correctamente o cuando hay un error.

Finally, we will use the `MessageService` in the template of the component here as well to display the same message. Let's modify `src/app/stock/create-stock/create-stock.component.html` as follows:

Finalmente, aquí también usaremos `MessageService` en la plantilla del componente para mostrar el mismo mensaje. Modifiquemos `src/app/stock/create-stock/create-stock.component.html` de la siguiente manera:

```
<h2>Create Stock Form</h2>

<div>{{messageService.message}}</div>
<div class="form-group">
  <form (ngSubmit)="createStock(stockForm)" #stockForm="ngForm">
    <div class="stock-name">
      <input type="text"
        placeholder="Stock Name"
        required
        name="stockName"
        #stockName="ngModel"
        [(ngModel)]="stock.name">
    </div>
  </form>
<!-- Remaining code as before, omitting for brevity -->
```

We have omitted most of the file, as the only change we did was add line 3, where we displayed the current value of `messageService.message` in the UI.

Hemos omitido la mayor parte del archivo, ya que el único cambio que hicimos fue agregar la línea 3, donde mostramos el valor actual de `messageService.message` en la interfaz de usuario.

Now when we run this, we should see our application, but with two new lines. One is the second line, which is part of our AppComponent, and one within the CreateStockComponent, both of which are displaying the initial value of the message in the UI. If you now fill up the form and create a stock, you will notice that both the messages change simultaneously. Hence, we are assured that there is only one instance of the MessageService, which is being shared across both the components.

Ahora, cuando ejecutemos esto, deberíamos ver nuestra aplicación, pero con dos líneas nuevas. Una es la segunda línea, que forma parte de nuestro AppComponent, y otra dentro de CreateStockComponent, las cuales muestran el valor inicial del mensaje en la interfaz de usuario. Si ahora completa el formulario y crea una acción, notará que ambos mensajes cambian simultáneamente. Por lo tanto, tenemos la seguridad de que solo hay una instancia de MessageService, que se comparte entre ambos componentes.

Now, let's modify the CreateStockComponent slightly as follows:

Ahora, modifiquemos ligeramente el CreateStockComponent de la siguiente manera:

```
import { Component, OnInit } from '@angular/core';
import { Stock } from 'app/model/stock';
import { StockService } from 'app/services/stock.service';
import { MessageService } from 'app/services/message.service';

@Component({
  selector: 'app-create-stock',
  templateUrl: './create-stock.component.html',
  styleUrls: ['./create-stock.component.css'],
  providers: [MessageService] ①
})
export class CreateStockComponent {

  public stock: Stock;
  public confirmed = false;
  public exchanges = ['NYSE', 'NASDAQ', 'OTHER'];
}
```

```

constructor(private stockService: StockService,
            public messageService: MessageService) {
    this.stock = new Stock('', '', 0, 0, 'NASDAQ');
    this.messageService.message = 'Component Level: Hello Message Service';
②
}

setStockPrice(price) {
    this.stock.price = price;
    this.stock.previousPrice = price;
}

createStock(stockForm) {
    /* Code as before, no change */
    /* Omitted for brevity */
}
}

```

- ① Agregando una declaración providers para MessageService
- ② Agregando un valor inicial al MessageService en el componente

We've made one addition to the @Component decorator for the CreateStockComponent class. We have added a providers declaration, and provided the MessageService at the component level. We also added an initial value to the MessageService in the constructor. We will not make any other changes. We will also leave the MessageService declared in the providers section of the main module.

Hicimos una adición al decorador @Component para la clase CreateStockComponent. Agregamos una declaración providers y proporcionamos el MessageService a nivel de componente. También agregamos un valor inicial a MessageService en el constructor. No haremos ningún otro cambio. También dejaremos el MessageService declarado en la sección providers del módulo principal.

Before we talk about what happens as a result of this, it is worth executing the application and seeing it yourself. Execute and note

the following:

Antes de hablar de lo que sucede como resultado de esto, vale la pena ejecutar la aplicación y verlo usted mismo. Ejecute y observe lo siguiente:

1. See the message in the AppComponent, which is the default we had set.

Vea el mensaje en AppComponent, que es el valor predeterminado que habíamos configurado.

2. See the message in the CreateStockComponent, which will be the value we set in the component in the constructor "Component Level: Hello Message Service".

Vea el mensaje en CreateStockComponent, que será el valor que estableceremos en el componente en el constructor "Component Level: Hello Message Service".

3. Now fill up the form and create a stock.

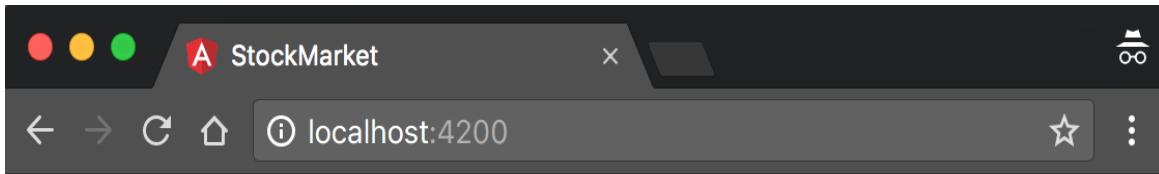
Ahora complete el formulario y cree una acción.

4. Note that the message is updated only in the CreateStockComponent, but the AppComponent message does not change.

Tenga en cuenta que el mensaje se actualiza solo en CreateStockComponent, pero el mensaje AppComponent no cambia.

When you do all of this, you should see something like [Figure 8-2](#).

Cuando haga todo esto, debería ver algo como la Figura 8-2.



Stock Market App

App level: Hello Message Service!

Test Stock Company (TSC) NASDAQ \$ 85 <input type="button" value="Add to Favorite"/>	Second Stock Company (SSC) NSE \$ 10 <input type="button" value="Add to Favorite"/>
Last Stock Company (LSC) NYSE \$ 876 <input type="button" value="Add to Favorite"/>	Test (TSS) NASDAQ \$ 44 <input type="button" value="Add to Favorite"/>

Create Stock Form

Successfully created stock with stock code: TSS

Stock Name

Stock Name is Mandatory

Stock Code

Stock Code is Mandatory

NASDAQ

I confirm that the information provided above is accurate!

Stock Name is { "name": "", "code": "", "price": 0, "previousPrice": 0, "exchange": "NASDAQ", "favorite": false }

Data has been confirmed: true

Figure 8-2. Angular application with services defined at component level

Figura 8-2. Aplicación angular con servicios definidos a nivel de componente.

This is Angular's hierarchical dependency injection at play. As mentioned before, Angular supports multiple dependency injectors within the same application. There is the root injector at the root AppModule level, which is where most services you create will be registered and made available. This makes the instance available across your entire application, and this was happening initially.

Esta es la inyección de dependencia jerárquica de Angular en juego. Como se mencionó anteriormente, Angular admite múltiples inyectores de dependencia dentro de la misma aplicación. Existe el inyector raíz en el nivel raíz AppModule, que es donde la mayoría de los servicios que cree se registrarán y estarán disponibles. Esto hace que la instancia esté disponible en toda la aplicación, y esto ocurría inicialmente.

Then, when we made the change to add the providers at the CreateStockComponent level, we brought the injector at the component level into play. Angular will create a chain of injectors all the way down, depending on the need and declarations. And all child components will inherit that injector, which will take precedence over the root injector. When we registered our MessageService provider at the CreateStockComponent level, it created a child injector at that level, with its own instance of the MessageService. What we injected into the CreateStockComponent is in fact this new instance, which has nothing to do with the original root-level MessageService. So we, in fact, have two separate instances of MessageService with nothing in common.

Luego, cuando hicimos el cambio para agregar providers en el nivel CreateStockComponent, pusimos en juego el inyector en el nivel de componente. Angular creará una cadena de inyectores hasta el final, según la necesidad y las declaraciones. Y todos los componentes secundarios heredarán ese inyector, que tendrá prioridad sobre el

inyector raíz. Cuando registramos nuestro proveedor `MessageService` en el nivel `CreateStockComponent`, creó un inyector secundario en ese nivel, con su propia instancia de `MessageService`. Lo que inyectamos en `CreateStockComponent` es, de hecho, esta nueva instancia, que no tiene nada que ver con el nivel raíz original `MessageService`. Entonces, de hecho, tenemos dos instancias separadas de `MessageService` sin nada en común.

Just like an Angular application is a tree of components, there is a parallel tree of injectors at play. For most components, these injectors might just be a reference or a proxy to the parent injector. But they may not, as we have just seen.

Así como una aplicación Angular es un árbol de componentes, hay un árbol paralelo de inyectores en juego. Para la mayoría de los componentes, estos inyectores pueden ser simplemente una referencia o un proxy del inyector principal. Pero puede que no sea así, como acabamos de ver.

Whenever a component asks for a dependency, Angular will check the closest injector in the tree to see whether it can satisfy it. If it can (like in `CreateStockComponent`), it will provide it. If not, it will check with the parent injector, all the way to the root injector.

Siempre que un componente solicite una dependencia, Angular comprobará el inyector más cercano en el árbol para ver si puede satisfacerla. Si puede (como en `CreateStockComponent`), lo proporcionará. De lo contrario, lo comprobará con el inyector principal, hasta llegar al inyector raíz.

Now when will or can you use this? Usually, it won't even matter and you will just register all your services at the root level. But there might be cases when you want to use this capability of the Angular injector, such as:

Ahora, ¿cuándo podrás o podrás usar esto? Por lo general, ni siquiera importará y simplemente registrará todos sus servicios en el

nivel raíz. Pero puede haber casos en los que desee utilizar esta capacidad del inyector Angular, como por ejemplo:

- You want to scope services to only certain components and modules, to ensure that a change in the service doesn't impact the entire application.

Desea limitar los servicios a determinados componentes y módulos para garantizar que un cambio en el servicio no afecte a toda la aplicación.

- You might want different instances of a service in different components. This might be common where you might want to override things, or have different services for different instances, and the like.

Es posible que desee diferentes instancias de un servicio en diferentes componentes. Esto puede ser común cuando desee anular cosas o tener diferentes servicios para diferentes instancias, etc.

- You want to override a particular service with a more specific implementation for a section of your application. Maybe you want a cached version of your service for a certain section as opposed to the general pass-through implementation.

Quiere anular un servicio en particular con una implementación más específica para una sección de su aplicación. Tal vez desee una versión en caché de su servicio para una determinada sección en lugar de la implementación general de transferencia.

There is more detail in how we can instantiate a service and the various ways we can provide them. You can read up on all the details and ways in the [official Angular docs](#).

Hay más detalles sobre cómo podemos crear instancias de un servicio y las diversas formas en que podemos brindarlos. Puede leer todos los detalles y formas en los documentos oficiales de Angular.

The completed version of this code is available in *chapter8/di-example* in the GitHub repository.

La versión completa de este código está disponible en el capítulo 8/di-example en el repositorio de GitHub.

RxJS and Observables: Moving to Asynchronous Operations

RxJS y Observables: pasando a operaciones asincrónicas

The last change we will make before we conclude this particular chapter on services is to work with asynchronous code. The data that we returned from our service in our example was hardcoded, mock data. We returned it as is, and displayed it immediately. But in a real-world application, that would not be the case, as most of the times we would be fetching the data from a server.

El último cambio que haremos antes de concluir este capítulo particular sobre servicios es trabajar con código asincrónico. Los datos que devolvimos de nuestro servicio en nuestro ejemplo eran datos simulados codificados. Lo devolvimos tal cual y lo mostramos inmediatamente. Pero en una aplicación del mundo real, ese no sería el caso, ya que la mayoría de las veces estaríamos obteniendo los datos de un servidor.

Handling responses and data from a server becomes slightly different from what we have done. We might make a call to fetch the list of stocks from the server. And then at some later point in time, after our server finishes processing the request, we would be provided with a list of stocks. Thus, the stocks would not be available immediately on demand, but at a later point in time, asynchronously.

El manejo de respuestas y datos de un servidor se vuelve ligeramente diferente de lo que hemos hecho hasta ahora. Podríamos hacer una llamada para recuperar la lista de acciones del servidor. Y luego, en algún momento posterior, después de que nuestro servidor termine de procesar la solicitud, se nos proporcionará una lista de existencias. Por lo tanto, las existencias no estarían disponibles inmediatamente a pedido, sino en un momento posterior, de forma asincrónica.

In AngularJS, we used to handle these situations using promises. Promises were a better way to handle asynchronous behavior than callbacks, which was the traditional way, **for multiple reasons**. That said, there are a few drawbacks that Angular tries to do away with, by switching to using *observables*.

En AngularJS, solíamos manejar estas situaciones mediante promesas. Las promesas eran una mejor manera de manejar el comportamiento asincrónico que las devoluciones de llamada, que era la forma tradicional, por múltiples razones. Dicho esto, hay algunos inconvenientes que Angular intenta eliminar cambiando al uso de observables.

Observables are a **ReactiveX** concept that allows us to deal with streams which emit data. Any interested party can then be an observer on this stream, and perform operations and transformations on the events emitted by the stream.

Los observables son un concepto de ReactiveX que nos permite tratar con flujos que emiten datos. Cualquier parte interesada puede ser observador de esta secuencia y realizar operaciones y transformaciones en los eventos emitidos por la secuencia.

There are a few differences between observables and promises, primarily:

Existen algunas diferencias entre observables y promesas, principalmente:

- Promises operate on a single asynchronous event, while observables allow us to deal with a stream of zero or more asynchronous events.

Las promesas operan en un único evento asincrónico, mientras que los observables nos permiten lidiar con un flujo de cero o más eventos asincrónicos.

- Unlike promises, observables can be canceled. That is, a promise's success or error handler will eventually be called, while we can cancel a subscription and not process data if we don't care about it.

A diferencia de las promesas, los observables se pueden cancelar. Es decir, eventualmente se llamará al controlador de éxito o error de una promesa, mientras que podemos cancelar una suscripción y no procesar datos si no nos importa.

- Observables allow us to compose and create a chain of transformations easily. The operators it provides out of the box allow for some strong and powerful compositions, and operations like retry and replay make handling some common use cases trivial. All of this while being able to reuse our subscription code.

Los observables nos permiten componer y crear una cadena de transformaciones fácilmente. Los operadores que proporciona de fábrica permiten algunas composiciones sólidas y poderosas, y operaciones como reintentar y reproducir hacen que el manejo de algunos casos de uso comunes sea trivial. Todo ello pudiendo reutilizar nuestro código de suscripción.

That said, promises are good for single event cases, and are still an option when you work with Angular. An observable can be converted into a promise and then handled in Angular. But it is recommended to use observables, as Angular provides a lot of out-of-the-box support for RxJS and its extensions within its framework.

Dicho esto, las promesas son buenas para casos de eventos únicos y siguen siendo una opción cuando se trabaja con Angular. Un observable se puede convertir en una promesa y luego manejar en Angular. Pero se recomienda utilizar observables, ya que Angular proporciona una gran cantidad de soporte listo para usar para RxJS y sus extensiones dentro de su marco.

LEARNING RXJS AND OBSERVABLES APRENDIZAJE DE RXJS Y OBSERVABLES

We will touch upon the bare minimum of observables and RxJS features in this book, but this book itself is not intended to be a way to learn RxJS and reactive programming in general. There are excellent tutorials and books available, starting with the [official documentation for ReactiveX](#).

En este libro abordaremos el mínimo de observables y características de RxJS, pero este libro en sí no pretende ser una forma de aprender RxJS y la programación reactiva en general. Hay excelentes tutoriales y libros disponibles, comenzando con la documentación oficial de ReactiveX.

Let's now take our example and move it step by step to using observables and make it ready for our future use cases. You can use the base codebase from *chapter8/simple-service* if you have not been coding along.

Ahora tomemos nuestro ejemplo y avancemos paso a paso para usar observables y prepararlo para nuestros casos de uso futuros. Puede utilizar el código base del capítulo 8/simple-service si no ha estado codificando.

First, we will modify our StockService to start returning an asynchronous observable to get us ready for the future when we start integrating with servers. Then we will modify our components to subscribe to these observables and deal with success and error cases.

Primero, modificaremos nuestro StockService para comenzar a devolver un observable asíncrono para prepararnos para el futuro cuando comencemos a integrarnos con los servidores. Luego modificaremos nuestros componentes para suscribirnos a estos observables y lidiar con casos de éxito y error.

Let's change the `src/app/services/stock.service.ts` file as follows:

Cambiemos el archivo `src/app/services/stock.service.ts` de la siguiente manera:

```
import { Injectable } from '@angular/core';

import { Observable } from 'rxjs/Observable'; ❶
import { _throw as ObservableThrow } from 'rxjs/observable/throw'; ❷
import { of as ObservableOf } from 'rxjs/observable/of';
import { Stock } from 'app/model/stock';

@Injectable()
export class StockService {

  private stocks: Stock[];
  constructor() {
    this.stocks = [
      new Stock('Test Stock Company', 'TSC', 85, 80, 'NASDAQ'),
      new Stock('Second Stock Company', 'SSC', 10, 20, 'NSE'),
      new Stock('Last Stock Company', 'LSC', 876, 765, 'NYSE')
    ];
  }

  getStocks(): Observable<Stock[]> { ❸
    return ObservableOf(this.stocks); ❹
  }

  createStock(stock: Stock): Observable<any> {
    let foundStock = this.stocks.find(each => each.code === stock.code);
    if (foundStock) {
      return ObservableThrow({msg: 'Stock with code ' + ❺
        stock.code + ' already exists'});
    }
    this.stocks.push(stock);
    return ObservableOf({msg: 'Stock with code ' + stock.code +
      ' successfully created'});;
  }
}
```

```

    toggleFavorite(stock: Stock): Observable<Stock> {
      let foundStock = this.stocks.find(each => each.code === stock.code);
      foundStock.favorite = !foundStock.favorite;
      return Observable.of(foundStock);
    }
}

```

- ① Importing observable
- ② Importing methods from the Observable API like `of` and `throw`
- ③ Changing the return type of `getStock` to an observable
- ④ Returning an observable for data that's pulled
- ⑤ Throwing an exception to the observer

We completely overhauled the `StockService`, so let's walk through the major changes one by one so we understand what changed and more importantly, why:

Revisamos completamente el `StockService`, así que repasemos los cambios principales uno por uno para entender qué cambió y, lo que es más importante, por qué:

- The first thing we do is import `Observable` from the RxJS library. Note that we import operators and classes individually from the respective files, rather than import the entire RxJS library.
- Lo primero que hacemos es importar `Observable` de la biblioteca RxJS. Tenga en cuenta que importamos operadores y clases individualmente desde los archivos respectivos, en lugar de importar toda la biblioteca RxJS.
- We then make sure to import and add the operators we are planning to use from RxJS to ensure we have them available in our application. In this case, we are simply planning to use the `of` and `throw` operators on the core `Observable` class.

Luego nos aseguramos de importar y agregar los operadores que planeamos usar desde RxJS para asegurarnos de tenerlos disponibles en nuestra aplicación. En este caso, simplemente planeamos utilizar los operadores `of` y `throw` en la clase principal `Observable`.

- We then change the return type of each of the methods in the service to return an observable instead of a synchronous value. This is to ensure a consistent API interface to the user of the service. Once we make this change, we can change the implementation underneath (say, changing from mock data to making a server call) without having to change each and every component.

Luego cambiamos el tipo de retorno de cada uno de los métodos del servicio para devolver un valor observable en lugar de un valor sincrónico. Esto es para garantizar una interfaz API coherente para el usuario del servicio. Una vez que realizamos este cambio, podemos cambiar la implementación subyacente (por ejemplo, cambiar de datos simulados a realizar una llamada al servidor) sin tener que cambiar todos y cada uno de los componentes.

- For the time being, we convert our return value to an observable by using the `Observable.of` operator. The `of` takes a value and returns an observable of that type which is triggered only once.

Por el momento, convertimos nuestro valor de retorno en un observable usando el operador `Observable.of`. El `of` toma un valor y devuelve un observable de ese tipo que se activa solo una vez.

- In the `createStock` method, we also change the functionality to throw an exception on the observable if the stock already exists.

En el método `createStock`, también cambiamos la funcionalidad para generar una excepción en el observable si el stock ya existe.

TIP CONSEJO

Instead of importing each class and operator from RxJS manually, we also have the option to import the entire RxJS library and access the classes and operators through it, like so:

En lugar de importar cada clase y operador desde RxJS manualmente, también tenemos la opción de importar toda la biblioteca RxJS y acceder a las clases y operadores a través de ella, así:

```
import { Rx } from 'rxjs/Rx';
```

We would then be able to access `Rx.Observable` and so on. But this comes with a downside, which is that Angular won't be able to optimize your build, as it would not be able to understand which parts of RxJS are being used at the time of compilation. RxJS as a library is large, and most applications would only end up using bits and pieces of it.

Entonces podremos acceder a `Rx.Observable` y así sucesivamente. Pero esto tiene una desventaja, y es que Angular no podrá optimizar su compilación, ya que no podrá comprender qué partes de RxJS se están utilizando en el momento de la compilación. RxJS como biblioteca es grande y la mayoría de las aplicaciones solo terminarían usando fragmentos de ella.

Thus I would always recommend using individual imports as a general practice.

Por lo tanto, siempre recomendaría utilizar importaciones individuales como práctica general.

Now let's change the components to integrate with the new asynchronous APIs in the service. First, we will change the `StockListComponent`, to read the list of stocks from the observable instead of reading the array directly. We will change the `src/app/stock/stock-list/stock-list.component.ts` file as follows:

Ahora cambiemos los componentes para integrarlos con las nuevas API asíncronas del servicio. Primero, cambiaremos StockListComponent para leer la lista de acciones del observable en lugar de leer la matriz directamente. Cambiaremos el archivo src/app/stock/stock-list/stock-list.component.ts de la siguiente manera:

```
/** Imports skipped for brevity */
export class StockListComponent implements OnInit {

  public stocks: Stock[];
  constructor(private stockService: StockService) { }

  ngOnInit() {
    this.stockService.getStocks()
      .subscribe(stocks => {
        this.stocks = stocks;
      });
  }

  onToggleFavorite(stock: Stock) {
    this.stockService.toggleFavorite(stock);
  }
}
```

We have made one small change, which is to the ngOnInit block of the component. Instead of directly assigning the return value of the stockService.getStocks() call to the stocks array, we now instead subscribe to the observable that it returns. The observable gets triggered with the array of stocks once, at which point we assign the value to our local array. There is no change as such to the onToggleFavorite, though we should also subscribe to the observable it returns for proper handling.

Hemos realizado un pequeño cambio, que es en el bloque ngOnInit del componente. En lugar de asignar directamente el valor de retorno de la llamada stockService.getStocks() a la matriz stocks, ahora nos suscribimos al observable que devuelve. El observable se activa con la matriz de acciones una vez, momento en el que

asignamos el valor a nuestra matriz local. No hay ningún cambio como tal en `onToggleFavorite`, aunque también debemos suscribirnos al observable que devuelve para un manejo adecuado.

Let's also change the `CreateStockComponent` and see how the `src/app/stock/create-stock/create-stock.component.ts` file changes:

Cambiemos también el `CreateStockComponent` y veamos cómo cambia el archivo `src/app/stock/create-stock/create-stock.component.ts`:

```
/** Imports skipped for brevity */
export class CreateStockComponent {

    /** No changes, skipping for brevity */

    createStock(stockForm) {
        if (stockForm.valid) {
            this.stockService.createStock(this.stock)
                .subscribe((result: any) => {    ①
                    this.message = result.msg;
                    this.stock = new Stock('', '', 0, 0, 'NASDAQ');
                }, (err) => {
                    this.message = err.msg;
                });
        } else {
            console.error('Stock form is in an invalid state');
        }
    }
}
```

① Subscribing to the observable

Again, most of our changes are focused on one method, the `createStock`. Instead of handling all the work immediately after triggering the `stockService.createStock()`, we now subscribe to the observable. In the previous case, we just handled the success case, but the `subscribe` method allows us to take two functions as arguments. The first argument is called in the case of a successful call, while the second is the error handler callback.

Nuevamente, la mayoría de nuestros cambios se centran en un método, el `createStock`. En lugar de manejar todo el trabajo inmediatamente después de activar `stockService.createStock()`, ahora nos suscribimos al observable. En el caso anterior, solo manejamos el caso de éxito, pero el método `subscribe` nos permite tomar dos funciones como argumentos. El primer argumento se llama en caso de una llamada exitosa, mientras que el segundo es la devolución de llamada del controlador de errores.

Both of our flows return an object with a `msg` key, so we handle it accordingly and update the `message` with the value returned.

Nuestros dos flujos devuelven un objeto con una clave `msg`, por lo que lo manejamos en consecuencia y actualizamos el `message` con el valor devuelto.

At this point, we can now run our application to see it working. When you run it, you shouldn't see any difference in the functionality, but all our stocks should be visible and you should be able to add stocks.

En este punto, ya podemos ejecutar nuestra aplicación para verla funcionando. Cuando lo ejecute, no debería ver ninguna diferencia en la funcionalidad, pero todas nuestras acciones deberían estar visibles y debería poder agregar acciones.

Let's alter our code one final time to make it simpler and easier to read. In a lot of cases, we simply want to make a call to our server, and display the return value in our UI. We don't need to process the data, make any transformations, or anything else. In those cases, Angular gives us a slight shortcut that we can use.

Modifiquemos nuestro código por última vez para hacerlo más simple y fácil de leer. En muchos casos, simplemente queremos realizar una llamada a nuestro servidor y mostrar el valor de retorno en nuestra interfaz de usuario. No necesitamos procesar los datos,

realizar ninguna transformación ni nada más. En esos casos, Angular nos da un pequeño atajo que podemos usar.

We'll change the `src/app/stock/stock-list/stock-list.component.ts` file first:

Primero cambiaremos el archivo `src/app/stock/stock-list/stock-list.component.ts`:

```
/** Imports skipped for brevity */
export class StockListComponent implements OnInit {

  public stocks$: Observable<Stock[]>;           ❶
  constructor(private stockService: StockService) { }

  ngOnInit() {
    this.stocks$ = this.stockService.getStocks();        ❷
  }

  onToggleFavorite(stock: Stock) {
    this.stockService.toggleFavorite(stock);
  }
}
```

❶ Storing the observable as a member variable.

❷ Calling `getStocks()` directly on the observable.

We have made two changes to our `StockListComponent` class:

Hemos realizado dos cambios en nuestra clase `StockListComponent`:

- Instead of having an array of stocks as a member variable, we now have an `Observable<Stock[]>` as the member. That is, we are saving the observable that the API returns directly, instead of its underlying return value.

En lugar de tener una matriz de stocks como variable miembro, ahora tenemos un `Observable<Stock[]>` como miembro. Es decir, estamos guardando el observable que la API devuelve directamente, en lugar de su valor de retorno subyacente.

- In ngOnInit, we simply save the observable returned by our stockService.getStocks() call.

En ngOnInit, simplemente guardamos el observable devuelto por nuestra llamada stockService.getStocks().

Given this, how do we display our array of stocks in the template? How do we handle this asynchronous behavior? Let's look at what we can do in the template to take care of this:

Teniendo esto en cuenta, ¿cómo mostramos nuestra variedad de acciones en la plantilla? ¿Cómo manejamos este comportamiento asincrónico? Veamos qué podemos hacer en la plantilla para solucionar esto:

```
<app-stock-item *ngFor="let stock of stocks$ | async"
  [stock]="stock"
  (toggleFavorite)="onToggleFavorite($event)">
</app-stock-item>
```

We have made one tweak here, which is to use a Pipe in the ngFor expression. Angular provides a pipe called `async`, which allows us to bind to Observable. Angular would then be responsible for waiting for events to be emitted on the observable and displaying the resultant value directly. It saves us that one step of having to manually subscribe to the observable.

Hemos realizado un ajuste aquí, que consiste en utilizar un Pipe en la expresión ngFor. Angular proporciona una tubería llamada `async`, que nos permite vincularnos a Observable. Angular entonces sería responsable de esperar a que se emitan eventos en el observable y mostrar el valor resultante directamente. Nos ahorra ese paso de tener que suscribirnos manualmente al observable.

Again, this is useful in only a few situations where the data returned by an API call is something we can directly display. But it does save a few lines of code, leaving it to the framework to handle most of the boilerplate. Run your application to make sure that this still

works, and you should see the same application running without any issues.

Nuevamente, esto es útil solo en algunas situaciones en las que los datos devueltos por una llamada API son algo que podemos mostrar directamente. Pero guarda algunas líneas de código, dejando que el marco maneje la mayor parte del texto estándar. Ejecute su aplicación para asegurarse de que todavía funcione y debería ver la misma aplicación ejecutándose sin ningún problema.

The completed version of this code is available in *chapter8/observables* in the GitHub repository.

La versión completa de este código está disponible en el capítulo 8/observables en el repositorio de GitHub.

Conclusion

Conclusión

In this chapter, we started understanding Angular services in more detail. We covered what Angular services are and some common uses for them, primarily:

En este capítulo, comenzamos a comprender los servicios Angular con más detalle. Cubrimos qué son los servicios Angular y algunos usos comunes de ellos, principalmente:

- Abstraction of the data-fetching aspects
Abstracción de los aspectos de obtención de datos.
- Encapsulation of shared application logic
Encapsulación de lógica de aplicación compartida
- Sharing of data across components

Compartir datos entre componentes

We also discussed Angular's dependency injection system and how its hierarchical dependency injection works with an example. Finally, we looked at how Observable works and how to integrate very simple observables into our application.

También analizamos el sistema de inyección de dependencia de Angular y cómo funciona su inyección de dependencia jerárquica con un ejemplo. Finalmente, vimos cómo funciona Observable y cómo integrar observables muy simples en nuestra aplicación.

In the next chapter, we will dig into how to work with and make HTTP calls and deal with their responses. We will also cover some common use cases that we have when working with servers and how to build solutions for them.

En el próximo capítulo, profundizaremos en cómo trabajar y realizar llamadas HTTP y manejar sus respuestas. También cubriremos algunos casos de uso comunes que tenemos cuando trabajamos con servidores y cómo crear soluciones para ellos.

Exercise

Ejercicio

Take the finished exercise from [Chapter 6](#) (available in `chapter6/exercise/e-commerce`). Try to accomplish the following:

Realice el ejercicio terminado del Capítulo 6 (disponible en el capítulo 6/ejercicio/comercio electrónico). Intente lograr lo siguiente:

1. Create a common service backing the `ProductListComponent` and the `CreateProductComponent` called `ProductService`.

Cree un servicio común que respalte el `ProductListComponent` y el `CreateProductComponent` llamado `ProductService`.

2. Make the components simple and move any logic into the service. Register the service correctly at the module level (either using the CLI or manually).

Simplifique los componentes y traslade cualquier lógica al servicio. Registre el servicio correctamente a nivel de módulo (ya sea usando la CLI o manualmente).

3. Start from the beginning with observables and make all the components deal with asynchronous APIs.

Comience desde el principio con observables y haga que todos los componentes traten con API asincrónicas.

4. Use the `async` pipe where possible instead of manually subscribing to the results.

Utilice la canalización `async` siempre que sea posible en lugar de suscribirse manualmente a los resultados.

All of this can be accomplished using concepts covered in this chapter. You can check out the finished solution in `chapter8/exercise/ecommerce`.

Todo esto se puede lograr utilizando los conceptos tratados en este capítulo. Puede consultar la solución terminada en el capítulo 8/ejercicio/ecommerce.

Chapter 9. Making HTTP Calls in Angular

Capítulo 9. Realizar Llamadas HTTP en Angular

In the previous chapter, we started our groundwork on Angular services. In particular, we took a look at what Angular services are and when to use them. We then dealt with creating Angular services and using them in our application, followed by a very cursory glance at dealing with asynchronous behavior in Angular using observables.

En el capítulo anterior, comenzamos nuestro trabajo preliminar sobre los servicios Angular. En particular, analizamos qué son los servicios Angular y cuándo usarlos. Luego nos ocupamos de la creación de servicios Angular y su uso en nuestra aplicación, seguido de un vistazo muy superficial a cómo lidiar con el comportamiento asincrónico en Angular usando observables.

In this chapter, we will build on that base and start using the built-in Angular modules and services to make and parse HTTP calls to a server. We will use that to explore common paradigms, the API options, and how to chain and really use the power of observables in our application.

En este capítulo, nos basaremos en esa base y comenzaremos a utilizar los módulos y servicios integrados de Angular para realizar y analizar llamadas HTTP a un servidor. Lo usaremos para explorar

paradigmas comunes, las opciones de API y cómo encadenar y usar realmente el poder de los observables en nuestra aplicación.

Introducing HttpClient

Presentamos HttpClient

In this section, we will start using Angular's HttpClient to make GET and POST calls to a server. Through this, we will see how to set up our application so that we can make the calls, walk through the process of actually making the calls and dealing with the response, and then go into the API signature and all the various options that we have to tweak it to our needs.

En esta sección, comenzaremos a usar HttpClient de Angular para realizar llamadas GET y POST a un servidor. A través de esto, veremos cómo configurar nuestra aplicación para que podamos realizar las llamadas, recorrer el proceso de realizar las llamadas y manejar la respuesta, y luego entraremos en la firma API y todas las opciones que tenemos. para adaptarlo a nuestras necesidades.

As for the server, we won't be spending any time building it out, but rather using a prebuilt server for this application. It is a Node.js server, and available in the repository in case you are interested in digging deeper into it, but it is not required to understand this part of the book.

En cuanto al servidor, no dedicaremos tiempo a construirlo, sino que utilizaremos un servidor prediseñado para esta aplicación. Es un servidor Node.js y está disponible en el repositorio en caso de que esté interesado en profundizar en él, pero no es necesario comprender esta parte del libro.

HTTPCLIENT VERSUS HTTP

HTTPCLIENT FRENTE A HTTP

If you happened upon some older tutorials and examples, you might encounter a slightly different way of making HTTP calls, by directly importing from `@angular/http` and then making calls. This was the old way of working with HTTP in Angular, before `HttpClient` was introduced in Angular version 4.3. In version 5 of Angular, the old `http` service was deprecated in favor of `HttpClient`, so just use the method described in this chapter in your applications.

Si se topó con algunos tutoriales y ejemplos más antiguos, es posible que encuentre una forma ligeramente diferente de realizar llamadas HTTP, importando directamente desde `@angular/http` y luego realizando llamadas. Esta era la antigua forma de trabajar con HTTP en Angular, antes de que se introdujera `HttpClient` en la versión 4.3 de Angular. En la versión 5 de Angular, el antiguo servicio `http` quedó obsoleto en favor de `HttpClient`, así que utilice el método descrito en este capítulo en sus aplicaciones.

We will continue building on our application, and try to move it over to communicate with a real server instead of our mock data that it was working with so far. In particular, in this section, we will switch over all three service calls (getting a list of stocks, creating a stock, and toggling the favorite on a stock level) to server calls using HTTP GET/POST. By the end of this section, we should not be operating with any mock data on our client side.

Continuaremos desarrollando nuestra aplicación e intentaremos moverla para comunicarnos con un servidor real en lugar de nuestros datos simulados con los que estaba trabajando hasta ahora. En particular, en esta sección, cambiaremos las tres llamadas de servicio (obtener una lista de acciones, crear una acción y alternar el favorito en un nivel de acción) a llamadas al servidor usando HTTP GET/POST. Al final de esta sección, no deberíamos estar operando con ningún dato simulado en nuestro lado del cliente.

Server Setup

Configuración del servidor

As mentioned, the server we will be working with is already developed and available in the repository in the *chapter9/server* folder. Before we start any web development, let's get our server up and running.

Como se mencionó, el servidor con el que trabajaremos ya está desarrollado y disponible en el repositorio en la carpeta capítulo9/servidor. Antes de comenzar cualquier desarrollo web, pongamos en funcionamiento nuestro servidor.

Checkout and browse to the *chapter9/server* folder in the [GitHub repository](#). From within the folder, execute the following commands in your terminal :

Pague y busque la carpeta capítulo9/servidor en el repositorio de GitHub. Desde dentro de la carpeta, ejecute los siguientes comandos en su terminal:

```
npm i  
node index.js
```

This installs all the necessary dependencies for our Node.js server, and then starts the server on port 3000. Keep this server running in the background; don't kill it. This will be what our application hits to fetch and save stocks.

Esto instala todas las dependencias necesarias para nuestro servidor Node.js y luego inicia el servidor en el puerto 3000. Mantenga este servidor ejecutándose en segundo plano; no lo mates. Esto será lo que nuestra aplicación haga para buscar y guardar acciones.

WARNING ADVERTENCIA

Note that this is a very simplistic server with an in-memory data store. Anything you create or save will be reset if you restart the server.

Tenga en cuenta que este es un servidor muy simplista con un almacén de datos en memoria. Todo lo que cree o guarde se restablecerá si reinicia el servidor.

Using HttpClientModule

Usando HttpClientModule

In case you are not coding along, you can get the base code from the *chapter8/observables* folder in the GitHub repository.

En caso de que no esté codificando, puede obtener el código base de la carpeta capítulo8/observables en el repositorio de GitHub.

Now let's get to our web application, and see step by step how to convert our local web application to talk to a server and fetch its data. While doing this, we will also see how easy this switch is, since we are already using observables in our code.

Ahora vayamos a nuestra aplicación web y veamos paso a paso cómo convertir nuestra aplicación web local para comunicarse con un servidor y recuperar sus datos. Mientras hacemos esto, también veremos qué tan fácil es este cambio, ya que ya estamos usando observables en nuestro código.

The very first thing we will do is add a dependency on `HttpClientModule` in our `AppModule`. Let's modify the `src/app/app.module.ts` file as follows:

Lo primero que haremos será agregar una dependencia de `HttpClientModule` en nuestro `AppModule`. Modifiquemos el archivo `src/app/app.module.ts` de la siguiente manera:

```

import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/common/http'; ①

import { AppComponent } from './app.component';
import { StockItemComponent } from './stock/stock-item/stock-item.component';
import { CreateStockComponent } from './stock/create-stock/create-
stock.component';
import { StockListComponent }
  from './stock/stock-list/stock-list.component';
import { StockService } from 'app/services/stock.service';

@NgModule({
  declarations: [
    AppComponent,
    StockItemComponent,
    CreateStockComponent,
    StockListComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpClientModule ②
  ],
  providers: [
    StockService,
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

- ① Import the `HttpClientModule` instead of `HttpModule`
- ② Add `HttpClientModule` to the imports array

Making HTTP GET/POST Calls

Realizar Llamadas HTTP GET/POST

Next, we will change the implementation of our `StockService` to actually make an HTTP service call instead of just returning an observable of mock data. To do this, we get the `HttpClient` service

injected into the constructor (thanks, Angular dependency injection!), and then use it to make our calls. Let's see how we can modify the `src/app/services/stock.service.ts` file:

A continuación, cambiaremos la implementación de nuestro StockService para realizar una llamada de servicio HTTP en lugar de simplemente devolver un observable de datos simulados. Para hacer esto, inyectamos el servicio HttpClient en el constructor (igual que la inyección de dependencia angular!) y luego lo usamos para realizar nuestras llamadas. Veamos cómo podemos modificar el archivo `src/app/services/stock.service.ts`:

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

import { Observable } from 'rxjs/Observable';

import { Stock } from 'app/model/stock';

@Injectable()
export class StockService {

  constructor(private http: HttpClient) {}

  getStocks(): Observable<Stock[]> {
    return this.http.get<Stock[]>('/api/stock');
  }

  createStock(stock: Stock): Observable<any> {
    return this.http.post('/api/stock', stock);
  }

  toggleFavorite(stock: Stock): Observable<Stock> {
    return this.http.patch<Stock>('/api/stock/' + stock.code,
      {
        favorite: !stock.favorite
      });
  }
}
```

Our server exposes three APIs:

Nuestro servidor expone tres API:

- GET on `/api/stock` to get a list of stocks
OBTÉN `/api/stock` para obtener una lista de acciones
- POST on `/api/stock` with the new stock as a body to create a stock on the server
PUBLICAR en `/api/stock` con el nuevo stock como cuerpo para crear un stock en el servidor
- PATCH on `/api/stock/:code` with the stock code in the URL and the new favorite status in the body of the request, to change the state of favorite for the particular stock.
PATCH en `/api/stock/:code` con el código de stock en la URL y el nuevo estado de favorito en el cuerpo de la solicitud, para cambiar el estado de favorito para el stock en particular.

Our StockService mirrors this API, with each of the three methods making the respective call. The HttpClient APIs directly mirror the HTTP methods, as we can call `httpClient.get`, `httpClient.post`, and `httpClient.patch` directly. Each of them take the URL as the first argument, and a request body as the second (if the method supports it).

Nuestro StockService refleja esta API, con cada uno de los tres métodos realizando la llamada respectiva. Las API HttpClient reflejan directamente los métodos HTTP, ya que podemos llamar a `httpClient.get`, `httpClient.post` y `httpClient.patch` directamente. Cada uno de ellos toma la URL como primer argumento y el cuerpo de la solicitud como segundo (si el método lo admite).

One important thing to note is that the HttpClient can give you type-assurance across your code. We leverage this feature in the `getStocks()` and the `toggleFavorite()` methods.

Una cosa importante a tener en cuenta es que HttpClient puede brindarle seguridad de tipo en todo su código. Aprovechamos esta

característica en los métodos `getStocks()` y `toggleFavorite()`.

One effect of this is that we need to change our `stock.ts` from a TypeScript class to a TypeScript interface. Why is this? While we don't need to, Angular does a simple typecast of the response body into the type we have defined. But TypeScript (and ECMAScript underneath it) has no nice and easy way to convert a simple plain-old JavaScript object into a prototypical JavaScript/TypeScript class object. This means that while our response from `StockService` will have all the properties of the class `Stock`, it would not have the functions (in particular, `isPositiveChange()`) available.

Un efecto de esto es que necesitamos cambiar nuestro archivo `stock.ts` de una clase TypeScript a una interfaz TypeScript. ¿Por qué es esto? Si bien no es necesario, Angular realiza una conversión simple del cuerpo de la respuesta al tipo que hemos definido. Pero TypeScript (y ECMAScript debajo de él) no tiene una manera fácil y agradable de convertir un simple objeto JavaScript antiguo en un objeto de clase JavaScript/TypeScript prototípico. Esto significa que si bien nuestra respuesta de `StockService` tendrá todas las propiedades de la clase `Stock`, no tendrá las funciones (en particular, `isPositiveChange()`) disponibles.

We could write a converter, but it is only worth it in very specific cases. It is easier to simply leverage TypeScript for type safety and work with other ways of encapsulation (either at a component level, or maybe as an Angular service).

Podríamos escribir un conversor, pero sólo merece la pena en casos muy concretos. Es más fácil simplemente aprovechar TypeScript para la seguridad de tipos y trabajar con otras formas de encapsulación (ya sea a nivel de componente o tal vez como un servicio Angular).

For these reasons, let's switch our `Stock` class to an interface, by editing `src/app/model/stock.ts` as follows:

Por estos motivos, cambiemos nuestra clase Stock a una interfaz, editando src/app/model/stock.ts de la siguiente manera:

```
export interface Stock {
  name: string;
  code: string;
  price: number;
  previousPrice: number;
  exchange: string;
  favorite: boolean;
}
```

We have simply converted it to an interface, and defined all the properties on it. No more constructor, no more built-in functions. With that groundwork done, let's now move to changing the components over to using the service correctly. We will first start with StockListComponent, which has minimal changes. All we will do is remove the toggling of favorites functionality away from this component, and let each individual stock component handle it. Let's see the changes to *src/app/stock/stock-list/stock-list.component.ts*:

Simplemente lo convertimos en una interfaz y definimos todas las propiedades en él. No más constructores, no más funciones integradas. Una vez hecho el trabajo preliminar, pasemos a cambiar los componentes para utilizar el servicio correctamente. Primero comenzaremos con StockListComponent, que tiene cambios mínimos. Todo lo que haremos es eliminar la funcionalidad de alternancia de favoritos de este componente y dejar que cada componente de stock individual se encargue de ello. Veamos los cambios en src/app/stock/stock-list/stock-list.component.ts:

```
import { Component, OnInit } from '@angular/core';
import { StockService } from 'app/services/stock.service';
import { Stock } from 'app/model/stock';
import { Observable } from 'rxjs/Observable';

@Component({
  selector: 'app-stock-list',
  templateUrl: './stock-list.component.html',
```

```

    styleUrls: ['./stock-list.component.css']
  })
export class StockListComponent implements OnInit {

  public stocks$: Observable<Stock[]>;
  constructor(private stockService: StockService) { }

  ngOnInit() {
    this.stocks$ = this.stockService.getStocks();
  }
}

```

Our StockListComponent becomes simplified with this change. We also have to make a slight tweak to its template, to remove the toggleFavorite event binding. Change *src/app/stock/stock-list/stock-list.component.html* as follows:

Nuestro StockListComponent se simplifica con este cambio. También tenemos que hacer un ligero ajuste a su plantilla para eliminar el enlace del evento `toggleFavorite`. Cambie *src/app/stock/stock-list/stock-list.component.html* de la siguiente manera:

```

<app-stock-item *ngFor="let stock of stocks$ | async"
  [stock]="stock">
</app-stock-item>

```

Next, let's move on to the StockItemComponent. We will move the `toggleFavorite` logic into the component here, and ask each individual stock to make the respective server call through `StockService.toggleFavorite` directly, and handle the response. We will remove the `EventEmitter` while we are at it. The finished *src/app/stock/stock-item/stock-item.component.ts* file should look like this:

A continuación, pasemos al StockItemComponent. Moveremos la lógica `toggleFavorite` al componente aquí y le pediremos a cada acción individual que realice la llamada al servidor respectivo a través de `StockService.toggleFavorite` directamente y manejaremos la respuesta. Eliminaremos el `EventEmitter` mientras

estamos en ello. El archivo src/app/stock/stock-item/stock-item.component.ts terminado debería verse así:

```
import { Component, OnInit, Input } from '@angular/core';

import { Stock } from '../model/stock';
import { StockService } from 'app/services/stock.service';

@Component({
  selector: 'app-stock-item',
  templateUrl: './stock-item.component.html',
  styleUrls: ['./stock-item.component.css']
})
export class StockItemComponent {

  @Input() public stock; ❶

  constructor(private stockService: StockService) {} ❷

  onToggleFavorite(event) { ❸
    this.stockService.toggleFavorite(this.stock)
      .subscribe((stock) => this.stock.favorite = !this.stock.favorite);
  }
}
```

- ❶ Only toggle favorite on the client side
- ❷ Inject stock service into the constructor
- ❸ Change onToggleFavorite for a call to service instead

Note that we are taking responsibility for toggling the local state of favorite on the stock on successful toggle favorite call. Without it, it would change the state on the server, but the local browser state would not be changed. We could also have chosen to refresh the entire list of stocks on every toggle favorite, by keeping the EventEmiter and asking the parent StockListComponent to refetch the list of stocks every time. That is a call we can make as we develop our applications, depending on our needs. There will be times when we want the latest and greatest information from the server, and times when handling changes locally is acceptable.

Tenga en cuenta que asumimos la responsabilidad de alternar el estado local de `favorite` en la acción si se realiza con éxito la llamada favorita. Sin él, cambiaría el estado del servidor, pero no se cambiaría el estado del navegador local. También podríamos haber optado por actualizar la lista completa de acciones en cada favorito de alternancia, manteniendo el `EventEmitter` y pidiendo al padre `StockListComponent` que vuelva a buscar la lista de acciones cada vez. Ésa es una decisión que podemos hacer a medida que desarollamos nuestras aplicaciones, según nuestras necesidades. Habrá momentos en los que queramos obtener la información más reciente y mejor del servidor, y momentos en los que sea aceptable manejar los cambios localmente.

There is also a change in our template for the `StockItemComponent`, as we no longer have access to the `isPositiveChange()` function on the stock. We instead directly calculate whether it is positive or not using the underlying properties in our template. Our changed `src/app/stock/stock-item/stock-item.component.html` should look as follows:

También hay un cambio en nuestra plantilla para `StockItemComponent`, ya que ya no tenemos acceso a la función `isPositiveChange()` en la acción. En su lugar, calculamos directamente si es positivo o no utilizando las propiedades subyacentes en nuestra plantilla. Nuestro `src/app/stock/stock-item/stock-item.component.html` modificado debería tener el siguiente aspecto:

```
<div class="stock-container">
  <div class="name">{{stock.name + ' (' + stock.code + ')'}}</div>
  <div class="exchange">{{stock.exchange}}</div>
  <div class="price"
    [class.positive]="stock.price > stock.previousPrice" ①
    [class.negative]="stock.price <= stock.previousPrice"> ②
    $ {{stock.price}}>
  </div>
  <button (click)="onToggleFavorite($event)"
    *ngIf="!stock.favorite">Add to Favorite</button>
```

```

<button (click)="onToggleFavorite($event)"
         *ngIf="stock.favorite">Remove from Favorite</button>
</div>

```

- ❶ Agregar la clase `highlighted` basada en el precio de las acciones
- ❷ Agregar la clase `highlighted` basada en el precio de las acciones

Now, we will move to the last component, the `CreateStockComponent`. We only have to accommodate the change over from the class to the interface, which means we don't have a nice constructor for a `Stock` object. While we are at it, we will refactor the class for reuse. But there is no change required with respect to `HttpClient`, since we abstracted out the logic and used observables from the beginning. The finished `src/app/stock/create-stock/create-stock.component.ts` file should look like this:

Ahora, pasaremos al último componente, el `CreateStockComponent`. Solo tenemos que acomodar el cambio de la clase a la interfaz, lo que significa que no tenemos un buen constructor para un objeto `Stock`. Mientras lo hacemos, refactorizaremos la clase para su reutilización. Pero no se requiere ningún cambio con respecto a `HttpClient`, ya que abstrajimos la lógica y usamos observables desde el principio. El archivo `src/app/stock/create-stock/create-stock.component.ts` terminado debería verse así:

```

import { Component, OnInit } from '@angular/core';
import { Stock } from 'app/model/stock';
import { StockService } from 'app/services/stock.service';

@Component({
  selector: 'app-create-stock',
  templateUrl: './create-stock.component.html',
  styleUrls: ['./create-stock.component.css']
})
export class CreateStockComponent {

  public stock: Stock;
  public confirmed = false;
  public message = null;
}

```

```

public exchanges = ['NYSE', 'NASDAQ', 'OTHER'];
constructor(private stockService: StockService) {
  this.initializeStock();           ①
}

initializeStock() {               ②
  this.stock = {
    name: '',
    code: '',
    price: 0,
    previousPrice: 0,
    exchange: 'NASDAQ',
    favorite: false
  };
}

setStockPrice(price) {
  this.stock.price = price;
  this.stock.previousPrice = price;
}

createStock(stockForm) {
  if (stockForm.valid) {
    this.stockService.createStock(this.stock)
      .subscribe((result: any) => {
        this.message = result.msg;
        this.initializeStock();          ③
      }, (err) => {
        this.message = err.error.msg;
      });
  } else {
    console.error('Stock form is in an invalid state');
  }
}
}

```

- ① Creamos la función initializeStock para crear el stock en la instancia de stock
- ② Definir el método de stock en la clase para su reutilización
- ③ Usar initializeStock después de que el stock sea ya creado correctamente

We just pulled out an `initializeStock` function, which we call both from the constructor as well as after the stock is successfully

created. The only other change we did in this class is how we handled the error. With `HttpResponse` for errors, the response body is actually available in the `error` key inside the response, so we grab the `msg` key from there instead. There are no other changes we need to do in this class. The template also remains as is.

Acabamos de sacar una función `initializeStock`, a la que llamamos tanto desde el constructor como después de que el stock se haya creado correctamente. El único otro cambio que hicimos en esta clase es cómo manejamos el error. Con `HttpResponse` para errores, el cuerpo de la respuesta en realidad está disponible en la clave `error` dentro de la respuesta, por lo que tomamos la clave `msg` de allí. No hay otros cambios que debamos hacer en esta clase. La plantilla también permanece tal cual.

We are now almost ready to run our application, but we have one final change we need to make. The browser, for security reasons, does not allow you to make calls across domains and origins. Thus, even while both our server and the Angular application are running on localhost, they are running on different ports, and thus the browser treats them as different origins. To get around this, the Angular CLI allows us to set up a proxy, so that our requests would be sent to the server, which would then proxy it to our final endpoint.

Ahora estamos casi listos para ejecutar nuestra aplicación, pero tenemos que realizar un último cambio. El navegador, por motivos de seguridad, no permite realizar llamadas entre dominios y orígenes. Por lo tanto, aunque tanto nuestro servidor como la aplicación Angular se ejecutan en localhost, se ejecutan en puertos diferentes y, por lo tanto, el navegador los trata como orígenes diferentes. Para solucionar esto, Angular CLI nos permite configurar un proxy, de modo que nuestras solicitudes se envíen al servidor, que luego las enviará a nuestro punto final.

To do this, we will create a file *proxy.conf.json* in the main folder of our Angular application, with the following contents:

Para ello crearemos un archivo proxy.conf.json en la carpeta principal de nuestra aplicación Angular, con el siguiente contenido:

```
{
  "/api": {
    "target": "http://localhost:3000",
    "secure": false
  }
}
```

What we have done is simply asked Angular to proxy all requests made to the server with the path starting with */api* to our Node.js server running on port 3000 locally. Technically, this filename can be anything, but we are just following a certain pattern. This file supports a lot more configuration, but we will not get into it in this book. You can read up on it in the [official Angular CLI docs](#).

Lo que hemos hecho es simplemente pedirle a Angular que envíe todas las solicitudes realizadas al servidor con la ruta que comienza con */api* a nuestro servidor Node.js que se ejecuta en el puerto 3000 localmente. Técnicamente, este nombre de archivo puede ser cualquier cosa, pero solo seguimos un patrón determinado. Este archivo admite muchas más configuraciones, pero no lo abordaremos en este libro. Puede leer más sobre esto en los documentos oficiales de Angular CLI.

Now we are finally ready to run our application. So far, we have been running the application using `ng serve`. Now, we need to run it with the following command:

Ahora finalmente estamos listos para ejecutar nuestra aplicación. Hasta ahora, hemos estado ejecutando la aplicación usando `ng serve`. Ahora necesitamos ejecutarlo con el siguiente comando:

```
ng serve --proxy-config proxy.conf.json
```

This will let Angular run our application, but taking our proxy configuration into account. Now, when you browse to our application (<http://localhost:4200>), you should see the list of stocks coming from the server. Note that if you create a new stock, you will see a message saying it has been added successfully, but you will have to manually refresh the page to see it updated in the UI. Toggling the stock should also work, and this should be persisted even after you refresh the page. The application should look exactly the same as it has been looking so far, as we have not made any UI-specific changes.

Esto permitirá que Angular ejecute nuestra aplicación, pero teniendo en cuenta nuestra configuración de proxy. Ahora, cuando navegue a nuestra aplicación (<http://localhost:4200>), debería ver la lista de acciones provenientes del servidor. Tenga en cuenta que si crea una nueva acción, verá un mensaje que indica que se agregó correctamente, pero tendrá que actualizar manualmente la página para verla actualizada en la interfaz de usuario. Alternar el stock también debería funcionar, y esto debería persistir incluso después de actualizar la página. La aplicación debería verse exactamente igual a como se ha visto hasta ahora, ya que no hemos realizado ningún cambio específico en la interfaz de usuario.

The finished code sample for this section is available in the GitHub repository in the *chapter9/simple-http* folder.

El ejemplo de código terminado para esta sección está disponible en el repositorio de GitHub en la carpeta Chapter9/simple-http.

Advanced HTTP

HTTP avanzado

In the previous section, we started with the basics of Angular's `HttpClient` and learned how to make simple HTTP GET and POST

calls to our server, and deal with its response. In this section, we will dig a little bit deeper into our HTTP API, and other features that are supported by Angular's HTTP module.

En la sección anterior, comenzamos con los conceptos básicos de HttpClient de Angular y aprendimos cómo realizar llamadas HTTP GET y POST simples a nuestro servidor y manejar su respuesta. En esta sección, profundizaremos un poco más en nuestra API HTTP y otras características compatibles con el módulo HTTP de Angular.

Options—Headers/Params

Opciones: encabezados/parámetros

First, let's take a closer look at the HTTP API we invoke. So far, we passed it a URL, along with a body for the request if necessary. The HTTP API also allows us to pass an `options` object as the second (or third in case the API allows for a body like POST and PATCH) argument to the function. Again, let's modify our existing application to see these options, and add in some common requirements as well. We will use the codebase from the previous section, which can be obtained from the `chapter9/simple-http` folder in case you are not coding along, as a base.

Primero, echemos un vistazo más de cerca a la API HTTP que invocamos. Hasta ahora, le pasamos una URL, junto con un cuerpo para la solicitud, si es necesario. La API HTTP también nos permite pasar un objeto `options` como segundo argumento (o tercero en caso de que la API permita un cuerpo como POST y PATCH) de la función. Nuevamente, modifiquemos nuestra aplicación existente para ver estas opciones y agreguemos también algunos requisitos comunes. Usaremos el código base de la sección anterior, que se puede obtener de la carpeta capítulo9/simple-http en caso de que no esté codificando, como base.

First, one of the common tasks that any developer has with an HTTP API is to send additional query parameters, or certain HTTP headers along with the request. Let's see how we can accomplish this using the `HttpClient`. We will change the `src/app/services/stock.service.ts` file as follows:

En primer lugar, una de las tareas comunes que tiene cualquier desarrollador con una API HTTP es enviar parámetros de consulta adicionales o ciertos encabezados HTTP junto con la solicitud. Veamos cómo podemos lograr esto usando `HttpClient`. Cambiaremos el archivo `src/app/services/stock.service.ts` de la siguiente manera:

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders } from '@angular/common/http';

import { Observable } from 'rxjs/Observable';

import { Stock } from 'app/model/stock';

@Injectable()
export class StockService {

  constructor(private http: HttpClient) {}

  getStocks(): Observable<Stock[]> {
    return this.http.get<Stock[]>('/api/stock', {
      headers: new HttpHeaders() ①
        .set('Authorization', 'MyAuthorizationHeaderValue')
        .set('X-EXAMPLE-HEADER', 'TestValue'),
      params: {
        q: 'test',
        test: 'value'
      }
    });
  }

  /** Unchanged after this, omitted for brevity */
}
```

- ① Agregue `headers` a `HttpClient` para una cabecera saliente

② Agregar los parámetros de solicitud en la llamada saliente

In the preceding code, we have made two changes. We have added a second argument to the `http.get` call, which is an options object. There are certain keys we can pass to it, to configure the outgoing HTTP request. In the code, we have added two options:

En el código anterior, hemos realizado dos cambios. Hemos agregado un segundo argumento a la llamada `http.get`, que es un objeto de opciones. Hay ciertas claves que podemos pasarle para configurar la solicitud HTTP saliente. En el código, hemos agregado dos opciones:

headers

encabezados

We can set the outgoing/request HTTP headers. There are two ways we can set both the headers as well as the parameters. One option, as we have done in the code, is to pass an `HttpHeaders` object, which is a typed class instance, on which we can call `set` to set the correct headers. It follows a builder pattern, so you can chain multiple headers as we have done. The second option is to just pass it a plain-old JavaScript object. Of course, for the values, we have hardcoded it here, but you can just as well access it from some variable, or even some other service (in case you need to get authorization headers from, say, an `AuthService`).

Podemos configurar los encabezados HTTP salientes/de solicitud. Hay dos formas en que podemos configurar tanto los encabezados como los parámetros. Una opción, como hemos hecho en el código, es pasar un objeto `HttpHeaders`, que es una instancia de clase escrita, en la que podemos llamar a `set` para configurar los encabezados correctos. Sigue un patrón de creación, por lo que puedes encadenar varios encabezados como

lo hemos hecho nosotros. La segunda opción es simplemente pasarle un objeto JavaScript antiguo. Por supuesto, para los valores, los hemos codificado aquí, pero también puedes acceder a ellos desde alguna variable o incluso algún otro servicio (en caso de que necesites obtener encabezados de autorización de, por ejemplo, un AuthService) .

params

parámetros

Just like the headers, HTTP query parameters can also be configured in two ways: using the typed, built-in `HttpParams` class, or using a plain-old JavaScript object.

Al igual que los encabezados, los parámetros de consulta HTTP también se pueden configurar de dos maneras: usando la clase `HttpParams` incorporada escrita o usando un objeto JavaScript antiguo.

Now, when we run this code (making sure that the Node.js server is also running in parallel), open your network inspector to see the outgoing requests. You should see something like [Figure 9-1](#).

Ahora, cuando ejecutemos este código (asegurándonos de que el servidor Node.js también se esté ejecutando en paralelo), abra el inspector de red para ver las solicitudes salientes. Debería ver algo como la Figura 9-1.

▼ General

Request URL: http://localhost:4200/api/stock?q=test&test=value

Request Method: GET

Status Code: 304 Not Modified

Remote Address: 127.0.0.1:4200

Referrer Policy: no-referrer-when-downgrade

▼ Response Headers

[view source](#)

Access-Control-Allow-Origin: *

connection: close

date: Tue, 23 Jan 2018 10:01:02 GMT

etag: W/"14a-0nHBtk/F4S8UTL5AKv8cEceKDns"

x-powered-by: Express

▼ Request Headers

[view source](#)

Accept: application/json, text/plain, */*

Accept-Encoding: gzip, deflate, br

Accept-Language: en-US, en;q=0.9

Authorization: MyAuthorizationHeaderValue

Connection: keep-alive

Cookie: X-RESTOK-AUTH-TOKEN=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJJuYW1lIjoiU2h5YW0iLCJwaZSI6Ijk4MzM5MTEzMzc1LCJpZCI6NiwidHlwZSI6IlJFVEFJTEVSIiwicmV0YWlsZXJJZCI6NywiawF0IjoxNTE2MTIxLCJleHAiOjE1MjM5NjExMjF9.gF6ELDDe6lbXaOSsxyMUltX86QLtAmgzZ02YrgDdOSSI

DNT: 1

Host: localhost:4200

If-None-Match: W/"14a-0nHBtk/F4S8UTL5AKv8cEceKDns"

Referer: http://localhost:4200/

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3322.3 Safari/537.36

X-EXAMPLE-HEADER: TestValue

Figure 9-1. Sending query params and headers with an Angular HTTP request

Figura 9-1. Envío de parámetros de consulta y encabezados con una solicitud HTTP angular

The finished code is available in the *chapter9/http-api* folder.

El código terminado está disponible en la carpeta Chapter9/http-api.

Options—Observe/Response Type

Opciones: Observar/Tipo de respuesta

We will next move on to two other options on the HTTP request, which give you flexibility in a variety of use cases. The first one we will focus on is the `observe` property on the `options` parameter.

A continuación, pasaremos a otras dos opciones de la solicitud HTTP, que le brindan flexibilidad en una variedad de casos de uso. La primera en la que nos centraremos es la propiedad `observe` en el parámetro `options`.

The `observe` parameter takes one of three values:

El parámetro `observe` toma uno de tres valores:

`body`

This is the default value, which ensures that the observable's `subscribe` gets called with the body of the response. This body is auto-casted to the return type specified when calling the API. This is what we have been using so far.

Este es el valor predeterminado, que garantiza que el `subscribe` del observable sea llamado con el cuerpo de la respuesta. Este cuerpo se convierte automáticamente al tipo de retorno especificado al llamar a la API. Esto es lo que hemos estado usando hasta ahora.

`response`

This changes the response type of the HTTP APIs from returning the entire `HttpResponse` instead of just the body. The response still holds a typed response, so you can still access it underneath, but you also get access to the headers of the response and the status code. Note that instead of a raw `HttpResponse`, what you really get is an `HttpResponse<T>`, where `T` is the type of the body. So in the case of `getStocks`, what you would end up returning is actually an observable of `HttpResponse<Stock[]>`.

Esto cambia el tipo de respuesta de las API HTTP para que devuelvan el `HttpResponse` completo en lugar de solo el cuerpo. La respuesta aún contiene una respuesta escrita, por lo que aún puedes acceder a ella debajo, pero también obtienes acceso a los encabezados de la respuesta y al código de estado. Tenga en cuenta que en lugar de un `HttpResponse` sin formato, lo que realmente obtiene es un `HttpResponse<T>`, donde `T` es el tipo de cuerpo. Entonces, en el caso de `getStocks`, lo que terminarías devolviendo es en realidad un observable de `HttpResponse<Stock[]>`.

events

This is similar to `response`, but gets triggered on all `HttpEvents`. This would include the initialization event, as well as the request finished event. These correspond to the `XMLHttpRequest` states. This is more useful when we have an API that sends progress events, as we can catch and listen to progress events as well with the `observe` parameter set to `events`.

Esto es similar a `response`, pero se activa en todos los `HttpEvents`. Esto incluiría el evento de inicialización, así como el evento de finalización de la solicitud. Estos corresponden a los estados `XMLHttpRequest`. Esto es más útil cuando tenemos una API que envía eventos de progreso, ya que también podemos capturar y escuchar eventos de progreso con el parámetro `observe` establecido en `events`.

The second parameter that we will explore is the `responseType` property on the `options` parameter. This can take one of four values:

El segundo parámetro que exploraremos es la propiedad `responseType` en el parámetro de opciones. Esto puede tomar uno de cuatro valores:

json

This is the default, which basically ensures that the response body is parsed and treated as a JSON object. In most cases, you wouldn't need to change this from the default.

Este es el valor predeterminado, que básicamente garantiza que el cuerpo de la respuesta se analice y se trate como un objeto JSON. En la mayoría de los casos, no será necesario cambiar el valor predeterminado.

text

This allows you to treat the response body as a string, without parsing it at all. In this case, the response from your HTTP request would be an `Observable<string>` instead of a typed response.

Esto le permite tratar el cuerpo de la respuesta como una cadena, sin analizarlo en absoluto. En este caso, la respuesta de su solicitud HTTP sería un `Observable<string>` en lugar de una respuesta escrita.

blob

Both this and the next option are more useful when dealing with binary responses that you need to handle in your application. `blob` gives you a file-like object containing the raw immutable data, which you can then process as you see fit.

Tanto esta como la siguiente opción son más útiles cuando se trata de respuestas binarias que necesita manejar en su

aplicación. blob le proporciona un objeto similar a un archivo que contiene los datos sin procesar e inmutables, que luego puede procesar como mejor le parezca.

arraybuffer

The last option gives us the underlying raw binary data directly.

La última opción nos proporciona directamente los datos binarios sin procesar subyacentes.

Let's take a look at some of these in action. We will change the StockService temporarily to try the same API with a few of these values set as options. Change the *src/app/services/stock.service.ts* file as follows:

Echemos un vistazo a algunos de ellos en acción. Cambiaremos el StockService temporalmente para probar la misma API con algunos de estos valores configurados como opciones. Cambie el archivo *src/app/services/stock.service.ts* de la siguiente manera:

```
/** No Change in Imports */

@Injectable()
export class StockService {

  constructor(private http: HttpClient) {}

  getStocks(): Observable<Stock[]> {
    return this.http.get<Stock[]>('/api/stock', {
      headers: new HttpHeaders()
        .set('Authorization', 'MyAuthorizationHeaderValue')
        .set('X-EXAMPLE-HEADER', 'TestValue'),
      params: {
        q: 'test',
        test: 'value'
      },
      observe: 'body' ❶
    });
  }

  getStocksAsResponse(): Observable<HttpResponse<Stock[]>> {
    return this.http.get<Stock[]>('/api/stock', {
```

```

        observe: 'response'          ②
    });
}

getStocksAsEvents(): Observable<HttpEvent<any>> {
    return this.http.get('/api/stock', {
        observe: 'events'           ③
    });
}

getStocksAsString(): Observable<string> {
    return this.http.get('/api/stock', {
        responseType: 'text'       ④
    });
}

getStocksAsBlob(): Observable<Blob> {
    return this.http.get('/api/stock', {
        responseType: 'blob'        ⑤
    });
}

/** Remaining code unchanged, omitted for brevity */
}

```

① Observar el ~~body~~ de la respuesta

② Observar la respuesta completa

③ Observa todos los eventos

④ Respuesta debe ser texto.

⑤ Respuesta debe ser un blob

We have added four new methods to the StockService as follows:

Hemos agregado cuatro nuevos métodos a StockService de la siguiente manera:

- getStocksAsResponse makes the same HTTP call, but sets the observe value to response. This also changes the response of the function to return an Observable<HttpResponse<Stock[]>>.

`getStocksAsResponse` realiza la misma llamada HTTP, pero establece el valor de `observe` en `response`. Esto también cambia la respuesta de la función para devolver un `Observable<HttpResponse<Stock[]>`.

- `getStocksAsEvents` makes the same HTTP call, but sets the `observe` value to `events`. This also changes the response of the function to return an `Observable<HttpEvent<any>`. This is because we will get multiple instances of `HttpEvent` that are not just the response, but also progress, initialization, etc. Thus the format of the response is not defined.

`getStocksAsEvents` realiza la misma llamada HTTP, pero establece el valor de `observe` en `events`. Esto también cambia la respuesta de la función para devolver un `Observable<HttpEvent<any>`. Esto se debe a que obtendremos múltiples instancias de `HttpEvent` que no son solo la respuesta, sino también el progreso, la inicialización, etc. Por lo tanto, el formato de la respuesta no está definido.

- `getStocksAsString` makes the same HTTP call, but sets the `responseType` value to `text`. We also change the response type of the function to return an `Observable<string>`, and the string is the entire body as a string.

`getStocksAsString` realiza la misma llamada HTTP, pero establece el valor de `responseType` en `text`. También cambiamos el tipo de respuesta de la función para devolver un `Observable<string>`, y la cadena es el cuerpo completo como una cadena.

- `getStocksAsBlob` makes the same HTTP call, but sets the `responseType` value to `blob`. We also change the response type of the function to return an `Observable<Blob>` to allow the subscriber to work with the blob once we get a response from the server.

`getStocksAsBlob` realiza la misma llamada HTTP, pero establece el valor de `responseType` en `blob`. También cambiamos el tipo de respuesta de la función para devolver un `Observable<Blob>` para permitir que el suscriptor trabaje con el blob una vez que obtengamos una respuesta del servidor.

Now, let's hook this up in the component so that we see the effect of calling each of these slightly different APIs. We will modify the `StockListComponent` to make calls to each of these APIs so that we can see and compare them side by side. Modify the `src/app/stock/stock-list/stock-list.component.ts` file as follows:

Ahora, conectemos esto en el componente para que veamos el efecto de llamar a cada una de estas API ligeramente diferentes. Modificaremos el `StockListComponent` para realizar llamadas a cada una de estas API para que podamos verlas y compararlas una al lado de la otra. Modifique el archivo `src/app/stock/stock-list/stock-list.component.ts` de la siguiente manera:

```
import { Component, OnInit } from '@angular/core';
import { StockService } from 'app/services/stock.service';
import { Stock } from 'app/model/stock';
import { Observable } from 'rxjs/Observable';

@Component({
  selector: 'app-stock-list',
  templateUrl: './stock-list.component.html',
  styleUrls: ['./stock-list.component.css']
})
export class StockListComponent implements OnInit {

  public stocks$: Observable<Stock[]>;
  constructor(private stockService: StockService) { }

  ngOnInit() {
    this.stocks$ = this.stockService.getStocks();
    this.stockService.getStocksAsResponse()
      .subscribe((response) => {
        console.log('OBSERVE "response" RESPONSE is ', response);
      });
  }
}
```

```

    this.stockService.getStocksAsEvents()
      .subscribe((response) => {
        console.log('OBSERVE "events" RESPONSE is ', response);
      });

    this.stockService.getStocksAsString()
      .subscribe((response) => {
        console.log('Response Type "text" RESPONSE is ', response);
      });

    this.stockService.getStocksAsBlob()
      .subscribe((response) => {
        console.log('Response Type "blob" RESPONSE is ', response);
      });
  }
}

```

In the StockListComponent, we leave the original call untouched, and simply add calls to each of the other functions we added in the service below it. For each, we simply subscribe to the response, and then print it (with a respective log so that we can identify them separately). Now, when you run it, make sure you have your browser developer tools open so that you can see the logs it prints. You should see something like [Figure 9-2](#).

En StockListComponent, dejamos la llamada original intacta y simplemente agregamos llamadas a cada una de las otras funciones que agregamos en el servicio debajo de ella. Para cada uno, simplemente nos suscribimos a la respuesta y luego la imprimimos (con un registro respectivo para que podamos identificarlos por separado). Ahora, cuando lo ejecute, asegúrese de tener abiertas las herramientas de desarrollo de su navegador para poder ver los registros que imprime. Debería ver algo como la Figura 9-2.

The screenshot shows the Chrome DevTools Console tab with five log entries:

- OBSERVE "events" RESPONSE is `Object` Response of `getStocksAsEvents`
- Angular is running in the development mode. Call enableProdMode() to run in the production mode. `core.js:3675`
- OBSERVE "response" RESPONSE is `HttpResponse` Response of `getStocksAsResponse`
 - `body: (3) [{}]`
 - `headers: HttpHeaders {normalizedNames: Map(0), lazyUpdate: null, lazyInit: false}`
 - `ok: true`
 - `status: 200`
 - `statusText: "OK"`
 - `type: 4`
 - `url: "http://localhost:4200/api/stock"`
 - `_proto_: HttpResponseBase`
- OBSERVE "events" RESPONSE is `HttpResponse` Response of `getStocksAsEvents again`
 - `body: (3) [{}]`
 - `headers: HttpHeaders {normalizedNames: Map(0), lazyUpdate: null, lazyInit: false}`
 - `ok: true`
 - `status: 200`
 - `statusText: "OK"`
 - `type: 4`
 - `url: "http://localhost:4200/api/stock"`
 - `_proto_: HttpResponseBase`
- Response Type "text" RESPONSE is `[{"name": "Test Stock Company", "code": "TSC", "price": 85, "previousPrice": 80, "exchange": "NASDAQ", "favorite": false}, {"name": "Second Stock Company", "code": "SSC", "price": 10, "previousPrice": 20, "exchange": "NSE", "favorite": false}, {"name": "Last Stock Company", "code": "LSC", "price": 876, "previousPrice": 765, "exchange": "NYSE", "favorite": false}]` stock-list.component.ts:30
- Response Type "blob" RESPONSE is `Blob(330)` Response of `getStocksAsBlob`
 - `size: 330`
 - `type: "application/json"`
 - `_proto_: Blob`

Figure 9-2. Different kind of response types in Angular

Figura 9-2. Diferentes tipos de respuesta en Angular

There are a few interesting things to note in the example we ran:

Hay algunas cosas interesantes a tener en cuenta en el ejemplo que ejecutamos:

- For the `getStocksAsEvents()` call, our subscription is actually called twice, once for the initialization/request being sent and once when the actual information has been loaded with the response. The second event is the one that has the actual data. If our API supported progress, then the subscription would get called for progress events as well.

Para la llamada `getStocksAsEvents()`, nuestra suscripción en realidad se llama dos veces, una para la inicialización/solicitud que se envía y otra cuando la información real se ha cargado con la respuesta. El segundo evento es el que tiene los datos reales. Si nuestra API admitiera el progreso, entonces también se solicitaría la suscripción para eventos de progreso.

- The `getStocksAsResponse()` call is similar to our initial `getStocks()` call, except it gives the body along with other HTTP request fields like status, headers, etc.

La llamada `getStocksAsResponse()` es similar a nuestra llamada inicial `getStocks()`, excepto que proporciona el cuerpo junto con otros campos de solicitud HTTP como estado, encabezados, etc.

- The `getStocksAsString()` call is similar to our original call, but instead of getting a typed JSON response, we get our entire JSON response as a string. As mentioned before, this is more useful for working with non-JSON APIs.

La llamada `getStocksAsString()` es similar a nuestra llamada original, pero en lugar de obtener una respuesta JSON escrita,

obtenemos nuestra respuesta JSON completa como una cadena. Como se mencionó anteriormente, esto es más útil para trabajar con API que no son JSON.

- The final `getStocksAsBlob()` call returns a blob containing our data that we can then work with. This is more useful for working with binary data and the like, rather than JSON APIs.

La llamada final a `getStocksAsBlob()` devuelve un blob que contiene nuestros datos con los que luego podemos trabajar. Esto es más útil para trabajar con datos binarios y similares, en lugar de API JSON.

For the most part, you will only need the default setting of observing the body and using the `json` response type. These other options exist for that 5% of use cases where the default is not enough, and you need access to something more. The new `HttpClient` gives you that flexibility to work with your APIs the way you want to, while making it easy for the most common use cases.

En su mayor parte, solo necesitará la configuración predeterminada de observar el `body` y usar el tipo de respuesta `json`. Estas otras opciones existen para ese 5% de los casos de uso en los que el valor predeterminado no es suficiente y necesita acceso a algo más. El nuevo `HttpClient` le brinda esa flexibilidad para trabajar con sus API de la manera que desee, al mismo tiempo que facilita los casos de uso más comunes.

The finished code sample for this section is available in the GitHub repository in the *chapter9/http-api* folder.

El ejemplo de código terminado para esta sección está disponible en el repositorio de GitHub en la carpeta Chapter9/http-api.

Interceptors

Interceptores

One other common use case is to be able to hook into all incoming and outgoing requests to be able to either modify them in flight, or listen in on all responses to accomplish things like logging, handling authentication failures in a common manner, etc.

Otro caso de uso común es poder conectarse a todas las solicitudes entrantes y salientes para poder modificarlas en el momento o escuchar todas las respuestas para realizar cosas como iniciar sesión, manejar fallas de autenticación de manera común, etc.

With the `HttpClient` API, Angular allows easily defining interceptors, which can conceptually sit between the `HttpClient` and the server, allowing it to transform all outgoing requests, and listen in and transform if necessary all incoming responses before passing them on. Let's see how we might create a very simple interceptor in Angular that allows us to:

Con la API `HttpClient`, Angular permite definir fácilmente interceptores, que conceptualmente pueden ubicarse entre `HttpClient` y el servidor, lo que le permite transformar todas las solicitudes salientes y escuchar y transformar, si es necesario, todas las respuestas entrantes antes de pasarlos en. Veamos cómo podríamos crear un interceptor muy simple en Angular que nos permita:

- Modify all outgoing requests by adding a header if we have an authorization token.

Modifique todas las solicitudes salientes agregando un encabezado si tenemos un token de autorización.

- Log all incoming responses before passing them along.

Registre todas las respuestas entrantes antes de transmitirlas.

- In case the request ends up in a failure (a non-200 or non-300 response), we will log it differently.

En caso de que la solicitud termine en un error (una respuesta que no sea 200 o que no sea 300), la registraremos de manera diferente.

We will fetch the authorization token from another service, which we will create just for the purpose of storing and returning our authorization information. To complete this example, we will build on our codebase from the first section. So if you want, you can obtain the base code from the *chapter9/simple-http* folder. Note that we are not using the additions we did while we were exploring the HTTP API.

Obtendremos el token de autorización de otro servicio, que crearemos solo con el fin de almacenar y devolver nuestra información de autorización. Para completar este ejemplo, nos basaremos en nuestro código base de la primera sección. Entonces, si lo desea, puede obtener el código base de la carpeta capítulo9/simple-http. Tenga en cuenta que no estamos utilizando las adiciones que hicimos mientras explorábamos la API HTTP.

We will first add a very simple service that acts as a holder of authorization-related information. We can add it simply by running this from the terminal:

Primero agregaremos un servicio muy simple que actúa como poseedor de información relacionada con la autorización. Podemos agregarlo simplemente ejecutando esto desde la terminal:

```
ng generate service services/auth --module=app
```

This will create an AuthService for you, and also hook up the provider in the AppModule. We will just make a small change to the generated service to hold a property that we can use in our

interceptor. The changed service in *src/app/services/auth.service.ts* should look like the following:

Esto creará un AuthService para usted y también conectará el proveedor en AppModule. Simplemente haremos un pequeño cambio en el servicio generado para mantener una propiedad que podamos usar en nuestro interceptor. El servicio modificado en *src/app/services/auth.service.ts* debería tener el siguiente aspecto:

```
import { Injectable } from '@angular/core';

@Injectable()
export class AuthService {

  public authToken: string;

  constructor() { }
}
```

We have added a public property on the class called authToken in the AuthService. Because we generated the service using the Angular CLI, it automatically added the provider in the AppModule. Otherwise, we would have had to do it ourselves. We will add one more HTTP API call in the StockService to an API that always returns a 403. Make the change so that the *src/app/services/stock.service.ts* file looks like this:

Hemos agregado una propiedad pública en la clase llamada authToken en AuthService. Debido a que generamos el servicio usando Angular CLI, agregó automáticamente el proveedor en AppModule. De lo contrario, hubiéramos tenido que hacerlo nosotros mismos. Agregaremos una llamada API HTTP más en StockService a una API que siempre devuelve un 403. Realice el cambio para que el archivo *src/app/services/stock.service.ts* tenga este aspecto:

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
```

```

import { Observable } from 'rxjs/Observable';

import { Stock } from 'app/model/stock';

@Injectable()
export class StockService {

  constructor(private http: HttpClient) {}

  /** No changes to other calls, omitting for brevity */

  makeFailingCall() {
    return this.http.get('/api/fail');
  }
}

```

We only added the last method, `makeFailingCall()`, to the `StockService`. The remaining code remains untouched. We will then change the `StockListComponent` to display a few extra buttons, which we will hook up to the `StockService` and the `AuthService` that we have just created. First, change the `src/app/stock/stock-list/stock-list.component.ts` file as follows:

Solo agregamos el último método, `makeFailingCall()`, al `StockService`. El código restante permanece intacto. Luego cambiaremos el `StockListComponent` para mostrar algunos botones adicionales, que conectaremos al `StockService` y al `AuthService` que acabamos de crear. Primero, cambie el archivo `src/app/stock/stock-list/stock-list.component.ts` de la siguiente manera:

```

import { Component, OnInit } from '@angular/core';
import { StockService } from 'app/services/stock.service';
import { Stock } from 'app/model/stock';
import { Observable } from 'rxjs/Observable';
import { AuthService } from 'app/services/auth.service';

@Component({
  selector: 'app-stock-list',
  templateUrl: './stock-list.component.html',
  styleUrls: ['./stock-list.component.css']
})
export class StockListComponent implements OnInit {

```

```

public stocks$: Observable<Stock[]>;
constructor(private stockService: StockService,
            private authService: AuthService) { }

ngOnInit() {
  this.fetchStocks();
}

fetchStocks() {
  this.stocks$ = this.stockService.getStocks();
}

setAuthToken() {
  this.authService.authToken = 'TESTING';
}

resetAuthToken() {
  this.authService.authToken = null;
}

makeFailingCall() {
  this.stockService.makeFailingCall().subscribe(
    (res) => console.log('Successfully made failing call', res),
    (err) => console.error('Error making failing call', err));
}
}

```

We made a slight change to the initialization logic, and added a few new methods to the StockListComponent. First, we pulled out the stock\$ Observable subscription to a new method called fetchStocks(), and just called it in the ngOnInit. Next, we have added a few more methods, namely setAuthToken(), resetAuthToken(), and makeFailingCall(), all of which are trivial and basically call out to AuthService and StockService, respectively.

Hicimos un ligero cambio en la lógica de inicialización y agregamos algunos métodos nuevos al StockListComponent. Primero, retiramos la suscripción stock\$ Observable a un nuevo método llamado fetchStocks() y simplemente lo llamamos en ngOnInit. A continuación, hemos agregado algunos métodos más, a saber, setAuthToken(), resetAuthToken() y makeFailingCall(), todos los

cuales son triviales y básicamente llaman a AuthService y StockService, respectivamente.

Next, we will hook these four new methods to buttons in the template for the StockListComponent. Let's change `src/app/stock/stock-list/stock-list.component.html` as follows:

A continuación, conectaremos estos cuatro nuevos métodos a los botones de la plantilla para StockListComponent. Cambiemos `src/app/stock/stock-list/stock-list.component.html` de la siguiente manera:

```
<app-stock-item *ngFor="let stock of stocks$ | async"
                 [stock]="stock">
</app-stock-item>

<div>
  <button type="button"
         (click)="fetchStocks()">
    Refetch Stocks
  </button>
  <button type="button"
         (click)="makeFailingCall()">
    Make Failing Call
  </button>
  <button type="button"
         (click)="setAuthToken()">
    Set Auth Token
  </button>
  <button type="button"
         (click)="resetAuthToken()">
    Reset Auth Token
  </button>
</div>
```

We have added four new buttons to the template, with each one calling out to one of the new methods we just added.

Hemos agregado cuatro botones nuevos a la plantilla, cada uno de los cuales llama a uno de los nuevos métodos que acabamos de agregar.

So far, we have not added anything related to interceptors, but rather just set up our application to demonstrate various things related to the interceptors and how we might be able to use them in our application. Let's create our interceptor now. Sadly, the Angular CLI does not yet support generating the skeleton of the interceptor, so we will do it manually. Create a file at `src/app/services/stock-app.interceptor.ts` with the following content:

Hasta ahora, no hemos agregado nada relacionado con los interceptores, sino que simplemente configuramos nuestra aplicación para demostrar varias cosas relacionadas con los interceptores y cómo podríamos usarlos en nuestra aplicación. Creemos nuestro interceptor ahora. Lamentablemente, Angular CLI aún no admite la generación del esqueleto del interceptor, por lo que lo haremos manualmente. Cree un archivo en `src/app/services/stock-app.interceptor.ts` con el siguiente contenido:

```
import {Injectable} from '@angular/core';
import {HttpEvent, HttpInterceptor, HttpResponse} from '@angular/common/http';
import {HttpHandler, HttpRequest, HttpErrorResponse} from
  '@angular/common/http'

import {Observable} from 'rxjs/Observable';

@Injectable()
export class StockAppInterceptor implements HttpInterceptor { ❶

  constructor() {}

  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> { ❷
    console.log('Making a request to ', req.url);
    return next.handle(req); ❸
  }
}
```

❶ Implementando la interfaz `HttpInterceptor`

❷ Implementando la API de la API

③ Continúe la cadena llamando `handler` con la solicitud

We have a mostly straightforward StockAppInterceptor class that implements the Angular `HttpInterceptor` interface. This interface provides one method, `intercept`, which we implement in the class. The `intercept` method is called with two arguments, the `HttpRequest` and an `HttpHandler`.

Tenemos una clase `StockAppInterceptor` prácticamente sencilla que implementa la interfaz Angular `HttpInterceptor`. Esta interfaz proporciona un método, `intercept`, que implementamos en la clase. El método `intercept` se llama con dos argumentos, el `HttpRequest` y un `HttpHandler`.

A good way to think of `HttpInterceptor` is that it is a chain. Each interceptor is called with the request, and it is up to the interceptor to decide whether it continues in the chain or not. Within this context, each interceptor can decide to modify the request as well. It can continue the chain by calling the handler provided to the `intercept` method with a request object. If there is only one interceptor, then the handler would simply call our backend with the request object. If there are more, it would proceed to the next interceptor in the chain.

Una buena forma de pensar en `HttpInterceptor` es que es una cadena. Se llama a cada interceptor con la solicitud y le corresponde al interceptor decidir si continúa en la cadena o no. Dentro de este contexto, cada interceptor también puede decidir modificar la solicitud. Puede continuar la cadena llamando al controlador proporcionado al método `intercept` con un objeto de solicitud. Si solo hay un interceptor, entonces el controlador simplemente llamará a nuestro backend con el objeto de solicitud. Si hay más, pasaría al siguiente interceptor de la cadena.

Let's hook up this simple interceptor, which at this point simply logs all outgoing requests to the console, to the application. We do so in

the AppModule (available in `src/app/app.module.ts`) as follows:

Conectemos este interceptor simple, que en este punto simplemente registra todas las solicitudes salientes a la consola, a la aplicación. Lo hacemos en el AppModule (disponible en `src/app/app.module.ts`) de la siguiente manera:

```
/** Skipping other standard imports for brevity */
import { HttpClientModule, HTTP_INTERCEPTORS } from '@angular/common/http';
import { AuthService } from './services/auth.service';
import { StockAppInterceptor } from './services/stock-app.interceptor';

@NgModule({
  declarations: [
    /** Skipping for brevity */
  ],
  imports: [
    /** Skipping for brevity */
  ],
  providers: [
    StockService,
    AuthService,
    {
      provide: HTTP_INTERCEPTORS,
      useClass: StockAppInterceptor,
      multi: true,
    }
  ],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

The major focus is on the last element in the providers section, which is how we provide the HttpInterceptor. Here, we are using the basic form of a provider, where we mention what we are providing (using the `provide` key), how to provide it (using the `useClass`, which points at our newly created `StockAppInterceptor`), and the fact that it is an array of interceptors (`multi: true` takes care of this). We can similarly add another entry for each interceptor we want.

La atención se centra principalmente en el último elemento de la sección providers, que es cómo proporcionamos el `HttpInterceptor`. Aquí, utilizamos la forma básica de un proveedor, donde mencionamos lo que proporcionamos (usando la clave `provide`), cómo proporcionarlo (usando la `useClass`, que apunta a nuestro `useClass` recién creado `StockAppInterceptor`), y el hecho de que es una matriz de interceptores (`multi: true` se encarga de esto). De manera similar, podemos agregar otra entrada para cada interceptor que queramos.

Now we can run this application (using `ng serve --proxy-config proxy.conf.json`, after making sure our Node.js server is running). When you run this, open the console in your developer tools and you should see a log entry each time a server call is made. You can trigger more server calls by clicking the Refetch Stocks button.

Ahora podemos ejecutar esta aplicación (usando `ng serve --proxy-config proxy.conf.json`, después de asegurarnos de que nuestro servidor Node.js esté ejecutándose). Cuando ejecute esto, abra la consola en sus herramientas de desarrollador y debería ver una entrada de registro cada vez que se realice una llamada al servidor. Puede activar más llamadas al servidor haciendo clic en el botón Volver a buscar acciones.

Now, let's extend to make the interceptor nontrivial. Modify the `src/app/services/stock-app.interceptor.ts` file as follows:

Ahora, ampliemos para hacer que el interceptor no sea trivial. Modifique el archivo `src/app/services/stock-app.interceptor.ts` de la siguiente manera:

```
import { Injectable } from '@angular/core';
import { HttpEvent, HttpInterceptor, HttpResponse } from '@angular/common/http';
import { HttpHandler, HttpRequest, HttpHeaders } from '@angular/common/http'

import { Observable } from 'rxjs/Observable';
import 'rxjs/add/operator/do';
```

```

import { AuthService } from './auth.service';

@Injectable()
export class StockAppInterceptor implements HttpInterceptor {

  constructor(private authService: AuthService) {}

  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    ①
    if (this.authService.authToken) { ②
      const authReq = req.clone({
        headers: req.headers.set(
          'Authorization',
          this.authService.authToken
        )
      });
      console.log('Making an authorized request');
      ③
      req = authReq;
    }
    ④
    return next.handle(req)
      .do(event => this.handleResponse(req, event),
           error => this.handleError(req, error));
  }

  handleResponse(req: HttpRequest<any>, event) { ⑤
    console.log('Handling response for ', req.url, event);
    if (event instanceof HttpResponse) {
      console.log('Request for ', req.url,
                  ' Response Status ', event.status,
                  ' With body ', event.body);
    }
  }

  handleError(req: HttpRequest<any>, event) { ⑥
    console.error('Request for ', req.url,
                 ' Response Status ', event.status,
                 ' With error ', event.error);
  }
}

```

- ① Implementar la API `intercept`
- ② Verificar la presencia del token de autenticación en el servicio

- ③ Cambie la solicitud a una solicitud autorizada con encabezados adicionales
- ④ Envíe la API en la solicitud para continuar la cadena
- ⑤ Manejará su respuesta exitosa
- ⑥ Manejará un error

We have changed a few things of note in the interceptor now:

Hemos cambiado algunas cosas importantes en el interceptor ahora:

- First, we check for the presence of the authToken in the AuthService. If it is available, we then modify the request to add an Authorization header with the current value from the AuthService.

Primero, verificamos la presencia de authToken en AuthService. Si está disponible, modificamos la solicitud para agregar un encabezado Authorization con el valor actual de AuthService.

- We have added an operator to Observable using the `import 'rxjs/add/operator/do'`; statement. We then use this on the `next.handle`.

Hemos agregado un operador a Observable usando la declaración `import 'rxjs/add/operator/do'`. Luego usamos esto en el `next.handle`.

- The do operator on an observable allows us to hook onto the results of an observable in a pass-through manner, but still make changes or have side effects. It is useful for cases like these where we want to see the response and possibly even make changes to it or log it.

El operador do en un observable nos permite conectarnos a los resultados de un observable de manera transferible, pero aún así realizar cambios o tener efectos secundarios. Es útil para

casos como estos en los que queremos ver la respuesta y posiblemente incluso realizar cambios o registrarla.

- We subscribe to both the success and error events on the observable, and add the appropriate logging in both.

Nos suscribimos a los eventos de éxito y error en el observable y agregamos el registro apropiado en ambos.

WHY CLONE THE REQUEST? ¿POR QUÉ CLONAR LA SOLICITUD?

One very important thing to note is how we handle adding headers (or making any other changes) to the outgoing request.

Una cosa muy importante a tener en cuenta es cómo manejamos la adición de encabezados (o cómo realizamos otros cambios) a la solicitud saliente.

`HttpRequest` (and `HttpResponse`) instances are *immutable*. This means that once created, its values cannot be changed. We want the `HttpRequest` and `HttpResponse` to be immutable because there are various observable operators, some of which might want to retry the request.

Las instancias `HttpRequest` (y `HttpResponse`) son inmutables. Esto significa que una vez creado, sus valores no se pueden cambiar. Queremos que `HttpRequest` y `HttpResponse` sean inmutables porque hay varios operadores observable, algunos de los cuales podrían querer volver a intentar la solicitud.

For example, assume a simple flow where before we make the request, we add a counter that counts the number of requests from our client. If the request was retried, then the count might get updated again, even though it is still the original request that was being retried.

Por ejemplo, supongamos un flujo simple en el que antes de realizar la solicitud, agregamos un contador que cuenta la cantidad de solicitudes de nuestro cliente. Si se volvió a intentar la solicitud, es posible que el recuento se actualice nuevamente, aunque todavía sea la solicitud original la que se estaba reintentando.

If request instances were mutable, then on retrying the request through an interceptor chain, the request might be completely different. For this reason, both `HttpRequest` and `HttpResponse` instances are immutable. Thus, any changes we make on them result in a new immutable instance. This allows us to ensure that retrying a request through an interceptor chain would result in exactly the same request being made, and not result in unexpected behavior because of mutability.

Si las instancias de solicitud fueran mutables, al volver a intentar la solicitud a través de una cadena interceptor, la solicitud podría ser completamente diferente. Por este motivo, tanto la instancia `HttpRequest` como la `HttpResponse` son inmutables. Por lo tanto, cualquier cambio que hagamos en ellos dará como resultado una nueva instancia inmutable. Esto nos permite garantizar que volver a intentar una solicitud a través de una cadena de interceptor resulte en realizar exactamente la misma solicitud y no en un comportamiento inesperado debido a la mutabilidad.

We therefore call `clone()` on the request to get a new instance with modified/updated properties that we pass to the instance. This gives us back a new instance with updated values. We pass this instance to the handler instead of the original request.

Por lo tanto, llamamos a `clone()` en la solicitud para obtener una nueva instancia con propiedades modificadas/actualizadas que le pasamos a la instancia. Esto nos devuelve una nueva instancia con valores actualizados. Pasamos esta instancia al controlador en lugar de la solicitud original.

Now when you run your application, take the following actions:

Ahora, cuando ejecute su aplicación, realice las siguientes acciones:

1. See the initial request to get the list of stocks in the network inspector. Ensure that there is no Authorization header in the request.

Consulte la solicitud inicial para obtener la lista de acciones en el inspector de red. Asegúrese de que no haya ningún encabezado Authorization en la solicitud.

2. Click the Set Auth Token button. Now click the Refetch Stocks button. Notice that the Authorization header is now set and sent in the request. Corresponding logs will also be printed.

Haga clic en el botón Establecer token de autenticación. Ahora haga clic en el botón Volver a buscar acciones. Observe que el encabezado Authorization ahora está configurado y enviado en la solicitud. También se imprimirán los registros correspondientes.

3. Click Reset Auth Token to ensure that header is removed and try Refetch Stocks once again.

Haga clic en Restablecer token de autenticación para asegurarse de que se elimine el encabezado e intente recuperar acciones una vez más.

4. Click the Make Failing Call button and ensure that the error handler in the interceptor gets called and the logs are correctly printed.

Haga clic en el botón Realizar llamada fallida y asegúrese de que se llame al controlador de errores en el interceptor y que los registros se impriman correctamente.

SERVICE DEPENDENCIES AND INTERCEPTORS

DEPENDENCIAS DE SERVICIOS E INTERCEPTORES

Note that we pulled in an AuthService dependency in the interceptor via dependency injection. But there is one exception that you would need to watch out for, which is if the dependency you are pulling in within the interceptor in turn depends on HttpClient. This would compile, but when you run the application in your browser, you will see an error about *circular dependency*. This is because the HttpClient underneath requires all the interceptors, and when we add a service as a dependency that requires HttpClient, we end up with a circular ask that breaks our application.

Tenga en cuenta que incorporamos una dependencia AuthService en el interceptor mediante inyección de dependencia. Pero hay una excepción a la que debes tener cuidado, que es si la dependencia que estás incorporando dentro del interceptor depende a su vez de HttpClient. Esto se compilaría, pero cuando ejecute la aplicación en su navegador, verá un error sobre la dependencia circular. Esto se debe a que el HttpClient debajo requiere todos los interceptores, y cuando agregamos un servicio como una dependencia que requiere HttpClient, terminamos con una solicitud circular que interrumpe nuestra aplicación.

How do we fix this? There are a few approaches we can take:

¿Cómo arreglamos esto? Hay algunos enfoques que podemos adoptar:

- Split apart your service if possible into a data service that does not depend on HttpClient and one that makes server calls using HttpClient. Then you can pull in the first service as a dependency only, which won't cause this circular dependency. This is usually the easier and preferred approach.

Si es posible, divida su servicio en un servicio de datos que no dependa de HttpClient y uno que realice llamadas al servidor usando HttpClient. Luego puede incorporar el primer servicio solo como dependencia, lo que no causará esta dependencia circular. Este suele ser el enfoque más fácil y preferido.

- Do not inject your HttpClient dependency in the constructor, but rather lazily inject it later as and when you need it. To do this, you would need the Injector injected in the constructor. You can then do `this.injector.get(MyService)` to get a handle on your service and use it. Refer to [this GitHub issue](#) for more details.

No inyectes tu dependencia HttpClient en el constructor, sino más bien inyéctala perezosamente más tarde cuando la necesites. Para hacer esto, necesitaría inyectar Injector en el constructor. Luego puede hacer `this.injector.get(MyService)` para controlar su servicio y usarlo. Consulte este número de GitHub para obtener más detalles.

The finished code is available in *chapter9 interceptors*.

El código terminado está disponible en el capítulo 9/interceptores.

Advanced Observables

Observables avanzados

In this final section, we will go a little bit more in depth into how we can accomplish some usually complex things using observables. We will also see some common pitfalls that you should be aware of when using observables. From an application perspective, we will try to add the capability to search stocks as you type from the application. This will allow us to see many things in actions. Search is the prototypical example for demonstrating the power of ReactiveX as it really demonstrates how observables can make certain tasks simpler, especially when you treat everything as a stream of events.

En esta sección final, profundizaremos un poco más en cómo podemos lograr algunas cosas generalmente complejas utilizando observables. También veremos algunos errores comunes que debes tener en cuenta al utilizar observables. Desde la perspectiva de la aplicación, intentaremos agregar la capacidad de buscar acciones a medida que escribe desde la aplicación. Esto nos permitirá ver muchas cosas en acciones. La búsqueda es el ejemplo prototípico para demostrar el poder de ReactiveX, ya que realmente demuestra cómo los observables pueden simplificar ciertas tareas, especialmente cuando se trata todo como un flujo de eventos.

Search-as-you-type is generally difficult, for the following reasons:

La búsqueda mientras se escribe suele ser difícil por los siguientes motivos:

- If you trigger an HTTP call every time someone performs a keypress, then you would end up with a lot of calls, most of which will need to be ignored.

Si activa una llamada HTTP cada vez que alguien presiona una tecla, terminará con muchas llamadas, la mayoría de las cuales deberán ignorarse.

- Users rarely type correctly in one shot, often having to erase and retype. This will also result in unnecessary duplicate calls.

Los usuarios rara vez escriben correctamente de una sola vez, y a menudo tienen que borrar y volver a escribir. Esto también resultará en llamadas duplicadas innecesarias.

- You need to worry about how to deal with out-of-order responses. If the result for the previous query returns after the current one, you need to handle it in your application logic.

Debe preocuparse por cómo lidiar con las respuestas desordenadas. Si el resultado de la consulta anterior regresa después de la actual, debe manejarlo en la lógica de su aplicación.

Thankfully, observables give us powerful operators to handle each of these. Before we get into how to fix these using observables, let's start by first adding a line to show how many search results we are seeing. This will also demonstrate one thing to watch out for when using observables.

Afortunadamente, los observables nos brindan operadores poderosos para manejar cada uno de ellos. Antes de ver cómo solucionar estos problemas usando observables, comencemos agregando una línea para mostrar cuántos resultados de búsqueda estamos viendo. Esto también demostrará algo a tener en cuenta al utilizar observables.

We will again use the code from the first finished example, which is available in *chapter9/simple-http*. We will build on this for the rest of

the section.

Usaremos nuevamente el código del primer ejemplo terminado, que está disponible en el capítulo 9/simple-http. Nos basaremos en esto para el resto de la sección.

First, let's edit `src/app/stock/stock-list.component.html` to show the number of stocks we have in addition to the stocks themselves. We might end up with something like:

Primero, editemos `src/app/stock/stock-list.component.html` para mostrar la cantidad de acciones que tenemos además de las acciones mismas. Podríamos terminar con algo como:

```
<h2>
  We have found {{(stocks$ | async)?.length}} stocks!
</h2>

<app-stock-item *ngFor="let stock of stocks$ | async"
  [stock]="stock">
</app-stock-item>
```

We have added a `div`, which shows the length (if present, which is what the `?` syntax does—it marks the element as optional, thus preventing failure in case of nulls, etc.). Now if you run the application, you will see the list of stocks as well as "We have found 3 stocks!"

Hemos agregado un `div`, que muestra la longitud (si está presente, que es lo que hace la sintaxis `?` : marca el elemento como opcional, evitando así fallas en caso de valores nulos, etc.). Ahora, si ejecuta la aplicación, verá la lista de acciones y también "¡Hemos encontrado 3 acciones!"

But open up the network inspector and you will notice an interesting thing. There will actually be two different calls going for fetching the list of stocks. Why? Because Angular observables are cold by default. And so every time someone subscribes to it, the observable is triggered. In this case, we have two subscribers, which are the two

Async pipes, one for the `ngFor` and one for the `length`. Thus, instead of using the same observable, we end up with two different calls being made.

Pero abre el inspector de red y notarás algo interesante. En realidad, habrá dos llamadas diferentes para obtener la lista de acciones. ¿Por qué? Porque los observables angulares son fríos por defecto. Y así, cada vez que alguien se suscribe, se activa el observable. En este caso, tenemos dos suscriptores, que son las dos tuberías Async, una para el `ngFor` y otra para el `length`. Por lo tanto, en lugar de utilizar el mismo observable, terminamos realizando dos llamadas diferentes.

COLD VERSUS HOT OBSERVABLES OBSERVABLES FRÍOS VERSUS CALIENTES

We just mentioned that Angular observables are cold observables by default. What does this mean for us? Fundamentally, an observable is nothing but a function that connects a producer to a consumer.

Acabamos de mencionar que los observables angulares son observables fríos por defecto. ¿Qué significa esto para nosotros? Fundamentalmente, un observable no es más que una función que conecta a un productor con un consumidor.

A cold observable is responsible for creating the producer as well, while a hot observable ends up sharing the producer.

Un observable frío también es responsable de crear al productor, mientras que un observable caliente termina compartiendo al productor.

For us, this just means that if whenever someone subscribes to an observable in Angular, the producer is created for that instance. This is why for each subscribe, we end up with a new producer.

Para nosotros, esto solo significa que si cada vez que alguien se suscribe a un observable en Angular, se crea el productor para esa instancia. Es por eso que por cada suscripción, terminamos con un nuevo productor.

You can read up more on hot and cold observables in [this article](#).

Puede leer más sobre observables fríos y calientes en este artículo.

Now how can we solve it? We have a few options:

Ahora ¿cómo podemos solucionarlo? Tenemos algunas opciones:

- We can tell Angular to share the same observable, thus preventing two calls.

Podemos decirle a Angular que comparta el mismo observable, evitando así dos llamadas.

- We can manually subscribe to the observable in the component, and capture the event response and save it to a class variable. Then the template can access the class variable instead of relying on the Async pipe.

Podemos suscribirnos manualmente al observable en el componente, capturar la respuesta del evento y guardarla en una variable de clase. Entonces la plantilla puede acceder a la variable de clase en lugar de depender de la tubería Async.

- We can choose not to use observables and instead use the promise to get at the underlying value. Promises do work in Angular, and you can convert any observable into a promise by calling `toPromise` on the observable (after adding the operator `toPromise` first of course by importing it).

Podemos optar por no utilizar observables y, en su lugar, utilizar la promesa para obtener el valor subyacente. Las promesas funcionan en Angular y puedes convertir cualquier observable en una promesa llamando a `toPromise` en el observable (después de agregar primero el operador `toPromise`, por supuesto, importándolo).

All of these are acceptable options, depending on the use case. While Angular pushes us toward using observables, there is no hard-and-fast rule that you can't convert it into a promise. And there are cases where it makes sense to deal with a promise instead of an observable.

Todas estas son opciones aceptables, según el caso de uso. Si bien Angular nos empuja a usar observables, no existe una regla estricta que indique que no se puede convertir en una promesa. Y hay casos en los que tiene sentido tratar con una promesa en lugar de un observable.

We won't go into code snippets for the latter two, though. Let's see how we can share the same observable, by changing the `src/app/stock/stock-list/stock-list.component.ts` file as follows:

Sin embargo, no entraremos en fragmentos de código para los dos últimos. Veamos cómo podemos compartir el mismo observable, cambiando el archivo `src/app/stock/stock-list/stock-list.component.ts` de la siguiente manera:

```
import { Component, OnInit } from '@angular/core';
import { StockService } from 'app/services/stock.service';
import { Stock } from 'app/model/stock';
import { Observable } from 'rxjs/Observable';

import { share } from 'rxjs/operators';

@Component({
  selector: 'app-stock-list',
  templateUrl: './stock-list.component.html',
  styleUrls: ['./stock-list.component.css']
})
export class StockListComponent implements OnInit {

  public stocks$: Observable<Stock[]>;

  constructor(private stockService: StockService) { }

  ngOnInit() {
    this.stocks$ = this.stockService.getStocks()
      .pipe(share());
  }
}
```

We have simply imported and used the `share` operator from RxJS. Then, in the `ngOnInit`, instead of saving the `getStocks()` observable directly, we pipe our base observable and add the `share()` operator

to our pipe. We save this observable as a member variable. This ensures that regardless of how many subscriptions are present on the observable, there would be only one underlying trigger to the server. Now when you run the application, you should see only one request being made to fetch the list of stocks, and still see both the number of stocks as well as the individual stocks.

Simplemente importamos y utilizamos el operador `share` de RxJS. Luego, en `ngOnInit`, en lugar de guardar el observable `getStocks()` directamente, canalizamos nuestro observable base y agregamos el operador `share()` a nuestra canalización. Guardamos este observable como una variable miembro. Esto garantiza que, independientemente de cuántas suscripciones haya presentes en el observable, solo habrá un activador subyacente para el servidor. Ahora, cuando ejecute la aplicación, debería ver solo una solicitud para obtener la lista de acciones y aún así ver tanto el número de acciones como las acciones individuales.

TIP CONSEJO

Be careful when you use AsyncPipe in your application, as using multiple async pipes on the same observable without sharing the underlying observable underneath would result in multiple server calls.

Tenga cuidado cuando use AsyncPipe en su aplicación, ya que usar múltiples canalizaciones async en el mismo observable sin compartir el observable subyacente debajo resultaría en múltiples llamadas al servidor.

Another option is to use the as operator along with the async pipe. This assigns the variable to a template variable for easy access and use. We can do something like:

Otra opción es utilizar el operador as junto con la tubería async. Esto asigna la variable a una variable de plantilla para facilitar el acceso y el uso. Podemos hacer algo como:

```
<li *ngFor="let stock of stocks$ | async as stocks;  
        index as i">  
    {{ stock.name }} ({{ i }} of {{ stocks.length }})  
</li>
```

The problem is that the template variable is scoped to the element, and cannot be accessed outside like in our example.

El problema es que la variable de plantilla tiene como ámbito el elemento y no se puede acceder a ella desde fuera como en nuestro ejemplo.

Next, let's add a simple search field, and see how we might fetch a list of stocks based on the search term from the server. We will take this step by step, by first updating our service to make the modified server call, and then adding a search field in the UI.

A continuación, agreguemos un campo de búsqueda simple y veamos cómo podemos obtener una lista de acciones según el término de búsqueda del servidor. Haremos esto paso a paso, primero actualizando nuestro servicio para realizar la llamada al servidor modificado y luego agregando un campo de búsqueda en la interfaz de usuario.

First, let's change our StockService to support searching for stocks with a query string. We will update the `src/app/services/stock.service.ts` file as follows:

Primero, cambiemos nuestro StockService para admitir la búsqueda de acciones con una cadena de consulta. Actualizaremos el archivo `src/app/services/stock.service.ts` de la siguiente manera:

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

import { Observable } from 'rxjs/Observable';

import { Stock } from 'app/model/stock';

@Injectable()
export class StockService {

  constructor(private http: HttpClient) {}

  getStocks(query: string) : Observable<Stock[]> {
    return this.http.get<Stock[]>(`/api/stock?q=${query}`);
  }

  /** Remaining same, omitted for brevity */
}
```

We have changed the definition of the `getStocks` method to take a `query` parameter, which is then passed to the `/api/stock` server call as a query parameter. Note that we could have passed it as an `options` object as the second argument as well instead of appending it to the URL.

Hemos cambiado la definición del método `getStocks` para tomar un parámetro `query`, que luego se pasa a la llamada al servidor `/api/stock` como parámetro de consulta. Tenga en cuenta que también podríamos haberlo pasado como un objeto de opciones como segundo argumento en lugar de agregarlo a la URL.

Now we need to change our StockListComponent to make the updated call. While we are doing that, we will also add an input field

bound to a model in the component, which will drive our search-as-you-type capability. Let's modify the template first to add the new form field, by changing `src/app/stock/stock-list/stock-list.component.html` as follows:

Ahora necesitamos cambiar nuestro `StockListComponent` para realizar la llamada actualizada. Mientras hacemos eso, también agregaremos un campo de entrada vinculado a un modelo en el componente, lo que impulsará nuestra capacidad de búsqueda mientras escribe. Primero modifiquemos la plantilla para agregar el nuevo campo de formulario, cambiando `src/app/stock/stock-list/stock-list.component.html` de la siguiente manera:

```
<div>
  <input name="searchBox"
    [(ngModel)]="searchString"
    placeholder="Search Here"
    (keyup)="search()">
</div>

<h2>
  We have found {{(stocks$ | async)?.length}} stocks!
</h2>

<app-stock-item *ngFor="let stock of stocks$ | async"
  [stock]="stock">
</app-stock-item>
```

We have added an `input` field named `searchBox` to the template, and bound it using `ngModel` to a member variable called `searchString`. In addition, on every `keyup` event, we are triggering a method called `search()`. Let's see how the component in `src/app/stock/stock-list/stock-list.component.ts` has to evolve:

Agregamos un campo `input` llamado `searchBox` a la plantilla y lo vinculamos usando `ngModel` a una variable miembro llamada `searchString`. Además, en cada evento `keyup`, activamos un método llamado `search()`. Veamos cómo tiene que evolucionar el componente en `src/app/stock/stock-list/stock-list.component.ts`:

```

import { Component, OnInit } from '@angular/core';
import { StockService } from 'app/services/stock.service';
import { Stock } from 'app/model/stock';
import { Observable } from 'rxjs/Observable';

import { share } from 'rxjs/operators';

@Component({
  selector: 'app-stock-list',
  templateUrl: './stock-list.component.html',
  styleUrls: ['./stock-list.component.css']
})
export class StockListComponent implements OnInit {

  public stocks$: Observable<Stock[]>;
  public searchString: string = '';

  constructor(private stockService: StockService) { }

  ngOnInit() {
    this.stocks$ = this.stockService.getStocks(this.searchString)
      .pipe(share());
  }

  search() {
    this.stocks$ = this.stockService.getStocks(this.searchString)
      .pipe(share());
  }
}

```

We've not done much at this point; we've simply replicated our logic of fetching the stocks in another method called `search()`. We have also ensured that we are passing along the query term to our `StockService`. If you run the application at this point, you will see the new search box. And if you type into the box, it does actually make a request to the server and search. But opening the network inspector in the developer tools tells the underlying story, which is that:

No hemos hecho mucho en este momento; simplemente hemos replicado nuestra lógica de recuperar las acciones en otro método llamado `search()`. También nos hemos asegurado de pasar el término de consulta a nuestro `StockService`. Si ejecuta la aplicación

en este punto, verá el nuevo cuadro de búsqueda. Y si escribe en el cuadro, en realidad realiza una solicitud al servidor y realiza una búsqueda. Pero abrir el inspector de red en las herramientas de desarrollo cuenta la historia subyacente, que es la siguiente:

- For every keystroke, we make a request to the server. This is horrifyingly inefficient.

Por cada pulsación de tecla, realizamos una solicitud al servidor. Esto es terriblemente ineficiente.

- We make requests for duplicate values even if we had already made a request for the previous one.

Realizamos solicitudes de valores duplicados incluso si ya habíamos realizado una solicitud para el anterior.

Thankfully, because we only hold the reference to the latest observable, we don't have to worry about out-of-order responses, but if we were subscribing to the observable in our component, then we would have to ensure that the response we are working with is the latest and not an older, out-of-date response.

Afortunadamente, como solo tenemos la referencia al observable más reciente, no tenemos que preocuparnos por respuestas desordenadas, pero si nos suscribíramos al observable en nuestro componente, entonces tendríamos que asegurarnos de que la respuesta que estamos trabajando es la respuesta más reciente y no una respuesta más antigua y desactualizada.

Now let's see how we can leverage observable operators to solve these problems in a clean, simple manner. We will change the component's implementation to the following now:

Ahora veamos cómo podemos aprovechar los operadores observables para resolver estos problemas de una manera limpia y sencilla. Ahora cambiaremos la implementación del componente a lo siguiente:

```

import { Component, OnInit } from '@angular/core';
import { StockService } from 'app/services/stock.service';
import { Stock } from 'app/model/stock';
import { Observable } from 'rxjs/Observable';
import { Subject } from 'rxjs/Subject';

import { debounceTime, switchMap,
         distinctUntilChanged, startWith,
         share } from 'rxjs/operators';

@Component({
  selector: 'app-stock-list',
  templateUrl: './stock-list.component.html',
  styleUrls: ['./stock-list.component.css']
})
export class StockListComponent implements OnInit {

  public stocks$: Observable<Stock[]>;
  public searchString: string = '';

  private searchTerms: Subject<string> = new Subject();
  constructor(private stockService: StockService) { }

  ngOnInit() {
    this.stocks$ = this.searchTerms.pipe(
      startWith(this.searchString),
      debounceTime(500),
      distinctUntilChanged(),
      switchMap((query) => this.stockService.getStocks(query)),
      share()
    );
  }

  search() {
    this.searchTerms.next(this.searchString);
  }
}

```

We have given the component a major overhaul, so let's talk through the various changes and why we have done them:

Le hemos dado al componente una revisión importante, así que hablemos de los diversos cambios y por qué los hemos realizado:

- The first major thing we have done is that we have introduced a member variable called `searchTerms`, which is a `Subject`. A `Subject` is a special type in RxJS that acts as both an observer as well as an observable. That is, it is capable of both emitting events as well as subscribing to one. We will use this `searchTerms` subject to trigger an event whenever the user types in our search box.

Lo primero que hemos hecho es introducir una variable miembro llamada `searchTerms`, que es un `Subject`. Un `Subject` es un tipo especial en RxJS que actúa como observador y observable. Es decir, es capaz tanto de emitir eventos como de suscribirse a uno. Usaremos este asunto `searchTerms` para activar un evento cada vez que el usuario escriba en nuestro cuadro de búsqueda.

- At this point, our `ngOnInit` now starts with the `Subject` we created, rather than the `StockService`. This means our chain of observable operators will be triggered each time the `Subject` gets a new term. The `Subject` is hooked in the `search()` to emit an event each time the user enters a letter.

En este punto, nuestro `ngOnInit` ahora comienza con el `Subject` que creamos, en lugar del `StockService`. Esto significa que nuestra cadena de operadores observables se activará cada vez que `Subject` obtenga un nuevo término. El `Subject` está enganchado en el `search()` para emitir un evento cada vez que el usuario ingresa una letra.

- Of course, we don't want to trigger the server call each time the user enters a key, so we introduce the `debounceTime()` operator in the chain. To chain, we use the `pipe` operator on the observable, and then can add any number of operators separated as arguments to the `pipe` function. Here, we instruct the stream to hold until there are no more new events for a

period of 500 milliseconds. This ensures that we only send a call once the user stops typing for half a second.

Por supuesto, no queremos activar la llamada al servidor cada vez que el usuario ingresa una clave, por lo que introducimos el operador `debounceTime()` en la cadena. Para encadenar, usamos el operador `pipe` en el observable y luego podemos agregar cualquier número de operadores separados como argumentos a la función `pipe`. Aquí, le indicamos a la transmisión que se mantenga en espera hasta que no haya más eventos nuevos durante un período de 500 milisegundos. Esto garantiza que solo enviamos una llamada una vez que el usuario deja de escribir durante medio segundo.

- The next thing we want to do is to avoid unnecessary calls, like if the user enters a search term (say, "test"), then types a few more characters and then erases them to land back at the same word he started with. Instead of adding checks and local variables, we use another observable operator called `distinctUntilChanged()`. This ensures that the event is only emitted if the new value is different from the previous value, thus saving a few more network calls.

Lo siguiente que queremos hacer es evitar llamadas innecesarias, como si el usuario ingresa un término de búsqueda (por ejemplo, "prueba"), luego escribe algunos caracteres más y luego los borra para regresar a la misma palabra con la que comenzó. En lugar de agregar comprobaciones y variables locales, utilizamos otro operador observable llamado `distinctUntilChanged()`. Esto garantiza que el evento solo se emita si el nuevo valor es diferente del valor anterior, ahorrando así algunas llamadas de red más.

- So far, we are using a Subject that emits string values. But our observable that we bind to is one of `Stock[]`. Converting an observable chain from one type to another is usually the work of

the `map` operator. But we will use a particular type of `map` operator called the `switchMap`. The `switchMap` has a nice behavior that in addition to converting from one type of observable to another, it also has the capability to cancel old, in-flight subscriptions. This helps to solve our out-of-order response problem in a nice, clean manner. Note that it does not necessarily cancel the underlying HTTP request, but simply drops the subscription.

Hasta ahora, estamos usando un `Subject` que emite valores `string`. Pero nuestro observable al que nos unimos es uno de `Stock[]`. La conversión de una cadena observable de un tipo a otro suele ser obra del operador `map`. Pero usaremos un tipo particular de operador `map` llamado `switchMap`. El `switchMap` tiene un comportamiento agradable que, además de convertir de un tipo de observable a otro, también tiene la capacidad de cancelar suscripciones antiguas en vuelo. Esto ayuda a resolver nuestro problema de respuesta desordenada de una manera agradable y limpia. Tenga en cuenta que no necesariamente cancela la solicitud HTTP subyacente, sino que simplemente cancela la suscripción.

- If we leave it at just this, our chain will only start the first moment the user starts typing into the search box, and will result in an empty list of stocks when the page loads. We can solve this by using an operator called `startWith`, which sets the initial value with which the observable chain is to be triggered. We start it with an empty string, which ensures that when the page is loaded, we see our original list of stocks.

Si lo dejamos así, nuestra cadena solo comenzará en el primer momento en que el usuario comience a escribir en el cuadro de búsqueda y dará como resultado una lista vacía de acciones cuando se cargue la página. Podemos resolver esto usando un operador llamado `startWith`, que establece el valor inicial con el

que se activará la cadena observable. Lo comenzamos con una cadena vacía, lo que garantiza que cuando se carga la página, veamos nuestra lista original de acciones.

At this point, with a chain of four RxJS operators, we can now run our application. When you type in the search box, it will wait for you to stop typing for a period before making the request. If you type and erase to land back at the starting value, it will not make the request at all.

En este punto, con una cadena de cuatro operadores RxJS, ya podemos ejecutar nuestra aplicación. Cuando escriba en el cuadro de búsqueda, esperará a que deje de escribir durante un período antes de realizar la solicitud. Si escribe y borra para volver al valor inicial, no realizará la solicitud en absoluto.

There are many more operators available in RxJS than what I can reasonably cover in this book, so I won't even try. This section is more to give you an idea of what RxJS is really capable of.

Whenever you are trying to do any complex work, see if you can tackle it using operators rather than doing it manually. The [official RxJS documentation](#) is a great place to start on learning them.

Hay muchos más operadores disponibles en RxJS de los que razonablemente puedo cubrir en este libro, así que ni siquiera lo intentaré. Esta sección es más para darle una idea de lo que RxJS es realmente capaz de hacer. Siempre que intente realizar un trabajo complejo, vea si puede realizarlo utilizando operadores en lugar de hacerlo manualmente. La documentación oficial de RxJS es un excelente lugar para comenzar a aprenderlos.

Conclusion

Conclusión

In this chapter, we learned how to make HTTP calls using the `HttpClient` in Angular. We started with GET and POST calls using the `HttpClient`, before diving deeper into the API provided in the `HttpClient`. This included working with headers and query parameters, as well as working with different levels of detail in the response and different response types. We then moved on to handling common use cases of hooking onto every HTTP request and response using interceptors, and saw how to create and hook our own interceptor to the Angular HTTP chain. Finally, we saw how to leverage observables to accomplish some complex tasks in a simple, efficient manner with an example of search-as-you-type.

En este capítulo, aprendimos cómo realizar llamadas HTTP usando `HttpClient` en Angular. Comenzamos con llamadas GET y POST usando `HttpClient`, antes de profundizar en la API proporcionada en `HttpClient`. Esto incluyó trabajar con encabezados y parámetros de consulta, así como trabajar con diferentes niveles de detalle en la respuesta y diferentes tipos de respuesta. Luego pasamos a manejar casos de uso comunes de vinculación a cada solicitud y respuesta HTTP mediante interceptores, y vimos cómo crear y vincular nuestro propio interceptor a la cadena HTTP Angular. Finalmente, vimos cómo aprovechar los observables para realizar algunas tareas complejas de una manera simple y eficiente con un ejemplo de búsqueda mientras se escribe.

In the next chapter, we will take a step back and learn how to write unit tests for services, as well as how to unit-test flows where HTTP calls are being made through the `HttpClient`.

En el próximo capítulo, daremos un paso atrás y aprenderemos cómo escribir pruebas unitarias para servicios, así como también

cómo realizar pruebas unitarias de flujos donde se realizan llamadas HTTP a través de `HttpClient`.

Exercise

Ejercicio

Take the finished exercise from [Chapter 8](#) (available in `chapter8/exercise/e-commerce`). Install and run the server in the `chapter9/exercise/server` folder by running:

Realice el ejercicio terminado del Capítulo 8 (disponible en el capítulo 8/ejercicio/comercio electrónico). Instale y ejecute el servidor en la carpeta capitulo9/ejercicio/servidor ejecutando:

```
npm i  
node index.js
```

from within the folder. This will start a local Node.js server, which can work with products (similar to one we had for stocks). It exposes the following APIs:

desde dentro de la carpeta. Esto iniciará un servidor Node.js local, que puede funcionar con productos (similar al que teníamos para las existencias). Expone las siguientes API:

- GET on `/api/product` to get a list of products. It can also take an optional query param `q`, which is the product name to search for.

OBTÉN `/api/product` para obtener una lista de productos. También puede tomar un parámetro de consulta opcional `q`, que es el nombre del producto a buscar.

- POST on `/api/product` with product information in the body to create a product on the server (in-memory of course; restarting the server would lose all created products).

PUBLICAR en `/api/product` con información del producto en el cuerpo para crear un producto en el servidor (en memoria, por supuesto; reiniciar el servidor perdería todos los productos creados).

- PATCH on `/api/product/:id` with the product ID in the URL and a field `changeInQuantity` in the body would change the quantity in cart of the product by that amount.

PATCH en `/api/product/:id` con el ID del producto en la URL y un campo `changeInQuantity` en el cuerpo cambiaría la cantidad en el carrito del producto en esa cantidad.

Given this, now try to accomplish the following:

Teniendo esto en cuenta, ahora intenta lograr lo siguiente:

1. Change over the ProductService to make HTTP calls instead of responding with mock data. Support searching for a list of products as well.

Cambie el ProductService para realizar llamadas HTTP en lugar de responder con datos simulados. También admite la búsqueda de una lista de productos.

2. Implement both product listing and search-as-you-type using the power of observables.

Implemente tanto el listado de productos como la búsqueda mientras escribe utilizando el poder de los observables.

3. Handle product creation, as well as change in quantity of an individual product and hook it up end-to-end.

Maneje la creación de productos, así como el cambio en la cantidad de un producto individual y conéctelo de un extremo a

otro.

4. See if you can continue using the same observable chain to now reload the entire list of products each time a product is created or the quantity is changed.

Vea si puede continuar usando la misma cadena observable para recargar ahora la lista completa de productos cada vez que se crea un producto o se cambia la cantidad.

Most of this can be accomplished using concepts covered in this chapter. The only tricky thing is how to reload the list of products when it happens in a different component, for which you can leverage template reference variables to access the component and make a call. The other thing you might need to use is the `merge` operator in the observable so that you can leverage the same observable for loading the list, searching products, and reloading the list. You can check out the finished solution in [*chapter9/exercise/ecommerce*](#).

La mayor parte de esto se puede lograr utilizando los conceptos tratados en este capítulo. Lo único complicado es cómo recargar la lista de productos cuando ocurre en un componente diferente, para lo cual puede aprovechar las variables de referencia de la plantilla para acceder al componente y realizar una llamada. La otra cosa que quizás necesites usar es el operador `merge` en el observable para poder aprovechar el mismo observable para cargar la lista, buscar productos y recargar la lista. Puede consultar la solución terminada en el capítulo 9/ejercicio/ecommerce.

Chapter 10. Unit Testing Services

Capítulo 10. Servicios de pruebas unitarias

In the previous two chapters, we started understanding what Angular services are, when to create them, and how to use them. We also started learning how to make HTTP calls and handle the various use cases that crop up when working with servers.

En los dos capítulos anteriores, comenzamos a comprender qué son los servicios Angular, cuándo crearlos y cómo usarlos. También comenzamos a aprender cómo realizar llamadas HTTP y manejar los diversos casos de uso que surgen al trabajar con servidores.

In this chapter, we will take a step back and try to see how we can unit test these services. We will first see how to unit test a service, followed by understanding how to leverage the Angular dependency injection system to mock out service dependencies in unit tests. Finally, we will dig into writing unit tests when we are working with `HttpClient`.

En este capítulo, daremos un paso atrás e intentaremos ver cómo podemos realizar pruebas unitarias de estos servicios. Primero veremos cómo realizar una prueba unitaria de un servicio, y luego comprenderemos cómo aprovechar el sistema de inyección de dependencia Angular para simular las dependencias del servicio en

pruebas unitarias. Finalmente, profundizaremos en la escritura de pruebas unitarias cuando trabajemos con `HttpClient`.

If you want to quickly recap what unit tests are and how to write them for components, you can refer to [Chapter 5](#).

Si desea recapitular rápidamente qué son las pruebas unitarias y cómo escribirlas para componentes, puede consultar el Capítulo 5.

How to Unit Test Services

Cómo realizar pruebas unitarias de servicios

The first thing we will start with is learning how to unit test very simple services. These might be services without any dependencies that act as encapsulators of business logic or functionality that needs to be reused across our application.

Lo primero que comenzaremos es aprender a realizar pruebas unitarias de servicios muy simples. Estos pueden ser servicios sin dependencias que actúan como encapsuladores de lógica empresarial o funcionalidad que debe reutilizarse en nuestra aplicación.

We will start with testing the very simple service we built in [Chapter 8](#). You can use the codebase in `chapter8/simple-service` as the base for this section. The finished code is available in `chapter10/simple-service`.

Comenzaremos probando el servicio muy simple que creamos en el Capítulo 8. Puede usar el código base en el capítulo 8/simple-service como base para esta sección. El código terminado está disponible en el capítulo 10/servicio simple.

Any time we are unit testing our service, we must do a few things over and above what we did for testing components. Namely:

Cada vez que realizamos pruebas unitarias de nuestro servicio, debemos hacer algunas cosas además de lo que hicimos para probar los componentes. A saber:

- Configure the Angular TestBed with a provider for the service we want to test.

Configurar el Angular TestBed con un proveedor para el servicio que queremos probar.

- Inject an instance of the service we want to test either into our test or as a common instance in the beforeEach.

Inyecte una instancia del servicio que queremos probar en nuestra prueba o como una instancia común en beforeEach.

When you generate a service using the Angular CLI, this initial skeleton is generated for you out of the box. Regardless, let's first take a look at the skeleton spec that was generated in `src/app/services/stock.service.spec.ts`:

Cuando genera un servicio utilizando Angular CLI, este esqueleto inicial se genera de forma inmediata. De todos modos, primero echemos un vistazo a la especificación del esqueleto que se generó en `src/app/services/stock.service.spec.ts`:

```
import { TestBed, inject } from '@angular/core/testing';

import { StockService } from './stock.service';

describe('StockService', () => {
  beforeEach(() => {
    TestBed.configureTestingModule({
      providers: [StockService]
    });
  });

  it('should be created', inject([StockService],
```

```
(service: StockService) => {
  expect(service).toBeTruthy();
});
});
```

The basic skeleton itself allows us to walk through some of the initial setup that we mentioned. Let's cover the most important bits:

El esqueleto básico en sí nos permite recorrer parte de la configuración inicial que mencionamos. Cubramos las partes más importantes:

- In the `beforeEach`, like we used to register the components, we now register the provider for the `StockService`. This ensures that the test is now running in the context of our testing module.

En el `beforeEach`, como solíamos registrar los componentes, ahora registramos el provider para el `StockService`. Esto garantiza que la prueba ahora se esté ejecutando en el contexto de nuestro módulo de prueba.

- In the `it`, which is the actual test, instead of just passing the test function to it, we call `inject`, which is a function provided by the Angular testing utilities. We pass an array to it as the first argument, which are the Angular services that need to be injected into our test. The second argument is a function that gets the arguments in the same order that we passed to the array. We write our actual test within this function.

En `it`, que es la prueba real, en lugar de simplemente pasarle la función de prueba, llamamos a `inject`, que es una función proporcionada por las utilidades de prueba de Angular. Le pasamos una matriz como primer argumento, que son los servicios de Angular que deben inyectarse en nuestra prueba. El segundo argumento es una función que obtiene los argumentos en el mismo orden en que pasamos a la matriz. Escribimos nuestra prueba real dentro de esta función.

In the skeleton code, we instantiate the StockService provider with the testing module, and then just make sure that when we inject it in the test, we get an instantiated version of the service for use in the test.

En el código esqueleto, creamos una instancia del proveedor StockService con el módulo de prueba y luego nos aseguramos de que cuando lo inyectemos en la prueba, obtengamos una versión instanciada del servicio para usar en la prueba.

Before we get on to the actual test, let's quickly review the service that we will be testing. The StockService currently looks like the following:

Antes de pasar a la prueba real, revisemos rápidamente el servicio que probaremos. El StockService actualmente se parece a lo siguiente:

```
import { Injectable } from '@angular/core';
import { Stock } from 'app/model/stock';

@Injectable()
export class StockService {

  private stocks: Stock[];
  constructor() {
    this.stocks = [
      new Stock('Test Stock Company', 'TSC', 85, 80, 'NASDAQ'),
      new Stock('Second Stock Company', 'SSC', 10, 20, 'NSE'),
      new Stock('Last Stock Company', 'LSC', 876, 765, 'NYSE')
    ];
  }

  getStocks(): Stock[] {
    return this.stocks;
  }

  createStock(stock: Stock) {
    let foundStock = this.stocks.find(each => each.code === stock.code);
    if (foundStock) {
      return false;
    }
    this.stocks.push(stock);
  }
}
```

```

        return true;
    }

    toggleFavorite(stock: Stock) {
        let foundStock = this.stocks.find(each => each.code === stock.code);
        foundStock.favorite = !foundStock.favorite;
    }
}

```

The StockService has three stocks that are instantiated by default. When we call `getStocks()`, this list of three stocks is returned. We can also add stocks by calling `createStock`, which checks for the presence of the stock and then adds it.

El StockService tiene tres acciones de las que se crean instancias de forma predeterminada. Cuando llamamos a `getStocks()`, se devuelve esta lista de tres acciones. También podemos agregar acciones llamando a `createStock`, que verifica la presencia de las acciones y luego las agrega.

Now let's improve it to actually test the service, which had the capability of getting a list of stocks and adding stocks. We will add two tests for these two methods:

Ahora mejorémoslo para probar realmente el servicio, que tenía la capacidad de obtener una lista de acciones y agregar acciones. Agregaremos dos pruebas para estos dos métodos:

```

/** Other imports skipped for brevity */
import { Stock } from 'app/model/stock';

describe('StockService', () => {
    var stockService: StockService;
    beforeEach(() => {
        /* No change in first beforeEach, skipping for brevity */
    });

    beforeEach(inject([StockService]),
        (service: StockService) => {
            stockService = service;
    ));
}

```

①

```

it('should allow adding stocks', () => {
  expect(stockService.getStocks().length).toEqual(3);          ②
  let stock = new Stock('Testing A New Company', 'TTT',
    850, 800, 'NASDAQ');
  expect(stockService.createStock(stock)).toBeTruthy();        ③
  expect(stockService.getStocks().length).toEqual(4);           ④
  expect(stockService.getStocks()[3].code).toEqual('TTT')
});

it('should fetch a list of stocks', () => {
  expect(stockService.getStocks().length).toEqual(3);          ⑤
  expect(stockService.getStocks()[0].code).toEqual('TSC');      ⑥
  expect(stockService.getStocks()[1].code).toEqual('SSC');
  expect(stockService.getStocks()[2].code).toEqual('LSC');
});
});

```

- ❶ Inject StockService into another beforeEach and save it for ~~to jest a stock service~~ en otro beforeEach y guárdelo para acceder a él en todas las pruebas.
- ❷ Ensure that we start with the original three stocks from our ~~service~~ se de comenzar con las tres acciones originales de nuestro servicio.
- ❸ Add the stock and ensure that it returns true ~~to be truthy~~
- ❹ Verify the presence of the stock we just added in the service.
- ❺ Ensure that we start with the original three stocks from our ~~service~~ se de comenzar con las tres acciones originales de nuestro servicio.
- ❻ Verify each stock to make sure it's the same as what we expect los datos que esperamos.

We have made a slight change to the initialization logic, and added two pretty mundane, run-of-the-mill tests:

Hicimos un ligero cambio en la lógica de inicialización y agregamos dos pruebas comunes y corrientes bastante mundanas:

- Instead of writing an `inject` block in each test (each `it` block), we have moved that logic to another `beforeEach` block, which is solely responsible for setting up local variables that will be repeatedly used across all the tests. Note that this does not mean the same instance is used in all the tests, but that we don't have to inject the service into each test individually.

En lugar de escribir un bloque `inject` en cada prueba (cada bloque `it`), hemos movido esa lógica a otro bloque `beforeEach`, que es el único responsable de configurar las variables locales que se repetirán repetidamente. utilizado en todas las pruebas. Tenga en cuenta que esto no significa que se utilice la misma instancia en todas las pruebas, sino que no tenemos que inyectar el servicio en cada prueba individualmente.

- We have added two tests, one to test adding of stocks and another to check the default `getStocks()` call. Again, note that the two tests are independent. Our adding a stock in the first stock does not result in there being four stocks in the second test. Before each test, we are creating a new testing module with a new instance of the service.

Hemos agregado dos pruebas, una para probar la adición de acciones y otra para verificar la llamada predeterminada `getStocks()`. Nuevamente, tenga en cuenta que las dos pruebas son independientes. Agregar una acción en la primera acción no da como resultado que haya cuatro acciones en la segunda prueba. Antes de cada prueba, creamos un nuevo módulo de prueba con una nueva instancia del servicio.

Other than this, the test itself is pretty straightforward and self-explanatory. To run this test, simply execute:

Aparte de esto, la prueba en sí es bastante sencilla y se explica por sí misma. Para ejecutar esta prueba, simplemente ejecute:

```
ng test
```

That should automatically start Karma, capture Chrome, run the tests, and report the results right in your terminal window.

Eso debería iniciar Karma automáticamente, capturar Chrome, ejecutar las pruebas e informar los resultados directamente en la ventana de su terminal.

HANDLING SERVICE DEPENDENCIES? ¿MANEJO DE DEPENDENCIAS DE SERVICIOS?

What if our service itself had a dependency on another service? Well, we would handle it exactly the same way. We have a few options:

¿Qué pasaría si nuestro propio servicio dependiera de otro servicio? Bueno, lo manejaríamos exactamente de la misma manera. Tenemos algunas opciones:

- Register the dependency as a service with the Angular TestBed module, and let Angular be responsible for injecting it into the service we are testing.

Registre la dependencia como un servicio con el módulo Angular TestBed y deje que Angular sea responsable de inyectarla en el servicio que estamos probando.

- Override/mock the dependent service by registering a fake/stub provider with the TestBed, and rely on that instead of the original service.

Anule o simule el servicio dependiente registrando un proveedor falso/stub con TestBed y confíe en él en lugar del servicio original.

For both of these, we would simply add another provider in the `TestBed.configureTestingModule` call for the other service. We will see this principle in action in the next section.

Para ambos, simplemente agregariamos otro provider en la llamada `TestBed.configureTestingModule` para el otro servicio. Veremos este principio en acción en la siguiente sección.

Testing Components with a Service Dependency

Prueba de componentes con dependencia del servicio

Next, let's see how to deal with two slightly different situations:

A continuación, veamos cómo afrontar dos situaciones ligeramente diferentes:

- If we had to test a component, and actually use the real service underneath in the test.

Si tuviéramos que probar un componente y utilizar el servicio real que se encuentra debajo en la prueba.

- If we had to test a component, and we wanted to mock out the service it depends on in the test.

Si tuviéramos que probar un componente y quisiéramos simular el servicio del que depende en la prueba.

Testing Components with a Real Service

Probar componentes con un servicio real

First, let's take a look at the test for StockListComponent if we were using the real service underneath. Our *src/app/stock/stock-list/stock-list.component.spec.ts* file would look like the following:

Primero, echemos un vistazo a la prueba de StockListComponent si estuviéramos usando el servicio real que se encuentra debajo.

Nuestro archivo *src/app/stock/stock-list/stock-list.component.spec.ts* tendría el siguiente aspecto:

```

import { async, ComponentFixture, TestBed } from '@angular/core/testing';
import { StockListComponent } from './stock-list.component';
import { StockService } from 'app/services/stock.service';
import { StockItemComponent } from 'app/stock/stock-item/stock-item.component';
import { Stock } from 'app/model/stock';

describe('StockListComponent With Real Service', () => {
  let component: StockListComponent;
  let fixture: ComponentFixture<StockListComponent>;

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [ StockListComponent, StockItemComponent ],
      providers: [ StockService ]           ①
    })
    .compileComponents();                      ②
  }));

  beforeEach(() => {
    fixture = TestBed.createComponent(StockListComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('should load stocks from real service on init', () => {
    expect(component).toBeTruthy();
    expect(component.stocks.length).toEqual(3);          ③
  });
});

```

- ① Add stock component to the declarations arrays
- ② Add stock service to the providers arrays
- ③ Ensure that the stocks in the component are loaded from the service
Asegúrese de que las existencias en el componente se carguen desde el servicio.

Most of the test is the autogenerated skeleton from the Angular CLI.
The changes we have made in particular are:

La mayor parte de la prueba es el esqueleto autogenerado de Angular CLI. Los cambios que hemos realizado en particular son:

- We have added a declaration of StockItemComponent in addition to the StockListComponent. This is because the template for our StockListComponent uses the StockItemComponent and hence it is needed for our test to succeed.

Hemos agregado una declaración de StockItemComponent además del StockListComponent. Esto se debe a que la plantilla para nuestro StockListComponent usa StockItemComponent y, por lo tanto, es necesaria para que nuestra prueba tenga éxito.

- We have added StockService in the array passed to the providers section of the testing module created. This ensures that in our test, we use the actual underlying StockService whenever it is asked for by any component.

Hemos agregado StockService en la matriz pasada a la sección providers del módulo de prueba creado. Esto garantiza que en nuestra prueba utilicemos el StockService subyacente real siempre que lo solicite cualquier componente.

- Finally, we have added an assertion to ensure that we load the three stocks that are returned by our StockService when the component is initialized.

Finalmente, hemos agregado una aserción para asegurarnos de cargar las tres acciones que devuelve nuestro StockService cuando se inicializa el componente.

If you notice, this should be similar to how we tested the service itself in the previous section. We simply add a provider for our service and then we have the real live service in our tests accessible.

Si lo notas, esto debería ser similar a cómo probamos el servicio en la sección anterior. Simplemente agregamos un proveedor para

nuestro servicio y luego tenemos accesible el servicio real en vivo en nuestras pruebas.

Testing Components with a Mock Service

Prueba de componentes con un servicio simulado

Next, let's see how we could write a similar test, but instead of using the real service in the test, how to mock out certain calls instead of creating a brand-new fake service just for our test. This approach is useful when we do actually want to use most of the service, but just override/mock out certain calls. This is also less cumbersome than creating and maintaining a full parallel fake implementation.

A continuación, veamos cómo podríamos escribir una prueba similar, pero en lugar de utilizar el servicio real en la prueba, cómo simular ciertas llamadas en lugar de crear un nuevo servicio falso solo para nuestra prueba. Este enfoque es útil cuando realmente queremos utilizar la mayor parte del servicio, pero simplemente anulamos o simulamos ciertas llamadas. Esto también es menos engorroso que crear y mantener una implementación falsa paralela completa.

Let's see how we could use the real service with just some calls mocked out in our test. Our revamped test in `src/app/stock/stock-list/stock-list.component.spec.ts` would look like this:

Veamos cómo podríamos usar el servicio real con solo algunas llamadas simuladas en nuestra prueba. Nuestra prueba renovada en `src/app/stock/stock-list/stock-list.component.spec.ts` se vería así:

```
/** Standard imports, skipping for brevity */

describe('StockListComponent With Mock Service', () => {
  let component: StockListComponent;
  let fixture: ComponentFixture<StockListComponent>;
  let stockService: StockService;

  beforeEach(async(() => {
```

```

    /** No change in TestBed configuration, skipping for brevity */
  });

beforeEach(() => {
  fixture = TestBed.createComponent(StockListComponent);
  component = fixture.componentInstance;
  // Always get the Service from the Injector!
  stockService = fixture.debugElement.injector.get(StockService); ①
  let spy = spyOn(stockService, 'getStocks') ②
    .and.returnValue([
      new Stock('Mock Stock', 'MS', 800, 900, 'NYSE')
    ]);
  fixture.detectChanges();
});

it('should load stocks from mocked service on init', () => {
  expect(component).toBeTruthy();
  expect(component.stocks.length).toEqual(1); ③
  expect(component.stocks[0].code).toEqual('MS');
});
});

```

- ① Get the `componentInstance` through the `component's injector`
- ② componente
- ③ Mock out the `getStocks()` call and return a hardcoded value
- ④ Sustituir la llamada `getStocks()` y devuelva un valor codificado en su lugar
- ⑤ Ensure that the stocks are provided from our mocked-out call
- ⑥ Asegúrese de que las existencias se proporcionen a partir de nuestra convocatoria simulada ahora

Our test looks mostly similar to the test in the previous section, especially in how we hook up the service as a provider to the TestBed. The major difference is in the second `beforeEach`, where we use the injector in the test fixture to get a handle on the StockService.

Nuestra prueba se parece en gran medida a la prueba de la sección anterior, especialmente en cómo conectamos el servicio como

proveedor al TestBed. La principal diferencia está en el segundo beforeEach, donde usamos el inyector en el dispositivo de prueba para controlar el StockService.

There are two ways for us to get the service instance in our tests. We can either rely on the `inject` function from the Angular testing utilities to inject the service instance, like we did in our test for StockService, or we can use the `injector` reference on the element, like we just did here.

Hay dos formas de obtener la instancia de servicio en nuestras pruebas. Podemos confiar en la función `inject` de las utilidades de prueba de Angular para injectar la instancia de servicio, como hicimos en nuestra prueba para StockService, o podemos usar la referencia `injector` en el elemento. , como acabamos de hacer aquí.

Once we have a handle on the service instance, we can use *Jasmine spies* to spy on different methods on the service. A spy (whether it is from Jasmine or any other framework) allows us to stub any function or method, and track any calls to it along with its arguments and also define our own return values.

Una vez que controlamos la instancia del servicio, podemos usar los espías Jasmine para espiar diferentes métodos del servicio. Un espía (ya sea de Jasmine o de cualquier otro marco) nos permite detectar cualquier función o método y rastrear cualquier llamada junto con sus argumentos y también definir nuestros propios valores de retorno.

In this case, we use `spyOn` to spy on a particular method in the service (the `getStocks()` call), and use it to change the return value to what we want, rather than the original underlying call. This way, the actual service call never gets invoked.

En este caso, usamos `spyOn` para espiar un método particular en el servicio (la llamada `getStocks()`) y lo usamos para cambiar el valor de retorno a lo que queremos, en lugar de la llamada subyacente

original. De esta manera, nunca se invoca la llamada de servicio real.

Our test then just has assertions to make sure the return value is from our mocked service call rather than the original service.

Luego, nuestra prueba solo tiene afirmaciones para garantizar que el valor de retorno provenga de nuestra llamada de servicio simulada y no del servicio original.

Testing Components with a Fake Service

Probar componentes con un servicio falso

The last option we have when we are writing tests for components or services that depend on other services is to replace the actual service with a fake that we have created just for the purpose of testing. This allows us to create a service tuned just for the test that gives us maybe more access, or just tracks what APIs are called with what values. Most of this could also be accomplished with Jasmine spies, but if you have a repeated use case, then it might make more sense to create a fake that can be reused.

La última opción que tenemos cuando escribimos pruebas para componentes o servicios que dependen de otros servicios es reemplazar el servicio real con uno falso que hemos creado solo con el propósito de realizar pruebas. Esto nos permite crear un servicio ajustado solo para la prueba que tal vez nos brinde más acceso, o simplemente rastrea qué API se llaman con qué valores. La mayor parte de esto también podría lograrse con los espías Jasmine, pero si tiene un caso de uso repetido, entonces podría tener más sentido crear una falsificación que pueda reutilizarse.

For us, a fake would simply be an object that we create, which has the same API as the service we are mocking, but our own (hardcoded) implementation of the methods underneath. For

example, our fake could simply return a different hardcoded list of stocks instead of making a server call in a test.

Para nosotros, una falsificación sería simplemente un objeto que creamos, que tiene la misma API que el servicio del que nos estamos burlando, pero nuestra propia implementación (codificada) de los métodos subyacentes. Por ejemplo, nuestra falsificación podría simplemente devolver una lista diferente de acciones codificada en lugar de realizar una llamada al servidor en una prueba.

Let's see how we could use a fake service in our test. Our revamped test in `src/app/stock/stock-list/stock-list.component.spec.ts` would look like this:

Veamos cómo podríamos utilizar un servicio falso en nuestra prueba. Nuestra prueba renovada en `src/app/stock/stock-list/stock-list.component.spec.ts` se vería así:

```
/** Skipping imports for brevity **/
```

```
describe('StockListComponent With Fake Service', () => {
  let component: StockListComponent;
  let fixture: ComponentFixture<StockListComponent>;

  beforeEach(async(() => {
    let stockServiceFake = {
      getStocks: () => {
        return [new Stock('Fake Stock', 'FS', 800, 900, 'NYSE')];
      }
    };
    TestBed.configureTestingModule({
      declarations: [ StockListComponent, StockItemComponent ],
      providers: [ {
        provide: StockService,
        useValue: stockServiceFake
      } ]
    })
    .compileComponents();
  }));
  
```

❶

```
  beforeEach(() => {
    fixture = TestBed.createComponent(StockListComponent);
  });
  
```

❷

```
  it('should display the correct stock information', () => {
    const compiled = fixture.debugElement.nativeElement;
    expect(compiled.querySelector('.stock-item').textContent).toContain('Fake Stock');
  });
});
```

```

        component = fixture.componentInstance;
        fixture.detectChanges();
    });

    it('should load stocks from fake service on init', () => {
        expect(component).toBeTruthy();
        expect(component.stocks.length).toEqual(1);
        expect(component.stocks[0].code).toEqual('FS'); ③
    });
});

```

- ❶ Define a stockServiceFake JavaScript object that implements a ~~StockService~~ interface. Definir un objeto JavaScript stockServiceFake que implemente un método getStocks()
- ❷ Specify the instance type when registering the provider. Especificar la instancia tipo cuando se registre el proveedor.
- ❸ Assign that the values are coming from the fake service. Asignar que los valores vienen del servicio falso.

There are a lot of differences in the test from the previous sections, so let's walk through some of the major changes:

Hay muchas diferencias en la prueba con respecto a las secciones anteriores, así que repasemos algunos de los cambios principales:

- We first create a fake service instance, called stockServiceFake. Note that we initialize it with only one of the methods (getStocks()), and not necessarily all the APIs in the service.

Primero creamos una instancia de servicio falsa, llamada stockServiceFake. Tenga en cuenta que lo inicializamos con solo uno de los métodos (getStocks()), y no necesariamente con todas las API del servicio.

- When we configure the testing module using the TestBed, instead of registering the StockService, we register a provider. We tell Angular that whenever someone asks for StockService (using the provide key), provide the value stockServiceFake

(using the `useValue`) key. This overrides the default behavior of providing the class instance.

Cuando configuramos el módulo de prueba usando `TestBed`, en lugar de registrar `StockService`, registramos un proveedor. Le decimos a Angular que cada vez que alguien solicite `StockService` (usando la clave `provide`), proporcione el valor `stockServiceFake` (usando la clave `useValue`). Esto anula el comportamiento predeterminado de proporcionar la instancia de clase.

PROVIDING AN INSTANCE OF A SERVICE PROPORCIONAR UNA INSTANCIA DE UN SERVICIO

Generally, when we identify a class as a provider to Angular, Angular would be responsible for instantiating an instance of the class and providing it when a component or a service depends on it.

Generalmente, cuando identificamos una clase como provider para Angular, Angular sería responsable de crear una instancia de la clase y proporcionarla cuando un componente o servicio dependa de ella.

In certain cases, we don't want Angular to instantiate it, but rather we want to define what value to use. That is when we use the mechanism we used in the preceding code, where we can define what class is being provided (using the `provide` key) and specify the instance to use, rather than letting Angular create the instance (using the `useValue` key).

En ciertos casos, no queremos que Angular lo cree una instancia, sino que queremos definir qué valor usar. Ahí es cuando usamos el mecanismo que usamos en el código anterior, donde podemos definir qué clase se proporciona (usando la clave `provide`) y especificar la instancia a usar, en lugar de permitir que Angular cree la instancia (usando la clave `useValue`).

You can read up more on the different ways you can configure Providers in the [official Angular docs](#).

Puede leer más sobre las diferentes formas en que puede configurar proveedores en los documentos oficiales de Angular.

Other than this, our test looks similar. We again assert that the data returned to the component is from our fake service, and not from the original service.

Aparte de esto, nuestra prueba es similar. Nuevamente afirmamos que los datos devueltos al componente provienen de nuestro servicio falso y no del servicio original.

ALWAYS GET SERVICES FROM THE INJECTOR OBTENGA SIEMPRE SERVICIOS DEL INYECTOR

Note that the recommended way to get the handle on an instance of a service, even when using fakes, is through the injector. This is because the `fakeStockService` instance we create in our test will not be the same as the instance provided by the Angular dependency injector. Thus, even with a fake, if you want to assert, for example, that a stock was added successfully to the service, you would want to do it against the service returned by the injector, and not the original instance we used.

Tenga en cuenta que la forma recomendada de controlar una instancia de un servicio, incluso cuando se utilizan falsificaciones, es a través del inyector. Esto se debe a que la instancia `fakeStockService` que creamos en nuestra prueba no será la misma que la instancia proporcionada por el inyector de dependencia Angular. Por lo tanto, incluso con una falsificación, si desea afirmar, por ejemplo, que una acción se agregó correctamente al servicio, deberá hacerlo contra el servicio devuelto por el inyector, y no contra la instancia original que utilizamos.

You can either let Angular's dependency injection provide it for you using the `inject` method from the Angular testing utilities or use the `injector` instance on the element created.

Puede dejar que la inyección de dependencia de Angular se lo proporcione utilizando el método `inject` de las utilidades de prueba de Angular o utilizar la instancia `injector` en el elemento creado.

Angular's dependency injector creates a clone of the stub/fake you provide to it and injects that instead of the original instance.

El inyector de dependencia de Angular crea un clon del código auxiliar/falso que usted le proporciona y lo inyecta en lugar de la instancia original.

You can check this out by trying to write the test against the service instance in the test versus the way we have written it here, and see the behavior of the test. You will see that the test fails, because the original instance does not change.

Puede comprobar esto intentando escribir la prueba en la instancia de servicio en la prueba en comparación con la forma en que la hemos escrito aquí, y ver el comportamiento de la prueba. Verás que la prueba falla porque la instancia original no cambia.

All three of these tests are available in `chapter10/simple-service/src/app/stock/stock-list/stock-list.component.spec.ts`, one after the other as three `describe` blocks.

Estas tres pruebas están disponibles en el capítulo 10/simple-service/src/app/stock/stock-list/stock-list.component.spec.ts, una tras otra como tres bloques describe.

Unit Testing Async

Prueba unitaria asíncrona

So far, we have seen how we might test simple and straightforward services, with or without further dependencies, as well as test components. But the services we have worked with so far have been synchronous. In this section, we will dig a little bit into how to handle and write tests when the underlying service or code deals with asynchronous flows.

Hasta ahora, hemos visto cómo podemos probar servicios simples y directos, con o sin dependencias adicionales, así como componentes de prueba. Pero los servicios con los que hemos trabajado hasta ahora han sido sincrónicos. En esta sección, profundizaremos un poco en cómo manejar y escribir pruebas cuando el servicio o código subyacente trata con flujos asíncronos.

With any code that touches an asynchronous flow, we have to be careful in our tests and recognize at which point the hand-off happens from synchronous flow to asynchronous, and deal with it accordingly. Thankfully, Angular provides enough helpers to abstract out some of the complexities of this in our test.

Con cualquier código que toque un flujo asíncrono, debemos tener cuidado en nuestras pruebas y reconocer en qué punto se produce el traspaso del flujo sincrónico al asíncrono, y tratarlo en consecuencia. Afortunadamente, Angular proporciona suficientes ayudas para abstraer algunas de las complejidades de esto en nuestra prueba.

Fundamentally, we have to make sure our test itself is identified and running as an asynchronous test. Secondly, we must ensure that we identify when we have to wait for the asynchronous part to finish, and validate the changes after that.

Fundamentalmente, debemos asegurarnos de que nuestra prueba en sí esté identificada y ejecutándose como una prueba asíncrona. En segundo lugar, debemos asegurarnos de identificar cuándo tenemos que esperar a que finalice la parte asíncrona y validar los cambios después de eso.

Let's see how a test for the `CreateStockComponent` might look, when we had just switched to using observables (but not HTTP yet). We will use the codebase from *chapter8/observables* as the base on which to write our tests.

Veamos cómo se vería una prueba para `CreateStockComponent`, cuando acabábamos de cambiar al uso de observables (pero aún no HTTP). Usaremos el código base del capítulo 8/observables como base sobre la cual escribir nuestras pruebas.

We will add (or modify if the skeleton already exists) the `src/app/stock/create-stock/create-stock.component.spec.ts` file, and change it as follows:

Agregaremos (o modificaremos si el esqueleto ya existe) el archivo `src/app/stock/create-stock/create-stock.component.spec.ts` y lo cambiaremos de la siguiente manera:

```
import { async, ComponentFixture, TestBed } from '@angular/core/testing';

import { CreateStockComponent } from './create-stock.component';
import { StockService } from 'app/services/stock.service';
import { Stock } from 'app/model/stock';
import { FormsModule } from '@angular/forms';
import { By } from '@angular/platform-browser';

describe('CreateStockComponent', () => {
  let component: CreateStockComponent;
```

```

let fixture: ComponentFixture<CreateStockComponent>;

beforeEach(async(() => {
  TestBed.configureTestingModule({
    declarations: [ CreateStockComponent ],
    providers: [ StockService ],
    imports: [ FormsModule ]           ①
  })
  .compileComponents();
}));

beforeEach(() => {
  fixture = TestBed.createComponent(CreateStockComponent);
  component = fixture.componentInstance;
  fixture.detectChanges();
});

it('should create stock through service', async(() => { ②
  expect(component).toBeTruthy();
  component.stock = new Stock(
    'My New Test Stock', 'MNTS', 100, 120, 'NYSE');

  component.createStock({valid: true});

  fixture.whenStable().then(() => {          ③
    fixture.detectChanges();                  ④
    expect(component.message)
      .toEqual('Stock with code MNTS successfully created');
    const messageEl = fixture.debugElement.query(
      By.css('.message')).nativeElement;
    expect(messageEl.textContent)
      .toBe('Stock with code MNTS successfully created');
  });
});
});

```

- ① El componente CreateStockComponent necesita el módulo FormsModule para funcionar.
- ② Pass the return value of calling `async` as the second param to the `it` function. El valor de retorno de llamar a `async` como segundo parámetro a la función `it`.
- ③ Esperar que el dispositivo fija el resultado de los cambios de ejecutar flujos asíncronicos.

④ ~~Update and run tests after the changes~~ cambios.

The early part of the unit test remains pretty much the same, from initializing the TestBed with the module to the beforeEach. We also have to import the FormsModule for the CreateStockComponent to work.

La primera parte de la prueba unitaria sigue siendo prácticamente la misma, desde la inicialización del TestBed con el módulo hasta el beforeEach. También tenemos que importar el FormsModule para que funcione el CreateStockComponent.

The first difference comes in the it declaration itself of our asynchronous test. Instead of simply passing the function containing our test code to the it block, we now pass an async function, which in turn is passed the function containing our test code. Do not forget this part when writing an asynchronous unit test.

La primera diferencia viene en la declaración it de nuestra prueba asincrónica. En lugar de simplemente pasar la función que contiene nuestro código de prueba al bloque it, ahora pasamos una función async, a la que a su vez se le pasa la función que contiene nuestro código de prueba. No olvide esta parte al escribir una prueba unitaria asincrónica.

The second major difference is after calling the function under test, createStock(), which actually triggers the asynchronous flow. Normally, we would write our assertions right after this. In the case of an asynchronous flow, we need to ask Angular's test fixture to stabilize (that is, wait for the asynchronous parts to finish). The whenStabilize returns a promise that on completion allows us to run the remaining part of the test. In this case, we tell Angular to detect any changes and update the UI, and then make our assertions.

La segunda diferencia importante se produce después de llamar a la función bajo prueba, createStock(), que en realidad desencadena el flujo asincrónico. Normalmente, escribiríamos nuestras afirmaciones

justo después de esto. En el caso de un flujo asíncrono, debemos pedirle al dispositivo de prueba de Angular que se estabilice (es decir, esperar a que finalicen las partes asíncronas). El `whenStabilize` devuelve una promesa que, una vez completada, nos permite ejecutar la parte restante de la prueba. En este caso, le decimos a Angular que detecte cualquier cambio y actualice la interfaz de usuario, y luego hagamos nuestras afirmaciones.

NOTE

NOTA

In this particular case, even if we had skipped the `whenStabilize` bit and directly written our assertions, our test might still have passed. But that is only because our actual underlying service is also synchronous, even though it does return an observable. If it was truly asynchronous, then the `whenStable` becomes critical for us. Thus, it is generally a good practice to always use it when it comes to `async` tests.

En este caso particular, incluso si nos hubiéramos saltado el bit `whenStabilize` y hubiéramos escrito directamente nuestras afirmaciones, nuestra prueba aún podría haber pasado. Pero eso se debe únicamente a que nuestro servicio subyacente real también es sincrónico, aunque devuelva un observable. Si fue realmente asíncrono, entonces `whenStable` se vuelve crítico para nosotros. Por lo tanto, generalmente es una buena práctica usarlo siempre cuando se trata de pruebas `async`.

The finished code for this example is available in the `chapter10/observables` folder in the GitHub repository.

El código terminado para este ejemplo está disponible en la carpeta `capítulo10/observables` en el repositorio de GitHub.

ASYNC VERSUS FAKEASYNC

ASÍNCRONO VERSUS FAKEASYNC

In the preceding test, we passed an `async` function to the `it` block, which is our test. This allowed us to deal with any asynchronous behavior in the test and still be able to assert everything we needed to.

En la prueba anterior, pasamos una función `async` al bloque `it`, que es nuestra prueba. Esto nos permitió lidiar con cualquier comportamiento asincrónico en la prueba y aún poder afirmar todo lo que necesitábamos.

Angular also provides a `fakeAsync` function, which can be used instead of the `async` function. Both are very similar and used for the same purpose, which is to abstract away some of Angular's internals and handling of asynchronous behavior in our code. The `async` function still exposes some of the underlying asynchronous behavior, since we have to deal with promises in our test with the `whenStable()` function.

Angular también proporciona una función `fakeAsync`, que se puede usar en lugar de la función `async`. Ambos son muy similares y se usan para el mismo propósito, que es abstraer algunos de los aspectos internos de Angular y el manejo del comportamiento asincrónico en nuestro código. La función `async` aún expone parte del comportamiento asincrónico subyacente, ya que tenemos que lidiar con promesas en nuestra prueba con la función `whenStable()`.

The `fakeAsync` does away with all of that. It allows us to write our unit test (for `async` and `sync` code) in a completely synchronous manner (almost that is, unless we are actually making XHR requests in our tests). Here is how the same test might look if rewritten using `fakeAsync`:

El fakeAsync acaba con todo eso. Nos permite escribir nuestra prueba unitaria (para código asíncrono y de sincronización) de manera completamente sincrónica (casi, a menos que realmente estemos realizando solicitudes XHR en nuestras pruebas). Así es como se vería la misma prueba si se reescribiera usando fakeAsync:

```
import { async, fakeAsync, tick,
         ComponentFixture, TestBed } from '@angular/core/testing';

import { CreateStockComponent } from './create-stock.component';
import { StockService } from 'app/services/stock.service';
import { Stock } from 'app/model/stock';
import { FormsModule } from '@angular/forms';
import { By } from '@angular/platform-browser';

describe('CreateStockComponent', () => {
  /* Skipped for brevity, same as before */

  it('should create stock through service', fakeAsync(() => { ❶
    expect(component).toBeTruthy();
    component.stock = new Stock(
      'My New Test Stock', 'MNTS', 100, 120, 'NYSE');

    component.createStock({valid: true});

    tick(); ❷
    fixture.detectChanges();
    expect(component.message)
      .toEqual('Stock with code MNTS successfully created');
    const messageEl = fixture.debugElement.query(
      By.css('.message')).nativeElement;
    expect(messageEl.textContent)
      .toBe('Stock with code MNTS successfully created');
  }));
});
```

- ❶ Using the fakeAsync helper instead of async
- ❷ Using tick() to simulate and finalize the asynchronous behavior

We pass a `fakeAsync` function, and instead of the `whenStable` call, we now have a simple `tick()` function call that does similar work. It allows the code to look linear and makes it more readable.

Pasamos una función `fakeAsync` y, en lugar de la llamada `whenStable`, ahora tenemos una llamada de función `tick()` simple que realiza un trabajo similar. Permite que el código parezca lineal y lo hace más legible.

There are in fact two methods to simulate a passage of time in a `fakeAsync` test, namely `tick()` and `flush()`. `tick` simulates the passage of time (and can take the number of milliseconds as an argument). `flush`, on the other hand, takes the number of turns as an argument, which is basically how many times the queue of tasks is to be drained.

De hecho, existen dos métodos para simular el paso del tiempo en una prueba `fakeAsync`, a saber, `tick()` y `flush()`. `tick` simula el paso del tiempo (y puede tomar el número de milisegundos como argumento). `flush`, por otro lado, toma como argumento el número de turnos, que es básicamente cuántas veces se vaciará la cola de tareas.

So why use one over the other? If you would like to keep your code linear and abstract away the `async` behavior, then the `fakeAsync` is great. But if you would like to ensure that you are thinking about how your code flows, and want to keep it similar to your actual code, then you can use the `async` method. It is completely a matter of preference. Feel free to pick whichever works better for your style of coding!

Entonces, ¿por qué utilizar uno sobre el otro? Si desea mantener su código lineal y abstraer el comportamiento asincrónico, entonces `fakeAsync` es excelente. Pero si desea asegurarse de que está pensando en cómo fluye su código y desea mantenerlo similar a su código real, puede usar el método `async`. Es

completamente una cuestión de preferencia. ¡Siéntete libre de elegir el que mejor se adapte a tu estilo de codificación!

Unit Testing HTTP

Prueba unitaria HTTP

The last thing we will take a look at in this chapter is to see how to test HTTP communication. In particular, we will dig into how to mock out server calls. We will see how to leverage Angular's built-in testing utilities that it provides to test HTTP communication.

Lo último que veremos en este capítulo es cómo probar la comunicación HTTP. En particular, profundizaremos en cómo simular llamadas al servidor. Veremos cómo aprovechar las utilidades de prueba integradas de Angular que proporciona para probar la comunicación HTTP.

For this section, we will use the code from *chapter9/simple-http* as the base, and see how we can test both GET calls like fetching a list of stocks as well as POST calls that we make to create stocks.

Para esta sección, usaremos el código del capítulo 9/simple-http como base y veremos cómo podemos probar tanto las llamadas GET, como la obtención de una lista de acciones, como las llamadas POST que realizamos para crear acciones.

First, let's test the StockListComponent to see how we can test the initialization logic of fetching the list of stocks from our server. We want to ensure the entire flow of making a server call, getting the list of stocks, and then displaying it.

Primero, probemos StockListComponent para ver cómo podemos probar la lógica de inicialización para obtener la lista de acciones de

nuestro servidor. Queremos garantizar todo el flujo de realizar una llamada al servidor, obtener la lista de acciones y luego mostrarla.

We will modify `src/app/stock/stock-list/stock-list.component.spec.ts` as follows:

Modificaremos `src/app/stock/stock-list/stock-list.component.spec.ts` de la siguiente manera:

```
/** Standard imports, skipping for brevity **/


import { HttpClientModule } from '@angular/common/http';
import { HttpClientTestingModule, HttpTestingController } from '@angular/common/http/testing';
import { By } from '@angular/platform-browser';

describe('StockListComponent With Real Service', () => {
  let component: StockListComponent;
  let fixture: ComponentFixture<StockListComponent>;
  let httpBackend: HttpTestingController; ①

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [ StockListComponent, StockItemComponent ],
      providers: [ StockService ],
      imports: [
        HttpClientModule,
        HttpClientTestingModule ②
      ]
    })
    .compileComponents();
  ));

  beforeEach(inject([HttpTestingController], ③
    (backend: HttpTestingController) => {
    httpBackend = backend;
    fixture = TestBed.createComponent(StockListComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
    httpBackend.expectOne({ url: '/api/stock', method: 'GET' }) ④
      .flush([{
        name: 'Test Stock 1',
        code: 'TS1',
      }]);
  ));
```

```

        price: 80,
        previousPrice: 90,
        exchange: 'NYSE'
    }, {
        name: 'Test Stock 2',
        code: 'TS2',
        price: 800,
        previousPrice: 900,
        exchange: 'NYSE'
    }]);
}))));

it('should load stocks from real service on init',
  async(() => {
    expect(component).toBeTruthy();
    expect(component.stocks$).toBeTruthy();

    fixture.whenStable().then(() => {          ⑤
      fixture.detectChanges();
      const stockItems = fixture.debugElement.queryAll(
        By.css('app-stock-item'));
      expect(stockItems.length).toEqual(2);
    });
  }));

afterEach(() => {                         ⑥
  httpBackend.verify();
});
});

```

- ① Include the `HttpTestingController` as variable variable
- ② Import the `HttpClientModule` and the `HttpTestingController` in ~~the module~~ HttpclientModule y el HttpTestingController en el módulo
- ③ Set expectation that one call to `/api/stock` will be made as part of ~~Establecer~~ la expectativa de que se realizará una llamada a `/api/stock` como parte de la prueba.
- ④ Define a list of hardcoded stocks to return when the GET call is ~~Defined~~ Dada una lista de acciones codificadas que se devolverán cuando se realice la llamada GET

- ⑤ Esperar que Algo de la tarea sea de Angular y proceder continúe
- ⑥ Verify that the GET call to `/api/stock` was actually made as part of ~~the test~~ Verifique que la llamada GET a `/api/stock` realmente se realizó como parte de la prueba

While the test seems very long, the changes to support testing HTTP calls are actually pretty straightforward. The major change when it comes to writing tests for code that makes XHR calls is the use of the `HttpTestingController`. In our unit test, while we want to test the code flow that makes XHR calls, we don't really want to make the actual underlying call. Any network call in a test adds unreliable dependencies, and makes our tests brittle and possibly nondeterministic.

Si bien la prueba parece muy larga, los cambios para admitir las pruebas de llamadas HTTP son en realidad bastante sencillos. El cambio más importante cuando se trata de escribir pruebas para código que realiza llamadas XHR es el uso de `HttpTestingController`. En nuestra prueba unitaria, si bien queremos probar el flujo de código que realiza llamadas XHR, en realidad no queremos realizar la llamada subyacente real. Cualquier llamada de red en una prueba agrega dependencias poco confiables y hace que nuestras pruebas sean frágiles y posiblemente no deterministas.

For this reason, we actually mock out the XHR calls in our tests, just check that the right XHR calls are made, and that if the response was a certain value, then it is handled correctly. Therefore, a lot of our tests would simply be setting up through the `HttpTestingController` what calls to expect and what responses to send when those calls are made. You even have control over whether the server responds with a 200 success response, or whether it is an error (a 400 or 500 response, which corresponds to a client- or server-side error, respectively).

Por esta razón, en realidad simulamos las llamadas XHR en nuestras pruebas, solo verificamos que se realicen las llamadas XHR correctas y que, si la respuesta fue un valor determinado, entonces se maneja correctamente. Por lo tanto, muchas de nuestras pruebas simplemente consistirían en configurar a través de `HttpTestingController` qué llamadas esperar y qué respuestas enviar cuando se realicen esas llamadas. Incluso tienes control sobre si el servidor responde con una respuesta de éxito 200 o si es un error (una respuesta 400 o 500, que corresponde a una respuesta del lado del cliente o del servidor). `error`, respectivamente).

With that context, let's walk through the points of interest in the preceding test:

Con ese contexto, repasemos los puntos de interés de la prueba anterior:

1. The first and most important change is that we need to import the `HttpClientModule` so that our service can initialize with the injected `HttpClient`.

El primer y más importante cambio es que necesitamos importar el `HttpClientModule` para que nuestro servicio pueda inicializarse con el `HttpClient` injectado.

2. The second thing is that we include the `HttpClientTestingModule` from `@angular/common/http/testing`. This testing module helps automatically mock out the actual server calls, and replaces it with a `HttpTestingController` that can intercept and mock all server calls.

Lo segundo es que incluimos el `HttpClientTestingModule` de `@angular/common/http/testing`. Este módulo de prueba ayuda a simular automáticamente las llamadas al servidor reales y lo reemplaza con un `HttpTestingController` que puede interceptar y simular todas las llamadas al servidor.

3. We hook these two modules up in the `imports` section of the `TestBed` module configuration.

Conectamos estos dos módulos en la sección `imports` de la configuración del módulo `TestBed`.

4. Then in our test (or in this case, the `beforeEach`), we can inject an instance of the `HttpTestingController` to set our expectations and verify server calls.

Luego, en nuestra prueba (o en este caso, `beforeEach`), podemos inyectar una instancia de `HttpTestingController` para establecer nuestras expectativas y verificar las llamadas al servidor.

5. After initializing the component instance as usual, we then start setting our expectations on the `HttpTestingController`. In this case, we expect that one GET call to the URL `/api/stock`. We also pass it a human-readable text to print in the logs in case the test fails or that call is not made.

Después de inicializar la instancia del componente como de costumbre, comenzamos a establecer nuestras expectativas en `HttpTestingController`. En este caso, esperamos que se realice una llamada GET a la URL `/api/stock`. También le pasamos un texto legible por humanos para imprimir en los registros en caso de que la prueba falle o no se realice esa llamada.

6. In addition to setting expectations on calls made, we can also define the response Angular should send when those calls are made. Here, we return an array of two stocks by calling the `flush` method. The first argument to `flush` is the response body.

Además de establecer expectativas sobre las llamadas realizadas, también podemos definir la respuesta que Angular debe enviar cuando se realizan esas llamadas. Aquí, devolvemos

una matriz de dos acciones llamando al método flush. El primer argumento de flush es el cuerpo de la respuesta.

7. The rest of the test is straightforward. We expect the component to be initialized. Then, we wait for the change detection to stabilize (by calling fixture.whenStable()), and then ensure that the response we returned is actually rendered in the template correctly.

El resto de la prueba es sencillo. Esperamos que el componente se inicialice. Luego, esperamos a que se estabilice la detección de cambios (llamando a fixture.whenStable()) y luego nos aseguramos de que la respuesta que devolvimos se represente correctamente en la plantilla.

8. In the afterEach, we call httpBackend.verify(). This ensures that all the expectations we set on the HttpTestingController were actually satisfied during the run of the test. It is generally good practice to do this in the afterEach, to ensure that our code doesn't make extra or fewer calls.

En afterEach, llamamos a httpBackend.verify(). Esto garantiza que todas las expectativas que establecimos en el HttpTestingController se cumplieron durante la ejecución de la prueba. En general, es una buena práctica hacer esto en afterEach, para garantizar que nuestro código no realice llamadas adicionales o menores.

Let's quickly write one more test to see how to handle POST calls, and how to send non-200 responses from the server. We will create the *src/app/stock/create-stock/create-stock.component.spec.ts* file as follows:

Escribamos rápidamente una prueba más para ver cómo manejar las llamadas POST y cómo enviar respuestas que no sean 200 desde el servidor. Crearemos el archivo *src/app/stock/create-stock/create-stock.component.spec.ts* de la siguiente manera:

```

/** Standard imports, skipping for brevity **/


describe('CreateStockComponent With Real Service', () => {
  let component: CreateStockComponent;
  let fixture: ComponentFixture<CreateStockComponent>;
  let httpBackend: HttpTestingController;

  beforeEach(async(() => {
    /* TestBed configuration similar to before, skipping for brevity */
  }));

  beforeEach(inject([HttpTestingController],
    (backend: HttpTestingController) => {
      httpBackend = backend;
      fixture = TestBed.createComponent(CreateStockComponent);
      component = fixture.componentInstance;
      fixture.detectChanges();
    }));
}

it('should make call to create stock and handle failure',
  async(() => {
    expect(component).toBeTruthy();
    fixture.detectChanges();

    component.stock = {
      name: 'Test Stock',
      price: 200,
      previousPrice: 500,
      code: 'TSS',
      exchange: 'NYSE',
      favorite: false
    };

    component.createStock({valid: true});

    let httpReq = httpBackend.expectOne({          ①
      url: '/api/stock',
      method: 'POST'
    }, 'Create Stock with Failure');
    expect(httpReq.request.body).toEqual(component.stock); ②
    httpReq.flush({msg: 'Stock already exists.'},        ③
      {status: 400, statusText: 'Failed!!'});
  })
);

fixture.whenStable().then(() => {
  fixture.detectChanges();
  const messageEl = fixture.debugElement.query(
    By.css('.message')).nativeElement;
  expect(messageEl.textContent).toEqual('Stock already exists.'); ④
})

```

```

    });
  });

  afterEach(() => {
    httpBackend.verify();      ⑤
  });
});

```

- ① Espera que POST redirija la solicitud POST a /api/stock/stock durante la prueba
- ② Ensure that the body of the POST request is the same as the stock we send to the component. La solicitud POST sea el mismo que el stock que creamos en el componente
- ③ Define the response for the POST request, which is a failure 400. Responde respuesta para la solicitud POST, que es una respuesta fallida 400
- ④ Check that the server response is shown correctly by the component. Comprueba que la respuesta del servidor se muestra correctamente por el componente
- ⑤ Asegúrate de que la POST solicitud POST se delay durante la prueba.

Most of the preceding test should be very similar to the test we wrote for the StockListComponent. Let's talk about the major differences in detail:

La mayor parte de la prueba anterior debería ser muy similar a la prueba que escribimos para StockListComponent. Hablemos de las principales diferencias en detalle:

- Instead of immediately flushing the response when we set the expectation on the httpBackend for the POST call, we save it to a local variable.

En lugar de descartar inmediatamente la respuesta cuando establecemos la expectativa en `httpBackend` para la llamada `POST`, la guardamos en una variable local.

- We can then write expectations on the various parts of the HTTP request, like the method, URL, body, headers, and so on.

Luego podemos escribir expectativas sobre las distintas partes de la solicitud HTTP, como el método, la URL, el cuerpo, los encabezados, etc.

- We then ensure that the body of the request matches the stock in our component.

Luego nos aseguramos de que el cuerpo de la solicitud coincida con el stock de nuestro componente.

- Finally, we flush a response, but instead of just flushing the body, we pass a second `options` argument to configure the response further. We mark the response as a `400` response to trigger the error condition.

Finalmente, desechamos una respuesta, pero en lugar de simplemente desechar el cuerpo, pasamos un segundo argumento `options` para configurar aún más la respuesta. Marcamos la respuesta como respuesta `400` para activar la condición de error.

- The rest of the test doesn't change, from waiting for the fixture to stabilize, asserting on the elements, to verifying that the calls were made on the `httpBackend`.

El resto de la prueba no cambia, desde esperar a que se estabilice el dispositivo, hacer valer los elementos, hasta verificar que las llamadas se realizaron en el `httpBackend`.

WARNING ADVERTENCIA

`httpBackend.expectOne` also takes an `HttpRequest` object instead of passing the URL and the method as a config object. In such a case, we can actually configure the `HttpRequest` object with the body of the POST request as well. Note that this does not enforce that the POST request is made with that body. You will still need to manually check it the way we did in the previous example. Do not assume that the body is automatically matched and forget to verify it.

`httpBackend.expectOne` también toma un objeto `HttpRequest` en lugar de pasar la URL y el método como un objeto de configuración. En tal caso, también podemos configurar el objeto `HttpRequest` con el cuerpo de la solicitud POST. Tenga en cuenta que esto no exige que la solicitud POST se realice con ese cuerpo. Aún necesitarás verificarlo manualmente como lo hicimos en el ejemplo anterior. No asuma que el cuerpo coincide automáticamente y olvide verificarlo.

The finished code is available in the `chapter10/simple-http` folder of the GitHub repository.

El código terminado está disponible en la carpeta `capítulo10/simple-http` del repositorio de GitHub.

Conclusion

Conclusión

In this chapter, we dug deeper into testing services in Angular. We looked at how we could test services, as well as components that use services. We explored the various techniques and options to deal with services, from using the real service, to mocking it or stubbing it out. Then we moved onto seeing how we could handle async behavior when it came to unit tests, and then finally how to deal with XHR and HTTP in our tests.

En este capítulo, profundizamos en las pruebas de servicios en Angular. Analizamos cómo podríamos probar servicios, así como componentes que utilizan servicios. Exploramos las diversas técnicas y opciones para abordar los servicios, desde utilizar el servicio real hasta burlarse de él o eliminarlo. Luego pasamos a ver cómo podíamos manejar el comportamiento asíncrono cuando se trataba de pruebas unitarias y, finalmente, cómo lidiar con XHR y HTTP en nuestras pruebas.

In the next chapter, we will switch back to seeing how we can enhance our Angular applications with the ability to deep-link to certain pages and components. We will see how to start setting up our routes, as well as protect certain routes to be accessible under only certain conditions.

En el próximo capítulo, volveremos a ver cómo podemos mejorar nuestras aplicaciones Angular con la capacidad de establecer enlaces profundos a ciertas páginas y componentes. Veremos cómo empezar a configurar nuestras rutas, así como proteger ciertas rutas para que sean accesibles solo bajo ciertas condiciones.

Exercise

Ejercicio

Take the finished exercise from [Chapter 9](#) (available in `chapter9/exercise/ecommerce`). Given this, now try to accomplish the following:

Realice el ejercicio terminado del Capítulo 9 (disponible en el capítulo 9/ejercicio/comercio electrónico). Teniendo esto en cuenta, ahora intenta lograr lo siguiente:

1. Update the `ProductListComponent` tests. Remove the isolated unit tests. Update the Angular tests to use the

`HttpTestingController` to provide the list of stocks as well as handle quantity change.

Actualice las pruebas `ProductListComponent`. Retire las pruebas unitarias aisladas. Actualice las pruebas de Angular para usar `HttpTestingController` para proporcionar la lista de existencias y manejar el cambio de cantidad.

2. Ensure that the list of stocks is reloaded when the quantity changes.

Asegúrese de que la lista de existencias se recargue cuando cambie la cantidad.

3. Add tests for the positive and negative cases of the `CreateProductComponent`. Check that the event is emitted when a product is successfully created as well.

Agregar pruebas para los casos positivos y negativos del `CreateProductComponent`. Compruebe que el evento también se emita cuando un producto se crea correctamente.

Most of this can be accomplished using concepts covered in this chapter. You can check out the finished solution in [*chapter10/exercise/ecommerce*](#).

La mayor parte de esto se puede lograr utilizando los conceptos tratados en este capítulo. Puede consultar la solución terminada en el capítulo 10/ejercicio/ecommerce.

Chapter 11. Routing in Angular

Capítulo 11. Enrutamiento en Angular

In the previous few chapters, we saw how to extend our application and make reusable services. We also saw how to integrate and deal with HTTP calls in our application using the `HttpClient` module, and deal with the asynchronous flows using observables and RxJS.

En los capítulos anteriores, vimos cómo ampliar nuestra aplicación y crear servicios reutilizables. También vimos cómo integrar y manejar llamadas HTTP en nuestra aplicación usando el módulo `HttpClient` y manejar los flujos asincrónicos usando observables y RxJS.

In this chapter, we will deal with another common requirement of web applications, which is to encapsulate various pages and pieces under different routes, and be able to deep-link to them when needed. We will implement Angular's built-in routing module. In addition, we will also dig into how to secure our application using AuthGuards and other features of the router.

En este capítulo, nos ocuparemos de otro requisito común de las aplicaciones web, que es encapsular varias páginas y piezas bajo diferentes rutas, y poder establecer vínculos profundos con ellas cuando sea necesario. Implementaremos el módulo de enrutamiento integrado de Angular. Además, también profundizaremos en cómo

proteger nuestra aplicación usando AuthGuards y otras funciones del enrutador.

Setting Up Angular Routing

Configurar el enrutamiento angular

For this chapter, we are going to build against a pre-coded server, as well as use a codebase that has most of the base components built so that we can focus only on the key aspects. We are going to continue extending our application that we have been working on across the chapters by adding routing capability to it. We are going to try to add four routes: one for the stock list, one to create a stock, one for registering, and one for logging in. Furthermore, we will protect the stock list route and the create stock route so that you can access them only if you are logged in. Finally, we will add protections to ensure that we don't lose our work by navigating away from a filled-in form.

Para este capítulo, vamos a construir sobre un servidor precodificado, así como también usaremos una base de código que tiene la mayoría de los componentes básicos creados para que podamos centrarnos solo en los aspectos clave. Continuaremos ampliando nuestra aplicación en la que hemos estado trabajando a lo largo de los capítulos agregándole capacidad de enrutamiento. Vamos a intentar agregar cuatro rutas: una para la lista de existencias, una para crear una existencia, una para registrarse y otra para iniciar sesión. Además, protegeremos la ruta de la lista de existencias y la ruta de creación de existencias para que pueda acceder a ellos solo si ha iniciado sesión. Finalmente, agregaremos protecciones para garantizar que no perdamos nuestro trabajo al salir de un formulario completo.

Server Setup

Configuración del servidor

As mentioned earlier, the server we will be working with is already developed and available in the repository in the *chapter11/server* folder. Before we start any web development, let's get our server up and running. Note that this server has more functionality than the previous ones, so please use this one instead of continuing to keep the previous server running.

Como se mencionó anteriormente, el servidor con el que trabajaremos ya está desarrollado y disponible en el repositorio en la carpeta capítulo11/servidor. Antes de comenzar cualquier desarrollo web, pongamos en funcionamiento nuestro servidor. Tenga en cuenta que este servidor tiene más funciones que los anteriores, así que utilice este en lugar de continuar ejecutando el servidor anterior.

Checkout and browse to the *chapter11/server* folder in the [GitHub repository](#). From within the folder, execute the following commands in your terminal:

Pague y busque la carpeta capítulo11/servidor en el repositorio de GitHub. Desde dentro de la carpeta, ejecute los siguientes comandos en su terminal:

```
npm i  
node index.js
```

This installs all the necessary dependencies for our Node.js server, and then starts the server on port 3000. Keep this server running in the background. This will be what our application hits to fetch and save stocks, in addition to logging in and registering users.

Esto instala todas las dependencias necesarias para nuestro servidor Node.js y luego inicia el servidor en el puerto 3000. Mantenga este

servidor ejecutándose en segundo plano. Esto será lo que haga nuestra aplicación para buscar y guardar acciones, además de iniciar sesión y registrar usuarios.

WARNING ADVERTENCIA

Note that this server is a very simplistic, dummy server with an in-memory data store. Anything you create/save will be reset if you restart the server. This includes any usernames you might register.

Tenga en cuenta que este servidor es un servidor ficticio muy simplista con un almacén de datos en memoria. Todo lo que cree/garde se restablecerá si reinicia el servidor. Esto incluye cualquier nombre de usuario que pueda registrar.

Starting Codebase

Iniciando código base

Similarly, instead of spending time building all the components for the remaining routes, and reviewing concepts we have already covered in previous chapters, we will instead use the pre-coded base Angular application that has a few more components. If you're planning to keep coding along, do note the following additions and make sure you add them to your application.

De manera similar, en lugar de dedicar tiempo a construir todos los componentes para las rutas restantes y revisar los conceptos que ya hemos cubierto en capítulos anteriores, usaremos la aplicación Angular base precodificada que tiene algunos componentes más. Si planea seguir codificando, tenga en cuenta las siguientes adiciones y asegúrese de agregarlas a su aplicación.

The code is available for you in the *chapter11/base-code-base* folder. The major additions are:

El código está disponible para usted en la carpeta capítulo11/base-code-base. Las principales incorporaciones son:

- A LoginComponent and RegisterComponent
Un LoginComponent y RegisterComponent
- A UserService that makes HTTP calls to login and register a user
Un UserService que realiza llamadas HTTP para iniciar sesión y registrar un usuario
- A UserStoreService that stores whether a user is logged in or not, along with the token
Un UserStoreService que almacena si un usuario ha iniciado sesión o no, junto con el token
- A StockAppInterceptor that will be used to send the authentication token if it exists with every request
Un StockAppInterceptor que se utilizará para enviar el token de autenticación si existe con cada solicitud.

All of these are also registered in the main AppModule.

Todos estos también están registrados en el AppModule principal.

Importing the Router Module

Importación del módulo de enrutador

With all the set up done, we can now get into how to set up routing in our application. The very first thing is to set up our *index.html* to ensure that it is able to provide enough context to Angular on how to set up its navigation. We do this by using the `base` tag within the `head` element in *index.html*. If the application is being served from

the root (like we are doing so far), then it is enough to add the following to your *index.html*:

Una vez realizada toda la configuración, ahora podemos ver cómo configurar el enrutamiento en nuestra aplicación. Lo primero es configurar nuestro index.html para garantizar que pueda proporcionar suficiente contexto a Angular sobre cómo configurar su navegación. Hacemos esto usando la etiqueta base dentro del elemento head en index.html. Si la aplicación se sirve desde la raíz (como lo estamos haciendo hasta ahora), entonces es suficiente agregar lo siguiente a su index.html:

```
<base href="/">
```

This is automatically done by the Angular CLI, so you only need to change it in case you are serving your application from a non-root location. The next thing to do is to import and set up the RouterModule, as routing is an optional module in Angular. Before we can add the RouterModule, we need to define the routes of our application. So we will first look at how to define the routes. We will then come back to seeing how to import and add the RouterModule.

Angular CLI hace esto automáticamente, por lo que solo necesita cambiarlo en caso de que esté sirviendo su aplicación desde una ubicación que no sea raíz. Lo siguiente que debe hacer es importar y configurar RouterModule, ya que el enrutamiento es un módulo opcional en Angular. Antes de que podamos agregar RouterModule, debemos definir las rutas de nuestra aplicación. Así que primero veremos cómo definir las rutas. Luego volveremos a ver cómo importar y agregar el RouterModule.

We will define a separate routes module file, *app-routes.module.ts*, instead of defining it in the same *app.module.ts*. This is generally good practice, as you want to keep it separate and modular, even if you only have a few routes initially. While we are just defining a separate module for our routes, it would eventually make sense for

us to define a separate module and routes for each feature. This would also allow us to lazy load feature modules and certain routes instead of loading all our code up front.

Definiremos un archivo de módulo de rutas separado, `app-routes.module.ts`, en lugar de definirlo en el mismo `app.module.ts`. En general, esta es una buena práctica, ya que desea mantenerlo separado y modular, incluso si inicialmente solo tiene unas pocas rutas. Si bien solo estamos definiendo un módulo separado para nuestras rutas, eventualmente tendría sentido para nosotros definir un módulo y rutas separados para cada característica. Esto también nos permitiría cargar módulos de funciones de forma diferida y ciertas rutas en lugar de cargar todo nuestro código por adelantado.

We could choose to manually create our new module and hook it up to the main `AppModule`, or let the Angular CLI do it for us, by running:

Podríamos optar por crear manualmente nuestro nuevo módulo y conectarlo al `AppModule` principal, o dejar que Angular CLI lo haga por nosotros ejecutando:

```
ng generate module app-routes --flat --module=app
```

This would generate an `app-routes.module.ts` file in the main `app` folder. We can drop the basic `CommonModule` import from it, as we won't be declaring any components as part of our routing module. Our final `app-routing.module.ts` might look something like the following:

Esto generaría un archivo `app-routes.module.ts` en la carpeta principal de la aplicación. Podemos eliminar la importación básica de `CommonModule`, ya que no declararemos ningún componente como parte de nuestro módulo de enrutamiento. Nuestro `app-routing.module.ts` final podría verse así:

```

import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';

import { CreateStockComponent }
  from './stock/create-stock/create-stock.component';
import { StockListComponent } from './stock/stock-list/stock-list.component';
import { LoginComponent } from './user/login/login.component';
import { RegisterComponent } from './user/register/register.component';

const appRoutes: Routes = [
  { path: 'login', component: LoginComponent },
  { path: 'register', component: RegisterComponent },
  { path: 'stocks/list', component: StockListComponent },
  { path: 'stocks/create', component: CreateStockComponent },
];

@NgModule({
  imports: [
    RouterModule.forRoot(appRoutes),
  ],
  exports: [
    RouterModule
  ],
})
export class AppRoutesModule { }

```

- ❶ Declarar la **comunidad de rutas** para la aplicación.
- ❷ Importar y **register the routes** para la aplicación
- ❸ Export the RouterModule so that any module importing
Export the RouterModule para que cualquier módulo que importe
AppRoutesModule tenga acceso a las directivas del enrutador

This is the first time we have created another module, separate from the auto-generated one of the Angular CLI. The AppRoutesModule (annotated with `@NgModule`) simply imports the RouterModule and then exports it so that all modules get access to router directives (which we will use in a bit). While importing the RouterModule, we mark it for the root module by calling the `forRoot` method on it, with the routes we are defining.

Esta es la primera vez que creamos otro módulo, separado del generado automáticamente por Angular CLI. El AppRoutesModule (anotado con `@NgModule`) simplemente importa el `RouterModule` y luego lo exporta para que todos los módulos tengan acceso a las directivas del enrutador (que usaremos en un momento). Mientras importamos el `RouterModule`, lo marcamos para el módulo raíz llamando al método `forRoot` en él, con las rutas que estamos definiendo.

The routes we pass to the `forRoot` method are nothing but an array of `Routes`. Each route is simply a configuration that defines the path for the route, as well as the component to be loaded when the route is loaded. We define four routes, one for each of the components.

Las rutas que pasamos al método `forRoot` no son más que una matriz de `Routes`. Cada ruta es simplemente una configuración que define el path para la ruta, así como el componente que se cargará cuando se cargue la ruta. Definimos cuatro rutas, una para cada uno de los componentes.

Next, we just need to hook up this module to our main module, by modifying the `app.module.ts` file as follows:

A continuación, sólo necesitamos conectar este módulo a nuestro módulo principal, modificando el archivo `app.module.ts` de la siguiente manera:

```
/** Other imports not changed, skipping for brevity */
import { AppRoutesModule } from './app-routes.module';

@NgModule({
  declarations: [
    /** No change, skipping for brevity */
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpClientModule,
    AppRoutesModule,
```

```
  ],
  providers: [
    /** No change, skipping for brevity */
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

① Importing the newly created AppRoutingModule module

There are a lot more components and services, but these were created before we started hooking up the routing. These were part of the base codebase we used to build on top of.

Hay muchos más componentes y servicios, pero se crearon antes de que comenzáramos a conectar el enrutamiento. Estos eran parte del código base que utilizamos para construir.

Displaying the Route Contents

Visualización del contenido de la ruta

The last thing we need to do to get our routing app up and running is to tell Angular where to load the components when a certain route or path is matched. If you consider what we have done so far, we have defined the base for our routing and set up the module and the routes.

Lo último que debemos hacer para que nuestra aplicación de enrutamiento esté en funcionamiento es decirle a Angular dónde cargar los componentes cuando una determinada ruta o ruta coincida. Si considera lo que hemos hecho hasta ahora, hemos definido la base para nuestro enrutamiento y configurado el módulo y las rutas.

The final thing we do is to mark out where Angular is to load the components, and we do that by using the `RouterOutlet` directive

that is made available as part of the `RouterModule`. We will change the `src/app.component.html` file as follows:

Lo último que hacemos es marcar dónde Angular debe cargar los componentes, y lo hacemos usando la directiva `RouterOutlet` que está disponible como parte de `RouterModule`. Cambiaremos el archivo `src/app.component.html` de la siguiente manera:

```
<div>
  <span><a href="/login">Login</a></span>
  <span><a href="/register">Register</a></span>
  <span><a href="/stocks/list">Stock List</a></span>
  <span><a href="/stocks/create">Create Stock</a></span>
</div>
<router-outlet></router-outlet>
```

Previously, we used to have the `StockListComponent` and the `CreateStockComponent` as part of this HTML file. Instead, now we are telling Angular to load the relevant component based on the URL and path it matches. We have also added a bunch of links to the various pages we added.

Anteriormente, teníamos `StockListComponent` y `CreateStockComponent` como parte de este archivo HTML. En cambio, ahora le decimos a Angular que cargue el componente relevante según la URL y la ruta que coincide. También hemos agregado un montón de enlaces a las distintas páginas que agregamos.

We are at a point when we can now run the application, and see the various routes in action. Run it with the following command (making sure you proxy to the node server you have running):

Estamos en un punto en el que ya podemos ejecutar la aplicación y ver las distintas rutas en acción. Ejecútelo con el siguiente comando (asegúrándose de utilizar el servidor de nodo que está ejecutando):

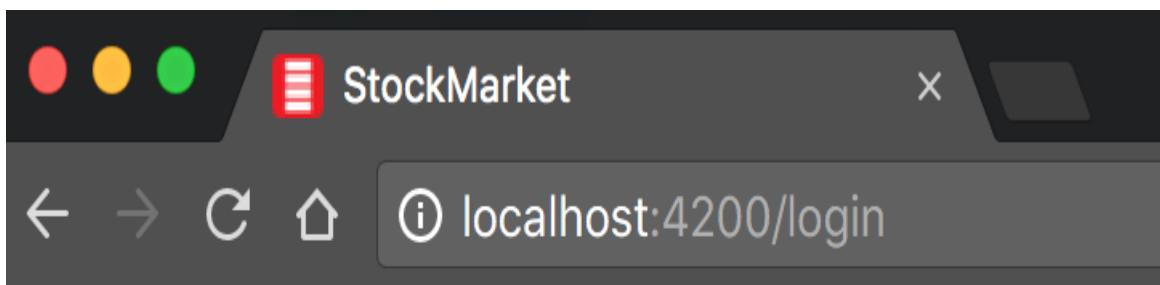
```
ng serve --proxy proxy.conf.json
```

You should see something like **Figure 11-1** when you browse to `http://localhost:4200` in your browser.

Debería ver algo como la Figura 11-1 cuando navega hasta `http://localhost:4200` en su navegador.

Clicking any of the links should open that particular component's page in your browser.

Al hacer clic en cualquiera de los enlaces, se debería abrir la página de ese componente en particular en su navegador.



[Login](#) [Register](#) [Stock List](#) [Create Stock](#)

Username

Username is Mandatory

Password

Password is Mandatory

Figure 11-1. Angular application with routing

Figura 11-1. Aplicación angular con enrutamiento.

Navigating Within the Application

Navegando dentro de la aplicación

If you do click any of the links at the top, and open the network inspector simultaneously, you will see something interesting and unexpected. You will see that the entire page reloads, rather than behaving like how we might expect a Single-Page Application routing to work. What we would want and expect is that when the route changes, only the component is loaded, and its respective XHR calls (if any) are executed. So how do we accomplish this?

Si hace clic en cualquiera de los enlaces en la parte superior y abre el inspector de red simultáneamente, verá algo interesante e inesperado. Verá que se recarga toda la página, en lugar de comportarse como podríamos esperar que funcione el enrutamiento de una aplicación de una sola página. Lo que queremos y esperamos es que cuando cambie la ruta, solo se cargue el componente y se ejecuten sus respectivas llamadas XHR (si las hay). Entonces, ¿cómo logramos esto?

Angular provides a directive that allows us to navigate within the application. The modified *app.component.html* would look as follows:

Angular proporciona una directiva que nos permite navegar dentro de la aplicación. El *app.component.html* modificado tendría el siguiente aspecto:

```
<div class="links">
  <span>
    <a routerLink="/login" routerLinkActive="active">
      Login
    </a>
  </span>
  <span>
    <a routerLink="/register" routerLinkActive="active">
      Register
    </a>
```

```

</span>
<span>
  <a routerLink="/stocks/list" routerLinkActive="active">
    Stock List
  </a>
</span>
<span>
  <a routerLink="/stocks/create" routerLinkActive="active">
    Create Stock
  </a>
</span>
</div>
<router-outlet></router-outlet>

```

We have slightly changed the content, and also added some styling to make it look nicer. From a functionality perspective, the major changes are as follows:

Cambiamos ligeramente el contenido y también agregamos algunos estilos para que se vea mejor. Desde una perspectiva de funcionalidad, los principales cambios son los siguientes:

- We have replaced the `href` links with an Angular directive `routerLink`. This ensures that all navigation happens within Angular.

Hemos reemplazado los enlaces `href` con una directiva Angular `routerLink`. Esto garantiza que toda la navegación se realice dentro de Angular.

- We have also added another Angular directive, `routerLinkActive`, which adds the argument passed to it (`active` in our case) as a CSS class when the current link in the browser matches the `routerLink` directive. It is a simple way of adding a class when the current link is selected.

También agregamos otra directiva Angular, `routerLinkActive`, que agrega el argumento que se le pasa (`active` en nuestro caso) como una clase CSS cuando el enlace actual en el navegador coincide con la directiva `routerLink`. Es una forma

sencilla de agregar una clase cuando se selecciona el enlace actual.

We also add some CSS to *app.component.css* as follows:

También agregamos algo de CSS a *app.component.css* de la siguiente manera:

```
.links, .links a {  
  padding: 10px;  
}  
  
.links a.active {  
  background-color: grey;  
}
```

We have added a `background-color` to the currently active link. This class will automatically get added to the link based on the current URL.

Hemos agregado un `background-color` al enlace actualmente activo. Esta clase se agregará automáticamente al enlace según la URL actual.

Now when you run the application, you should see the screen in **Figure 11-2** in your browser.

Ahora, cuando ejecute la aplicación, debería ver la pantalla de la Figura 11-2 en su navegador.

By default, if you open `http://localhost:4200` in your browser, you will see an empty page with only the links at the top. If you click any of the links (say, Login), then the respective components will be loaded.

De forma predeterminada, si abre `http://localhost:4200` en su navegador, verá una página vacía con solo los enlaces en la parte superior. Si hace clic en cualquiera de los enlaces (por ejemplo, Iniciar sesión), se cargarán los componentes respectivos.

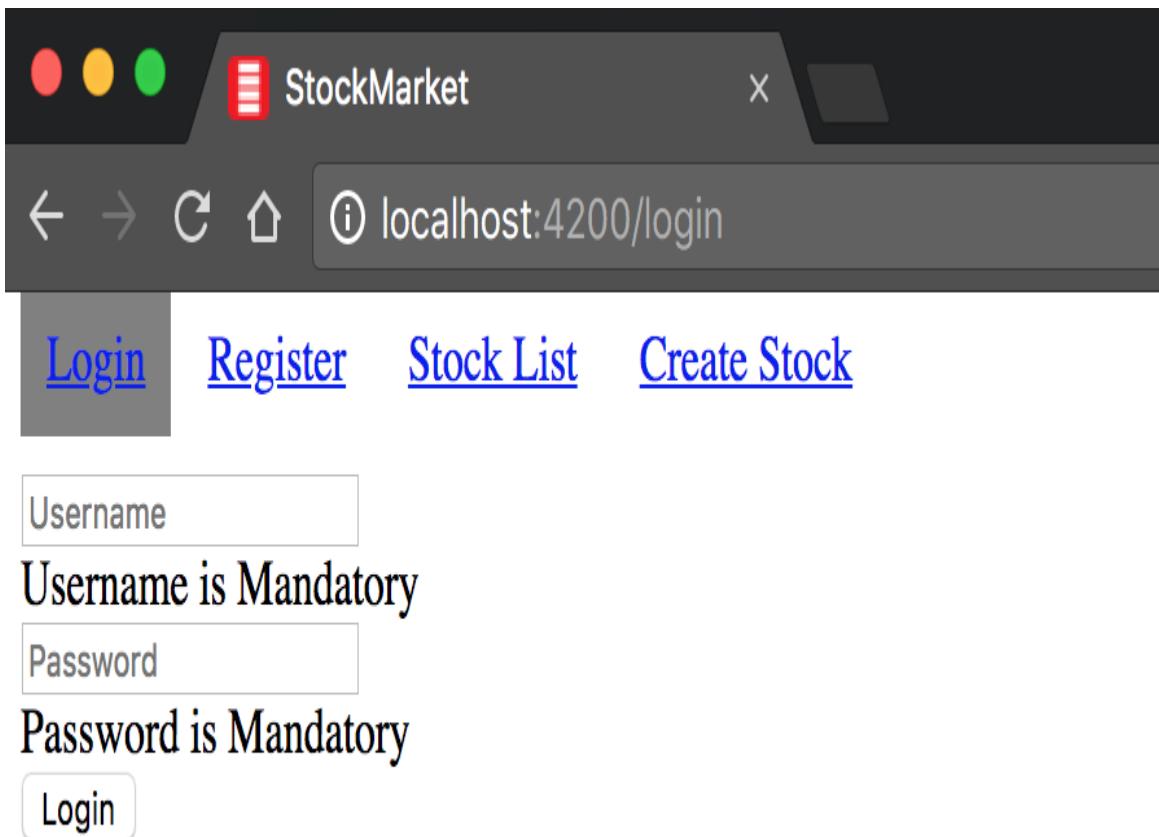


Figure 11-2. Angular application with routing and highlighting

Figura 11-2. Aplicación angular con enrutamiento y resaltado.

Wildcards and Defaults

Comodines y valores predeterminados

The one last thing we will take care of before we wrap up this example is handling the initial load. If we open up `http://localhost:4200` in our browser, we are treated with an empty page. Similarly, if we try to navigate to a URL that does not exist, it results in an error (in the developer console) and a redirect to the home page automatically. Let's see how we might tackle these in our application.

Lo último de lo que nos ocuparemos antes de concluir este ejemplo es manejar la carga inicial. Si abrimos `http://localhost:4200` en nuestro navegador, se nos trata con una página vacía. De manera similar, si intentamos navegar a una URL que no existe, se produce un error (en la consola del desarrollador) y una redirección a la página de inicio automáticamente. Veamos cómo podemos abordarlos en nuestra aplicación.

For both of these, we will go back to our `AppRoutesModule`. We will modify the `app-routes.module.ts` file as follows:

Para ambos, volveremos a nuestro `AppRoutesModule`. Modificaremos el archivo `app-routes.module.ts` de la siguiente manera:

```
/** No change in imports, skipping for brevity */

const appRoutes: Routes = [
  { path: '', redirectTo: '/login', pathMatch: 'full' }, ❶
  { path: 'login', component: LoginComponent },
  { path: 'register', component: RegisterComponent },
  { path: 'stocks/list', component: StockListComponent },
  { path: 'stocks/create', component: CreateStockComponent },
];

/** No change in remaining file, skipping for brevity */
export class AppRoutesModule { }
```

- ❶ Agregue una nueva entrada de ruta para redirigir a la página de inicio de sesión

We have added one more entry to our routes array. Here, we match the empty path and ask Angular to redirect us to the login route. Note that for any path, instead of asking Angular to use a component, we can redirect to another already defined path as well. Also note the `pathMatch` key, which is set as `full`. This ensures that only if the remaining path matches the empty string do we redirect to the login route.

Hemos agregado una entrada más a nuestra matriz de rutas. Aquí, hacemos coincidir la ruta vacía y le pedimos a Angular que nos redirija a la ruta de inicio de sesión. Tenga en cuenta que para cualquier ruta, en lugar de pedirle a Angular que use un componente, también podemos redirigir a otra ruta ya definida. Tenga en cuenta también la clave pathMatch, que está configurada como full. Esto garantiza que sólo si la ruta restante coincide con la cadena vacía redirigiremos a la ruta de inicio de sesión.

PATHMATCH RUTA COINCIDENCIA

The default pathMatch is prefix, which would check if a URL starts with the given path. If we had the first route as the default matching one, and we forgot to add pathMatch: full, then every URL would match and redirect to the login path. Thus, both the ordering of routes as well as the pathMatch value is important.

El pathMatch predeterminado es prefix, lo que verificaría si una URL comienza con el path dado. Si tuviéramos la primera ruta como la predeterminada y nos olvidáramos de agregar pathMatch: full, entonces cada URL coincidiría y redirigiría a la ruta de inicio de sesión. Por tanto, tanto el orden de las rutas como el valor pathMatch son importantes.

You can check this out by changing the pathMatch to prefix. When you do this, all the links will end up linking to the Login page.

Puedes comprobar esto cambiando pathMatch a prefix. Cuando haga esto, todos los enlaces terminarán enlazando a la página de inicio de sesión.

The final piece we will see is how to handle if the user types in a wrong URL, or we end up having bad links in our application. It is always useful to have a catch-all route that leads to a “Page not found” page or a redirect to some other page. Let’s see how we can have such a capability in our application. Again, we will only modify the *app-routes.module.ts* file:

La última parte que veremos es cómo manejar si el usuario escribe una URL incorrecta o terminamos teniendo enlaces incorrectos en nuestra aplicación. Siempre es útil tener una ruta general que conduzca a una página de "Página no encontrada" o a una redirección a otra página. Veamos cómo podemos tener esa capacidad en nuestra aplicación. Nuevamente, solo modificaremos el archivo app-routes.module.ts:

```
/** No change in imports, skipping for brevity **/ 

const appRoutes: Routes = [
  { path: '', redirectTo: '/login', pathMatch: 'full' },
  { path: 'login', component: LoginComponent },
  { path: 'register', component: RegisterComponent },
  { path: 'stocks/list', component: StockListComponent },
  { path: 'stocks/create', component: CreateStockComponent },
  { path: '**', redirectTo: '/register' }           ①
];

/** No change in remaining file, skipping for brevity */
export class AppRoutesModule { }
```

- ① ~~Addeguatcachaaalutau general route se redirige a la Registro~~ Agreguataa aluta general route se redirige a la Registro

Our catch-all route is added by matching the path **. On matching this route, we then have the option of loading a component (like the other routes). Alternatively, we can redirect again, as we have done here, to another route. We redirect to the /register route in case we can't match the URL.

Nuestra ruta general se agrega haciendo coincidir el path **. Al hacer coincidir esta ruta, tenemos la opción de cargar un componente (como las otras rutas). Alternativamente podemos redirigir nuevamente, como hemos hecho aquí, a otra ruta. Redireccionamos a la ruta /register en caso de que no podamos coincidir con la URL.

The end result of both of these changes is that if you just open `http://localhost:4200` in your browser, you will be automatically taken to the login route. In case we type in some random URL that doesn't exist, we will be redirected to the register route.

El resultado final de ambos cambios es que si simplemente abre `http://localhost:4200` en su navegador, accederá automáticamente a la ruta de inicio de sesión. En caso de que escribamos alguna URL aleatoria que no existe, seremos redirigidos a la ruta de registro.

The entire finished example is available in the `chapter11/simple-routing` folder in the GitHub repository.

El ejemplo completo está disponible en la carpeta capítulo11/simple-routing en el repositorio de GitHub.

Common Routing Requirements

Requisitos de enrutamiento comunes

In this section, we will continue digging into Angular routing capabilities, and see how we might accomplish common tasks that are needed in the course of building a web application. In particular, we will focus on having and using routes with parameters (both required and optional), as well as understand the various ways we might navigate within our application, whether it be through the template or via our component code.

En esta sección, continuaremos profundizando en las capacidades de enrutamiento de Angular y veremos cómo podemos realizar tareas comunes necesarias en el curso de la creación de una aplicación web. En particular, nos centraremos en tener y utilizar rutas con parámetros (tanto obligatorios como opcionales), así como en comprender las diversas formas en que podemos navegar dentro de

nuestra aplicación, ya sea a través de la plantilla o mediante el código de nuestro componente.

Required Route Params

Parámetros de ruta requeridos

Let's first see how we add a route where we might depend on the route to decide what to load. The simplest thing we can do within the context of our example is to build a details page for our stock.

Primero veamos cómo agregamos una ruta donde podríamos depender de la ruta para decidir qué cargar. Lo más sencillo que podemos hacer en el contexto de nuestro ejemplo es crear una página de detalles para nuestro stock.

For this section, we will use the previous section code as the base and build on top of it. In case you are not coding along, you can grab the codebase from the *chapter11/simple-routing* folder in the GitHub repository. Also, make sure you keep the Node server running in the background, and you proxy your requests to it when you serve your Angular application.

Para esta sección, usaremos el código de la sección anterior como base y construiremos sobre él. En caso de que no esté codificando, puede obtener el código base de la carpeta *capítulo11/simple-routing* en el repositorio de GitHub. Además, asegúrese de mantener el servidor Node ejecutándose en segundo plano y de enviarle sus solicitudes cuando entregue su aplicación Angular.

For the purpose of keeping our example focused on the routing, a completed (simplistic) *StockDetailsComponent* is already created in the *src/app/stocks* folder. It is already registered in the *AppModule*. All it shows is a repetition of the individual *StockItemComponent*, but without the favoriting logic. Before we look at it, let's first see how

our routes definition would change to include this new route. We would modify the `app-routes.module.ts` file as follows:

Con el fin de mantener nuestro ejemplo centrado en la ruta, ya se ha creado un `StockDetailsComponent` completo (simplista) en la carpeta `src/app/stocks`. Ya está registrado en el `AppModule`. Lo único que muestra es una repetición del individuo `StockItemComponent`, pero sin la lógica de favorecimiento. Antes de verlo, veamos primero cómo cambiaría nuestra definición de rutas para incluir esta nueva ruta. Modificariamos el archivo `app-routes.module.ts` de la siguiente manera:

```
/** No change in imports, skipping for brevity **/const appRoutes: Routes = [  { path: '', redirectTo: '/login', pathMatch: 'full' },  { path: 'login', component: LoginComponent },  { path: 'register', component: RegisterComponent },  { path: 'stocks/list', component: StockListComponent },  { path: 'stocks/create', component: CreateStockComponent },  { path: 'stock/:code', component: StockDetailsComponent }, ①  { path: '**', redirectTo: '/register' }];/** No change in remaining file, skipping for brevity **/export class AppRoutesModule { }
```

① Añadimos una ruta al path para los datos de detalle de una acción

There is only one addition to the route, which is a path to `StockDetailsComponent`. Note the path, which is `stock/:code`. This includes a variable in the URL, which can change based on which stock needs to be loaded. Our component in this case, the `StockDetailsComponent`, can then when it is loaded read the `code` from the route, and then load the corresponding stock from our server. Let's see how the `StockDetailsComponent` looks:

Sólo hay una adición a la ruta, que es una ruta a `StockDetailsComponent`. Tenga en cuenta la ruta, que es

stock/:code. Esto incluye una variable en la URL, que puede cambiar según el material que se debe cargar. Nuestro componente en este caso, el StockDetailsComponent, cuando está cargado puede leer el code de la ruta y luego cargar el stock correspondiente desde nuestro servidor. Veamos cómo queda el StockDetailsComponent:

```
import { Component, OnInit } from '@angular/core';
import { StockService } from '../services/stock.service';
import { ActivatedRoute } from '@angular/router';
import { Stock } from '../model/stock';

@Component({
  selector: 'app-stock-details',
  templateUrl: './stock-details.component.html',
  styleUrls: ['./stock-details.component.css']
})
export class StockDetailsComponent implements OnInit {

  public stock: Stock;
  constructor(private stockService: StockService,
    private route: ActivatedRoute) { }

  ngOnInit() {
    const stockCode = this.route.snapshot.paramMap.get('code'); ②
    this.stockService.getStock(stockCode).subscribe(stock => this.stock =
      stock);
  }
}
```

① Injecting the activated route into the constructor

② Using the activated route to get the code from the URL

Most of the component is similar to the remaining components we have seen so far, except for two differences:

La mayor parte del componente es similar al resto de componentes que hemos visto hasta ahora, excepto por dos diferencias:

- We inject what we call an ActivatedRoute into the constructor. The ActivatedRoute is a context-specific service that holds the

information about the currently activated route, and knows how to parse and retrieve information from it.

Inyectamos lo que llamamos ActivatedRoute en el constructor. El ActivatedRoute es un servicio específico del contexto que contiene información sobre la ruta activada actualmente y sabe cómo analizar y recuperar información de ella.

- We then use the ActivatedRoute to read the stock code (code) from the URL. Note that we read it from a snapshot. The paramMap in the snapshot is a map of all the URL parameters. We will talk about what that implies in just a bit.

Luego usamos ActivatedRoute para leer el código bursátil (code) de la URL. Tenga en cuenta que lo leemos de un snapshot. El paramMap en snapshot es un mapa de todos los parámetros de URL. Hablaremos de lo que eso implica en un momento.

Then we use the code to make a service call, and store the return value in a variable. This is then used to display information in the UI, like we have done so far.

Luego usamos el código para realizar una llamada de servicio y almacenamos el valor de retorno en una variable. Luego se usa para mostrar información en la interfaz de usuario, como lo hemos hecho hasta ahora.

The StockService.getStock code is a trivial addition to the *src/app/services/stock.service.ts* file as follows:

El código StockService.getStock es una adición trivial al archivo *src/app/services/stock.service.ts* de la siguiente manera:

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders, HttpResponse } from '@angular/common/http';

import { Observable } from 'rxjs/Observable';
```

```

import { Stock } from 'app/model/stock';
import { HttpEvent } from '@angular/common/http/src/response';
import { UserStoreService } from './user-store.service';

@Injectable()
export class StockService {

  /** Skipped for brevity */

  getStock(code: string): Observable<Stock> {
    return this.http.get<Stock>('/api/stock/' + code);
  }

  /** Skipped for brevity */
}

```

The preceding code skips most of the implementation, other than the new addition. Make sure you add it to the existing service. `getStock` simply makes a GET request with the provided `code` to the server and returns the relevant stock.

El código anterior omite la mayor parte de la implementación, excepto la nueva incorporación. Asegúrese de agregarlo al servicio existente. `getStock` simplemente realiza una solicitud GET con el `code` proporcionado al servidor y devuelve el stock correspondiente.

At this point, if we run the application, we would have a route that corresponds the stock details. But if you navigate via URL to it (say `http://localhost:4200/stock/TSC`), you would see an empty page with just the \$ sign. Inspect the Network tab in developer tools, and you will actually see a request to get the stock details, but it is responding with a 403 at this point because the user is not logged in right now. A 403 status response is usually sent when the access is forbidden to a certain resource, usually because the user is not logged in or doesn't have access to the data. We will see how to deal with it in the section "**Route Guards**".

En este punto, si ejecutamos la aplicación, tendríamos una ruta que corresponde a los detalles del stock. Pero si navega a través de una URL (digamos `http://localhost:4200/stock/TSC`), verá una página

vacía con solo el signo \$. Inspeccione la pestaña Red en las herramientas para desarrolladores y verá una solicitud para obtener los detalles del stock, pero en este momento responde con un 403 porque el usuario no ha iniciado sesión en este momento. Generalmente se envía una respuesta de estado 403 cuando el acceso está prohibido a un determinado recurso, generalmente porque el usuario no ha iniciado sesión o no tiene acceso a los datos. Veremos cómo solucionarlo en el apartado "Guardias de Ruta".

Navigating in Your Application

Navegando en su aplicación

Now that we have the route for details of a stock, let's hook up the navigation within the application. These are the few activities we want to enable:

Ahora que tenemos la ruta para obtener detalles de una acción, conectemos la navegación dentro de la aplicación. Estas son las pocas actividades que queremos habilitar:

- Navigate to the login page on successful registration
Navegue a la página de inicio de sesión si se registra correctamente
- Navigate to the stock list page on successful login
Navegue a la página de lista de existencias al iniciar sesión correctamente
- Navigate to the stock detail page when we click any stock item in the stock list
Navegue a la página de detalles de existencias cuando hagamos clic en cualquier artículo de existencias en la lista de existencias.

Of these, two need to be handled in the TypeScript code, while the other is handled in the template HTML. Let's see how we can achieve this.

De estos, dos deben manejarse en el código TypeScript, mientras que el otro se maneja en la plantilla HTML. Veamos cómo podemos lograrlo.

First, we'll modify the RegisterComponent by changing `src/app/user/register/register.component.ts` as follows:

Primero, modificaremos RegisterComponent cambiando `src/app/user/register/register.component.ts` de la siguiente manera:

```
import { Component } from '@angular/core';
import { UserService } from '../services/user.service';
import { Router } from '@angular/router';

@Component({
  selector: 'app-register',
  templateUrl: './register.component.html',
  styleUrls: ['./register.component.css']
})
export class RegisterComponent {

  public username: string = '';
  public password: string = '';

  public message: string = '';
  constructor(private userService: UserService,
    private router: Router) {} ①

  register() {
    this.userService.register(this.username, this.password)
      .subscribe((resp) => {
        console.log('Successfully registered');
        this.message = resp.msg;
        this.router.navigate(['login']); ②
      }, (err) => {
        console.error('Error registering', err);
        this.message = err.error.msg;
      });
  }
}
```

- ① Inject the Router into the component
- ② Navigate to a path using the Router

We have made some minor changes in the RegisterComponent. Primarily, we have now injected an instance of the Angular Router into our constructor, which gives us the capabilities to navigate within our application. Then, when we make a successful register call, at that point, we use the `router.navigate` call to navigate to the login page. The `navigate` method takes an array of commands, which together will resolve to a particular route.

Hemos realizado algunos cambios menores en RegisterComponent. Principalmente, ahora hemos injectado una instancia de Angular Router en nuestro constructor, lo que nos brinda la capacidad de navegar dentro de nuestra aplicación. Luego, cuando realizamos una llamada de registro exitosa, en ese momento usamos la llamada `router.navigate` para navegar a la página de inicio de sesión. El método `navigate` toma una serie de comandos, que juntos se resolverán en una ruta particular.

There are more intricacies to the `router.navigate` method. By default, any array of commands that we pass to it result in an absolute URL that Angular navigates to. So if we use `router.navigate(['stocks', 'list'])`, it would navigate to `stocks/list` route. But we can also specify the route it is relative to (for example, the current route, which we can obtain by injecting the current `ActivatedRoute` in the constructor). So if we wanted to navigate to the parent of the current route, we could execute `router.navigate(['..'], {relativeTo: this.route})`.

Hay más complejidades en el método `router.navigate`. De forma predeterminada, cualquier conjunto de comandos que le pasemos da como resultado una URL absoluta a la que navega Angular. Entonces, si usamos `router.navigate(['stocks', 'list'])`, navegaría a la ruta `stocks/list`. Pero también podemos especificar

la ruta con la que es relativa (por ejemplo, la ruta actual, que podemos obtener inyectando el ActivatedRoute actual en el constructor). Entonces, si quisiéramos navegar al padre de la ruta actual, podríamos ejecutar `router.navigate(['..'], {relativeTo: this.route})`.

We also have the capability to preserve the URL, skip changing the location, and so on. You can read up more on the other capabilities in [the official Angular Docs](#).

También tenemos la capacidad de conservar la URL, omitir cambiar la ubicación, etc. Puede leer más sobre las otras capacidades en los Angular Docs oficiales.

We can make a similar change to the LoginComponent as follows:

Podemos realizar un cambio similar en LoginComponent de la siguiente manera:

```
import { Component } from '@angular/core';
import { UserService } from '../services/user.service';
import { Router } from '@angular/router';

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})
export class LoginComponent {

  public username: string = '';
  public password: string = '';

  public message: string = '';
  constructor(private userService: UserService,
    private router: Router) {} ①

  login() {
    this.userService.login(this.username, this.password)
      .subscribe((resp) => {
        console.log('Successfully logged in');
        this.message = resp.msg;
        this.router.navigate(['stocks', 'list']); ②
      })
  }
}
```

```

    },
    (err) => {
      console.error('Error logging in', err);
      this.message = err.error.msg;
    });
}
}
}

```

① Inject the Router into the component

② Navigate to a path using the Router

Just like the RegisterComponent, we again inject the Router, and then use it on successful login to redirect to the stocks list page. Notice here that we use an array of commands to redirect to the correct stocks list page.

Al igual que RegisterComponent, volvemos a injectar Router y luego lo usamos al iniciar sesión correctamente para redirigir a la página de lista de acciones. Observe aquí que utilizamos una serie de comandos para redirigir a la página de lista de acciones correcta.

Finally, let's see how we might ensure that clicking a stock takes us to the details page for that stock. We have already created and hooked up the route for the StockDetailsComponent, which is at `stock/:code`. Let's modify `src/app/stock/stock-item/stock-item.component.html` as follows:

Finalmente, veamos cómo podemos asegurarnos de que al hacer clic en una acción nos lleve a la página de detalles de esa acción. Ya hemos creado y conectado la ruta para el StockDetailsComponent, que está en `stock/:code`. Modifiquemos `src/app/stock/stock-item/stock-item.component.html` de la siguiente manera:

```

<div class="stock-container" routerLink="/stock/{{stock.code}}">
  <div class="name">{{stock.name + ' (' + stock.code + ')'}}</div>
  <div class="exchange">{{stock.exchange}}</div>
  <div class="price"
    [class.positive]="stock.price > stock.previousPrice"
    [class.negative]="stock.price <= stock.previousPrice">
    $ {{stock.price}}
  </div>

```

```
</div>
<button (click)="onToggleFavorite($event)"
  *ngIf="!stock.favorite">Add to Favorite</button>
<button (click)="onToggleFavorite($event)"
  *ngIf="stock.favorite">Remove from Favorite</button>
</div>
```

The only change is on the first line, where we use the `routerLink` directive on the container `div` element itself. Note that unlike the links we had in the navigation bar, here we combine the `routerLink` directive with a binding. Thus, depending on the stock, the value of `routerLink` will change to have the correct code in it.

El único cambio está en la primera línea, donde usamos la directiva `routerLink` en el elemento contenedor `div`. Tenga en cuenta que, a diferencia de los enlaces que teníamos en la barra de navegación, aquí combinamos la directiva `routerLink` con un enlace. Por lo tanto, dependiendo de la acción, el valor de `routerLink` cambiará para tener el código correcto.

Now run the application, and perform the following steps in order:

Ahora ejecute la aplicación y realice los siguientes pasos en orden:

1. Open `http://localhost:4200` in your browser. It should redirect you to the login page.
Abra `http://localhost:4200` en su navegador. Debería redirigirlo a la página de inicio de sesión.
2. You can try entering a username and password, and it should update the UI with a message that the username and password is incorrect.
Puede intentar ingresar un nombre de usuario y contraseña, y debería actualizar la interfaz de usuario con un mensaje que indica que el nombre de usuario y la contraseña son incorrectos.
3. Click the Register link at the top. It should redirect you to the Register page, and also highlight the Register link at the top

(this is using the `routerLinkActive` directive if you remember).

Haga clic en el enlace Registrarse en la parte superior. Debería redirigirlo a la página Registrarse y también resaltar el enlace Registrarse en la parte superior (esto utiliza la directiva `routerLinkActive` si lo recuerda).

4. Enter a username and password, and click Register. It should redirect you to the Login page if successful.

Ingresar un nombre de usuario y contraseña y haga clic en Registrarse. Debería redirigirlo a la página de inicio de sesión si tiene éxito.

5. Enter the same username and password. It should now redirect you to the stocks list page, with three stocks present.

Ingresar el mismo nombre de usuario y contraseña. Ahora debería redirigirlo a la página de lista de acciones, con tres acciones presentes.

6. Click any of the stocks. It should open a page with only that stock (no new details though, we were lazy!). Notice that the URL has also changed.

Haga clic en cualquiera de las acciones. Debería abrir una página solo con ese stock (aunque no hay detalles nuevos, iéramos vagos!). Observe que la URL también ha cambiado.

A few things to note and keep in mind:

Algunas cosas a tener en cuenta y a tener en cuenta:

- We have not added any local storage capabilities in our application yet. Refreshing the page means you will have to login again!

Aún no hemos agregado ninguna capacidad de almacenamiento local en nuestra aplicación. ¡Actualizar la página significa que tendrás que iniciar sesión nuevamente!

- Restarting the node server will mean that you have to register again, as the Node server also keeps all data in memory.
Reiniciar el servidor de nodo significará que tendrá que registrarse nuevamente, ya que el servidor de nodo también mantiene todos los datos en la memoria.
- If you try to open the stocks list page or the stock details page directly via URL, expect to see a page with empty data, as the authentication token gets reset (between reloads of the page) and you have to go through login again.

Si intenta abrir la página de lista de acciones o la página de detalles de acciones directamente a través de una URL, espere ver una página con datos vacíos, ya que el token de autenticación se restablece (entre recargas de la página) y debe iniciar sesión nuevamente.

Optional Route Params

Parámetros de ruta opcionales

Before we wrap up this section and example, we will look at one last thing. There are routes where we might want additional params that may or may not be optional. These could be things like the current page number, the page size, or any filtering data that might be passed around, and we want to ensure that they can be bookmarked. First we will cover how to handle those cases, and then quickly look at another way in Angular to read both defined parameters and query parameters.

Antes de concluir esta sección y el ejemplo, veremos una última cosa. Hay rutas en las que es posible que queramos parámetros adicionales que pueden ser opcionales o no. Estos podrían ser elementos como el número de página actual, el tamaño de la página o cualquier dato de filtrado que pueda transmitirse, y queremos

asegurarnos de que se puedan marcar como favoritos. Primero, cubriremos cómo manejar esos casos y luego veremos rápidamente otra forma en Angular de leer tanto los parámetros definidos como los parámetros de consulta.

Let's assume that we wanted to pass a page number to the StockListComponent so that it can display the corresponding page. Now this parameter would be optional, so we want to pass it as a query parameter.

Supongamos que queremos pasar un número de página a StockListComponent para que pueda mostrar la página correspondiente. Ahora bien, este parámetro sería opcional, por lo que queremos pasarlo como parámetro de consulta.

First, let's modify the LoginComponent to pass in a page number to the route:

Primero, modifiquemos LoginComponent para pasar un número de página a la ruta:

```
/** Imports and Decorator unchanged, skipping for brevity */
export class LoginComponent {

  /** Code unchanged, skipping for brevity */

  login() {
    this.userService.login(this.username, this.password)
      .subscribe((resp) => {
        console.log('Successfully logged in');
        this.message = resp.msg;
        this.router.navigate(['stocks', 'list'], {
          queryParams: {page: 1} ❶
        });
      }, (err) => {
        console.error('Error logging in', err);
        this.message = err.error.msg;
      });
  }
}
```

- ❶ Pasa `queryParams` como parte del segundo argumento de `navigate`

We have made a slight change to the `router.navigate` in the `subscribe`, in that we pass a `queryParams` object as part of the second argument to the call. This will translate to query parameters in the route.

Hemos realizado un ligero cambio en `router.navigate` en `subscribe`, ya que pasamos un objeto `queryParams` como parte del segundo argumento de la llamada. Esto se traducirá en parámetros de consulta en la ruta.

Now let's see how we might read the query params when necessary in our component. We would modify the `StockListComponent` as follows:

Ahora veamos cómo podemos leer los parámetros de consulta cuando sea necesario en nuestro componente. Modificaríamos el `StockListComponent` de la siguiente manera:

```
/** Imports unchanged, skipping for brevity */
import { ActivatedRoute } from '@angular/router';

@Component({
  selector: 'app-stock-list',
  templateUrl: './stock-list.component.html',
  styleUrls: ['./stock-list.component.css']
})
export class StockListComponent implements OnInit {

  public stocks$: Observable<Stock[]>;
  constructor(private stockService: StockService,
              private userStore: UserStoreService,
              private route: ActivatedRoute) { }      ❶

  ngOnInit() {
    console.log('Page No. : ',
      this.route.snapshot.queryParamMap.get('page')); ❷
    this.stocks$ = this.stockService.getStocks();
  }
}
```

- ① Inject the `ActivatedRoute` and `RouteParams` into the constructor
- ② Read the page number parameter from the route's snapshot

Very similar to how we read the defined parameters, we can also read the query parameters from the `ActivatedRoute` snapshot. In this case, when we run the application, when we successfully login, we would see that our route in the browser becomes `http://localhost:4200/stocks/list?page=1`, and we get the page number printed in the console.

De manera muy similar a cómo leemos los parámetros definidos, también podemos leer los parámetros de consulta desde `ActivatedRoute` snapshot. En este caso, cuando ejecutamos la aplicación, cuando iniciamos sesión correctamente, veremos que nuestra ruta en el navegador se convierte en `http://localhost:4200/stocks/list?page=1`, y obtenemos el número de página impreso en la consola.

Before we wrap up this topic, there is one additional thing to note or be wary of when working with routes and route parameters (whether mandatory or query parameters). So far, we used the snapshot from the `ActivatedRoute` to read our parameters in the `ngOnInit` of our components. This is OK if the component is loaded only once, and we navigate from it to another component and route. But if there is a chance that the same component might need to be loaded with different parameters, then it is recommended that we do not rely on the snapshot.

Antes de concluir este tema, hay una cosa adicional que se debe tener en cuenta o de la que se debe tener cuidado al trabajar con rutas y parámetros de ruta (ya sean parámetros obligatorios o de consulta). Hasta ahora, usamos el `snapshot` del `ActivatedRoute` para leer nuestros parámetros en el `ngOnInit` de nuestros componentes. Esto está bien si el componente se carga solo una vez y navegamos desde él a otro componente y ruta. Pero si existe la posibilidad de

que sea necesario cargar el mismo componente con diferentes parámetros, se recomienda que no confiemos en `snapshot`.

Instead, we can treat the parameters and query parameters as an observable, just like service calls and HTTP requests. This way, the subscription will get triggered each time the URL changes, allowing us to reload the data rather than relying on the `snapshot`.

En cambio, podemos tratar los parámetros y los parámetros de consulta como observables, al igual que las llamadas de servicio y las solicitudes HTTP. De esta manera, la suscripción se activará cada vez que cambie la URL, lo que nos permitirá recargar los datos en lugar de depender del `snapshot`.

Let's change our `StockListComponent` slightly to see it in action. First we will add a button to our template to simulate moving to the next page, by modifying the `src/app/stock/stock-list/stock-list.component.html` file as follows:

Cambiemos ligeramente nuestro `StockListComponent` para verlo en acción. Primero agregaremos un botón a nuestra plantilla para simular el movimiento a la página siguiente, modificando el archivo `src/app/stock/stock-list/stock-list.component.html` de la siguiente manera:

```
<app-stock-item *ngFor="let stock of stocks$ | async"
  [stock]="stock">
</app-stock-item>
<div>
  <button type="button" (click)="nextPage()">Next page</button>
</div>
```

We simply added a button, which on click triggers the `nextPage()` method that we will create. Next, let's modify the component code to use a subscription to an observable, instead of relying on the `snapshot`. Our finished `src/app/stock/stock-list/stock-list.component.ts` would look like this:

Simplemente agregamos un botón, que al hacer clic activa el método nextPage() que crearemos. A continuación, modifiquemos el código del componente para usar una suscripción a un observable, en lugar de depender del snapshot. Nuestro src/app/stock/stock-list/stock-list.component.ts terminado se vería así:

```
/** Imports unchanged, skipping for brevity */
import { ActivatedRoute, Router } from '@angular/router';

@Component({
  selector: 'app-stock-list',
  templateUrl: './stock-list.component.html',
  styleUrls: ['./stock-list.component.css']
})
export class StockListComponent implements OnInit {

  public stocks$: Observable<Stock[]>;
  private page = 1;
  constructor(private stockService: StockService,
              private userStore: UserStoreService,
              private router: Router,           ①
              private route: ActivatedRoute) { } ②

  ngOnInit() {
    console.log('Page No. : ',               ③
      this.route.snapshot.queryParamMap.get('page'));
    this.route.queryParams.subscribe((params) => { ④
      console.log('Page : ', params.page);
      this.stocks$ = this.stockService.getStocks();
    });
  }

  nextPage() {
    this.router.navigate([], {               ⑤
      queryParams: {
        page: ++this.page
      }
    })
  }
}
```

① Injects the router into the constructor

② Injects ActivatedRoute into the constructor

- ③ Read ~~página de los parámetros de snapshot~~ la snapshot
- ④ Suscribirse a `queryParams` para cuando cambie
- ⑤ Navegar a la misma página mientras se mantiene la página en otra página

There are a few changes, so let's go over them one by one:

Hay algunos cambios, así que repasémoslos uno por uno:

- We added a local variable `page`, and initialized it to 1.
Agregamos una variable local `page` y la inicializamos en 1.
- We added a `nextPage()` method, which on click navigates (using `router.navigate`) to the next page. Note that we don't provide any commands, to keep it at the same page, but just change the query params.

Agregamos un método `nextPage()`, que al hacer clic navega (usando `router.navigate`) a la página siguiente. Tenga en cuenta que no proporcionamos ningún comando para mantenerlo en la misma página, solo cambiamos los parámetros de consulta.

- In the `ngOnInit`, we left the old `console.log` as it is reading from the snapshot. In addition, we subscribe to the `queryParams` observable. This subscription will trigger each time the page changes, while we remain on the same component.

En `ngOnInit`, dejamos el antiguo `console.log` tal como se lee en `snapshot`. Además, nos suscribimos al observable `queryParams`. Esta suscripción se activará cada vez que cambie `page`, mientras permanezcamos en el mismo componente.

Now, you can try the following:

Ahora puedes probar lo siguiente:

- Login (or register again, in case you restarted the server).
Inicie sesión (o regístrese nuevamente, en caso de que haya reiniciado el servidor).
- Note the developer tools console to see the initial page number in the snapshot as well as the observable subscription getting triggered.

Observe la consola de herramientas de desarrollador para ver el número de página inicial en snapshot, así como la suscripción observable que se activa.

- Click the “Next page” button a few times, to see the subscription getting triggered.

Haga clic en el botón "Página siguiente" varias veces para ver cómo se activa la suscripción.

Whether we subscribe to the `queryParams` or to `params`, the code doesn't change by much. This is a super useful approach in case you have a component where the data will get loaded for various parameters without the component getting reloaded.

Ya sea que nos suscribamos a `queryParams` o a `params`, el código no cambia mucho. Este es un enfoque muy útil en caso de que tenga un componente donde los datos se cargarán para varios parámetros sin que el componente se vuelva a cargar.

The entire completed example (including parameters, navigation, and query parameters including the subscription-based approach) can be found in the `chapter11/navigation-and-params` folder in the GitHub repository.

El ejemplo completo completo (incluidos los parámetros, la navegación y los parámetros de consulta, incluido el enfoque basado en suscripción) se puede encontrar en la carpeta `Chapter11/navigation-and-params` en el repositorio de GitHub.

Route Guards

Guardias de ruta

The next thing we will cover is the concept of route guards. Route guards in Angular are a way of protecting the loading or unloading of a route based on your own conditions. Route guards give you a lot of flexibility in the kinds of checks you want to add before a route opens or closes. In this section, we will deal with three in particular: a guard to prevent a route from opening, a guard to prevent a route from closing, and a guard that loads necessary data before a route is opened. We will keep the examples very simple, but these could be extended to do whatever is needed in your use case.

Lo siguiente que cubriremos es el concepto de guardias de ruta. Los guardias de ruta en Angular son una forma de proteger la carga o descarga de una ruta según tus propias condiciones. Los guardias de ruta le brindan mucha flexibilidad en los tipos de controles que desea agregar antes de que se abra o cierre una ruta. En esta sección, nos ocuparemos de tres en particular: una guardia para evitar que se abra una ruta, una guardia para evitar que se cierre una ruta y una guardia que carga los datos necesarios antes de que se abra una ruta. Mantendremos los ejemplos muy simples, pero podrían ampliarse para hacer lo que sea necesario en su caso de uso.

For this entire section, we will use the codebase from the previous section as a base to build on. In case you are not coding along, you can find the starter code in the *chapter11/navigation-and-params* folder in the GitHub repository.

Para toda esta sección, usaremos el código base de la sección anterior como base para construir. En caso de que no esté codificando, puede encontrar el código de inicio en la carpeta capítulo11/navegación-y-params en el repositorio de GitHub.

Authenticated-Only Routes

Rutas solo autenticadas

The first thing we will tackle is the issue we saw in the previous section, where if we tried to navigate to the Stock List component without logging in, we would end up seeing an empty page. What we would ideally want in this case is a message or error, and a redirection to the login page so that we can prompt the user to login.

Lo primero que abordaremos es el problema que vimos en la sección anterior, donde si intentábamos navegar al componente Lista de acciones sin iniciar sesión, terminaríamos viendo una página vacía. Lo que idealmente queríamos en este caso es un mensaje o error y una redirección a la página de inicio de sesión para que podamos solicitar al usuario que inicie sesión.

For this, we will rely on the `UserStoreService` to figure out if the user is currently logged in or not. Using this service, we will then create an authentication guard, which will kick in before we open a protected route. The authentication guard will then decide whether we can continue on to the route, or if we need to redirect to a different route.

Para esto, nos basaremos en `UserStoreService` para determinar si el usuario ha iniciado sesión actualmente o no. Al utilizar este servicio, crearemos una protección de autenticación, que se activará antes de que abramos una ruta protegida. El guardia de autenticación decidirá entonces si podemos continuar con la ruta o si necesitamos redirigir a una ruta diferente.

To accomplish this, the first thing we will do is create an `AuthGuard`. To get it kickstarted, you can of course use the Angular CLI (remember, `ng g guard guards/auth` will do the trick). Replace the content of the generated file (`src/app/guards/auth.guard.ts`) with the following:

Para lograr esto, lo primero que haremos será crear un AuthGuard. Para ponerlo en marcha, por supuesto, puede utilizar la CLI de Angular (recuerde, `ng g guard guards/auth` funcionará). Reemplace el contenido del archivo generado (`src/app/guards/auth.guard.ts`) con lo siguiente:

```
import { Injectable }      from '@angular/core';
import { CanActivate, Router }   from '@angular/router';
import { UserStoreService } from '../services/user-store.service';
import { Observable } from 'rxjs/Observable';

@Injectable()
export class AuthGuard implements CanActivate {

  constructor(private userStore: UserStoreService,
              private router: Router) {}

  canActivate(): boolean {
    console.log('AuthGuard#canActivate called');

    if (this.userStore.isLoggedIn()) { return true };

    console.log('AuthGuard#canActivate not authorized to access page');
    // Can store current route and redirect back to it
    // Store it in a service, add it to a query param
    this.router.navigate(['login']);

    return false;
  }
}
```

The AuthGuard class is straightforward, and looks and behaves just like an Angular service. The service looks pretty simple, but let's walk through the changes one by one:

La clase AuthGuard es sencilla y se ve y se comporta como un servicio Angular. El servicio parece bastante simple, pero veamos los cambios uno por uno:

- We implement an interface called `CanActivate` from the Angular router module.

Implementamos una interfaz llamada `CanActivate` desde el módulo del enrutador Angular.

- We inject both the `UserStoreService` and the `Router` as part of the constructor.

Inyectamos tanto el `UserStoreService` como el `Router` como parte del constructor.

- We then implement the `canActivate` method. The `canActivate` method can return either a boolean or an `Observable<boolean>`. If it resolves to true, then the route will activate. If not, the route will not open.

Luego implementamos el método `canActivate`. El método `canActivate` puede devolver un boolean o un `Observable<boolean>`. Si se resuelve en true, la ruta se activará. En caso contrario, la ruta no se abrirá.

- In the `canActivate`, we check the `UserStoreService` to see if the user is logged in. If he's not, we redirect the user to the Login page, and return false as well.

En `canActivate`, verificamos `UserStoreService` para ver si el usuario ha iniciado sesión. Si no lo está, redirigimos al usuario a la página de inicio de sesión y también devolvemos false.

In the last step is where we can add our custom logic, if needed. For example, we could preserve the URL we were trying to open. Once the user successfully logs in, we can then redirect to the saved URL rather than the default.

En el último paso es donde podemos agregar nuestra lógica personalizada, si es necesario. Por ejemplo, podríamos conservar la URL que intentábamos abrir. Una vez que el usuario inicia sesión correctamente, podemos redirigir a la URL guardada en lugar de a la predeterminada.

We also can access the newly activated route as well as the router snapshot as arguments to the `canActivate` method, in case we need to access any route or URL-specific values to make our decision.

También podemos acceder a la ruta recién activada, así como al enrutador snapshot como argumentos para el método `canActivate`, en caso de que necesitemos acceder a alguna ruta o valores específicos de URL para tomar nuestra decisión.

Another thing to note is that in our example, we are actually relying on synchronous state to decide whether to proceed or not. But as mentioned, `canActivate` can also return an observable or a promise, thus allowing you to make server calls to decide whether or not to proceed. Angular will wait for the service to return before making a decision on whether or not the route should activate.

Otra cosa a tener en cuenta es que en nuestro ejemplo, en realidad confiamos en el estado sincrónico para decidir si continuar o no. Pero como se mencionó, `canActivate` también puede devolver un observable o una promesa, lo que le permite realizar llamadas al servidor para decidir si continuar o no. Angular esperará a que regrese el servicio antes de tomar una decisión sobre si la ruta debe activarse o no.

Make sure that you hook up the service in the `AppModule` before you proceed further. This might be required even if you use the Angular CLI, as we have multiple modules in our application.

Asegúrese de conectar el servicio en `AppModule` antes de continuar. Esto podría ser necesario incluso si utiliza Angular CLI, ya que tenemos varios módulos en nuestra aplicación.

Now, let's hook up our `AuthGuard` to the routing. We will modify the `src/app-routes.module.ts` file as follows:

Ahora, conectemos nuestro `AuthGuard` a la ruta. Modificaremos el archivo `src/app-routes.module.ts` de la siguiente manera:

```

/** Imports unchanged, skipping for brevity */
import { AuthGuard } from './guards/auth.guard';

const appRoutes: Routes = [
  { path: '', redirectTo: '/login', pathMatch: 'full' },
  { path: 'login', component: LoginComponent },
  { path: 'register', component: RegisterComponent },
  { path: 'stocks/list', component: StockListComponent,
    canActivate: [AuthGuard] }, ①
  { path: 'stocks/create', component: CreateStockComponent,
    canActivate: [AuthGuard] }, ②
  { path: 'stock/:code', component: StockDetailsComponent,
    canActivate: [AuthGuard] }, ③
  { path: '**', redirectTo: '/register' }
];

/** Code unchanged, skipping for brevity */
export class AppRoutesModule { }

```

- ① Agregar el AuthGuard a la ruta de la lista de acciones**
- ② Agregar el AuthGuard a la ruta de la creación de acciones**
- ③ Agregar el AuthGuard a la ruta de los detalles de acciones**

To the three stock routes, we have added another key to the route definition. We have added a canActivate key, which takes an array of guards. We only have the AuthGuard, so we pass that as the only element of the array. Thus, only the routes we add to the canActivate guard will use the guard, and the others will continue to work as normal.

A las tres rutas estándar, hemos agregado otra clave para la definición de ruta. Hemos agregado una clave canActivate, que requiere una serie de guardias. Solo tenemos AuthGuard, por lo que lo pasamos como el único elemento de la matriz. Así, solo las rutas que agreguemos a la guardia canActivate usarán la guardia, y las demás seguirán funcionando normalmente.

Before we run this application, ensure that the AuthGuard is hooked up as a provider in the AppModule, which the ng generate guard

does not do by default. Your `src/app.module.ts` file should have the following in it:

Antes de ejecutar esta aplicación, asegúrese de que AuthGuard esté conectado como proveedor en AppModule, lo cual ng generate guard no hace de forma predeterminada. Su archivo `src/app.module.ts` debe contener lo siguiente:

```
/** Other imports unchanged, skipping for brevity */
import { AuthGuard } from './guards/auth.guard';

@NgModule({
  /** No changes to imports and declarations */
  providers: [
    /** No changes to other services */
    AuthGuard, ①
    {
      provide: HTTP_INTERCEPTORS,
      useClass: StockAppInterceptor,
      multi: true,
    }
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

① Add AuthGuard to the providers

If you run the application at this point, you should see that trying to navigate directly to the stock list or create stock page will end up redirecting you to the Login page. You can confirm that the guard is working by checking the web development console logs.

Si ejecuta la aplicación en este punto, debería ver que intentar navegar directamente a la lista de acciones o crear la página de acciones terminará redireccionándolo a la página de inicio de sesión. Puede confirmar que el guardia está funcionando consultando los registros de la consola de desarrollo web.

Preventing Unload

Prevenir la descarga

Similar to how we can prevent loading of a route, we can also prevent deactivation of a route using guards. The `canDeactivate` guard is most commonly used to prevent the user from losing data by navigating away unintentionally from a form page, or to autosave the data when the user navigates away from a page. Other creative uses for the `canDeactivate` guard could be logging and analytics.

De manera similar a como podemos evitar la carga de una ruta, también podemos evitar la desactivación de una ruta usando guardias. La protección `canDeactivate` se usa más comúnmente para evitar que el usuario pierda datos al salir involuntariamente de una página de formulario, o para guardar automáticamente los datos cuando el usuario sale de una página. Otros usos creativos del guardia `canDeactivate` podrían ser el registro y el análisis.

In this example, we will again keep it very simple to just demonstrate the point, but you can extend it for your purpose. We will simply always prompt the user when navigating away from the `CreateStockComponent`. But you could make it smarter by looking at the form state, and whether there are any changes before prompting.

En este ejemplo, nuevamente lo mantendremos muy simple para demostrar el punto, pero puedes ampliarlo según tu propósito. Simplemente siempre avisaremos al usuario cuando navegue fuera de `CreateStockComponent`. Pero podría hacerlo más inteligente observando el estado del formulario y si hay algún cambio antes de solicitarlo.

Create a `CreateStockDeactivateGuard` (again, your choice, either create it manually or using the Angular CLI). Don't forget to register

it in the providers array in the AppModule, and then replace the contents of the service with the following:

Cree un CreateStockDeactivateGuard (nuevamente, su elección, créelo manualmente o usando Angular CLI). No olvide registrarla en la matriz providers en AppModule y luego reemplazar el contenido del servicio con lo siguiente:

```
import { Injectable } from '@angular/core';
import { CanDeactivate, ActivatedRouteSnapshot, RouterStateSnapshot }
    from '@angular/router';
import { CreateStockComponent }
    from '../stock/create-stock/create-stock.component';
import { Observable } from 'rxjs/Observable';

@Injectable()
export class CreateStockDeactivateGuard
    implements CanDeactivate<CreateStockComponent> { ❶

    constructor() { }

    canDeactivate(component: CreateStockComponent, ❷
                  currentRoute: ActivatedRouteSnapshot, ❸
                  currentState: RouterStateSnapshot, ❹
                  nextState?: RouterStateSnapshot): ❺
        boolean | Observable<boolean> | Promise<boolean> {
        return window.confirm('Do you want to navigate away from this page?');
    }
}
```

- ❶ Implement the CanDeactivate interface, specific for our ~~CreateStockComponent~~ clase. La interfaz CanDeactivate, específica para nuestro CreateStockComponent
- ❷ Instance of the CreateStockComponent passed to the ~~last~~ método. El objeto CreateStockComponent pasado al método canDeactivate
- ❸ The currently ActivatedRoute snapshot passed to the canDeactivate method

El ActivatedRoute snapshot actualmente pasado al método canDeactivate

- ④ The current router state snapshot passed to the canDeactivate
El estado actual del enrutador snapshot pasado al método canDeactivate
- ⑤ The next state that is being navigated from the current state

Our CanDeactivate guard is slightly different from the CanActivate guard, and for a few reasons. The most important reason is that usually, the deactivation is in context of an existing component, and so the state of that component is usually important in deciding whether or not the router can deactivate the component and route.

Nuestra guardia CanDeactivate es ligeramente diferente de la guardia CanActivate y por varias razones. La razón más importante es que, por lo general, la desactivación se realiza en el contexto de un componente existente, por lo que el estado de ese componente suele ser importante para decidir si el enrutador puede desactivar el componente y la ruta.

Here, we implement a CanDeactivate guard that is specific to our CreateStockComponent. The advantage is that we can access state and methods from our component to make the decision (not that we are doing it in our example!). If we had the form state available in the component, this would be a great place to access it and check if the form was dirty or not. You can also refer to the current route and state, as well as what the transition is going to and factor all of these into your decision.

Aquí, implementamos una protección CanDeactivate que es específica de nuestro CreateStockComponent. La ventaja es que podemos acceder al estado y a los métodos de nuestro componente para tomar la decisión (no es que lo estemos haciendo en nuestro ejemplo!). Si tuviéramos el estado del formulario disponible en el componente, este sería un excelente lugar para acceder a él y

verificar si el formulario era `dirty` o no. También puede consultar la ruta y el estado actuales, así como el destino de la transición, y tenerlos en cuenta en su decisión.

You can return a simple boolean (like we are), or return an observable or a promise that translates to a boolean, and Angular will wait for the asynchronous behavior to finish before making a decision.

Puede devolver un boolean simple (como lo hacemos nosotros), o devolver un observable o una promesa que se traduzca en un boolean, y Angular esperará a que finalice el comportamiento asincrónico antes de tomar una decisión.

Now, let's hook up this guard to the `AppRoutesModule`:

Ahora, conectemos este protector al `AppRoutesModule`:

```
/** Imports unchanged, skipping for brevity */
import { CreateStockDeactivateGuard }
    from './guards/create-stock-deactivate.guard';

const appRoutes: Routes = [
  { path: '', redirectTo: '/login', pathMatch: 'full' },
  { path: 'login', component: LoginComponent },
  { path: 'register', component: RegisterComponent },
  { path: 'stocks/list', component: StockListComponent,
    canActivate: [AuthGuardService] },
  { path: 'stocks/create', component: CreateStockComponent,
    canActivate: [AuthGuardService],
    canDeactivate: [CreateStockDeactivateGuard] }, ①
  { path: 'stock/:code', component: StockDetailsComponent,
    canActivate: [AuthGuardService] },
  { path: '**', redirectTo: '/register' }
];

/** Code unchanged, skipping for brevity */
export class AppRoutesModule { }
```

- ① Agregue `CreateStockDeactivateGuard` a la lista de stock de stock

We have only added another guard (`canDeactivate`) on the `stocks/create` route, and hooked up the `CreateStockDeactivateGuardService` as the only item in the array.

Solo agregamos otro guardia (`canDeactivate`) en la ruta `stocks/create` y conectamos el `CreateStockDeactivateGuardService` como el único elemento en la matriz.

WARNING ADVERTENCIA

Make sure you hook up the new guard to the `AppModule` and register it with the `providers` array like the other guard. Don't forget to do this for any new service or guard if it is not automatically added by the Angular CLI.

Asegúrese de conectar el nuevo guardia al `AppModule` y registrarlo con la matriz `providers` como el otro guardia. No olvide hacer esto para cualquier servicio o guardia nuevo si Angular CLI no lo agrega automáticamente.

Once you do this, you can run your application. Log in and navigate to the create stock page, and then try clicking any of the links at the top. At that point, you should see the confirmation asking whether you really want to navigate away. Clicking "No" should leave you on the same page, while clicking "Yes" would allow you to navigate away.

Una vez que haga esto, podrá ejecutar su aplicación. Inicie sesión y navegue hasta la página de creación de acciones y luego intente hacer clic en cualquiera de los enlaces en la parte superior. En ese punto, debería ver la confirmación que le preguntará si realmente desea navegar. Hacer clic en "No" debería dejarlo en la misma página, mientras que hacer clic en "Sí" le permitirá navegar.

Again, remember that you are free to make the logic as complex as needed. You need to make a call to server to persist the data and

then only navigate away? Feel free to change the guard around. The only thing to note is that this will not detect if you change the URL manually yourself. It will only detect navigation within the application context.

Nuevamente, recuerde que es libre de hacer que la lógica sea tan compleja como sea necesario. ¿Necesita hacer una llamada al servidor para conservar los datos y luego simplemente navegar? Siéntete libre de cambiar la guardia. Lo único que hay que tener en cuenta es que esto no se detectará si usted mismo cambia la URL manualmente. Sólo detectará la navegación dentro del contexto de la aplicación.

A GENERIC CANDEACTIVATE GUARD? ¿UN PROTECTOR GENÉRICO PUEDE DESACTIVARSE?

We just saw that the `CanDeactivate` guard was specific to a component, and we get an instance of that component in the `canDeactivate` method. Then is it possible for us to create a generic guard?

Acabamos de ver que la protección `CanDeactivate` era específica de un componente y obtenemos una instancia de ese componente en el método `canDeactivate`. Entonces, ¿es posible que creemos una guardia genérica?

The answer is yes. One common technique is to create an interface (say, `DeactivateableComponent`), with a method (say, `canDeactivate`) that returns a `boolean`, a `Promise<boolean>`, or an `Observable<boolean>`.

La respuesta es sí. Una técnica común es crear una interfaz (por ejemplo, `DeactivateableComponent`), con un método (por ejemplo, `canDeactivate`) que devuelva un `boolean`, un `Promise<boolean>` o un `Observable<boolean>`.

We can then create a `CanDeactivate<DeactivateableComponent>` that relies on the return value of `canDeactivate` to decide whether to deactivate or not. Each component that needs this guard simply needs to implement this interface, and you can then reuse the guard as needed.

Luego podemos crear un `CanDeactivate<DeactivateableComponent>` que se base en el valor de retorno de `canDeactivate` para decidir si desactivarlo o no. Cada componente que necesita esta protección simplemente necesita implementar esta interfaz y luego puede reutilizar la protección según sea necesario.

Again, this is only useful for a select handful of cases, primarily if you have multiple components that need to decide whether they can deactivate or not, but all in different ways.

Nuevamente, esto solo es útil para un puñado de casos seleccionados, principalmente si tiene varios componentes que necesitan decidir si se pueden desactivar o no, pero todos de diferentes maneras.

Preloading Data Using Resolve

Precarga de datos usando Resolve

The last thing we will see in this section is how to preload data before a route is activated. There might be cases where we want to

make the service call to fetch its data before the component loads. Similarly, we might want to check if the data exists before even opening up the component. In these cases, it might make sense for us to try to prefetch the data before the component itself. In Angular, we do this using a Resolver.

Lo último que veremos en este apartado es cómo precargar datos antes de activar una ruta. Puede haber casos en los que queramos realizar la llamada de servicio para recuperar sus datos antes de que se cargue el componente. De manera similar, es posible que deseemos verificar si los datos existen incluso antes de abrir el componente. En estos casos, podría tener sentido que intentemos capturar previamente los datos antes que el propio componente. En Angular, hacemos esto usando un Resolver.

Let's take an example to demonstrate how a Resolver works and how you might implement one. Say we wanted to resolve the stock data even before we open up the details of a stock. This would also in some sense allow us to check if the stock with a particular code exists even before opening the stock details component.

Tomemos un ejemplo para demostrar cómo funciona Resolver y cómo podría implementar uno. Digamos que queremos resolver los datos de las acciones incluso antes de abrir los detalles de una acción. En cierto sentido, esto también nos permitiría verificar si la acción con un código particular existe incluso antes de abrir el componente de detalles de la acción.

To accomplish this, we would use a Resolver. Use the Angular CLI or manually create a StockLoadResolver, with the following content:

Para lograr esto, usariamos un Resolver. Utilice la CLI de Angular o cree manualmente un StockLoadResolver, con el siguiente contenido:

```
import { Injectable } from '@angular/core';
import { StockService } from './stock.service';
import { Resolve, ActivatedRouteSnapshot, RouterStateSnapshot }
```

```

from '@angular/router';
import { Stock } from '../model/stock';
import { Observable } from 'rxjs/Observable';

@Injectable()
export class StockLoadResolverService implements Resolve<Stock> {

  constructor(private stockService: StockService) { }

  resolve(route: ActivatedRouteSnapshot,
         state: RouterStateSnapshot):
    Stock | Observable<Stock> | Promise<Stock> {
    const stockCode = route.paramMap.get('code');
    return this.stockService.getStock(stockCode);
  }
}

```

A Resolver implements the Resolve interface, and is typed. In this case, we are building a Resolver that returns one individual stock. We inject the StockService into the constructor, and then implement the resolve method. Here we have access to the route and state, which allows us to fetch the parameter information from the URL.

Un Resolver implementa la interfaz Resolve y se escribe. En este caso, estamos creando un Resolver que devuelve una acción individual. Inyectamos el StockService en el constructor y luego implementamos el método resolve. Aquí tenemos acceso a la ruta y al estado, lo que nos permite obtener la información de los parámetros de la URL.

In the resolve, we load the stockCode from the URL, and then return an Observable<Stock> by making the service call to getStock for the given stockCode. That is all there is to the Resolver.

En resolve, cargamos el stockCode desde la URL y luego devolvemos un Observable<Stock> realizando la llamada de servicio a getStock para el stockCode dado. Eso es todo lo que hay en el Resolver.

Make sure you hook this up to the AppModule and register it with the providers array like the other guards.

Asegúrate de conectar esto al AppModule y registrarlo con la matriz providers como los otros guardias.

Now we can hook this up to the AppRoutingModule as follows:

Ahora podemos conectar esto al AppRoutingModule de la siguiente manera:

```
/** Imports unchanged, skipping for brevity */
import { StockLoadResolverService } from './resolver/stock-load-
resolver.service';

const appRoutes: Routes = [
  { path: '', redirectTo: '/login', pathMatch: 'full' },
  { path: 'login', component: LoginComponent },
  { path: 'register', component: RegisterComponent },
  { path: 'stocks/list', component: StockListComponent,
    canActivate: [AuthGuardService] },
  { path: 'stocks/create', component: CreateStockComponent,
    canActivate: [AuthGuardService],
    canDeactivate: [CreateStockDeactivateGuardService] },
  { path: 'stock/:code', component: StockDetailsComponent,
    canActivate: [AuthGuardService],
    resolve: { stock: StockLoadResolverService } }, ①
  { path: '**', redirectTo: '/register' }
];

/** Code unchanged, skipping for brevity */
export class AppRoutingModule { }
```

① Agregar resolver para que los detalles de stock

Again, this file has only changed in one line. To the stock/:code route, we have now added a resolve key, which is an object. For each key of the object, we map a Resolver implementation. In this case, we only have the stock that needs to be resolved using the StockLoadResolverService. This is one part of the work that needs to be done, which ensures that the stock with the given code (based on the URL) is prefetched.

Nuevamente, este archivo solo ha cambiado en una línea. A la ruta `stock/:code`, ahora le hemos agregado una clave `resolve`, que es un objeto. Para cada clave del objeto, asignamos una implementación `Resolver`. En este caso, solo tenemos el `stock` que debe resolverse usando el `StockLoadResolverService`. Esta es una parte del trabajo que debe realizarse, que garantiza que el stock con el código dado (basado en la URL) se obtenga previamente.

Next, let's see how to modify the `StockDetailsComponent` to use the prefetched information instead of making the service call itself:

A continuación, veamos cómo modificar `StockDetailsComponent` para usar la información precargada en lugar de hacer que el servicio se llame a sí mismo:

```
import { Component, OnInit } from '@angular/core';
import { StockService } from '../../services/stock.service';
import { ActivatedRoute } from '@angular/router';
import { Stock } from '../../model/stock';

@Component({
  selector: 'app-stock-details',
  templateUrl: './stock-details.component.html',
  styleUrls: ['./stock-details.component.css']
})
export class StockDetailsComponent implements OnInit {

  public stock: Stock;
  constructor(private route: ActivatedRoute) { }

  ngOnInit() {
    this.route.data.subscribe((data: {stock: Stock}) => {
      this.stock = data.stock;
    });
  }
}
```

The major change in the component is that we have now gotten rid of our dependency on the `StockService`. Instead, we just use the `ActivatedRoute`. In the `ngOnInit`, we subscribe to changes on the `data` element on the `ActivatedRoute`. The resolved data would be

made available in the data with the key that we used in the route (stock for us). We simply read the key and store the data for use.

El cambio más importante en el componente es que ahora nos hemos deshecho de nuestra dependencia del StockService. En su lugar, simplemente usamos ActivatedRoute. En ngOnInit, nos suscribimos a los cambios en el elemento data en ActivatedRoute. Los datos resueltos estarían disponibles en los datos con la clave que utilizamos en la ruta (stock para nosotros). Simplemente leemos la clave y almacenamos los datos para su uso.

You could easily extend the resolve to any data and any number of data items that you want to prefetch. The completed example is available in the *chapter11/route-guards* folder in the GitHub repository.

Puede extender fácilmente la resolución a cualquier dato y cualquier cantidad de elementos de datos que desee capturar previamente. El ejemplo completo está disponible en la carpeta capítulo11/route-guards en el repositorio de GitHub.

Conclusion

Conclusión

In this chapter, we took a deep dive into Angular routing. In particular, we saw how to start setting up the Angular router for any Angular application. We then dealt with handling different kinds of routes, and handling required and optional parameters in routing. We also dealt with handling protected routes, as well as ensuring that we don't lose data by navigating away from a filled-out form.

En este capítulo, profundizamos en el enrutamiento angular. En particular, vimos cómo comenzar a configurar el enrutador Angular para cualquier aplicación Angular. Luego nos ocupamos del manejo

de diferentes tipos de rutas y del manejo de parámetros requeridos yopcionales en el enrutamiento. También nos ocupamos del manejo de rutas protegidas, así como de garantizar que no perdamos datos al navegar fuera de un formulario completado.

In the next chapter, we will bring together all the topics we have covered, and then talk about what it takes to build a performant Angular application, and how to think about deploying it in production.

En el próximo capítulo, reuniremos todos los temas que hemos cubierto y luego hablaremos sobre lo que se necesita para crear una aplicación Angular de alto rendimiento y cómo pensar en implementarla en producción.

Exercise

Ejercicio

Take the base exercise from *chapter11/exercise/starter*. Install and run the server in the *chapter11/exercise/server* folder by running:

Realice el ejercicio básico del capítulo 11/ejercicio/inicio. Instale y ejecute el servidor en la carpeta capítulo11/ejercicio/servidor ejecutando:

```
npm i  
node index.js
```

from within the folder. This will start a local Node.js server which can work with products (similar to one we had for stocks). It exposes the following APIs:

desde dentro de la carpeta. Esto iniciará un servidor Node.js local que puede funcionar con productos (similar al que teníamos para las

existencias). Expone las siguientes API:

- GET on */api/product* to get a list of products. It can also take an optional query param q, which is the product name to search for.

OBTÉN /api/product para obtener una lista de productos.

También puede tomar un parámetro de consulta opcional q, que es el nombre del producto a buscar.

- GET on */api/product/:id* to get an individual product with an ID.

OBTÉN /api/product/:id para obtener un producto individual con una identificación.

- POST on */api/product* with a product information in the body to create a product on the server (in-memory of course; restarting the server would lose all created products).

PUBLICAR en /api/product con información del producto en el cuerpo para crear un producto en el servidor (en memoria, por supuesto; reiniciar el servidor perdería todos los productos creados).

- PATCH on */api/product/:id* with the product ID in the URL and a field changeInQuantity in the body would change the quantity in the cart of the product by that amount.

PATCH en /api/product/:id con el ID del producto en la URL y un campo changeInQuantity en el cuerpo cambiaría la cantidad en el carrito del producto en esa cantidad.

Given this, now try to accomplish the following:

Teniendo esto en cuenta, ahora intenta lograr lo siguiente:

- Hook up routing in the application. We want a login route, a register route, the product list route, a create product route, and a product details route. The components for the route are already created, as are the extra services you might need.

Conecte el enrutamiento en la aplicación. Queremos una ruta de inicio de sesión, una ruta de registro, una ruta de lista de productos, una ruta de creación de producto y una ruta de detalles del producto. Los componentes de la ruta ya están creados, al igual que los servicios extra que puedas necesitar.

- Only the create product route should be protected and accessible after login.

Sólo la ruta de creación del producto debe estar protegida y accesible después de iniciar sesión.

- In the product list and product details route, the add to cart should only be visible after login.

En la lista de productos y la ruta de detalles del producto, agregar al carrito solo debe ser visible después de iniciar sesión.

- See if you can adapt the login flow to remember that the user has logged in even through page refreshes.

Vea si puede adaptar el flujo de inicio de sesión para recordar que el usuario inició sesión incluso después de actualizar la página.

Most of this can be accomplished using concepts covered in this chapter. The only new thing might be trying to remember that you have logged in across page refreshes, for which you need to extend the service using `localStorage` or something similar. You can check out the finished solution in `chapter11/exercise/ecommerce`.

La mayor parte de esto se puede lograr utilizando los conceptos tratados en este capítulo. Lo único nuevo podría ser intentar recordar que has iniciado sesión durante las actualizaciones de la página, para lo cual necesitas ampliar el servicio usando `localStorage` o algo similar. Puede consultar la solución terminada en el capítulo 11/ejercicio/`ecommerce`.

Chapter 12. Productionizing an Angular App

Capítulo 12. Producción de una aplicación angular

In all the chapters so far, we have talked about the various bits and pieces that add up to an Angular application. We started at the very basic, and went to the more detailed and complex, from simple components to routing and server calls. But through all this, we focused on functionality and getting different bits to interact and play well together. At this point, you as a developer are ready to tackle 90% of most Angular application needs.

En todos los capítulos hasta ahora, hemos hablado sobre los distintos elementos que se suman a una aplicación Angular. Comenzamos desde lo más básico y pasamos a lo más detallado y complejo, desde componentes simples hasta enrutamiento y llamadas al servidor. Pero a través de todo esto, nos enfocamos en la funcionalidad y en lograr que diferentes partes interactúen y funcionen bien juntas. En este punto, usted, como desarrollador, está listo para abordar el 90% de la mayoría de las necesidades de las aplicaciones Angular.

In this chapter, we will focus on what it takes to get that application you have built out into production in a performant manner. We will cover all the things you will have to keep in mind when deploying your Angular applications to production, as well as some other concerns that you may not have thought of. We will cover how to

build an Angular app for production, how to reduce the size of the build, how to improve the performance, and even briefly cover other concerns like SEO.

En este capítulo, nos centraremos en lo que se necesita para que la aplicación que ha creado entre en producción de manera eficaz. Cubriremos todas las cosas que deberá tener en cuenta al implementar sus aplicaciones Angular en producción, así como algunas otras preocupaciones en las que quizás no haya pensado. Cubriremos cómo crear una aplicación Angular para producción, cómo reducir el tamaño de la compilación, cómo mejorar el rendimiento e incluso cubriremos brevemente otras preocupaciones como el SEO.

Building for Production

Edificio para la producción

So far, whenever we run our application, we usually ask the Angular CLI to serve our application, by running:

Hasta ahora, cada vez que ejecutamos nuestra aplicación, generalmente le pedimos a Angular CLI que entregue nuestra aplicación, ejecutando:

```
ng serve
```

This runs the Angular compiler, and builds and serves your Angular application using the internal server that the Angular CLI has. You could use the `build` command to generate the files to serve it for production as well. The command would simply be:

Esto ejecuta el compilador Angular y construye y sirve su aplicación Angular utilizando el servidor interno que tiene Angular CLI. También

puede usar el comando `build` para generar los archivos y servirlos para producción. El comando sería simplemente:

```
ng build
```

This would by default generate all your compiled files in a folder called `dist/`. You can then simply copy everything from this folder, put it on an HTTP server, and be off and running. But it is not what you should be doing! The build that gets generated by default is a suboptimal, overweight build that would make your production application slow to load and slow to run (comparatively, that is!). Angular allows you to build an optimized version of your application, so let's see how we might do that.

De forma predeterminada, esto generará todos los archivos compilados en una carpeta llamada `dist/`. Luego puede simplemente copiar todo lo que hay en esta carpeta, colocarlo en un servidor HTTP y listo. ¡Pero no es lo que deberías hacer! La compilación que se genera de forma predeterminada es una compilación subóptima y con sobrepeso que haría que su aplicación de producción se cargue y se ejecute lentamente (¡comparativamente, claro está!). Angular le permite crear una versión optimizada de su aplicación, así que veamos cómo podemos hacerlo.

Production Build

Construcción de producción

The simplest thing we can do to create a better build for production is to use the `prod` flag with the `ng build` command. Simply, you can run:

Lo más sencillo que podemos hacer para crear una mejor compilación para producción es usar el indicador `prod` con el comando `ng build`. Simplemente, puedes ejecutar:

```
ng build --prod
```

This does a few things of note:

Esto hace algunas cosas a tener en cuenta:

Bundling

agrupación

When we write our code, we like to keep it in nice, disparate files to make it easier to read, manage, and update. But for the end browser, loading 1,000 files is not as efficient as loading, say, 4 or 5 files. Angular CLI bundles all the application and library files into a few bundles to make it faster to load in the browser. Note that bundling happens both with and without the `--prod` flag.

Cuando escribimos nuestro código, nos gusta mantenerlo en archivos bonitos y dispares para que sea más fácil de leer, administrar y actualizar. Pero para el navegador final, cargar 1000 archivos no es tan eficiente como cargar, digamos, 4 o 5 archivos. Angular CLI agrupa todos los archivos de aplicaciones y bibliotecas en unos pocos paquetes para que se carguen más rápido en el navegador. Tenga en cuenta que la agrupación se produce tanto con como sin el indicador `--prod`.

Minification

Minificación

Spaces, indentation, and the like are useful for us developers, but the browser and systems running the code don't care.

Minification is the process of removing all unneeded spaces. The `--prod` does this for us, thus saving a few bytes of space in the final build.

Los espacios, las sangrías y similares son útiles para nosotros, los desarrolladores, pero al navegador y a los sistemas que ejecutan el código no les importa. La minificación es el proceso de eliminar

todos los espacios innecesarios. El `--prod` hace esto por nosotros, ahorrando así algunos bytes de espacio en la compilación final.

Uglification

Fealdad

The `--prod` flag uglifies the code as well, which is the process of replacing all nice, readable variable and function names with a smaller, two or three character name to save a few bytes. The overall code is much more efficient and smaller to load.

El indicador `--prod` tambiénfea el código, que es el proceso de reemplazar todos los nombres agradables y legibles de variables y funciones con un nombre más pequeño, de dos o tres caracteres, para ahorrar unos pocos bytes. El código general es mucho más eficiente y más pequeño de cargar.

AOT

AOT

We will talk about Ahead-of-Time (AOT) compilation in a little bit more detail in the following section, but in a nutshell, AOT compilation allows us to further reduce the size of code by dropping unused paths.

Hablaremos sobre la compilación anticipada (AOT) con un poco más de detalle en la siguiente sección, pero en pocas palabras, la compilación AOT nos permite reducir aún más el tamaño del código al eliminar las rutas no utilizadas.

Run Angular in production mode

Ejecute Angular en modo de producción

When we run Angular using `ng serve` (or build and run it without the `prod` flag), the Angular library performs some checks each time it renders in the UI. Think of these as training wheels, to

ensure that the developer doesn't end up developing something that invalidates or goes against Angular's patterns. These checks can add up to a few precious milliseconds during each render, and thus it is recommended that we turn these off in the final production build. The `--prod` does this for you.

Cuando ejecutamos Angular usando `ng serve` (o lo compilamos y ejecutamos sin el indicador `prod`), la biblioteca Angular realiza algunas comprobaciones cada vez que se representa en la interfaz de usuario. Piense en esto como ruedas de entrenamiento, para garantizar que el desarrollador no termine desarrollando algo que invalide o vaya en contra de los patrones de Angular. Estas comprobaciones pueden sumar unos preciosos milisegundos durante cada renderizado y, por lo tanto, se recomienda que las desactivemos en la compilación de producción final. El `--prod` hace esto por usted.

Dead code elimination

Eliminación de código muerto

There are times when you erroneously leave a module imported, or you haven't ended up using all of the functionality from a module. The build process removes all unused code and unreferenced modules, thus dropping the bundle size further.

Hay ocasiones en las que por error dejas un módulo importado o no terminas usando todas las funciones de un módulo. El proceso de compilación elimina todo el código no utilizado y los módulos sin referencia, lo que reduce aún más el tamaño del paquete.

At the end of this, you should have the files you need to deploy in the `dist` folder, each with an individual hash based on the contents. This would be a pretty optimal build that should be good in a majority of cases.

Al final de esto, debería tener los archivos que necesita implementar en la carpeta `dist`, cada uno con un hash individual basado en el

contenido. Esta sería una construcción bastante óptima que debería ser buena en la mayoría de los casos.

Ahead-of-Time (AOT) Compilation and Build Optimizer

Optimizador de compilación y compilación anticipada (AOT)

We briefly mentioned Ahead-of-Time compilation in the previous section. This mode has become enabled by default in any production build since the 1.5 version of the Angular CLI.

Mencionamos brevemente la compilación anticipada en la sección anterior. Este modo se ha habilitado de forma predeterminada en cualquier compilación de producción desde la versión 1.5 de Angular CLI.

Angular applications use what we call Just-in-Time (JIT) compilation, where the application is compiled at runtime in the browser before running. This is also the default when you run the Angular application using `ng serve` or `ng build`.

Las aplicaciones angulares utilizan lo que llamamos compilación Just-in-Time (JIT), donde la aplicación se compila en tiempo de ejecución en el navegador antes de ejecutarse. Este también es el valor predeterminado cuando ejecuta la aplicación Angular usando `ng serve` o `ng build`.

In production mode, Angular uses AOT for compilation, which means that Angular compiles as much of the application as possible upfront. Thus, when the application is served to the browser, it is already precompiled and optimal, thus allowing the browser to quickly render and execute the application.

En el modo de producción, Angular usa AOT para la compilación, lo que significa que Angular compila la mayor cantidad posible de la

aplicación por adelantado. Por lo tanto, cuando la aplicación se entrega al navegador, ya está precompilada y es óptima, lo que permite que el navegador represente y ejecute rápidamente la aplicación.

Furthermore, as part of the compilation process, all HTML templates and CSS are inlined within the application bundle, thus saving asynchronous requests to load them later.

Además, como parte del proceso de compilación, todas las plantillas HTML y CSS están integradas en el paquete de la aplicación, lo que ahorra solicitudes asincrónicas para cargarlas más tarde.

There is also a significant reduction in the size of the built bundle, as the Angular compiler, which constitutes almost half of the Angular library, can be omitted. The compiler's work, of checking templates, bindings, and the like, can now be done at compile time, thus catching them earlier rather than after deploying the application.

También hay una reducción significativa en el tamaño del paquete creado, ya que se puede omitir el compilador Angular, que constituye casi la mitad de la biblioteca Angular. El trabajo del compilador, de comprobar plantillas, enlaces y similares, ahora se puede realizar en tiempo de compilación, detectándolos antes en lugar de después de implementar la aplicación.

Build Optimizer is a webpack plug-in that was introduced by the Angular team to further optimize the bundle beyond what webpack is capable of. In particular, it focuses on removing some of the decorators and other code that is not relevant for the final build. It only works with AOT, so you shouldn't end up using it with non-AOT builds. Since Angular CLI 1.5, this has been enabled by default whenever you do a production build in Angular along with AOT.

Build Optimizer es un complemento de paquete web que fue introducido por el equipo de Angular para optimizar aún más el paquete más allá de lo que el paquete web es capaz de hacer. En particular, se centra en eliminar algunos de los decoradores y otro

código que no es relevante para la compilación final. Solo funciona con AOT, por lo que no deberías terminar usándolo con compilaciones que no sean AOT. Desde Angular CLI 1.5, esto se ha habilitado de forma predeterminada cada vez que realiza una compilación de producción en Angular junto con AOT.

There is a lot more to the AOT compiler, and the options it provides and how we can modify and play around with it. You can read up on it in more detail in [the official Angular docs](#).

Hay mucho más sobre el compilador AOT, las opciones que proporciona y cómo podemos modificarlo y jugar con él. Puede leerlo con más detalle en los documentos oficiales de Angular.

Thus, unless there is a very strong reason (and there usually isn't), leave the AOT enabled when you generate the final build, which should give you as close to an optimal build with minimal additional work.

Por lo tanto, a menos que haya una razón muy fuerte (y generalmente no la hay), deje el AOT habilitado cuando genere la compilación final, lo que debería brindarle una compilación lo más cercana posible a una compilación óptima con un mínimo de trabajo adicional.

Base Href

Referencia base

One additional concern when building and deploying any Single-Page Application is where it is served from. In cases where your application is served from the root domain (say, www.mytestpage.com), the default should work.

Una preocupación adicional al crear e implementar cualquier aplicación de una sola página es desde dónde se sirve. En los casos

en que su aplicación se sirve desde el dominio raíz (por ejemplo, `www.mytestpage.com`), el valor predeterminado debería funcionar.

In other cases though, where your application is served not from the root domain (say, `www.mytestpage.com/app` or something similar), then it is important that we update our `<base>` tag in the `index.html`.

Sin embargo, en otros casos, cuando su aplicación no se sirve desde el dominio raíz (por ejemplo, `www.mytestpage.com/app` o algo similar), es importante que actualicemos nuestra etiqueta `<base>` en `index.html`.

The HTML `base` tag is responsible for setting the base path for all relative URLs in our application. These include, but are not limited to, CSS/style files, our JavaScript application and library files, images, and more.

La etiqueta HTML `base` es responsable de establecer la ruta base para todas las URL relativas en nuestra aplicación. Estos incluyen, entre otros, archivos CSS/estilo, nuestra aplicación JavaScript y archivos de biblioteca, imágenes y más.

Let's take the case of serving an application from `www.mytestpage.com/app`, and see how the `base` tag would impact this:

Tomemos el caso de servir una aplicación desde `www.mytestpage.com/app` y veamos cómo la etiqueta `base` afectaría esto:

1. Let's assume that we didn't have a `<base>` tag, or the tag was `<base href="/">`. In this case, when we have the script tag `<script src="js/main.js">`, then the browser will make a request to `www.mytestpage.com/js/main.js`.

Supongamos que no teníamos una etiqueta `<base>` o que la etiqueta era `<base href="/">`. En este caso, cuando tengamos

la etiqueta de script `<script src="js/main.js">`, el navegador realizará una solicitud a `www.mytestpage.com/js/main.js`.

2. Now let's assume that we had the following base tag: `<base href="/app">`. In this case, when we have the script tag `<script src="js/main.js">`, then the browser will make a request to `www.mytestpage.com/app/js/main.js`.

Ahora supongamos que teníamos la siguiente etiqueta base: `<base href="/app">`. En este caso, cuando tengamos la etiqueta de secuencia de comandos `<script src="js/main.js">`, el navegador realizará una solicitud a `www.mytestpage.com/app/js/main.js`.

As you can clearly see, the second request is the correct one, and ensures that the necessary images and scripts are loaded correctly.

Como puede ver claramente, la segunda solicitud es la correcta y garantiza que las imágenes y scripts necesarios se carguen correctamente.

How does this play into our Angular application? When we build our Angular application using the Angular CLI, we can specify or overwrite the base href value. Continuing our example from earlier, we could build our application as follows:

¿Cómo influye esto en nuestra aplicación Angular? Cuando creamos nuestra aplicación Angular usando Angular CLI, podemos especificar o sobrescribir el valor base href. Continuando con nuestro ejemplo anterior, podríamos construir nuestra aplicación de la siguiente manera:

```
ng build --base-href /app/
```

This would ensure that the generated `index.html` has the correct base tag for your deployment.

Esto garantizaría que el index.html generado tenga la etiqueta base correcta para su implementación.

Deploying an Angular Application

Implementación de una aplicación angular

There are a ton more options when building your Angular application for deployment, and you can read up on all the options and their uses at [the Angular CLI build wiki](#).

Hay muchas más opciones al crear su aplicación Angular para su implementación, y puede leer sobre todas las opciones y sus usos en la wiki de compilación de Angular CLI.

But for the purpose of deploying a mostly performant Angular application, the options we just reviewed cover the majority of the use cases. At this point, you would have a *dist/* folder (unless you have overridden the default) with generated files. Each generated file would also have a hash representing the contents.

Pero con el fin de implementar una aplicación Angular con mayor rendimiento, las opciones que acabamos de revisar cubren la mayoría de los casos de uso. En este punto, tendrá una carpeta *dist/* (a menos que haya anulado el valor predeterminado) con los archivos generados. Cada archivo generado también tendría un hash que representa el contenido.

At this point, you should be able to take the entire folder, drop it in to your frontend server (be it nginx, Apache, or whatever), and start serving your Angular application. As long as you serve the entire folder, and your base path (as we saw in the preceding section) is set correctly, you should be off to the races. There are a few more concerns that we will cover in the next section, from handling caching, deep-linking, and others, but this would be the base you will build off of.

En este punto, debería poder tomar la carpeta completa, colocarla en su servidor frontend (ya sea nginx, Apache o lo que sea) y comenzar a servir su aplicación Angular. Siempre que sirva toda la carpeta y su ruta base (como vimos en la sección anterior) esté configurada correctamente, debería estar listo. Hay algunas preocupaciones más que cubriremos en la siguiente sección, desde el manejo del almacenamiento en caché, los enlaces profundos y otros, pero esta sería la base a partir de la cual construirá.

Other Concerns

Otras preocupaciones

In the previous section, we saw the bare minimum it would take to drop a mostly optimal build into our frontend server and have it start serving traffic. In this section, we will go slightly deeper into specific concerns that we need to think about to ensure better performance or proper functionality.

En la sección anterior, vimos lo mínimo que se necesitaría para colocar una compilación óptima en nuestro servidor frontend y hacer que comience a atender tráfico. En esta sección, profundizaremos un poco más en inquietudes específicas en las que debemos pensar para garantizar un mejor rendimiento o una funcionalidad adecuada.

Caching

Almacenamiento en caché

The first topic we will start with is caching. And in this section, we are particularly talking about frontend code caching, not the API response caching. That is, how and when should we cache our

index.html, our JS files and CSS files, and how long should they remain cached.

El primer tema con el que comenzaremos es el almacenamiento en caché. Y en esta sección, estamos hablando particularmente del almacenamiento en caché del código frontend, no del almacenamiento en caché de la respuesta API. Es decir, cómo y cuándo debemos almacenar en caché nuestro *index.html*, nuestros archivos JS y CSS, y cuánto tiempo deben permanecer almacenados en caché.

When we create our production builds, notice that our generated files (other than the *index.html*) have a hash in the filename (like *inline.62ca64ed6c08f96e698b.bundle.js*). This hash is generated based on the contents of the file, so if the content of the file changes, the hash in the filename also changes! Also, our generated *index.html* explicitly refers to these generated files and loads them as scripts or as styles.

Cuando creamos nuestras compilaciones de producción, observe que nuestros archivos generados (que no sean *index.html*) tienen un hash en el nombre del archivo (como *inline.62ca64ed6c08f96e698b.bundle.js*). Este hash se genera en función del contenido del archivo, por lo que si el contenido del archivo cambia, el hash en el nombre del archivo también cambia. Además, nuestro *index.html* generado se refiere explícitamente a estos archivos generados y los carga como scripts o estilos.

This gives us a great mechanism now for caching these files on the browser. Our simple rule of thumb for caching now becomes:

Esto nos brinda un excelente mecanismo ahora para almacenar en caché estos archivos en el navegador. Nuestra simple regla general para el almacenamiento en caché ahora es:

- We *never* cache our *index.html* file on the browser. This would mean setting the Cache-Control header on your server just for

the *index.html* to be set to `no-cache, no-store, must-revalidate`. Note that this is only for the *index.html* file, and not for all other files from your frontend server. The *index.html* file is tiny anyway, and we can quickly revalidate it if needed.

Nunca almacenamos en caché nuestro archivo *index.html* en el navegador. Esto significaría configurar el encabezado Cache-Control en su servidor solo para que *index.html* se establezca en `no-cache, no-store, must-revalidate`. Tenga en cuenta que esto es sólo para el archivo *index.html* y no para todos los demás archivos de su servidor frontend. El archivo *index.html* es pequeño de todos modos y podemos revalidarlo rápidamente si es necesario.

- We *always* cache all other asset files, like our JavaScript bundles and our CSS files, as long as possible. Again, because the asset filenames themselves will change, we can be guaranteed that our noncached *index.html* will always load the correct asset files. And those files can remain cached indefinitely. This ensures a great second load performance, where all asset files can be served from the cache.

Siempre almacenamos en caché todos los demás archivos de activos, como nuestros paquetes de JavaScript y nuestros archivos CSS, durante el mayor tiempo posible. Nuevamente, debido a que los nombres de los archivos de activos cambiarán, podemos tener la garantía de que nuestro *index.html* no almacenado en caché siempre cargará los archivos de activos correctos. Y esos archivos pueden permanecer almacenados en caché indefinidamente. Esto garantiza un excelente rendimiento de segunda carga, donde todos los archivos de activos se pueden servir desde la memoria caché.

Now, let's talk through a few potential user scenarios to make it clear how this caching strategy handles them:

Ahora, analicemos algunos escenarios de usuarios potenciales para dejar claro cómo los maneja esta estrategia de almacenamiento en caché:

First load, new user

Primera carga, nuevo usuario

For the very first request from a fresh user, the browser first requests the *index.html* from the server. The server returns it, after which the browser processes the *index.html*. Based on the files it then asks for, the browser makes request for the styles and scripts from the server. Since it is the first request, none of the data is cached, and all the required files are served by the server. Finally, the app is up and running after all the files are loaded.

Para la primera solicitud de un usuario nuevo, el navegador primero solicita el *index.html* del servidor. El servidor lo devuelve, después de lo cual el navegador procesa el *index.html*. En función de los archivos que solicita, el navegador solicita los estilos y scripts del servidor. Dado que es la primera solicitud, ninguno de los datos se almacena en caché y el servidor proporciona todos los archivos necesarios. Finalmente, la aplicación está en funcionamiento después de cargar todos los archivos.

Second load, repeat user

Segunda carga, repite usuario

The second time the user comes back to our application, the browser again requests the *index.html* from the frontend server. This is because the *index.html* has not been cached according to our Cache-Control headers. The frontend server responds with the latest content for *index.html*, which has not changed. Now, the browser looks at the script and style files it has to load, which has also not changed. But these files are perpetually cached on

the browser. Thus, the browser doesn't need to make server requests to load them, and instead loads them from the local cache. Our app is now up and running immediately almost as soon as the *index.html* finishes loading.

La segunda vez que el usuario regresa a nuestra aplicación, el navegador solicita nuevamente el index.html del servidor frontend. Esto se debe a que index.html no se ha almacenado en caché según nuestros encabezados Cache-Control. El servidor frontend responde con el contenido más reciente de index.html, que no ha cambiado. Ahora, el navegador mira los archivos de script y de estilo que tiene que cargar, los cuales tampoco han cambiado. Pero estos archivos se almacenan permanentemente en caché en el navegador. Por lo tanto, el navegador no necesita realizar solicitudes al servidor para cargarlos, sino que los carga desde la memoria caché local. Nuestra aplicación ya está operativa inmediatamente casi tan pronto como termina de cargarse index.html.

First load, new user, website updated

Primera carga, nuevo usuario, sitio web actualizado

In case a new user visits after we have updated our website source code, the flow followed is exactly the same as the first load case for a new user.

En caso de que un nuevo usuario visite después de haber actualizado el código fuente de nuestro sitio web, el flujo seguido es exactamente el mismo que el primer caso de carga para un nuevo usuario.

Second load, repeat user, website updated

Segunda carga, usuario repetido, sitio web actualizado

In this case, the user had visited our website in the past, and has a cached version of the styles and scripts on his browser. Now when we visit the website, after we have pushed out a new

update, the very first thing that happens will not change. The browser will make a request for *index.html* from the frontend server. The server will return the new version of the *index.html* that points to the new script and style tags. Now when the browser tries to load these, it will realize that it doesn't have a cached version of the script and style tags (because the content-hash in the filename has changed). It will again make a request to the server to load all these files, and thus the flow becomes very similar to the first flow where the user is visiting for the first time.

En este caso, el usuario había visitado nuestro sitio web en el pasado y tiene una versión en caché de los estilos y scripts en su navegador. Ahora, cuando visitamos el sitio web, después de haber lanzado una nueva actualización, lo primero que sucede no cambiará. El navegador realizará una solicitud de *index.html* desde el servidor frontend. El servidor devolverá la nueva versión de *index.html* que apunta a las nuevas etiquetas de script y estilo. Ahora, cuando el navegador intente cargarlos, se dará cuenta de que no tiene una versión en caché del script y las etiquetas de estilo (porque el hash de contenido en el nombre del archivo ha cambiado). Nuevamente realizará una solicitud al servidor para cargar todos estos archivos y, por lo tanto, el flujo se vuelve muy similar al primer flujo que el usuario visita por primera vez.

Thus, we can see that with this kind of caching mechanism, we get the best of both worlds, which is ensuring that we cache as much as possible, without breaking the user experience in case of pushing out updates to our web application.

Por lo tanto, podemos ver que con este tipo de mecanismo de almacenamiento en caché, obtenemos lo mejor de ambos mundos, que es garantizar que almacenemos en caché la mayor cantidad posible, sin alterar la experiencia del usuario en caso de publicar actualizaciones en nuestra aplicación web.

API/Server Calls and CORS

Llamadas API/servidor y CORS

The second topic worth covering is how to set up your web application so that it can make server calls successfully. The browser, for security reasons, prevents your web application from making asynchronous calls outside of its domain (this includes sub-domains as well). That is, your web application running on www.mytestpage.com cannot make an AJAX call to www.mytestapi.com, or even to api.mytestpage.com.

El segundo tema que vale la pena cubrir es cómo configurar su aplicación web para que pueda realizar llamadas al servidor con éxito. El navegador, por razones de seguridad, evita que su aplicación web realice llamadas asincrónicas fuera de su dominio (esto también incluye subdominios). Es decir, su aplicación web que se ejecuta en www.mytestpage.com no puede realizar una llamada AJAX a www.mytestapi.com, ni siquiera a api.mytestpage.com.

We get around this during development by using a proxy as part of the Angular CLI (remember `ng serve --proxy` `proxy.config.json`?). The proxy ensured that our requests were made to the same frontend server (and domain) serving our static files, and it then was responsible to proxy the calls to the actual API server.

Solucionamos esto durante el desarrollo usando un proxy como parte de Angular CLI (¿recuerdas `ng serve --proxy` `proxy.config.json`?). El proxy aseguró que nuestras solicitudes se realizaran al mismo servidor frontend (y dominio) que sirve nuestros archivos estáticos, y luego fue responsable de enviar las llamadas al servidor API real.

When we deploy our web application in production, you will also need to set up something similar to this. That is, your frontend

server will be the one getting the initial API calls, and it then has to proxy those requests forward to the actual API server.

Cuando implementemos nuestra aplicación web en producción, también necesitarás configurar algo similar a esto. Es decir, su servidor frontend será el que reciba las llamadas API iniciales y luego tendrá que enviar esas solicitudes al servidor API real.

Your frontend server would end up behaving something like shown in [Figure 12-1](#).

Su servidor frontend terminaría comportándose como se muestra en la Figura 12-1.

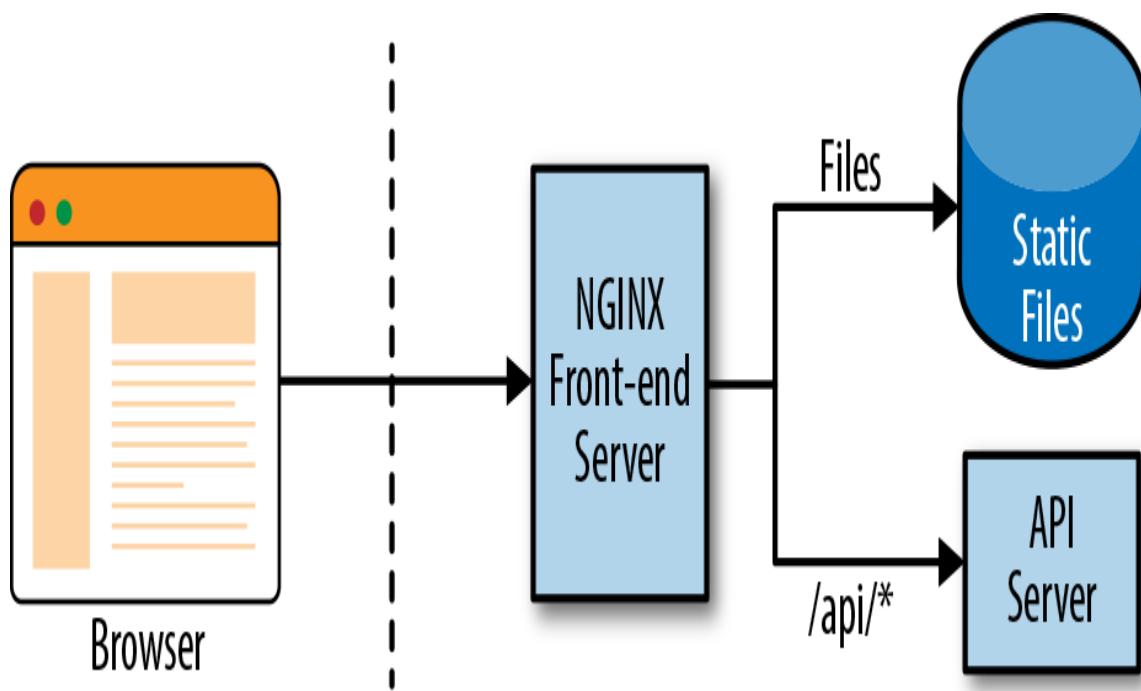


Figure 12-1. Simple end-to-end architecture for a web application

Figura 12-1. Arquitectura simple de un extremo a otro para una aplicación web

We have used NGINX as an example, but you could easily replace it with Apache or IIS configuration, which does exactly the same. We simply route all requests to our API server (`/api/`), and all others to our static files, which is our Angular application.

Hemos usado NGINX como ejemplo, pero puedes reemplazarlo fácilmente con la configuración Apache o IIS, que hace exactamente lo mismo. Simplemente enrutamos todas las solicitudes a nuestro servidor API (/api/) y todas las demás a nuestros archivos estáticos, que es nuestra aplicación Angular.

If you really can't do this (for whatever reason), then there is a second option, which again requires changes on your server. We can enable Cross-Origin Resource Sharing (CORS) on our server, which can allow pages from different origins (read domains or subdomains) to make requests, bypassing the browser security restrictions. It simply requires the API server to respond with an additional header:

Si realmente no puede hacer esto (por cualquier motivo), entonces existe una segunda opción, que nuevamente requiere cambios en su servidor. Podemos habilitar el uso compartido de recursos entre orígenes (CORS) en nuestro servidor, lo que puede permitir que páginas de diferentes orígenes (léase dominios o subdominios) realicen solicitudes, evitando las restricciones de seguridad del navegador. Simplemente requiere que el servidor API responda con un encabezado adicional:

```
Access-Control-Allow-Origin: *
```

You can also restrict this header to allow only requests from certain origins, instead of all origins like we have done here. Once you do this, then you can allow your web application to directly make requests to the API server. You can learn more about CORS [here](#), and see how to configure it for your particular server [here](#).

También puede restringir este encabezado para permitir solo solicitudes de ciertos orígenes, en lugar de todos los orígenes como lo hemos hecho aquí. Una vez que haga esto, podrá permitir que su aplicación web realice solicitudes directamente al servidor API. Puede obtener más información sobre CORS aquí y ver cómo configurarlo para su servidor en particular [aquí](#).

Different Environments

Diferentes entornos

Another common requirement when building an application is having different configuration for different environments. For example, you might have different API keys for your client-side tracking libraries, or you might even have different server URLs to configure for test versus production.

Otro requisito común al crear una aplicación es tener una configuración diferente para diferentes entornos. Por ejemplo, es posible que tenga diferentes claves API para las bibliotecas de seguimiento del lado del cliente, o incluso puede que tenga diferentes URL de servidor para configurar para prueba o producción.

In such cases, you can use the concept of *environments* in Angular. By default, when you create a new Angular application using the Angular CLI, it creates an *src/environments* folder, with one file per environment. By default, the Angular CLI makes the properties available in the *environment.ts* file available across your application. But you can override it by running Angular with:

En tales casos, puedes utilizar el concepto de entornos en Angular. De forma predeterminada, cuando crea una nueva aplicación Angular usando la CLI de Angular, crea una carpeta *src/environments*, con un archivo por entorno. De forma predeterminada, Angular CLI hace que las propiedades del archivo *Environment.ts* estén disponibles en toda su aplicación. Pero puedes anularlo ejecutando Angular con:

```
ng serve --env=prod
```

When you run it like this, it uses the value passed to *--env* flag, and loads and makes available the corresponding environments file. In

this case, it would make the `environment.prod.ts` file available.

Cuando lo ejecuta de esta manera, utiliza el valor pasado al indicador `--env` y carga y pone a disposición el archivo de entorno correspondiente. En este caso, el archivo `Environment.prod.ts` estaría disponible.

In your application, you can simply import the main `environment` file like so, and Angular will ensure you get the correct properties based on the flag:

En su aplicación, simplemente puede importar el archivo del entorno principal de esta manera, y Angular se asegurará de obtener las propiedades correctas según la bandera:

```
import { environment } from './environments/environment';
```

Handling Deep-Linking

Manejo de enlaces profundos

The last thing to have a really complete web application is to support deep-linking. We already saw how to enable routing for your Angular application in [Chapter 11](#). Once you deployed your Angular application, you might notice something weird or annoying. When you navigate to the base route of your application, your frontend server serves the `index.html`, and your application works. Any routes from that point on work as you would expect it to. But if you try to directly link to a route within your application, it might not.

Lo último para tener una aplicación web realmente completa es admitir enlaces profundos. Ya vimos cómo habilitar el enrutamiento para su aplicación Angular en el Capítulo 11. Una vez que haya implementado su aplicación Angular, es posible que note algo extraño o molesto. Cuando navega a la ruta base de su aplicación, su servidor frontend sirve el `index.html` y su aplicación funciona. Cualquier ruta desde ese punto en adelante funciona como es de

esperar. Pero si intenta vincular directamente a una ruta dentro de su aplicación, es posible que no sea así.

This is due to how we have set up our frontend server to serve the static files necessary for the Angular application. Let's take a closer look at what is happening here:

Esto se debe a cómo hemos configurado nuestro servidor frontend para servir los archivos estáticos necesarios para la aplicación Angular. Echemos un vistazo más de cerca a lo que está sucediendo aquí:

1. If you request for the base route, your frontend server translates that to serve the *index.html* file. This also loads all the relevant scripts and CSS, and bootstraps your Angular application.

Si solicita la ruta base, su servidor frontend la traduce para servir el archivo index.html. Esto también carga todos los scripts y CSS relevantes, y arranca su aplicación Angular.

2. After this, any link within your application is intercepted by Angular within the browser, and Angular serves the relevant content for that route. So while the route in your browser changes, it is actually not making a request to the server for that new route. It behaves like a Single-Page Application should.

Después de esto, Angular intercepta cualquier enlace dentro de su aplicación dentro del navegador y Angular ofrece el contenido relevante para esa ruta. Entonces, aunque la ruta en su navegador cambia, en realidad no realiza una solicitud al servidor para esa nueva ruta. Se comporta como debería hacerlo una aplicación de una sola página.

3. Now in the case that we want to directly open a certain route, when we enter that route in the browser, the browser makes that request to our frontend server. Unless you have set up your frontend server configuration correctly, it is not going to find any

match for this URL (which is a frontend only route). Thus it fails to serve and most often ends up serving the 404 page (or whatever you might have configured).

Ahora, en el caso de que queramos abrir directamente una determinada ruta, cuando ingresamos esa ruta en el navegador, el navegador realiza esa solicitud a nuestro servidor frontend. A menos que haya configurado correctamente la configuración de su servidor frontend, no encontrará ninguna coincidencia para esta URL (que es una ruta únicamente frontend). Por lo tanto, no funciona y la mayoría de las veces termina publicando la página 404 (o cualquier cosa que haya configurado).

To work around this, we need to set up our frontend server to serve requests as follows, in order of priority:

Para solucionar este problema, debemos configurar nuestro servidor frontend para atender solicitudes de la siguiente manera, en orden de prioridad:

1. Recognize all API requests, and proxy that to the actual backend server to serve those requests. Keep API requests under a common path, so that you can proxy all of them consistently and first (for example, always beginning API requests with /api).

Reconozca todas las solicitudes de API y envíelas al servidor backend real para atender esas solicitudes. Mantenga las solicitudes de API en una ruta común, de modo que pueda enviarlas todas de manera consistente y primero (por ejemplo, siempre comenzando las solicitudes de API con /api).

2. Match and serve any request that translates to a static file (say, a JS, CSS file, or something like those).

Haga coincidir y atienda cualquier solicitud que se traduzca en un archivo estático (por ejemplo, un archivo JS, CSS o algo así).

3. Either serve all remaining requests with the *index.html* file, or match all frontend routes (in addition to the base / route) with the *index.html*.

Atienda todas las solicitudes restantes con el archivo index.html o haga coincidir todas las rutas de interfaz (además de la ruta base /) con index.html.

An easy way to do this with an NGINX server would be to use the `try_files` directive to serve the *index.html* as a fallback in case a file with the path is not found.

Una manera fácil de hacer esto con un servidor NGINX sería usar la directiva `try_files` para servir el index.html como respaldo en caso de que no se encuentre un archivo con la ruta.

Once you have set up your frontend server as described, then a deep-linked route will end up matching the last category of requests, and the *index.html* will be served. Once the *index.html* loads and Angular is bootstrapped, Angular then takes over for all further routing and loading necessary content based on the route in the browser.

Una vez que haya configurado su servidor frontend como se describe, una ruta de enlace profundo terminará coincidiendo con la última categoría de solicitudes y se entregará el index.html. Una vez que se carga index.html y se inicia Angular, Angular se hace cargo de todo el enrutamiento adicional y carga el contenido necesario según la ruta en el navegador.

WARNING ADVERTENCIA

Make sure your base tag is set up correctly so that the web page knows where to load the static files and relative paths from. Otherwise, your Angular application will not work even if you have set up your frontend server correctly.

Asegúrese de que su etiqueta base esté configurada correctamente para que la página web sepa desde dónde cargar los archivos estáticos y las rutas relativas. De lo contrario, su aplicación Angular no funcionará incluso si ha configurado su servidor frontend correctamente.

You can look at the [official Angular docs](#) for an updated set of configurations that works for different frontend servers. Configurations for NGINX, Apache, IIS, and more are available there.

Puede consultar los documentos oficiales de Angular para obtener un conjunto actualizado de configuraciones que funcionen para diferentes servidores frontend. Las configuraciones para NGINX, Apache, IIS y más están disponibles allí.

Lazy Loading

Carga lenta

One more technique for a highly performant app, which we very briefly touched upon in [Chapter 11](#) when we were talking about routing in Angular, is lazy loading. Once we introduced the concept of routing into our Angular applications, you might have realized that not all of the routes are really necessary or need to be loaded.

Una técnica más para una aplicación de alto rendimiento, que abordamos brevemente en el Capítulo 11 cuando hablamos sobre enrutamiento en Angular, es la carga diferida. Una vez que introdujimos el concepto de enrutamiento en nuestras aplicaciones

Angular, es posible que se haya dado cuenta de que no todas las rutas son realmente necesarias o deben cargarse.

So one common trick that we use to increase the performance and reduce the initial load time is to try to load the bare minimum up front in the initial request, and defer loading everything else to as and when it's needed. We accomplish this by leveraging the Angular routing and using what we call child routes.

Entonces, un truco común que usamos para aumentar el rendimiento y reducir el tiempo de carga inicial es intentar cargar lo mínimo por adelantado en la solicitud inicial y posponer la carga de todo lo demás cuando sea necesario. Logramos esto aprovechando el enrutamiento angular y usando lo que llamamos rutas secundarias.

The technique in a nutshell is as follows:

La técnica en pocas palabras es la siguiente:

1. Instead of defining all our routes up front, we break up our application into smaller modules, each with their routes defined in self-contained units.

En lugar de definir todas nuestras rutas por adelantado, dividimos nuestra aplicación en módulos más pequeños, cada uno con sus rutas definidas en unidades independientes.

2. The respective components are now registered at these submodule level only, and not at the main application-level module.

Los componentes respectivos ahora se registran únicamente en el nivel de estos submódulos y no en el módulo principal de nivel de aplicación.

3. We register all these routes as child routes in each individual module.

Registraremos todas estas rutas como rutas secundarias en cada módulo individual.

4. At the application level, we change our routing to instead point certain subpaths at the new module, rather than the individual routes.

A nivel de aplicación, cambiamos nuestra ruta para señalar ciertas subrutas al nuevo módulo, en lugar de rutas individuales.

Now, when we run our application, Angular will load the bare minimal code up front, and load the remaining modules as and when we navigate to those routes.

Ahora, cuando ejecutamos nuestra aplicación, Angular cargará el código mínimo por adelantado y cargará los módulos restantes a medida que naveguemos hacia esas rutas.

Let's take our application from the previous chapter, and see how to convert it into a lazy-loading application. You can use the code from *chapter11/route-guards* as the base to convert into the lazy-loading application. Before we get into the nitty gritties, let's talk through the changes we will make:

Tomemos nuestra aplicación del capítulo anterior y veamos cómo convertirla en una aplicación de carga diferida. Puede utilizar el código del capítulo 11/route-guards como base para convertirlo en la aplicación de carga diferida. Antes de entrar en el meollo de la cuestión, hablemos de los cambios que haremos:

- We will create two new modules, a `UserModule` and a `StockModule`. The `UserModule` will hold the `Login` and `Register` components, and the routes for them. The `StockModule` would hold the routes and components related to showing and creating stocks. Note that for now, we will leave the services registered at the parent level, though you could optimize further and split them into related modules only.

Crearemos dos nuevos módulos, un `UserModule` y un `StockModule`. El `UserModule` contendrá los componentes de inicio de sesión y registro, y las rutas para ellos. El `StockModule` contendría las rutas y componentes relacionados con mostrar y crear acciones. Tenga en cuenta que, por ahora, dejaremos los servicios registrados en el nivel principal, aunque podría optimizarlos aún más y dividirlos solo en módulos relacionados.

- We will redefine our routes to have a nice parent path for all related and grouped routes. So our login and register routes will move under a `user` parent path, and the stock routes will move under a `stock` parent path. This also means that all our redirects and navigation within the app will have to change to refer to the new URLs.

Redefiniremos nuestras rutas para tener una ruta principal agradable para todas las rutas relacionadas y agrupadas. Por lo tanto, nuestras rutas de inicio de sesión y registro se moverán bajo una ruta principal `user`, y las rutas comunes se moverán bajo una ruta principal `stock`. Esto también significa que todas nuestras redirecciones y navegación dentro de la aplicación tendrán que cambiar para hacer referencia a las nuevas URL.

- Finally, we will change the main `AppModule` and the routes to use lazy routing and register only relevant components and services.

Finalmente, cambiaremos el `AppModule` principal y las rutas para usar enrutamiento diferido y registrar solo componentes y servicios relevantes.

Let's walk through these changes step by step, along with the respective code.

Repasemos estos cambios paso a paso, junto con el código respectivo.

First, we will generate two new modules, along with their corresponding routing module:

Primero, generaremos dos nuevos módulos, junto con su correspondiente módulo de enrutamiento:

```
ng generate module stock --routing  
ng generate module user --routing
```

This will generate the following four files:

Esto generará los siguientes cuatro archivos:

- *src/app/stock/stock.module.ts*
src/app/stock/stock.module.ts
- *src/app/stock/stock-routing.module.ts*
src/app/stock/stock-routing.module.ts
- *src/app/user/user.module.ts*
src/app/user/user.module.ts
- *src/app/user/user-routing.module.ts*
src/app/user/user-routing.module.ts

Now let's see how we will modify each one to set up our application for lazy loading. First, we'll start with the *user-routing.module.ts* file:

Ahora veamos cómo modificaremos cada uno para configurar nuestra aplicación para carga diferida. Primero, comenzaremos con el archivo *user-routing.module.ts*:

```
import { NgModule } from '@angular/core';  
import { Routes, RouterModule } from '@angular/router';  
import { LoginComponent } from './login/login.component';  
import { RegisterComponent } from './register/register.component';  
  
const routes: Routes = [
```

```

    { path: 'login', component: LoginComponent },
    { path: 'register', component: RegisterComponent },
];

@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule]
})
export class UserRoutingModule { }

```

We simply add our two routes for login and register to the `routes` array. These have been moved from the `app-routes.module.ts` file. Also, note one major difference. Previously, whenever we registered our routes, we registered them as `RouterModule.forRoot`. Now we have started registering them as child routes. This is how Angular differentiates between parent/root routes and child routes.

Simplemente agregamos nuestras dos rutas para iniciar sesión y registrarnos en la matriz `routes`. Estos se han movido del archivo `app-routes.module.ts`. Además, observe una diferencia importante. Anteriormente, cada vez que registrábamos nuestras rutas, las registrábamos como `RouterModule.forRoot`. Ahora hemos comenzado a registrarlas como rutas secundarias. Así es como Angular diferencia entre rutas principal/raíz y rutas secundarias.

Our `user.module.ts` will also change as follows:

Nuestro `user.module.ts` también cambiará de la siguiente manera:

```

import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';

import { LoginComponent } from './login/login.component';
import { RegisterComponent } from './register/register.component';

import { UserRoutingModule } from './user-routing.module';
import { FormsModule } from '@angular/forms';

@NgModule({
  imports: [
    CommonModule,

```

```

    FormsModule,
    UserRoutingModule
  ],
  declarations: [
    LoginComponent,
    RegisterComponent,
  ]
})
export class UserModule { }

```

We end up with a very simple `UserModule`, which just declares the two components: `LoginComponent` and the `RegisterComponent`. Also note that we have imported the `FormsModule`, because we use `ngModel` binding in the forms. We don't define the services here, because we rely on them from the main `AppModule` instead.

Terminamos con un `UserModule` muy simple, que simplemente declara los dos componentes: `LoginComponent` y `RegisterComponent`. También tenga en cuenta que hemos importado el `FormsModule`, porque usamos el enlace `ngModel` en los formularios. No definimos los servicios aquí, porque en su lugar dependemos de ellos desde el `AppModule` principal.

Our changes to the `stock-routing.module.ts` file are also similar:

Nuestros cambios en el archivo `stock-routing.module.ts` también son similares:

```

import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { StockListComponent } from './stock-list/stock-list.component';
import { AuthGuardService } from 'app/services/auth-guard.service';
import { CreateStockComponent }
  from './create-stock/create-stock.component';
import { CreateStockDeactivateGuardService }
  from 'app/services/create-stock-deactivate-guard.service';
import { StockDetailsComponent }
  from './stock-details/stock-details.component';
import { StockLoadResolverService }
  from 'app/services/stock-load-resolver.service';

const routes: Routes = [

```

```

    { path: 'list', component: StockListComponent,
      canActivate: [AuthGuardService] },
    { path: 'create', component: CreateStockComponent,
      canActivate: [AuthGuardService],
      canDeactivate: [CreateStockDeactivateGuardService] },
    { path: ':code', component: StockDetailsComponent,
      canActivate: [AuthGuardService],
      resolve: { stock: StockLoadResolverService } },
  ];
}

@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule]
})
export class StockRoutingModule { }

```

Very similar to the `UserRoutingModule`, we have simply moved the stock list, create, and details routes to the `StockRoutingModule`. Do note that we dropped the prefix from the paths and just kept it relative to the current module. Other than prefilling the `routes` array, everything else is just the autogenerated code.

Muy similar a `UserRoutingModule`, simplemente hemos movido la lista de acciones, la creación y las rutas detalladas a `StockRoutingModule`. Tenga en cuenta que eliminamos el prefijo de las rutas y simplemente lo mantuvimos en relación con el módulo actual. Aparte de completar previamente la matriz `routes`, todo lo demás es solo el código generado automáticamente.

Our `StockModule` change is also trivial and straightforward:

Nuestro cambio `StockModule` también es trivial y sencillo:

```

import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';

import { StockItemComponent } from './stock-item/stock-item.component';
import { CreateStockComponent } from './create-stock/create-stock.component';
import { StockListComponent } from './stock-list/stock-list.component';
import { StockDetailsComponent } from './stock-details/stock-
details.component';

```

```

import { StockRoutingModule } from './stock-routing.module';
import { FormsModule } from '@angular/forms';

@NgModule({
  imports: [
    CommonModule,
    FormsModule,
    StockRoutingModule
  ],
  declarations: [
    StockDetailsComponent,
    StockItemComponent,
    StockListComponent,
    CreateStockComponent,
  ]
})
export class StockModule { }

```

We import the `FormsModule` along with declaring all the stock-related components. Now let's take a look at the modified `AppModule` first before we go redefine the routes:

Importamos el `FormsModule` junto con declarar todos los componentes relacionados con las acciones. Ahora echemos un vistazo al `AppModule` modificado antes de redefinir las rutas:

```

/** No major changes in imports, skipping for brevity */

@NgModule({
  declarations: [
    AppComponent,
  ],
  imports: [
    BrowserModule,
    HttpClientModule,
    AppRoutingModule,
  ],
  providers: [
    StockService,
    UserService,
    UserStoreService,
    AuthGuardService,
    CreateStockDeactivateGuardService,
    StockLoadResolverService,
  ]
})

```

```

        provide: HTTP_INTERCEPTORS,
        useClass: StockAppInterceptor,
        multi: true,
    }
],
bootstrap: [AppComponent]
})
export class AppModule { }

```

The major change is the declarations array of the NgModule. All the components that we moved into the child modules have been removed from the declarations in the AppModule now. These components will now be loaded if necessary based on the route.

El cambio principal es la matriz declarations del NgModule. Todos los componentes que movimos a los módulos secundarios se han eliminado de las declaraciones en AppModule. Estos componentes ahora se cargarán si es necesario según la ruta.

Now we can finally move to the *app-routes.module.ts* file, which changes as follows:

Ahora finalmente podemos pasar al archivo app-routes.module.ts, que cambia de la siguiente manera:

```

/** Imports omitted for brevity **/


const appRoutes: Routes = [
  { path: '', redirectTo: 'user/login', pathMatch: 'full' },
  { path: 'stock', loadChildren: 'app/stock/stock.module#StockModule' },
  { path: 'user', loadChildren: 'app/user/user.module#UserModule' },
  { path: '**', redirectTo: 'user/register' }
];

@NgModule({
  imports: [
    RouterModule.forRoot(appRoutes),
  ],
  exports: [
    RouterModule
  ],
})
export class AppRoutesModule { }

```

The major change is again only restricted to the `appRoutes` array. Previously, we defined all our routes in this one file. Now, we use the `loadChildren` key to tell Angular that these routes are defined as part of a child module. This also means that our login and register routes have changed from `/login` to `/user/login` and so on, and similar for the stock routes. Make sure you make a pass through the entire application to fix all the routes changed, in particular the following files:

Nuevamente, el cambio principal solo se limita a la matriz `appRoutes`. Anteriormente, definimos todas nuestras rutas en este único archivo. Ahora, usamos la clave `loadChildren` para decirle a Angular que estas rutas están definidas como parte de un módulo secundario. Esto también significa que nuestras rutas de inicio de sesión y registro han cambiado de `/login` a `/user/login` y así sucesivamente, y es similar para las rutas estándar. Asegúrese de revisar toda la aplicación para corregir todas las rutas modificadas, en particular los siguientes archivos:

- `register.component.ts` to redirect after registering
Register.component.ts para redirigir después de registrarse
- `login.component.ts` to redirect after login
login.component.ts para redirigir después de iniciar sesión
- `app.component.html` to fix all the navigation links
app.component.html para arreglar todos los enlaces de navegación

Now, we can run our application (after making sure you start the Node.js server and proxy to it). When you run it, open up the network inspector of your browser, and see the requests getting made. Create/register a user, and then try logging in. Now if you have the network inspector open, you should see something like **Figure 12-2**.

Ahora, podemos ejecutar nuestra aplicación (después de asegurarnos de iniciar el servidor Node.js y utilizarlo como proxy). Cuando lo ejecute, abra el inspector de red de su navegador y vea las solicitudes que se realizan. Cree/registre un usuario y luego intente iniciar sesión. Ahora, si tiene abierto el inspector de red, debería ver algo como la Figura 12-2.

The screenshot shows a web browser window titled "StockMarket". The address bar displays "localhost:4200/stock/list?page=1". Below the header, there are navigation links: "Login", "Register", "Stock List" (which is highlighted in blue), and "Create Stock". The main content area displays three stock cards:

- Test Stock Company (TSC)**
NASDAQ
\$ 85
[Add to Favorite](#)
- Second Stock Company (SSC)**
NSE
\$ 10
[Add to Favorite](#)
- Last Stock Company (LSC)**
NYSE
\$ 876
[Add to Favorite](#)

At the bottom left, there is a link labeled "Next page".

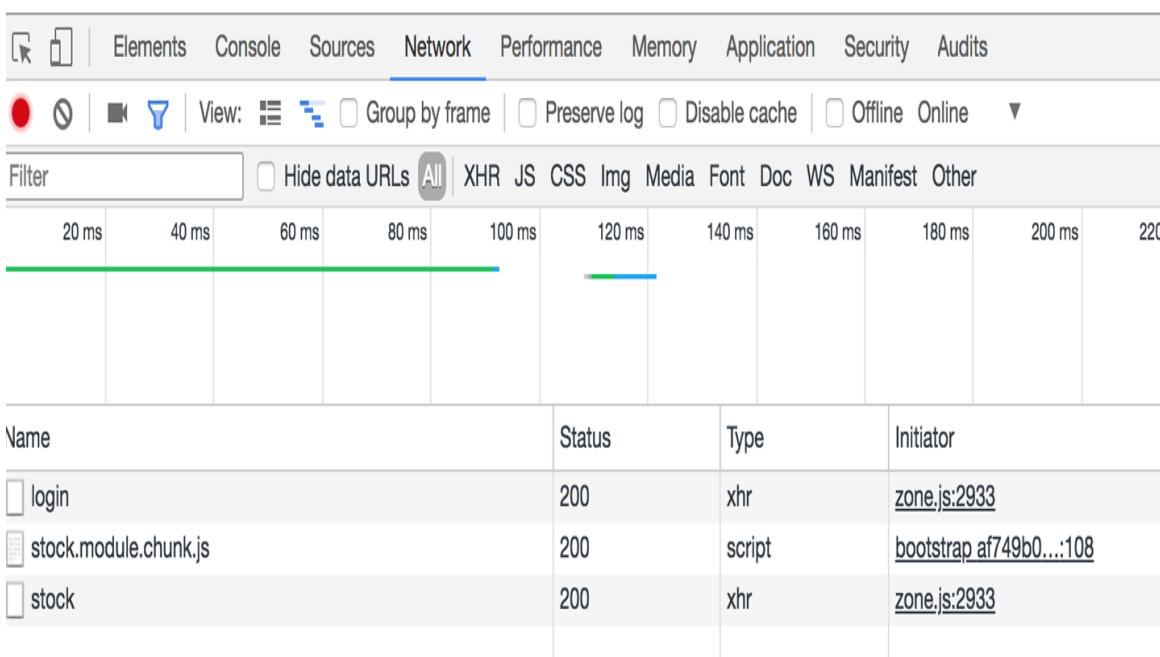


Figure 12-2. Angular lazy loading at work

Figura 12-2. Carga diferida angular en el trabajo

Notice the additional request to load `stock.module.chunk.js` when a login happens. This is lazy loading the StockModule chunk once we log in.

Observe la solicitud adicional para cargar `stock.module.chunk.js` cuando se produce un inicio de sesión. Esto es una carga diferida del fragmento StockModule una vez que iniciamos sesión.

You could extend this and configure it to your liking. You might choose to lazy load only the StockModule, and always load the UserModule, or vice versa. It just adds one more item in your toolbox to use to your liking.

Podrías ampliar esto y configurarlo a tu gusto. Puede optar por cargar de forma diferida solo el StockModule y cargar siempre el UserModule, o viceversa. Simplemente agrega un elemento más a su caja de herramientas para usarlo a su gusto.

The finished example that we created is available in the `chapter12/lazy-loading` folder in the GitHub repository.

El ejemplo terminado que creamos está disponible en la carpeta capítulo12/carga diferida en el repositorio de GitHub.

Lazy loading can really impact performance and speed up the initial load for a very large application with lots of routes and code flows. It also makes a lot of sense when there are routes you know most users will not load or open. And as we saw, it is pretty straightforward to implement and add to your application.

La carga diferida realmente puede afectar el rendimiento y acelerar la carga inicial para una aplicación muy grande con muchas rutas y flujos de código. También tiene mucho sentido cuando hay rutas que sabes que la mayoría de los usuarios no cargarán ni abrirán. Y como

vimos, es bastante sencillo de implementar y agregar a su aplicación.

Server-Side Rendering and Handling SEO

Representación del lado del servidor y manejo de SEO

We will look at one last thing in terms of performance before we wrap up this section. If we consider the life of the first request when we load any Angular (or any Single-Page Application in general) app in our browser, it looks something like this:

Veremos una última cosa en términos de rendimiento antes de concluir esta sección. Si consideramos la vida de la primera solicitud cuando cargamos cualquier aplicación Angular (o cualquier aplicación de una sola página en general) en nuestro navegador, se parece a esto:

1. A request is made with the base path to the server (say www.mytestpage.com).

Se realiza una solicitud con la ruta base al servidor (por ejemplo, www.mytestpage.com).

2. The server serves the *index.html* file for that path.

El servidor sirve el archivo index.html para esa ruta.

3. The browser starts loading the *index.html*, and then for every static file it needs (CSS, JS, etc.), it will make another request to the server for them.

El navegador comienza a cargar index.html y luego, para cada archivo estático que necesita (CSS, JS, etc.), realizará otra solicitud al servidor.

4. Once all the content has been loaded, Angular will bootstrap, parse the route, and load the necessary components.

Una vez que se haya cargado todo el contenido, Angular arrancará, analizará la ruta y cargará los componentes necesarios.

5. The components will then make necessary server calls to fetch the data, and then render them.

Luego, los componentes realizarán las llamadas necesarias al servidor para recuperar los datos y luego procesarlos.

It is only at this point that the final view is rendered to the user. As you can see, there are multiple hops back-and-forth between the client and the server before this view is rendered. Of course, for future routes and navigation, it is only the component and its calls that happen, so the first few hops are skipped in a Single-Page Application for further routes.

Sólo en este punto se presenta la vista final al usuario. Como puede ver, hay varios saltos entre el cliente y el servidor antes de que se represente esta vista. Por supuesto, para rutas y navegación futuras, solo ocurre el componente y sus llamadas, por lo que los primeros saltos se omiten en una aplicación de página única para rutas adicionales.

This back-and-forth nature also makes it difficult to handle when it comes to search engine optimization, as most search engine crawlers will not actually render and execute JavaScript when they try to crawl web pages (for various security reasons). Thus, more often than not, deep links and routes in a Single-Page Application also do not get indexed properly. One option to solve this is to use something like [Pre-render](#), which can render the web application in PhantomJS on the server, and serve the fully rendered HTML for search engines to index.

Esta naturaleza de ida y vuelta también hace que sea difícil de manejar cuando se trata de optimización de motores de búsqueda, ya que la mayoría de los rastreadores de motores de búsqueda en

realidad no procesan ni ejecutan JavaScript cuando intentan rastrear páginas web (por varias razones de seguridad). Por lo tanto, la mayoría de las veces, los enlaces profundos y las rutas en una aplicación de una sola página tampoco se indexan correctamente. Una opción para resolver esto es usar algo como Pre-render, que puede representar la aplicación web en PhantomJS en el servidor y servir el HTML completamente renderizado para que los motores de búsqueda lo indexen.

But with Angular, there is another option: rendering the application server-side. We accomplish this using something known as **Angular Universal**. With this approach, we render the Angular application for serving the initial request on the server itself, and then let Angular bootstrap and take over for the remaining application on the client-side. Thus, we get the best of both worlds, where the initial request does not have the usual back-and-forth, and future requests behave like a Single-Page Application. It also helps in reducing the perceived latency, as the user immediately gets the information he needs while the remaining libraries and framework load.

Pero con Angular, hay otra opción: renderizar la aplicación del lado del servidor. Logramos esto usando algo conocido como Angular Universal. Con este enfoque, renderizamos la aplicación Angular para atender la solicitud inicial en el servidor y luego dejamos que Angular arranque y se haga cargo de la aplicación restante en el lado del cliente. Por lo tanto, obtenemos lo mejor de ambos mundos, donde la solicitud inicial no tiene el habitual ir y venir, y las solicitudes futuras se comportan como una aplicación de una sola página. También ayuda a reducir la latencia percibida, ya que el usuario obtiene inmediatamente la información que necesita mientras se cargan las bibliotecas y el marco restantes.

In this section, we won't go too much into the details of exactly how Angular accomplishes this, but rather focus on what it takes for us to integrate and get it working. Let's see how we might take the Angular application we have been working on so far and make it into

an Angular Universal application that can seamlessly run on both client and server.

En esta sección, no entraremos demasiado en detalles sobre cómo Angular logra esto exactamente, sino que nos centraremos en lo que necesitamos para integrarlo y hacerlo funcionar. Veamos cómo podemos tomar la aplicación Angular en la que hemos estado trabajando hasta ahora y convertirla en una aplicación Angular Universal que pueda ejecutarse sin problemas tanto en el cliente como en el servidor.

WARNING ADVERTENCIA

Angular Universal is in its infancy, so expect to face lots of trouble as you go about integrating it, especially when it comes to working with third-party libraries and components. Even the online guides and tutorials will often be out-of-date, incomplete, or not working as the work is still in progress and prone to sudden changes.

Angular Universal está en su infancia, así que espere enfrentar muchos problemas a medida que lo integra, especialmente cuando se trata de trabajar con bibliotecas y componentes de terceros. Incluso las guías y tutoriales en línea a menudo estarán desactualizados, incompletos o no funcionarán, ya que el trabajo aún está en progreso y es propenso a cambios repentinos.

We will again use the codebase from *chapter11/routing-guard* as the base, and add server-side rendering capability to it. There are fundamentally five new files we will need to add to get Angular Universal working:

Usaremos nuevamente el código base del capítulo 11/routing-guard como base y le agregaremos capacidad de renderizado del lado del servidor. Básicamente, hay cinco archivos nuevos que necesitaremos agregar para que Angular Universal funcione:

- A bootstrapper for the server app (*main.server.ts*)

Un programa previo para la aplicación del servidor
(`main.server.ts`)

- TypeScript configuration for our server (`tsconfig.server.json`)
Configuración de TypeScript para nuestro servidor
(`tsconfig.server.json`)
- An application module for our server-side app
(`app.server.module.ts`)
Un módulo de aplicación para nuestra aplicación del lado del servidor (`app.server.module.ts`)
- An express web server to serve our application code (`server.ts`)
Un servidor web express para servir el código de nuestra aplicación (`server.ts`)
- Webpack server configuration to define how the build happens
(`webpack.server.config.js`)
Configuración del servidor Webpack para definir cómo ocurre la compilación (`webpack.server.config.js`)

In addition, we will be making changes to a few other files as we go along.

Además, realizaremos cambios en algunos otros archivos a medida que avancemos.

Dependencies

Dependencias

To get started, we will rely on a few Angular platform libraries and frameworks. We will need to install the following `npm` packages:

Para comenzar, nos basaremos en algunas bibliotecas y marcos de la plataforma Angular. Necesitaremos instalar los siguientes paquetes `npm`:

@angular/platform-server

@angular/plataforma-servidor

These provide the Angular server-side components to run and render our application code on the server.

Estos proporcionan los componentes del lado del servidor de Angular para ejecutar y representar el código de nuestra aplicación en el servidor.

@nguniversal/module-map-ngfactory-loader

@nguniversal/module-map-ngfactory-loader

In case we use lazy loading for routes, we use the factory loader to lazy load routes in the context of a server-side render.

En caso de que usemos carga diferida para rutas, usamos el cargador de fábrica para cargar rutas de forma diferida en el contexto de una representación del lado del servidor.

@nguniversal/express-engine

@nguniversal/express-motor

An express engine to integrate with Angular Universal and render our application.

Un motor rápido para integrarse con Angular Universal y renderizar nuestra aplicación.

ts-loader

cargador ts

TypeScript loader to transpile our server-side application into JavaScript and run it using Node.js.

Cargador TypeScript para transpilar nuestra aplicación del lado del servidor a JavaScript y ejecutarla usando Node.js.

You can install all of these using the following command:

Puede instalarlos todos usando el siguiente comando:

```
npm install --save @angular/platform-server  
@nguniversal/module-map-ngfactory-loader  
ts-loader@3.5.0 @nguniversal/express-engine
```

This will install and save all of these dependencies to your *package.json*. Note that we have installed a specific version of the *ts-loader*, because of a **bug** with the library and how it interacts with our Angular Universal application.

Esto instalará y guardará todas estas dependencias en su paquete.json. Tenga en cuenta que hemos instalado una versión específica de *ts-loader*, debido a un error con la biblioteca y cómo interactúa con nuestra aplicación Angular Universal.

Making the changes

Haciendo los cambios

The first thing we will do is modify the *AppModule* to have the capability to hook into a rendered server-side Angular application. We accomplish this by replacing the *BrowserModule* import in *src/app/app.module.ts* with the following line:

Lo primero que haremos es modificar *AppModule* para que tenga la capacidad de conectarse a una aplicación Angular renderizada del lado del servidor. Logramos esto reemplazando la importación *BrowserModule* en *src/app/app.module.ts* con la siguiente línea:

```
BrowserModule.withServerTransition({ appId: 'stock-app' }),
```

The *appId* is just for reference and a keyword for Angular to use when it renders server-side styles and the like. You can replace it with anything of your choice. We can also get runtime information about the current platform (whether Angular is running on the

server or the client) and the appId through Angular as well. We can add the following constructor to the AppModule:

El appId es solo como referencia y una palabra clave que Angular usa cuando representa estilos del lado del servidor y similares. Puedes reemplazarlo con cualquier cosa de tu elección. También podemos obtener información de tiempo de ejecución sobre la plataforma actual (si Angular se está ejecutando en el servidor o en el cliente) y appId a través de Angular también. Podemos agregar el siguiente constructor al AppModule:

```
import { NgModule, Inject, PLATFORM_ID, APP_ID } from '@angular/core';
import { isPlatformBrowser, APP_BASE_HREF } from '@angular/common';

@NgModule({
  /** Skipped for brevity */
  providers: [
    /* Skipping common ones for brevity */
    {provide: APP_BASE_HREF, useValue: ''}
  ]
})
export class AppModule {

  constructor(
    @Inject(PLATFORM_ID) private platformId: Object,
    @Inject(APP_ID) private appId: string) {
    const platform = isPlatformBrowser(platformId) ?
      'in the browser' : 'on the server';
    console.log(`Running ${platform} with appId=${appId}`);
  }
}
```

isPlatformBrowser is a useful check that you can use in other contexts as well, to selectively enable/disable certain flows and features in your application. There might be preloading, caching, and other flows you might want to keep only for the browser, and you can use isPlatformBrowser for this.

isPlatformBrowser es una verificación útil que también puede usar en otros contextos para habilitar/deshabilitar selectivamente ciertos flujos y funciones en su aplicación. Es posible que haya flujos de

precarga, almacenamiento en caché y otros que deseé conservar solo para el navegador, y puede usar `isPlatformBrowser` para esto.

The next major change is to the URLs to which we make HTTP calls. In the browser, relative URLs are fine. But in a Universal app, especially on the server side, the HTTP URLs must be absolute for Angular Universal to be able to resolve them correctly. One trick (which we will do here) is to use the `APP_BASE_HREF` token, which we can inject into our services. In the context of our browser, this will be what we have defined it to be, and in the server, it will have the entire URL. Another way to do it might be to again use `isPlatformBrowser` to check and change the URL. So in our browser-specific flow, we set the value of `APP_BASE_HREF` in the main module to be an empty string.

El siguiente cambio importante se refiere a las URL a las que realizamos llamadas HTTP. En el navegador, las URL relativas están bien. Pero en una aplicación Universal, especialmente en el lado del servidor, las URL HTTP deben ser absolutas para que Angular Universal pueda resolverlas correctamente. Un truco (queharemos aquí) es utilizar el token `APP_BASE_HREF`, que podemos inyectar en nuestros servicios. En el contexto de nuestro navegador, esto será lo que hemos definido y en el servidor tendrá la URL completa. Otra forma de hacerlo podría ser usar nuevamente `isPlatformBrowser` para verificar y cambiar la URL. Entonces, en nuestro flujo específico del navegador, configuraremos el valor de `APP_BASE_HREF` en el módulo principal para que sea una cadena vacía.

We will use the `APP_BASE_HREF` trick in the `stock.service.ts` file as follows:

Usaremos el truco `APP_BASE_HREF` en el archivo `stock.service.ts` de la siguiente manera:

```
import { Injectable, Optional, Inject } from '@angular/core';
import { APP_BASE_HREF } from '@angular/common';
```

```

/** Remaining imports skipped for brevity */

@ Injectable()
export class StockService {

    private baseUrl: string;

    constructor(private http: HttpClient,
                private userStore: UserStoreService,
                @Optional() @Inject(APP_BASE_HREF) origin: string) {
        this.baseUrl = `${origin}/api/stock`;
    }

    getStocks(): Observable<Stock[]> {
        return this.http.get<Stock[]>(this.baseUrl);
    }

/** Remaining skipped for brevity */
}

```

We would make the same change in the `user.service.ts` file as well, which we are skipping for brevity. You can always look up the changes in the finished example if you are unsure.

También haríamos el mismo cambio en el archivo `user.service.ts`, que omitiremos por brevedad. Siempre puedes buscar los cambios en el ejemplo terminado si no estás seguro.

Additions for the server side

Adiciones para el lado del servidor

Next, we will look at some of the additions we need to do on the server side for the application to actually run. We will first start with a parallel `AppServerModule` (created as `src/app/app.server.module.ts`) to the `AppModule`, which will be used by the server:

A continuación, veremos algunas de las adiciones que debemos hacer en el lado del servidor para que la aplicación realmente se ejecute. Primero comenzaremos con un `AppServerModule` paralelo

(creado como src/app/app.server.module.ts) al AppModule, que será utilizado por el servidor:

```
import { NgModule } from '@angular/core';
import { ServerModule } from '@angular/platform-server';
import { ModuleMapLoaderModule } from '@nguniversal/module-map-ngfactory-
loader';

import { AppModule } from './app.module';
import { AppComponent } from './app.component';
import { APP_BASE_HREF } from '@angular/common';

@NgModule({
  imports: [
    AppModule,
    ServerModule,
    ModuleMapLoaderModule
  ],
  providers: [
    // Add universal-only providers here
    {provide: APP_BASE_HREF, useValue: 'http://localhost:4000/'}
  ],
  bootstrap: [ AppComponent ],
})
export class AppServerModule {}
```

Notice that we import the original AppModule into our AppServerModule, and then add the ServerModule from Angular along with the ModuleMapLoaderModule to handle any lazy-loaded routes. We still bootstrap the AppComponent. We would set up any Universal-specific providers in the providers section, which would be for services that are only server-specific. In this case, we ensure that the value of APP_BASE_HREF is provided with an absolute path so that our server can actually make the correct requests.

Tenga en cuenta que importamos el AppModule original a nuestro AppServerModule y luego agregamos el ServerModule de Angular junto con el ModuleMapLoaderModule para manejar cualquier ruta con carga diferida. Todavía arrancamos el AppComponent. Configuraríamos cualquier proveedor específico de Universal en la

sección providers, que sería para servicios que son solo específicos del servidor. En este caso, nos aseguramos de que el valor de APP_BASE_HREF reciba una ruta absoluta para que nuestro servidor pueda realizar las solicitudes correctas.

We would also create a parallel *main.server.ts* that will be responsible as the entry point for our server-side Angular application, which would be very straightforward. Create it as *src/main.server.ts* with the following content:

También crearíamos un *main.server.ts* paralelo que será responsable como punto de entrada para nuestra aplicación Angular del lado del servidor, lo cual sería muy sencillo. Créelo como *src/main.server.ts* con el siguiente contenido:

```
export { AppServerModule } from './app/app.server.module';
```

Now we are ready to create our server. For the purpose of this example, we will use a Node.js express server, for which Angular Universal has out-of-the-box integration support. It is not necessary to understand the depths of this server code. Create a *server.ts* file in the main root folder of the application with the following content:

Ahora estamos listos para crear nuestro servidor. Para este ejemplo, usaremos un servidor express Node.js, para el cual Angular Universal tiene soporte de integración listo para usar. No es necesario comprender la profundidad de este código de servidor. Cree un archivo *server.ts* en la carpeta raíz principal de la aplicación con el siguiente contenido:

```
// These are important and needed before anything else
import 'zone.js/dist/zone-node';
import 'reflect-metadata';

import { enableProdMode } from '@angular/core';

import * as express from 'express';
import { join } from 'path';
```

```
import * as proxy from 'http-proxy-middleware';

// Faster server renders w/ Prod mode (dev mode never needed)
enableProdMode();

// Express server
const app = express();

const PORT = process.env.PORT || 4000;
const DIST_FOLDER = join(process.cwd(), 'dist');

// * NOTE :: leave this as require() since this file
// is built Dynamically from webpack
const { AppServerModuleNgFactory, LAZY_MODULE_MAP } =
  require('./dist/server/main.bundle');

// Express Engine
import { ngExpressEngine } from '@nguniversal/express-engine';
// Import module map for lazy loading
import { provideModuleMap } from '@nguniversal/module-map-ngfactory-loader';

app.engine('html', ngExpressEngine({
  bootstrap: AppServerModuleNgFactory,
  providers: [
    provideModuleMap(LAZY_MODULE_MAP)
  ]
}));

app.set('view engine', 'html');
app.set('views', join(DIST_FOLDER, 'browser'));

app.use('/api', proxy({
  target: 'http://localhost:3000',
  changeOrigin: true
}));

// Server static files from /browser
app.get('*.*', express.static(join(DIST_FOLDER, 'browser')));

// All regular routes use the Universal engine
app.get('*', (req, res) => {
  res.render(join(DIST_FOLDER, 'browser', 'index.html'), { req });
});

// Start up the Node server
```

```
app.listen(PORT, () => {
  console.log(`Node server listening on http://localhost:${PORT}`);
});
```

The preceding server is a very simplistic, insecure web server that serves your Angular application, but after rendering it on the server side. Again, *add your security and authorization checks* before you take it to production.

El servidor anterior es un servidor web muy simplista e inseguro que sirve su aplicación Angular, pero después de renderizarla en el lado del servidor. Nuevamente, agregue sus controles de seguridad y autorización antes de llevarlo a producción.

We make a few assumptions so that the whole process is easier for us. Primarily:

Hacemos algunas suposiciones para que todo el proceso nos resulte más fácil. Ante todo:

- The server uses the `ngExpressEngine` to convert all client requests into a server-rendered page. We pass it the `AppServerModule` that we wrote, which acts as the bridge between the server-side rendered application and our web application.

El servidor utiliza `ngExpressEngine` para convertir todas las solicitudes de los clientes en una página renderizada por el servidor. Le pasamos el `AppServerModule` que escribimos, que actúa como puente entre la aplicación renderizada del lado del servidor y nuestra aplicación web.

- We need to figure out what requests are for data, which are for static files, and which are Angular routes.

Necesitamos descubrir qué solicitudes son de datos, cuáles son de archivos estáticos y cuáles son rutas angulares.

- We expect all `/api/*` routes to be API/data routes, and the work to handle that if left incomplete.

Esperamos que todas las rutas `/api/*` sean rutas API/de datos, y el trabajo para manejarlas si se deja incompleto.

- We also expect that any request with an extension (say, `.js` or `.css`) will be for a static file, and serve that as a static file from a predefined folder.

También esperamos que cualquier solicitud con una extensión (por ejemplo, `.js` o `.css`) sea para un archivo estático y lo entregue como un archivo estático desde una carpeta predefinida.

- Finally, any request without an extension is then treated as an Angular route, and uses the `ngExpressEngine` to render the Angular server-side rendered page.

Finalmente, cualquier solicitud sin una extensión se trata como una ruta Angular y utiliza `ngExpressEngine` para representar la página renderizada del lado del servidor Angular.

Configuration

Configuración

Finally, we get to the configuration that pulls all of these together. The first thing is to write a configuration for TypeScript, which we can add as `src/tsconfig.server.json` with the following content:

Finalmente, llegamos a la configuración que reúne todo esto. Lo primero es escribir una configuración para TypeScript, que podemos agregar como `src/tsconfig.server.json` con el siguiente contenido:

```
{
  "extends": "../tsconfig.json",
  "compilerOptions": {
    "outDir": "../out-tsc/app",
    "baseUrl": "./",
    "allowJs": true,
    "skipLibCheck": true,
    "esModuleInterop": true,
    "allowSyntheticDefaultImports": true,
    "strict": true,
    "forceConsistentCasingInFileNames": true,
    "noFallthroughCasesInSwitch": true
  },
  "include": [
    "src/**/*.ts"
  ],
  "exclude": [
    "node_modules",
    "出力/dist"
  ]
}
```

```

    "module": "commonjs",
    "types": []
},
"exclude": [
  "test.ts",
  "**/*spec.ts"
],
"angularCompilerOptions": {
  "entryModule": "app/app.server.module#AppServerModule"
}
}
}

```

We extend the existing `tsconfig.json`, and point it to our new `AppServerModule` as the entry module. Also, the module *must* be set to `commonjs` for the Angular Universal application to work.

Ampliamos el `tsconfig.json` existente y lo señalamos a nuestro nuevo `AppServerModule` como módulo de entrada. Además, el módulo debe estar configurado en `commonjs` para que funcione la aplicación Angular Universal.

Next, we need the Webpack configuration for our server to compile and work. We add it as `webpack.server.config.js` at the root folder level, with the following content:

A continuación, necesitamos la configuración de Webpack para que nuestro servidor se compile y funcione. Lo agregamos como `webpack.server.config.js` en el nivel de la carpeta raíz, con el siguiente contenido:

```

const path = require('path');
const webpack = require('webpack');

module.exports = {
  entry: { server: './server.ts' },
  resolve: { extensions: ['.js', '.ts'] },
  target: 'node',
  // this makes sure we include node_modules and other third-party libraries
  externals: [/(node_modules|main\..*\.\js)/],
  output: {
    path: path.join(__dirname, 'dist'),
    filename: '[name].js'
  }
}

```

```

},
module: {
  rules: [{ test: /\.ts$/, loader: 'ts-loader' }]
},
plugins: [
  new webpack.ContextReplacementPlugin(
    /(.+)?angular(\||\.)core(.+)?/,
    path.join(__dirname, 'src'), // location of your src
    {} // a map of your routes
  ),
  new webpack.ContextReplacementPlugin(
    /(.+)?express(\||\.)(.+)?/,
    path.join(__dirname, 'src'),
    {}
  )
]
};

```

This is mainly so that our Node.js *server.ts* compiles into executable JavaScript code, with some fixes for some bugs in the Angular CLI. It also hooks up the *ts-loader* plug-in we installed so that it can convert our TypeScript into JavaScript correctly.

Esto se debe principalmente a que nuestro servidor Node.js.ts se compila en código JavaScript ejecutable, con algunas correcciones para algunos errores en Angular CLI. También conecta el complemento *ts-loader* que instalamos para que pueda convertir nuestro TypeScript a JavaScript correctamente.

We also need to make a small change to the Angular CLI configuration JSON file (*.angular-cli.json*) to make sure it outputs our application code correctly based on the platform. Add the following configuration as another entry into the *apps* array:

También debemos realizar un pequeño cambio en el archivo JSON de configuración de Angular CLI (*.angular-cli.json*) para asegurarnos de que genere el código de nuestra aplicación correctamente según la plataforma. Agregue la siguiente configuración como otra entrada a la matriz *apps*:

```
{
  "platform": "server",
  "root": "src",
  "outDir": "dist/server",
  "assets": [],
  "index": "index.html",
  "main": "main.server.ts",
  "polyfills": "polyfills.ts",
  "test": "test.ts",
  "tsconfig": "tsconfig.server.json",
  "testTsconfig": "tsconfig.spec.json",
  "prefix": "app",
  "styles": [
    "styles.css"
  ],
  "scripts": [],
  "environmentSource": "environments/environment.ts",
  "environments": {
    "dev": "environments/environment.ts",
    "prod": "environments/environment.prod.ts"
  }
}
```

While you are at it, also change the existing entry and change the `outDir` from `dist` to `dist/browser`. Your combined apps array in `.angular-cli.json` should look something like this:

Mientras lo hace, cambie también la entrada existente y cambie el `outDir` de `dist` a `dist/browser`. Su matriz combinada `apps` en `.angular-cli.json` debería verse así:

```
"apps": [
  {
    "root": "src",
    "outDir": "dist/browser",
    "assets": [
      "assets",
      "favicon.ico"
    ],
    "index": "index.html",
    "main": "main.ts",
    "polyfills": "polyfills.ts",
    "test": "test.ts",
    "tsconfig": "tsconfig.app.json",
    "testTsconfig": "tsconfig.spec.json",
    "prefix": "app"
  }
]
```

```

    "prefix": "app",
    "styles": [
      "styles.css"
    ],
    "scripts": [],
    "environmentSource": "environments/environment.ts",
    "environments": {
      "dev": "environments/environment.ts",
      "prod": "environments/environment.prod.ts"
    }
  },
  {
    "platform": "server",
    "root": "src",
    "outDir": "dist/server",
    "assets": [],
    "index": "index.html",
    "main": "main.server.ts",
    "polyfills": "polyfills.ts",
    "test": "test.ts",
    "tsconfig": "tsconfig.server.json",
    "testTsconfig": "tsconfig.spec.json",
    "prefix": "app",
    "styles": [
      "styles.css"
    ],
    "scripts": [],
    "environmentSource": "environments/environment.ts",
    "environments": {
      "dev": "environments/environment.ts",
      "prod": "environments/environment.prod.ts"
    }
  }
],

```

Once we do this, we can now finally add the executable scripts to our *package.json*. Add the following commands to the *scripts* section in your *package.json*:

Una vez que hagamos esto, finalmente podremos agregar los scripts ejecutables a nuestro paquete.json. Agregue los siguientes comandos a la sección *scripts* en su paquete.json:

```

"build:universal":
  "npm run build:client-and-server-bundles && npm run webpack:server",
"serve:universal": "node dist/server.js",

```

```
"webpack:server": "webpack --config webpack.server.config.js --progress --colors"
```

At this point, we should now be ready to run our Angular Universal application.

En este punto, deberíamos estar listos para ejecutar nuestra aplicación Angular Universal.

Running Angular Universal

Corriendo angular universal

Building our Angular Universal application is now as simple as executing:

Construir nuestra aplicación Angular Universal ahora es tan simple como ejecutar:

```
npm run build:universal
```

This runs and generates the build for both our server- and browser-side Angular applications. It creates two folders in our *dist* folder, one each for *browser* and for *server*.

Esto ejecuta y genera la compilación para nuestras aplicaciones Angular del lado del servidor y del navegador. Crea dos carpetas en nuestra carpeta dist, una para el navegador y otra para el servidor.

Now we can run our Angular application as:

Ahora podemos ejecutar nuestra aplicación Angular como:

```
npm run serve:universal
```

This should now start your application, and allow you to hit the application at <http://localhost:4000>.

Esto ahora debería iniciar su aplicación y permitirle acceder a la aplicación en <http://localhost:4000>.

When you open the URL in your browser, open the Network Inspector and look at the requests being made and the responses. In particular, notice the very first request. With a normal Angular application, you would see the barebones *index.html* being served, which would then load all the relevant source code and trigger Angular.

Cuando abra la URL en su navegador, abra el Inspector de red y observe las solicitudes que se realizan y las respuestas. En particular, observe la primera solicitud. Con una aplicación Angular normal, vería que se sirve el *index.html* básico, que luego cargaría todo el código fuente relevante y activaría Angular.

In this case though, you would see that the very first request itself comes with the route content preloaded, and the template HTML also loaded. It is then that the rest of Angular triggers and starts working in the background. This becomes even more apparent if you throttle the speed to 3G or below, to see the difference in perceived latency in an Angular Universal application versus a normal Angular application.

Sin embargo, en este caso, verá que la primera solicitud viene con el contenido de la ruta precargado y la plantilla HTML también cargada. Es entonces cuando el resto de Angular se activa y comienza a funcionar en segundo plano. Esto se vuelve aún más evidente si acelera la velocidad a 3G o menos, para ver la diferencia en la latencia percibida en una aplicación Angular Universal versus una aplicación Angular normal.

The finished code for this application is available in the *chapter12/server-side-rendering* folder in the GitHub repository.

El código terminado para esta aplicación está disponible en la carpeta *capítulo12/server-side-rendering* en el repositorio de GitHub.

Conclusion

Conclusión

This brings us to the end of our journey of learning Angular, step by step. In this chapter in particular, we covered what it takes to bring the Angular application you have been building so far to production. We covered all the steps from building to deploying, and then went in-depth into various concerns for performance, from caching to prerendering. We also touched upon lazy loading and saw how simple it is to take an Angular application and move it to lazy load certain sections of your application.

Esto nos lleva al final de nuestro viaje de aprendizaje de Angular, paso a paso. En este capítulo en particular, cubrimos lo que se necesita para llevar a producción la aplicación Angular que ha estado creando hasta ahora. Cubrimos todos los pasos, desde la creación hasta la implementación, y luego profundizamos en varias cuestiones relacionadas con el rendimiento, desde el almacenamiento en caché hasta el prerenderizado. También abordamos la carga diferida y vimos lo sencillo que es tomar una aplicación Angular y moverla para cargar de forma diferida ciertas secciones de su aplicación.

That said, we have just begun to scratch the surface of Angular. There are tons of things in Angular itself that we didn't cover in this book, from creating directives and pipes, to advanced concepts. But what we have done is build a solid base for you to build from, and covered about 80%–90% of the common tasks you will have in building any application. From here, you should be able to build pretty complex things, and leverage the official documentation to see you the rest of the way through.

Dicho esto, acabamos de empezar a arañar la superficie de Angular. Hay toneladas de cosas en Angular que no cubrimos en este libro, desde la creación de directivas y canalizaciones hasta conceptos

avanzados. Pero lo que hemos hecho es construir una base sólida para que usted pueda construir y cubrimos entre el 80% y el 90% de las tareas comunes que tendrá al crear cualquier aplicación. A partir de aquí, debería poder crear cosas bastante complejas y aprovechar la documentación oficial para completar el resto del proceso.

Index

Índice

Symbols

Símbolos

(pound sign), prefacing template reference variables, [Control Validity](#)

(signo de libra), variables de referencia de plantilla previas, validez de control

\$event variable, [Alternative to ngModel—Event and Property Binding](#), [ngModel](#)

\$ variable de evento, alternativa a ngModel: enlace de eventos y propiedades, ngModel

() (parentheses), event binding syntax, [Understanding Event Binding](#), [Understanding Event Binding](#)

() (paréntesis), sintaxis de enlace de eventos, Comprensión del enlace de eventos, Comprensión del enlace de eventos

* (asterisk), beginning structural directives, [Built-In Structural Directives](#)

* (asterisco), directivas estructurales iniciales, directivas estructurales integradas

@Component decorator, [Defining a Component](#)

Decorador @Component, Definición de un componente

@Injectable decorator, [Digging into the Example](#)
@Injectable decorador, profundizando en el ejemplo

@NgModule TypeScript annotation, [Main Module—app.module.ts](#)

Anotación @NgModule TypeScript, módulo principal:
app.module.ts

[(ngModel)] banana-in-a-box syntax, [ngModel](#)

[(ngModel)] sintaxis de banana-in-a-box, ngModel

[] (square brackets), property binding syntax, [Understanding Property Binding](#), [Understanding Event Binding](#)

[] (corchetes), sintaxis de enlace de propiedades, Comprensión del enlace de propiedades, Comprensión del enlace de eventos

{{ }} (double curly braces), interpolation syntax, [Root Component—AppComponent](#), [Understanding Data Binding](#)

replacing in components, [Others](#)

{{ }} (dobles llaves), sintaxis de interpolación, componente raíz: componente de aplicación, comprensión del enlace de datos y sustitución de componentes, otros

A

A

Access-Control-Allow-Origin: header, [API/Server Calls and CORS](#)
Access-Control-Allow-Origin: encabezado, API/llamadas al servidor y CORS

ActivatedRoute, [Required Route Params](#), [Navigating in Your Application](#)

[snapshot](#), [Optional Route Params](#)

subscribing to changes on data element, [Preloading Data Using Resolve](#)

Ruta activada, parámetros de ruta requeridos, navegación en la instantánea de su aplicación, parámetros de ruta opcionales, suscripción a cambios en el elemento de datos, precarga de datos usando Resolve

AfterContentChecked interface, [Interfaces and Functions](#)
Interfaz AfterContentChecked, interfaces y funciones

AfterContentInit interface, [Interfaces and Functions](#)
Interfaz AfterContentInit, interfaces y funciones

AfterViewChecked interface, [Interfaces and Functions](#)
Interfaz AfterViewChecked, interfaces y funciones

AfterViewInit interface, [Interfaces and Functions](#)
Interfaz AfterViewInit, interfaces y funciones

ahead-of-time (AOT) compilation, [Production Build](#)

and Build Optimizer, [Ahead-of-Time \(AOT\) Compilation and Build Optimizer](#)

compilación anticipada (AOT), compilación de producción y optimizador de compilación, compilación anticipada (AOT) y optimizador de compilación

Angular

Angular

advantages of, [Why Angular](#)

Ventajas de, ¿Por qué Angular?

and AngularJS, [Introducing Angular](#)
y AngularJS, presentando Angular
core fundamentals, [A Word on Web Application Development Today](#)
fundamentos básicos, unas palabras sobre el desarrollo de aplicaciones web hoy
shift to property and event binding, [Understanding Event Binding](#)
cambiar al enlace de propiedad y evento, Comprensión del enlace de eventos
starting a project, [Starting Your First Angular Project](#)
comenzando un proyecto, comenzando su primer proyecto angular
testing utilities, [Testing and Angular](#)
utilidades de prueba, Testing y Angular
versioning and upgrades, [Why Angular](#)
Versiones y actualizaciones, ¿Por qué Angular?
Angular services, [Angular Services-Exercise](#)
about, [What Are Angular Services?](#)
creating, [Creating Our Own Angular Service-Angular and Dependency Injection](#)
Angular and dependency injection, [Angular and Dependency Injection-Angular and Dependency Injection](#)
examining the code, [Digging into the Example-Digging into the Example](#)

introduction to dependency injection, [An Introduction to Dependency Injection](#)

moving to asynchronous operation, [RxJS and Observables: Moving to Asynchronous Operations-RxJS and Observables: Moving to Asynchronous Operations](#)

service dependencies and interceptors, [Interceptors](#)

unit testing, [Unit Testing Services-Exercise](#)

async services, [Unit Testing Async-Unit Testing Async](#)

HTTP calls, [Unit Testing HTTP-Conclusion](#)

testing components with a service dependency, [Testing Components with a Service Dependency-Testing Components with a Fake Service](#)

Servicios angulares, Servicios angulares-Ejercicioacerca de, ¿Qué son los servicios angulares?crear, crear nuestro propio servicio angular: inyección angular y de dependenciaInyección angular y de dependencia, inyección angular y de dependencia: inyección angular y de dependenciaexaminar el código, profundizar en el ejemplo: profundizar en el ejemplointroducción a la inyección de dependencias, Introducción a la inyección de dependencias, pasar a la operación asincrónica, RxJS y observables: pasar a operaciones asincrónicas: RxJS y observables: pasar a operaciones asincrónicasdependencias e interceptores de servicios, interceptorespruebas unitarias, servicios de pruebas unitarias-ejercicioservicios sincronizados, pruebas unitarias pruebas unitarias asíncronas Llamadas AsyncHTTP, Pruebas unitarias HTTP: prueba de conclusión de componentes con una dependencia del servicio, Prueba de componentes con una

dependencia del servicio: prueba de componentes con un servicio falso

Angular Universal, [Server-Side Rendering and Handling SEO](#)

caveats, [Server-Side Rendering and Handling SEO](#)

files needed to make it work, [Server-Side Rendering and Handling SEO](#)

running, [Running Angular Universal](#)

Angular Universal, Manejo y renderizado del lado del servidor
Advertencias de SEO, Manejo y renderizado del lado del servidor
Archivos SEO necesarios para que funcione, Manejo y
renderizado del lado del servidor SEO
running, Ejecución de
Angular Universal

animations, controlling for components, [Others](#)
animaciones, control de componentes, Otros

annotations, [Main Module—app.module.ts](#)
anotaciones, módulo principal: app.module.ts

API/server calls, [API/Server Calls and CORS](#)
Llamadas API/servidor, Llamadas API/servidor y CORS

AppComponent (example), [Root Component—AppComponent](#)
AppComponent (ejemplo), componente raíz: AppComponent

applications (Angular)
aplicaciones (angulares)

basics of, [Basics of an Angular Application-Root Component—AppComponent](#)

conceptos básicos de, Conceptos básicos de un
componente raíz de aplicación angular: AppComponent

creating on the CLI, [Starting Your First Angular Project](#)
creando en la CLI, comenzando su primer proyecto angular

AppRoutesModule, [Importing the Router Module](#)
AppRoutesModule, Importación del módulo de enrutador

APP_BASE_HREF, setting value, [Making the changes, Additions for the server side](#)

APP_BASE_HREF, valor de configuración, Realización de cambios, Adiciones para el lado del servidor

as operator, [Advanced Observables](#)
como operador, Observables Avanzados

async utility function, [Writing an Angular-Aware Unit Test, Unit Testing Async](#)

fakeAsync function vs., [Unit Testing Async](#)

función de utilidad asíncrona, Escritura de una prueba unitaria con reconocimiento angular, Prueba unitaria Función AsyncfakeAsync versus, Prueba unitaria Async

asynchronous operations in services, [RxJS and Observables: Moving to Asynchronous Operations-RxJS and Observables: Moving to Asynchronous Operations](#)

testing, [Unit Testing Async-Unit Testing Async](#)

operaciones asincrónicas en servicios, RxJS y Observables:
Pasar a Operaciones Asincrónicas-RxJS y Observables: Pasar a Operaciones Asincrónicaspruebas, Pruebas Unitarias Async-Unit Testing Async

asynchronous vs. synchronous forms, [Understanding the Differences](#)

Formas asincrónicas versus sincrónicas, Comprender las diferencias

AsyncPipe, [Advanced Observables](#)

caution with, [Advanced Observables](#)

AsyncPipe, Observables avanzados precaución con, Observables avanzados

attribute directives, [Directives and Components](#)

built-in, [Built-In Attribute Directives](#)-Alternative Class and Style Binding Syntax

alternative class and style binding syntax, [Alternative Class and Style Binding Syntax](#)

NgClass, [NgClass-NgClass](#)

NgStyle, [NgStyle-NgStyle](#)

directivas de atributos, directivas y componentes integrados, directivas de atributos integradas: sintaxis de enlace de estilos y clases alternativas sintaxis de enlace de estilos y clases alternativas NgClass, NgClass-NgClass NgStyle, NgStyle-NgStyle

attributes, case sensitive, [Input](#)

atributos, distingue entre mayúsculas y minúsculas, entrada

authenticated-only routes, [Authenticated-Only Routes](#)-
[Authenticated-Only Routes](#)

rutas solo autenticadas, Rutas solo autenticadas-Rutas solo autenticadas

B

B

banana-in-a-box syntax [()], [ngModel](#)
sintaxis de plátano en caja [()], ngModel

base HTML tag, [Base Href](#), [Handling Deep-Linking](#)
etiqueta HTML base, Href base, manejo de enlaces profundos

beforeEach function, [Writing an Angular-Aware Unit Test](#), [How to Unit Test Services](#)

función beforeEach, Escritura de una prueba unitaria con reconocimiento angular, Cómo realizar pruebas unitarias de servicios

behavior-driven development (BDD) frameworks, [Testing and Angular](#)

Marcos de desarrollo impulsado por el comportamiento (BDD), Testing y Angular.

bootstrap array, [Main Module—app.module.ts](#)
matriz de arranque, módulo principal: app.module.ts

Build Optimizer, [Ahead-of-Time \(AOT\) Compilation and Build Optimizer](#)

Optimizador de compilación, compilación anticipada (AOT) y optimizador de compilación

bundling app and library files, [Production Build](#)
agrupación de archivos de aplicaciones y bibliotecas, compilación de producción

C

C

caching, [Caching](#)

almacenamiento en caché, almacenamiento en caché

CanActivate interface, [Authenticated-Only Routes](#)

Interfaz CanActivate, rutas solo autenticadas

canActivate method, [Authenticated-Only Routes](#)

Método canActivate, rutas solo autenticadas

CanDeactivate guard, [Preventing Unload-Preventing Unload](#)

CanDeactivate guard, Prevención de descarga-Prevención de descarga

case-sensitive attributes, [Input](#)

atributos que distinguen entre mayúsculas y minúsculas,
Entrada

change detection for components, [Change Detection-Change Detection](#)

detección de cambios para componentes, Detección de cambios-Detección de cambios

changeDetection attribute, [Others](#)

atributo changeDetection, Otros

ChangeDetectionStrategy.Default, [Others, Change Detection](#)

ChangeDetectionStrategy.Default, Otros, Detección de cambios

ChangeDetectionStrategy.OnPush, [Others, Change Detection](#)

ChangeDetectionStrategy.OnPush, Otros, Detección de cambios

child routes, [Lazy Loading](#)

rutas secundarias, carga diferida

circular dependency, [Interceptors](#)
dependencia circular, interceptores

class member variables, declaring in component creation,
[Understanding Data Binding](#)
variables de miembros de clase, declaración en la creación de
componentes, comprensión del enlace de datos

classes
clases

CSS
CSS

alternative class binding syntax, [Alternative Class and Style Binding Syntax](#)

sintaxis de enlace de clase alternativa, sintaxis de
enlace de clase y estilo alternativo

applying/removing with NgClass directive, [NgClass-NgClass](#)

aplicar/eliminar con la directiva NgClass, NgClass-NgClass

documentation on TypeScript classes, [Using Models for Cleaner Code](#)

documentación sobre clases de TypeScript, uso de modelos
para código más limpio

generating TypeScript class on CLI, [Using Models for Cleaner Code](#)

generando clase TypeScript en CLI, usando modelos para
código más limpio

cloning HttpRequest, [Interceptors](#)
clonación de HttpRequest, interceptores

code examples from this book, [Using Code Examples](#)

getting the codebase, [Getting the Codebase](#)

ejemplos de código de este libro, Uso de ejemplos de código, obtención del código base, obtención del código base

cold vs. hot observables, [Advanced Observables](#)

Observables fríos versus calientes, Observables avanzados

command-line interface (CLI), [Angular CLI](#)

Angular CLI build wiki, [Deploying an Angular Application](#)

Angular CLI configuration JSON file, [Configuration](#)

generating a skeleton class on, [Using Models for Cleaner Code](#)

generating components and other Angular elements, [Steps in Creating New Components](#)

running an application on, [Running the Application](#)

running unit tests from, [Running the Tests](#)

understanding, [Understanding the Angular CLI](#)

interfaz de línea de comandos (CLI), [Angular CLI](#)Wiki de compilación de Angular CLI, Implementación de una aplicación Angular Archivo JSON de configuración de Angular CLI, Configuración, generación de una clase esqueleto, Uso de modelos para código más limpio, generación de componentes y otros elementos de Angular, Pasos para crear nuevos componentes, ejecución de una aplicación, Ejecución la aplicación: ejecutar pruebas unitarias desde, ejecutar las pruebas, comprender, comprender la CLI angular

components, [Understanding and Using Angular Components](#)-
Exercise

about, [Components—A Recap](#)

and modules, [Components and Modules](#)

Angular service sharing data between, [Digging into the Example](#)

component class, [Root Component—AppComponent](#)

creating, [Creating a Component-Using Our New Component](#)

steps in, [Steps in Creating New Components](#)

data binding, [Understanding Data Binding](#)-[Understanding Data Binding](#)

defining, [Defining a Component-Others](#)

other attributes, [Others](#)

selector, [Selector](#)

style encapsulation, [Style Encapsulation](#)

styles, [Styles](#)

template, [Template](#)

directives and, [Directives and Components](#)

event binding, [Understanding Event Binding](#)-[Understanding Event Binding](#)

input, [Input](#)

input and output, change detection, [Change Detection](#)-[Change Detection](#)

lifecycle, [Component Lifecycle](#)-[Interfaces and Functions](#)

interfaces and functions, [Interfaces and Functions](#)-
[Interfaces and Functions](#)

output, [Output](#)

property binding, [Understanding Property Binding](#)-
[Understanding Property Binding](#)

testing (see testing)

testing components with a service dependency, [Testing Components with a Service Dependency](#)-
[Testing Components with a Fake Service](#)

testing with a fake service, [Testing Components with a Fake Service](#)

testing with a mock service, [Testing Components with a Mock Service](#)

testing with a real service, [Testing Components with a Real Service](#)

using interfaces for cleaner code, [Using Models for Cleaner Code](#)-
[Using Models for Cleaner Code](#)

using our new component, [Using Our New Component](#)

view projection, [View Projection](#)-[View Projection](#)

componentes, comprensión y uso de componentes angulares:
ejercicio acerca de, componentes: un resumen y módulos,
componentes y módulos Servicio angular que comparte datos
entre, profundización en la clase de componente de ejemplo,
componente raíz: creación de componentes de aplicación,
creación de un componente: uso de nuestros nuevos
componentes, pasos para crear nuevos componentes datos

enlace, Comprensión del enlace de datos: comprensión del enlace de datos, definición, definición de un componente, otros atributos, otros selector, encapsulación de estilo de selector, estilos de encapsulación de estilo, plantilla de estilo, directivas de plantilla y, directivas y componentes, enlace de eventos, comprensión del enlace de eventos: comprensión del enlace de eventos, entrada, entrada y salida, detección de cambios, Detección de cambios-Detección de cambiosciclo de vida, Ciclo de vida de componentes-Interfaces y funcionesinterfaces y funciones, Interfaces y funciones-Interfaces y funcionessalida, Vinculación de propiedades de salida, Comprensión del enlace de propiedades-Comprensión del enlace de propiedadesPrueba (ver pruebas)Probar componentes con una dependencia de servicio, Probar componentes con una dependencia de servicio - Probar componentes con un servicio falso, probar con un servicio falso, probar componentes con un servicio falso, probar con un servicio simulado, probar componentes con un servicio simulado, probar con un servicio real, probar componentes con un servicio real, usar interfaces para un código más limpio, usar modelos para un código más limpio. -Usar modelos para un código más limpiousando nuestro nuevo componente, usando nuestro nuevo componentever proyección, Ver proyección-Ver proyección

comprehensive framework (Angular), [Why Angular](#) marco integral (Angular), por qué Angular

control state
estado de control

in reactive forms, [Control State, Validity, and Error Messages](#)

en formas reactivas, estado de control, validez y mensajes de error

in template-driven forms, [Control State-Control Validity](#)
en formularios basados en plantillas, Control de estado-
Control de validez

control validity
validez del control

in reactive forms, [Control State, Validity, and Error Messages](#)

en formas reactivas, estado de control, validez y mensajes de error

in template-driven forms, [Control Validity-Control Validity](#)
en formularios basados en plantillas, Controlar validez-
Controlar validez

Cross-Origin Resource Sharing (CORS), [API/Server Calls and CORS](#)

Intercambio de recursos entre orígenes (CORS), Llamadas API/servidor y CORS

CSS, [Starting Your First Angular Project](#)

alternative class and style binding syntax, [Alternative Class and Style Binding Syntax](#)

and Angular applications, [Understanding Data Binding](#)

app-selector, [Root Component—AppComponent](#)

app.component.css file, [Navigating Within the Application](#)

caching CSS files in Angular applications, [Caching](#)

classes for control state, [Control State, Control State, Validity, and Error Messages](#)

classes, applying/removing with NgClass directive, [NgClass-NgClass](#)

scoping with shadow DOM, [Style Encapsulation](#)
selector for components, [Selector](#)
styles/properties, manipulating with NgStyle directive,
[NgStyle-NgStyle](#)
styleUrls for templates, [Root Component—AppComponent](#)

CSS, Comenzando su primer proyecto angular Sintaxis de enlace de estilos y clases alternativas, Sintaxis de enlace de estilos y clases alternativas y aplicaciones Angular, Comprensión del selector de aplicaciones de enlace de datos, Componente raíz: archivo AppComponentapp.component.css, Navegación dentro de la aplicación Almacenamiento en caché de archivos CSS en aplicaciones Angular, Clases de almacenamiento en caché para estado de control, clases de estado de control, estado de control, validez y mensajes de error, aplicación/eliminación con directiva NgClass, alcance de NgClass-NgClass con DOM oculto, selector de encapsulaciones de estilo para componentes, estilos/propiedades de selector, manipulación con directiva NgStyle, NgStyle-NgStylestyleUrls para plantillas, raíz Componente: componente de aplicación

custom components, [Why Angular](#)
componentes personalizados, ¿Por qué Angular?

D

D

data binding, [Why Angular](#)
Angular form with, [A Complete Form](#)
component title, [Root Component—AppComponent](#)

in components (example), [Understanding Data Binding](#)-
[Understanding Data Binding](#)

two-way, with ngModel, [Setting Up Forms](#)

using form control binding instead of ngModel, [Form Controls](#)

enlace de datos, Por qué Angular forma angular con, Un título de componente de formulario completo, Componente raíz: componente de aplicación en componentes (ejemplo), Comprensión del enlace de datos: comprensión del enlace de datos bidireccional, con ngModel, Configuración de formularios usando el enlace de control de formulario en lugar de ngModel, Controles de formulario

data model in reactive forms, [Form and Data Model](#)-[Form and Data Model](#)

modelo de datos en formularios reactivos, modelo de formulario y datos-formulario y modelo de datos

deactivation of a route, preventing, [Preventing Unload](#)-
[Preloading Data Using Resolve](#)

desactivación de una ruta, impidiendo, Previendo la descarga-precarga de datos usando Resolve

declarations, [Main Module—app.module.ts](#)

adding new components to declarations array, [Steps in Creating New Components](#)

components used within modules, [Components and Modules](#)

declaraciones, módulo principal: app.module.tadición de nuevos

componentes a la matriz de declaraciones, pasos para crear nuevos componentes componentes utilizados dentro de módulos, componentes y módulos

declarations attribute, [Components and Modules](#)
atributos de declaraciones, componentes y módulos

decorators (TypeScript), [Main Module—app.module.ts](#)
decoradores (TypeScript), módulo principal: app.module.ts

deep-linking, handling in production apps, [Handling Deep-Linking](#)

enlaces profundos, manejo en aplicaciones de producción,
Manejo de enlaces profundos

default path (for routes), [Wildcards and Defaults](#)
ruta predeterminada (para rutas), comodines y valores
predeterminados

dependencies
dependencias

for server-side rendering and SEO, [Dependencies](#)
para renderizado del lado del servidor y SEO, Dependencias

handling service dependencies in unit testing, [How to Unit Test Services](#)

manejo de dependencias de servicios en pruebas unitarias,
Cómo realizar pruebas unitarias de servicios

service dependencies and interceptors, [Interceptors](#)
dependencias de servicio e interceptores, Interceptores

dependency injection, [Why Angular](#)

Angular dependency injection system, [Angular and Dependency Injection](#)-Angular and Dependency Injection

Injectable decorator and, [Digging into the Example](#)
introduction to, [An Introduction to Dependency Injection](#)

inyección de dependencia, por qué AngularSistema de inyección de dependencia angular, inyección angular y de dependencia-inyección angular y de dependenciadecorador inyectable y, profundizando en el ejemplo, introducción a, una introducción a la inyección de dependencia

deploying Angular applications, [Deploying an Angular Application](#)

implementar aplicaciones angulares, implementar una aplicación angular

development environment
entorno de desarrollo

getting started with, [Getting Started with Your Development Environment](#)

Angular CLI, [Angular CLI](#)

codebase, getting, [Getting the Codebase](#)

Node.js, [Node.js](#)

TypeScript, [TypeScript](#)

introducción, introducción a su entorno de desarrolloAngular CLI, Angular CLIconicodebase, obtención, obtención del código baseNode.js, Node.jsTypeScript, TypeScript

development mode, running application in, [Running the Application](#)

modo de desarrollo, ejecutar la aplicación en, Ejecutar la aplicación

directives, [Useful Built-In Angular Directives-Exercise](#), Components—A Recap

and components, [Directives and Components](#)

built-in attribute directives, [Built-In Attribute Directives-Alternative Class and Style Binding Syntax](#)

alternative class and style binding syntax, [Alternative Class and Style Binding Syntax](#)

[NgClass](#), [NgClass-NgClass](#)

[NgStyle](#), [NgStyle-NgStyle](#)

built-in structural directives, [Built-In Structural Directives-Multiple Sibling Structural Directives](#)

multiple sibling, [Multiple Sibling Structural Directives](#)

[NgFor](#), [NgFor-NgFor](#)

[NgIf](#), [NgIf-NgIf](#)

[NgSwitchCase](#) and [NgSwitchDefault](#), [NgSwitch](#)

directivas, ejercicio útil de directivas angulares integradas, componentes: resumen de componentes, directivas y componentes directivas de atributos integradas, directivas de atributos integradas: sintaxis de enlace de estilo y clase alternativa sintaxis de enlace de estilo y clase alternativa [NgClass](#), [NgClass-NgStyle](#), [NgStyle-NgStyle](#) directivas estructurales integradas, Directivas estructurales integradas-Directivas estructurales de varios hermanos varios hermanos, Directivas

estructurales de varios hermanos NgFor, NgFor-NgForNgIf, NgIf-NgIfNgSwitchCase y NgSwitchDefault, NgSwitch

DoCheck interface, [Interfaces and Functions](#)

Interfaz DoCheck, Interfaces y Funciones

DOM

DOMINGO

binding DOM properties, [Understanding Property Binding](#)-
[Understanding Property Binding](#)

vincular propiedades DOM, Comprender el enlace de
propiedades-Comprensión del enlace de propiedades

shadow DOM, [Style Encapsulation](#)

sombra DOM, encapsulación de estilo

E

mi

encapsulation attribute, Component decorator, [Style Encapsulation](#)

atributo de encapsulación, decorador de componentes,
encapsulación de estilo

environments, [Different Environments](#)

entornos, diferentes entornos

error handling, HTTP calls, [Making HTTP GET/POST Calls](#)

manejo de errores, llamadas HTTP, realización de llamadas
HTTP GET/POST

error messages, adding to reactive forms, [Control State, Validity, and Error Messages](#)

mensajes de error, agregar a formularios reactivos, estado de
control, validez y mensajes de error

event binding, [Understanding Event Binding](#)-[Understanding Event Binding](#)

input event binding in template-driven forms, [Alternative to ngModel—Event and Property Binding](#)

ngModelChange, [ngModel](#)

registering for events in components, [Output](#)

using in template-driven forms, [Alternative to ngModel—Event and Property Binding](#)

enlace de eventos, comprensión del enlace de eventos:
comprensión del enlace de eventos, enlace de eventos de
entrada en formularios basados en plantillas, alternativa a
ngModel: enlace de eventos y propiedades ngModelChange,
registro de ngModel para eventos en componentes, uso de
salida en formularios basados en plantillas, alternativa a
ngModel: enlace de eventos y propiedades

EventEmitter, [Making HTTP GET/POST Calls](#)

registering as output from a component, [Output](#)

EventEmitter, Realización de llamadas HTTP GET/POST, registro
como salida de un componente, Salida

exportAs attribute, [Others](#)

exportAs atributo, Otros

exporting components, [Others](#)

componentes exportadores, Otros

exports attribute, [Components and Modules](#)

Atributo de exportaciones, componentes y módulos.

F

F

fakeAsync function vs. async function, [Unit Testing Async](#)
Función fakeAsync frente a función asíncrona, prueba unitaria asíncrona

falsy values in JavaScript, [NgClass](#)
valores falsos en JavaScript, NgClass

fixture instance, [Writing an Angular-Aware Unit Test](#)
instancia de acceso, Escritura de una prueba unitaria con reconocimiento angular

fixture.debugElement function, [Writing an Angular-Aware Unit Test](#)

Función fix.debugElement, escritura de una prueba unitaria con reconocimiento angular

fixture.detectChanges function, [Writing an Angular-Aware Unit Test, Testing Component Interactions](#)

Función fixture.detectChanges, escritura de una prueba unitaria con reconocimiento angular, prueba de interacciones de componentes

FormArrays, [FormArrays-FormArrays](#)
FormArrays, FormArrays-FormArrays

FormBuilders, using in reactive forms, [Form Builders](#)
FormBuilders, utilizando en formularios reactivos, Form Builders

FormControls (reactive forms), [Form Controls-Form Controls](#)
checking control state, [Control State, Validity, and Error Messages](#)

Controles de formulario (formularios reactivos), controles de formulario-controles de formulario que verifican el estado del control, estado del control, validez y mensajes de error

FormGroup

Grupos de formularios

using in reactive forms, [Form Groups-Form Groups](#), [Form Builders](#)

adding to/removing from FormArrays, [FormArrays](#)

usar en formularios reactivos, grupos de formularios-grupos de formularios, creadores de formulariosagregar o eliminar de FormArrays, FormArrays

using in template-driven forms, [Working with FormGroup](#)-
[Working with FormGroup](#)

usar en formularios basados en plantillas, Trabajar con
FormGroup-Trabajar con FormGroup

forms, [Working with Template-Driven Forms](#)

(see also reactive forms; template-driven forms)

Create Stock Form (example), [Digging into the Example](#)

formularios, Trabajar con formularios basados en plantillas (consulte también formularios reactivos; formularios basados en plantillas) Crear formulario en stock (ejemplo), profundizar en el ejemplo

FormsModule, importing, [Setting Up Forms](#)

FormsModule, importar, configurar formularios

forRoot method, [Importing the Router Module](#)
Método forRoot, Importación del módulo enrutador

frameworks

marcos

advantages of using, [A Word on Web Application Development Today](#)

ventajas de usar, unas palabras sobre el desarrollo de aplicaciones web hoy

for testing, [Testing and Angular](#)
para pruebas, Testing y Angular

G

GRAMO

GET/POST calls (HTTP), making, [Making HTTP GET/POST Calls](#)-
[Making HTTP GET/POST Calls](#)

Llamadas GET/POST (HTTP), realización, Realización de llamadas HTTP GET/POST-Realización de llamadas HTTP GET/POST

Git repository for this book, [Getting the Codebase](#)
Repositorio Git para este libro, Obteniendo el código base

H

h

headers (HTTP), sending with request, [Options—Headers/Params](#)

encabezados (HTTP), envío con solicitud, Opciones:
encabezados/parámetros

hierarchical dependency injection, [Angular and Dependency Injection](#)

inyección de dependencia jerárquica, inyección angular y de dependencia

hooks in component lifecycle, [Understanding Data Binding](#)
ganchos en el ciclo de vida del componente, Comprensión del enlace de datos

hot vs. cold observables, [Advanced Observables](#)
Observables calientes versus fríos, Observables avanzados

HTML

HTML

and case-sensitive attributes, [Input](#)
y atributos que distinguen entre mayúsculas y minúsculas,
Entrada

native form validation from, [Control State](#)
validación de forma nativa desde, Estado de control

scoping with shadow DOM, [Style Encapsulation](#)
alcance con sombra DOM, encapsulación de estilo

HTML attributes vs. DOM properties, [Understanding Property Binding](#)

Atributos HTML frente a propiedades DOM, comprensión del enlace de propiedades

HTML elements, removing vs. hiding, [NgIf](#)
Elementos HTML, eliminar u ocultar, NgIf

HTTP calls in Angular, [Making HTTP Calls in Angular-Exercise](#)

advanced HTTP, [Advanced HTTP-Interceptors](#)
interceptors, [Interceptors-Interceptors](#)

options, headers/params, Options—Headers/Params-
Options—Headers/Params

options, observe/response type, Options—
Observe/Response Type-Options—Observe/Response
Type

advanced observables, Advanced Observables-Conclusion

HttpClient, Introducing HttpClient-Making HTTP GET/POST
Calls

server setup, Server Setup

making GET/POST calls, Making HTTP GET/POST Calls-
Making HTTP GET/POST Calls

unit testing, Unit Testing HTTP-Conclusion

using HttpClientModule, Using HttpClientModule

Llamadas HTTP en Angular, Realizar llamadas HTTP en Angular-
Ejercicio HTTP avanzado, Interceptores HTTP avanzados,
Interceptores-Opciones de interceptores,
encabezados/parámetros, Opciones—Encabezados/Parámetros-
Opciones—Encabezados/Parámetrosopciones, tipo de
observación/respuesta, Opciones—Tipo de
observación/respuesta -Opciones: Observar/Tipo de
respuestaObservables avanzados, Observables avanzados-
ConclusiónHttpClient, Presentación de HttpClient-Realización de
llamadas HTTP GET/POSTConfiguración del servidor,
Configuración del servidorRealización de llamadas GET/POST,
Realización de llamadas HTTP GET/POST-Realización de
llamadas HTTP GET/POSTPruebas unitarias, Pruebas unitarias
HTTP -ConclusiónUsando HttpClientModule, Usando
HttpClientModule

http service (deprecated), [Introducing HttpClient](#)
Servicio http (obsoleto), Presentación de HttpClient

httpBackend, [Unit Testing HTTP](#)
httpBackend, prueba unitaria HTTP

httpBackend.expectOne, [Unit Testing HTTP](#)
httpBackend.expectOne, prueba unitaria HTTP

HttpClient
Cliente HTTP

introduction to, [Introducing HttpClient-Making HTTP GET/POST Calls](#)

introducción a, Presentación de llamadas HTTP GET/POST para realizar HttpClient

service dependencies and interceptors, [Interceptors](#)
dependencias de servicio e interceptores, Interceptores

httpClient.get, httpClient.post, and httpClient.patch, [Making HTTP GET/POST Calls](#)

httpClient.get, httpClient.post y httpClient.patch, realización de llamadas HTTP GET/POST

HttpClientModule, [Unit Testing HTTP](#)
HttpClientModule, prueba unitaria HTTP

HttpClientTestingModule, [Unit Testing HTTP](#)
HttpClientTestingModule, prueba unitaria HTTP

HttpHandler, [Interceptors](#)
HttpHandler, interceptores

HttpHeaders object, [Options—Headers/Params](#)
Objeto HttpHeaders, Opciones: encabezados/parámetros

HttpInterceptor interface, [Interceptors](#)

Interfaz HttpInterceptor, Interceptores

HttpParams class, [Options—Headers/Params](#)

Clase HttpParams, Opciones: encabezados/parámetros

HttpRequest, [Interceptors](#), Unit Testing HTTP

immutability of, [Interceptors](#)

HttpRequest, Interceptores, Pruebas unitarias

HTTPInmutabilidad de, Interceptores

HttpResponse, [Making HTTP GET/POST Calls](#)

immutability of, [Interceptors](#)

HttpResponse, realización de llamadas HTTP GET/POST,

inmutabilidad de interceptores

HttpService, [An Introduction to Dependency Injection](#)

HttpService, una introducción a la inyección de dependencias

HttpTestingController, [Unit Testing HTTP](#)

HttpTestingController, prueba unitaria HTTP

I

I

imports, [Main Module—app.module.ts](#)

importaciones, módulo principal: app.module.ts

imports attribute, [Components and Modules](#)

Atributo de importaciones, componentes y módulos.

index.html file, [Root HTML—index.html](#)

caching, [Caching](#)

Archivo index.html, HTML raíz: almacenamiento en caché
index.html, almacenamiento en caché

inject function, [How to Unit Test Services](#)
función de inyección, Cómo realizar pruebas unitarias de servicios

injector, getting services from, [Testing Components with a Fake Service](#)

inyector, obtención de servicios, prueba de componentes con un servicio falso

inline styles, [Starting Your First Angular Project](#)
estilos en línea, Comenzando su primer proyecto angular

Input decorator, [Input](#)
Decorador de entrada, Entrada

input element
elemento de entrada

in reactive forms, [Form Controls](#)
en formas reactivas, Controles de formulario

in template-driven forms, [Alternative to ngModel—Event and Property Binding](#)
en formularios basados en plantillas, alternativa a ngModel:
enlace de eventos y propiedades

intercept method, [Interceptors](#)
método de intercepción, interceptores

interceptors, [Interceptors-Interceptors](#)
creating, [Interceptors](#)

service dependencies and, [Interceptors](#)

interceptores, [Interceptores-Creación de interceptores](#),
Dependencias de servicios de interceptores y, [Interceptores](#)
interfaces
interfaces

in component lifecycle, [Interfaces and Functions-Interfaces and Functions](#)

en el ciclo de vida de los componentes, [Interfaces y Funciones-Interfaces y Funciones](#)

using for cleaner code, [Using Models for Cleaner Code-Using Models for Cleaner Code](#)

usando para un código más limpio, Usando modelos para un código más limpio-Usando modelos para un código más limpio

interpolation, [Understanding Data Binding](#)

controlling in components, [Others](#)

interpolación, comprensión del enlace de datos, control en componentes, otros

isPlatformBrowser, [Making the changes](#)

isPlatformBrowser, realizando los cambios

J

j

Jasmine, [Testing and Angular](#)

matchers, documentation on, [An Isolated Unit Test](#)

spies, using to spy on a service, [Testing Components with a Mock Service](#)

writing unit test with, [Writing Unit Tests](#)

Jasmine, Testing y Angularmatchers, documentación sobre An Insulated Unit Testspies, uso para espiar un servicio, Prueba de componentes con un servicio simulado, escritura de pruebas unitarias con, Escritura de pruebas unitarias

JavaScript, [A Word on Web Application Development Today](#)

caching JS files in Angular applications, [Caching](#)

translation of TypeScript code to, [Basics of an Angular Application](#)

truthy and falsy values in, [NgClass](#)

JavaScript, una palabra sobre el desarrollo de aplicaciones web hoy Almacenamiento en caché de archivos JS en aplicaciones Angular, Almacenamiento en caché, traducción de código TypeScript a, Conceptos básicos de una aplicación Angular, valores verdaderos y falsos en, NgClass

just-in-time (JIT) compilation, [Ahead-of-Time \(AOT\) Compilation and Build Optimizer](#)

Compilación justo a tiempo (JIT), compilación anticipada (AOT) y optimizador de compilación.

K

k

Karma, [Testing and Angular](#)

Angular tests running via, in Chrome, [Running the Tests](#)
configuration, [Karma Config](#)
debugging tests using Chrome, [Debugging](#)
running service tests in, [How to Unit Test Services](#)

Karma, Pruebas y AngularPruebas angulares que se ejecutan a través de, en Chrome, Ejecución de la configuración de pruebas, Configuración de Karma, pruebas de depuración usando Chrome, Depuración y ejecución de pruebas de servicio en, Cómo realizar pruebas unitarias de servicios

L

I

lazy loading, [Lazy Loading-Lazy Loading](#)
carga diferida, carga diferida-carga diferida

lifecycle hooks for components, [Understanding Data Binding](#)
ganchos de ciclo de vida para componentes, Comprensión del enlace de datos

LoginComponent, [Navigating in Your Application](#), [Optional Route Params](#)
LoginComponent, navegación en su aplicación, parámetros de ruta opcionales

M

METRO

main.server.ts file, [Additions for the server side](#)

Archivo main.server.ts, Adiciones para el lado del servidor

main.ts file, [The Entry Point—main.ts](#)

Archivo main.ts, el punto de entrada: main.ts

matchers (Jasmine), [An Isolated Unit Test](#)

Matchers (Jasmine), una prueba unitaria aislada

message, displaying to user on stock creation, [Digging into the Example](#)

mensaje que se muestra al usuario sobre la creación de existencias, profundizando en el ejemplo

minification in production builds, [Production Build](#)

minificación en construcciones de producción, Construcción de producción

modules

módulos

components and, [Components and Modules](#)

componentes y, componentes y módulos

issues using a component, checking configuration for,

[Components and Modules](#)

problemas al usar un componente, verificar la configuración de componentes y módulos

N

norte

navigate method (router), [Navigating in Your Application](#)
método de navegación (enrutador), Navegando en su aplicación

ng build --prod command, [Production Build](#)

ng build --prod comando, compilación de producción

ng g guard guards/auth command, [Authenticated-Only Routes](#)
ng g guardias/comando de autenticación, rutas solo
autenticadas

ng g service services/message --module=app command,
[Angular and Dependency Injection](#)

ng g service servicios/mensaje --module=comando de
aplicación, inyección angular y de dependencia

ng generate component stock/stock-item command, [Steps in Creating New Components](#)

ng comando generar stock de componente/artículo de stock,
Pasos para crear nuevos componentes

ng generate module app-routes --flat --module=app command,
[Importing the Router Module](#)

ng generate module app-routes --flat --module=app comando,
Importación del módulo de enrutador

ng help command, [Starting Your First Angular Project](#)

ng comando de ayuda, Comenzando su primer proyecto angular

ng new command, arguments, [Starting Your First Angular Project](#)

ng nuevo comando, argumentos, Comenzando su primer
proyecto angular

ng serve --proxy proxy.config.json command, [API/Server Calls and CORS](#)

ng save --proxy comando proxy.config.json, llamadas API/servidor y CORS

ng serve command, [Running the Application](#)
Comando ng server, Ejecutando la aplicación

ng test command, [Running the Tests](#)
Comando ng test, Ejecución de las pruebas

ng-content element, [View Projection](#)
elemento ng-content, Ver proyección

ng-dirty class, [Control State, Control State, Validity, and Error Messages](#)

Clase ng-dirty, estado de control, estado de control, validez y mensajes de error

ng-invalid class, [Control State, Control State, Validity, and Error Messages](#)

Clase ng-invalid, estado de control, estado de control, validez y mensajes de error

ng-pristine class, [Control State, Control State, Validity, and Error Messages](#)

Clase ng-pristine, estado de control, estado de control, validez y mensajes de error

ng-untouched and ng-touched classes, [Control State, Control State, Validity, and Error Messages](#)

Clases ng-untouched y ng-touched, estado de control, estado de control, validez y mensajes de error

ng-valid class, [Control State, Control State, Validity, and Error Messages](#)

Clase ng-valid, estado de control, estado de control, validez y mensajes de error

ngAfterContentChecked function, [Interfaces and Functions](#)
Función ngAfterContentChecked, interfaces y funciones

ngAfterContentInit function, [Interfaces and Functions](#)
Función ngAfterContentInit, interfaces y funciones

ngAfterViewChecked function, [Interfaces and Functions](#)
Función ngAfterViewChecked, interfaces y funciones

ngAfterViewInit function, [Interfaces and Functions](#)
Función ngAfterViewInit, interfaces y funciones

NgClass directive, [NgClass-NgClass](#)
Directiva NgClass, NgClass-NgClass

ngDoCheck function, [Interfaces and Functions](#)
Función ngDoCheck, Interfaces y Funciones

ngExpressEngine, [Additions for the server side](#)
ngExpressEngine, Adiciones para el lado del servidor

NgFor directive, [NgFor-NgFor](#)

running NgIf with, [Multiple Sibling Structural Directives](#)

using to generate select options in form, [A Complete Form](#)

Directiva NgFor, NgFor-NgFor ejecutar NgIf con directivas estructurales de varios hermanos que se utilizan para generar opciones seleccionadas en el formulario, un formulario completo

ngForm object, [Control Validity](#)
Objeto ngForm, validez del control

NgForOf class, [NgFor](#)
NgForDe clase, NgFor

NgIf directive, [NgIf-NgIf](#)

running with NgFor, [Multiple Sibling Structural Directives](#)

Directiva NgIf, NgIf-NgIf running con NgFor, Directivas estructurales de varios hermanos

ngModel directive, [Setting Up Forms](#)

expanded syntax, use of, [ngModel](#)

two-way data binding in template-driven forms, [ngModel](#)

using event and property binding instead of, [Alternative to ngModel—Event and Property Binding](#)

Directiva ngModel, Configuración de sintaxis ampliada de Forms, uso de, enlace de datos bidireccional de ngModel en formularios basados en plantillas, ngModel que usa enlace de eventos y propiedades en lugar de, Alternativa a ngModel: enlace de eventos y propiedades

ngModelChange event binding, [ngModel](#)
Enlace de evento ngModelChange, ngModel

ngModelGroup directive, [Working with FormGroups](#)
Directiva ngModelGroup, Trabajar con FormGroups

NgModule, attributes affecting components, [Components and Modules](#)

NgModule, atributos que afectan a componentes, Componentes y Módulos

ngOnChanges function, [Interfaces and Functions](#)
Función ngOnChanges, Interfaces y Funciones

ngOnDestroy function, [Interfaces and Functions](#)
Función ngOnDestroy, interfaces y funciones

ngOnInit function, [Understanding Data Binding, Interfaces and Functions, An Isolated Unit Test](#)

Función ngOnInit, comprensión del enlace de datos, interfaces y funciones, una prueba de unidad aislada

NgStyle directive, [NgStyle-NgStyle](#)
Directiva NgStyle, NgStyle-NgStyle

ngSubmit event handler, [A Complete Form, Form Groups](#)
Controlador de eventos ngSubmit, un formulario completo, grupos de formularios

NgSwitch directive, [NgSwitch](#)
Directiva NgSwitch, NgSwitch

NgSwitchCase directive, [NgSwitch](#)
Directiva NgSwitchCase, NgSwitch

NgSwitchDefault directive, [NgSwitch](#)
NgSwitchDirectiva predeterminada, NgSwitch

ngValue directive, [A Complete Form](#)
Directiva ngValue, un formulario completo

Node.js server, [Server Setup](#)
Servidor Node.js, Configuración del servidor

Node.js, installing, [Node.js](#)
Node.js, instalación, Node.js

NPM, [Node.js](#)
scoped packages, Angular CLI

NPM, paquetes Node.js scoped, CLI angular

O

oh

Observable.do operator, [Interceptors](#)

Operador Observable.do, Interceptores

Observable.of operator, [RxJS and Observables: Moving to Asynchronous Operations](#)

Operador Observable.of, RxJS y Observables: paso a operaciones asincrónicas

[observables, RxJS and Observables: Moving to Asynchronous Operations](#)

[RxJS and Observables: Moving to Asynchronous Operations](#)

[Using HttpClientModule](#)

advanced operations with, [Advanced Observables-Conclusion](#)

cold vs. hot observables, [Advanced Observables](#)

importing from RxJS library, [RxJS and Observables: Moving to Asynchronous Operations](#)

subscribing to, [RxJS and Observables: Moving to Asynchronous Operations](#)

subscription to, using instead of ActivatedRoute snapshot, [Optional Route Params](#)

Observables, RxJS y Observables: pasar a operaciones asincrónicas: RxJS y Observables: pasar a operaciones asincrónicas, usar HttpClientModu, operaciones avanzadas con

Observables avanzados: conclusión de observables fríos versus observables calientes, Observables avanzados, importar desde la biblioteca RxJS, RxJS y Observables: pasar a operaciones asincrónicas, suscribirse a, RxJS y observables: pasar a la suscripción a operaciones asincrónicas, utilizando en lugar de la instantánea ActivatedRoute, parámetros de ruta opcionales

observe property, options parameter, [Options—](#)

[Observe/Response Type](#)

propiedad de observación, parámetro de opciones, Opciones: tipo de observación/respuesta

OnChanges interface, [Interfaces and Functions](#)

Interfaz OnChanges, Interfaces y Funciones

OnDestroy interface, [Interfaces and Functions](#)

Interfaz OnDestroy, Interfaces y Funciones

one-way data binding, [Understanding Data Binding](#)

enlace de datos unidireccional, comprensión del enlace de datos

OnInit hook, [Understanding Data Binding](#)

Gancho OnInit, comprensión del enlace de datos

OnInit interface, [Interfaces and Functions](#)

implementing, [Understanding Data Binding](#)

Interfaz OnInit, implementación de interfaces y funciones, comprensión del enlace de datos

onSubmit method, [Form Controls, Form and Data Model](#)

Método onSubmit, controles de formulario, formulario y modelo de datos

options object, HTTP headers and parameters, [Options—](#)

[Headers/Params—Headers/Params](#)

objeto de opciones, encabezados y parámetros HTTP, Opciones —Encabezados/Parámetros-Opciones—Encabezados/Parámetros

options parameter, observe/response type, Options—
Observe/Response Type-Options—Observe/Response Type
parámetro de opciones, tipo de observación/respuesta,
Opciones—Tipo de observación/respuesta-Opciones—Tipo de
observación/respuesta

Output decorator, Output Decorador de salida, Salida

P

PAG

packages, naming convention for, **Angular CLI**
paquetes, convención de nomenclatura para Angular CLI

parameters (HTTP queries), sending, Options—Headers/Params
parámetros (consultas HTTP), envío, Opciones:
encabezados/parámetros

paramMap, Required Route Params
paramMap, parámetros de ruta requeridos

parent/root routes and child routes, **Lazy Loading**
rutas padre/raíz y rutas secundarias, carga diferida

PATCH call (HTTP), making, [Making HTTP GET/POST Calls](#) Llamada PATCH (HTTP), realización, realización de llamadas HTTP GET/POST

path (for routes), Importing the Router Module, Wildcards and Defaults

ruta (para rutas), importación del módulo de enrutador, comodines y valores predeterminados

pathMatch key, **Wildcards and Defaults**

Tecla pathMatch, comodines y valores predeterminados

Pipe, using in ngFor expression, **RxJS and Observables: Moving to Asynchronous Operations**

Pipe, usando en la expresión ngFor, RxJS y Observables: pasando a operaciones asincrónicas

POST calls (HTTP), making, **Making HTTP GET/POST Calls**-
Making HTTP GET/POST Calls

Llamadas POST (HTTP), realización, Realización de llamadas
HTTP GET/POST-Realización de llamadas HTTP GET/POST

Pre-render, **Server-Side Rendering and Handling SEO**

Pre-renderizado, renderizado del lado del servidor y manejo de
SEO

prefix value (pathMatch), **Wildcards and Defaults**
valor de prefijo (pathMatch), comodines y valores
predeterminados

prefixing components, **Starting Your First Angular Project**
prefijando componentes, Comenzando su primer proyecto
angular

preserveWhitespaces attribute, **Others**
atributo preserveWhitespaces, Otros

production mode
modo de producción

running Angular in, **Production Build**

ejecutando Angular en, compilación de producción

running application in, **Running the Application**

ejecutando la aplicación en, Ejecutando la aplicación

productionizing Angular apps, Productionizing an Angular App-Conclusion

API/server calls and CORS, [API/Server Calls and CORS](#)

building for production, [Building for Production](#)-Deploying an Angular Application

ahead-of-time compilation and Build Optimizer, [Ahead-of-Time \(AOT\) Compilation and Build Optimizer](#)

base href tag, [Base Href](#)

deploying applications, [Deploying an Angular Application](#)

ng build --prod command on CLI, [Production Build](#)

caching, [Caching](#)

configurations for different environments, [Different Environments](#)

handling deep-linking, [Handling Deep-Linking](#)

lazy loading, [Lazy Loading](#)-Lazy Loading

server-side rendering and handling SEO, [Server-Side Rendering and Handling SEO](#)-Running Angular Universal

additions for the server side, [Additions for the server side](#)

configuration, [Configuration](#)

dependencies, [Dependencies](#)

making changes to the code, [Making the changes](#)

running Angular Universal, [Running Angular Universal](#)

producción de aplicaciones Angular, producción de una aplicación Angular: conclusión de llamadas API/servidor y CORS, llamadas API/servidor y CORS, construcción para producción, creación para producción: implementación de una aplicación Angular, compilación anticipada y optimizador de compilación anticipado (AOT) Compilación y optimización de compilación, etiqueta href base, implementación de aplicaciones Base Href, implementación de una aplicación angular, comando build --prod en CLI, almacenamiento en caché de compilación de producción, configuraciones de almacenamiento en caché para diferentes entornos, diferentes entornos que manejan enlaces profundos, manejo de enlaces profundos, carga diferida, carga diferida-carga diferida del lado del servidor renderizado y manejo de SEO, renderizado y manejo del lado del servidor SEO-Running Angular Universal Adiciones para el lado del servidor, Adiciones para la configuración del lado del servidor, Dependencias de configuración, Dependenciasrealización de cambios en el código, Realización de cambio ejecución de Angular Universal, Ejecución de Angular Universal

promises, converting observables to, [Advanced Observables](#) promesas, conversión de observables a observables avanzados

property binding, [Understanding Property Binding](#)-
[Understanding Property Binding](#)

Angular shift to, [Understanding Event Binding](#)

class property (example), [Using Models for Cleaner Code](#)
using in template-driven forms, [Alternative to ngModel—Event and Property Binding](#)

value property of input element, [Alternative to ngModel—Event and Property Binding](#)

enlace de propiedad, comprensión del enlace de propiedad: comprensión del enlace de propiedad, cambio angular a, comprensión de la propiedad de clase de enlace de eventos (ejemplo), uso de modelos para un código más limpio en formularios basados en plantillas, alternativa a ngModel: evento y propiedad de valor de enlace de propiedad del elemento de entrada, alternativa a ngModel: evento y vinculación de propiedad

Protractor, [Testing and Angular](#)

Transportador, Pruebas y Angular

providers, [Testing Components with a Fake Service](#)

providers array in Angular modules, [Digging into the Example](#)

Universal-specific, [Additions for the server side](#)

proveedores, Prueba de componentes con una matriz de proveedores de servicios falsos en módulos angulares, Profundización en el ejemplo específico de Universal, Adiciones para el lado del servidor

proxy server, setting up, [Making HTTP GET/POST Calls, API/Server Calls and CORS](#)

servidor proxy, configuración, realización de llamadas HTTP GET/POST, llamadas API/servidor y CORS

Q

q

queryParams object, [Optional Route Params](#), [Optional Route Params](#)

Objeto queryParams, Parámetros de ruta opcionales,
Parámetros de ruta opcionales

R

R

reactive forms, [Working with Reactive Forms-Exercise](#)

differences from template-driven forms, [Understanding the Differences](#)

form controls, [Form Controls-Form Controls](#)

FormArrays, [FormArrays-FormArrays](#)

FormBuilders, [Form Builders](#)

handling form data, [Form Data-Form and Data Model](#)

control state, validity, and error messages, [Control State, Validity, and Error Messages-Control State, Validity, and Error Messages](#)

form and data model, [Form and Data Model-Form and Data Model](#)

formularios reactivos, trabajar con formularios reactivos:
diferencias con ejercicios a partir de formularios basados en plantillas, comprender las diferencias en los controles de

formulario, controles de formulario: controles de formulario, arreglos de formularios, arreglos de formularios-arrays de formularios, constructores de formularios que manejan datos de formularios, mensajes de error, validez y estado de control de formularios y modelos de datos de formularios, Mensajes de estado de control, validez y error-Mensajes de estado de control, validez y errorformulario y modelo de datos, formulario y modelo de datos-formulario y modelo de datos

reactive programming, **Reactive Forms**

programación reactiva, formas reactivas

ReactiveFormsModule, importing, **Form Controls**

ReactiveFormsModule, importación, controles de formulario

RegisterComponent, **Navigating in Your Application**

RegisterComponent, navegando en su aplicación

resolve method, **Preloading Data Using Resolve**

método de resolución, Precarga de datos mediante Resolve

Resolver, preloading data with, **Preloading Data Using Resolve**-
Preloading Data Using Resolve

Resolver, precargando datos con, Precargando datos usando
Resolve-Precargando datos usando Resolve

responseType property, options parameter, **Options—**
Observe/Response Type

Propiedad ResponseType, parámetro de opciones, Opciones:
Observar/Tipo de respuesta

root component, **Root HTML—index.html**

AppComponent (example), **Root Component—**
AppComponent

componente raíz, HTML raíz: index.htmlAppComponent (ejemplo), componente raíz: AppComponent

Router object, [Navigating in Your Application](#)

injecting into LoginComponent and using for redirect, [Navigating in Your Application](#)

navigate method, [Navigating in Your Application](#), [Optional Route Params](#)

Objeto enrutador, Navegar en su aplicación Inyectar en LoginComponent y usarlo para redirigir, Navegar en su aplicación Método de navegación, Navegar en su aplicación, Parámetros de ruta opcionales

routerLink directive, [Navigating Within the Application](#), [Navigating in Your Application](#)

Directiva routerLink, Navegación dentro de la aplicación, Navegación en su aplicación

routerLinkActive directive, [Navigating Within the Application](#), [Navigating in Your Application](#)

directiva routerLinkActive, Navegar dentro de la aplicación, Navegar en su aplicación

RouterModule, importing and setting up, [Importing the Router Module](#)

RouterModule, importación y configuración, Importación del módulo enrutador

RouterOutlet directive, [Displaying the Route Contents](#)

Directiva RouterOutlet, Visualización del contenido de la ruta

routing, [Routing in Angular-Exercise](#)

common tasks in, [Common Routing Requirements](#)-[Optional Route Params](#)

navigating in your application, [Navigating in Your Application](#)

optional route params, [Optional Route Params](#)

required route params, [Required Route Params](#)

directly linking to routes within an application, [Handling Deep-Linking](#)

in lazy loading applications, [Lazy Loading](#)-[Lazy Loading, Additions for the server side](#)

route guards, [Route Guards](#)-[Preloading Data Using Resolve](#)

authenticated-only routes, [Authenticated-Only Routes](#)-[Authenticated-Only Routes](#)

preloading data using Resolver, [Preloading Data Using Resolve](#)-[Preloading Data Using Resolve](#)

preventing unload, [Preventing Unload](#)-[Preventing Unload](#)

setting up Angular routing, [Setting Up Angular Routing](#)-[Wildcards and Defaults](#)

displaying route contents, [Displaying the Route Contents](#)

importing RouterModule, [Importing the Router Module](#)

navigating within the application, [Navigating Within the Application](#)-[Navigating Within the Application](#)

server setup, [Server Setup](#)

starting codebase, [Starting Codebase](#)

wildcards and defaults, [Wildcards and Defaults](#)-
[Wildcards and Defaults](#)

enrutamiento, Enrutamiento en Angular-Ejercicio tareas comunes en, Requisitos de enrutamiento comunes-Parámetros de ruta opcionalesnavegar en su aplicación, Navegar en su aplicaciónparámetros de ruta opcionales, Parámetros de ruta opcionalesparámetros de ruta requeridos, Parámetros de ruta requeridos vinculación directa a rutas dentro de una aplicación, Manejo de enlaces profundos en carga diferida aplicaciones, Lazy Loading-Lazy Loading, Adiciones para los guardias de ruta lateral del servidor, Route Guards-Precarga de datos usando Resolve rutas solo autenticadas, Rutas solo autenticadas-Rutas solo autenticadas precarga de datos usando Resolver, Precarga de datos usando Resolve-Precarga de datos usando Resolveprevención de descarga, Prevención de la descarga-Prevención de la descargaConfiguración del enrutamiento angular, Configuración del enrutamiento angular: comodines y valores predeterminados, visualización del contenido de la ruta, visualización del contenido de la ruta, importación del módulo del enrutador, importación del módulo del enrutador, navegación dentro de la aplicación, navegación dentro de la aplicación, navegación dentro de la configuración del servidor de aplicaciones, configuración del servidor, inicio del código base, inicio Codebase comodines y valores predeterminados, comodines y valores predeterminados-comodines y valores predeterminados

routing modules, generating with ng new command, [Starting Your First Angular Project](#)

módulos de enrutamiento, generando con ng nuevo comando,
Comenzando su primer proyecto angular

running applications, [Running the Application](#)
ejecutar aplicaciones, Ejecutar la aplicación

S

S

scoped packages (NPM), [Angular CLI](#)
paquetes con alcance (NPM), CLI angular

scripts section in package.json, [Configuration](#)
sección de scripts en package.json, Configuración

search engine optimization (SEO), [Server-Side Rendering and Handling SEO-Running Angular Universal](#)

optimización de motores de búsqueda (SEO), procesamiento y
manejo del lado del servidor Ejecución de SEO Angular
Universal

search-as-you-type application (example), [Advanced Observables-Conclusion](#)

aplicación de búsqueda a medida que escribe (ejemplo),
conclusión de observables avanzados

select menu, template-driven form, [A Complete Form](#)
menú de selección, formulario basado en plantilla, un formulario
completo

selector attribute, components, [Selector](#)
atributo selector, componentes, selector

server-side rendering and handling SEO, [Server-Side Rendering and Handling SEO-Running Angular Universal](#)

additions for the server side, [Additions for the server side](#)
configuration, [Configuration](#)
dependencies, [Dependencies](#)
making changes to the code, [Making the changes](#)
running Angular Universal, [Running Angular Universal](#)

renderizado y manejo del lado del servidor SEO, renderizado y
manejo del lado del servidor SEO-Running Angular
Universaladiciones para el lado del servidor, Adiciones para la
configuración del lado del servidor, Dependencias de
configuración, Dependenciasrealizar cambios en el código,
Realizar los cambiosejecutar Angular Universal, Ejecutar Angular
Universal

server.ts file, [Additions for the server side](#)
Archivo server.ts, Adiciones para el lado del servidor
servers
servidores

API/server calls in production apps, [API/Server Calls and](#)
[CORS](#)

Llamadas API/servidor en aplicaciones de producción,
llamadas API/servidor y CORS

setting up frontend server to serve requests by priority,
[Handling Deep-Linking](#)

configurar un servidor frontend para atender solicitudes por
prioridad, manejar enlaces profundos

setup for Angular routing, [Server Setup](#)
configuración para enrutamiento angular, configuración del
servidor

setup for HTTP server, [Server Setup](#)
configuración para el servidor HTTP, Configuración del servidor

service providers (see providers)
proveedores de servicios (ver proveedores)

services, [Angular Services](#)

(see also Angular services)

providing an instance of, [Testing Components with a Fake Service](#)

servicios, servicios angulares (ver también servicios angulares) que proporcionan una instancia de prueba de componentes con un servicio falso

shadow DOM, [Style Encapsulation](#)
sombra DOM, encapsulación de estilo

Single-Page Applications (SPAs), [Introducing Angular](#)
Aplicaciones de una sola página (SPA), introducción a Angular

snapshot (ActivatedRoute), [Optional Route Params](#)
instantánea (ActivatedRoute), parámetros de ruta opcionales

specifications (.spec.ts files), [test.ts](#), [How to Unit Test Services](#)
especificaciones (archivos .spec.ts), test.ts, Cómo realizar pruebas unitarias de servicios

stock-item, generating, [Steps in Creating New Components](#)
artículo en stock, generación, pasos para crear nuevos componentes

structural directives, [Directives and Components](#), [Built-In Structural Directives](#)-[Multiple Sibling Structural Directives](#)

multiple sibling directives, [Multiple Sibling Structural Directives](#)

[NgFor](#), [NgFor-NgFor](#)

[NgIf](#), [NgIf-NgIf](#)

[NgSwitchCase](#) and [NgSwitchDefault](#), [NgSwitch](#)

Directivas estructurales, Directivas y componentes, Directivas estructurales integradas: Directivas estructurales de varios hermanos
Directivas de hermanos múltiples, Directivas estructurales de varios hermanos
[NgFor](#), [NgFor-NgFor](#)
[NgIf](#), [NgIf-NgIf](#)
[NgSwitchCase](#) y [NgSwitchDefault](#), [NgSwitch](#)

style binding
encuadernación de estilo

alternative syntax for, [Alternative Class and Style Binding Syntax](#)

sintaxis alternativa para, sintaxis de enlace de estilo y clase alternativa

documentation on, [Alternative Class and Style Binding Syntax](#)

documentación sobre sintaxis de enlace de estilo y clase alternativa

styles
estilos

defining for components, [Styles](#)

definición de componentes, estilos

encapsulation by Angular, [Style Encapsulation](#)

encapsulación por angular, encapsulación de estilo

styles attribute, [Styles](#)
atributo de estilos, Estilos

styleUrls attribute, [Root Component—AppComponent](#), [Styles](#)
Atributo styleUrls, componente raíz: AppComponent, estilos

synchronous vs. asynchronous forms, [Understanding the Differences](#)

Formas sincrónicas versus asincrónicas, Comprender las diferencias

T

t

template attribute, [Template](#)
atributo de plantilla, Plantilla

template reference variables, [Control Validity](#), [Control Validity](#)
variables de referencia de plantilla, Validez de control, Validez de control

template-driven forms, [Working with Template-Driven Forms-Exercise](#)

complete form (example), [A Complete Form-A Complete Form](#)

control state, [Control State](#)-[Control Validity](#)

control validity, [Control Validity](#)-[Control Validity](#)

differences from reactive forms, [Understanding the Differences](#)

event and property binding as alternative to ngModel,
[Alternative to ngModel—Event and Property Binding](#)

setting up forms, [Setting Up Forms](#)

using ngModel for two-way data binding, [ngModel](#)

working with FormGroups, [Working with FormGroups](#)-
[Working with FormGroups](#)

formularios basados en plantillas, Trabajar con formularios basados en plantillas-Ejercicio de formulario completo (ejemplo), Un formulario completo-Un formulario completo estado de control, Control de estado-Control de validez-Control de validez-Control de diferencias de validez de formularios reactivos, Comprensión de las diferencias Vinculación de eventos y propiedades como alternativa a ngModel, alternativa a ngModel: enlace de eventos y propiedades, configuración de formularios, configuración de formularios usando ngModel para enlace de datos bidireccional, ngModel trabajando con FormGroups, trabajando con FormGroups-Trabajando con FormGroups

templates, [Starting Your First Angular Project](#)

for components, [Template](#)

multiline, defining with backtick operator, [Template](#)

stock service (example), [Digging into the Example](#)

plantillas, Comenzando su primer proyecto angular para componentes, Plantilla multilínea, definición con operador de comillas invertidas, servicio Templatestock (ejemplo), Profundizando en el ejemplo

templateUrl attribute, [Root Component—AppComponent](#), [Template](#)

Atributo templateUrl, componente raíz: AppComponent, plantilla

TestBed, Writing an Angular-Aware Unit Test

configuring for service unit testing, How to Unit Test Services

handling service dependencies in, How to Unit Test Services

TestBed, Escritura de una prueba unitaria con reconocimiento angular, configuración para pruebas unitarias de servicio, Cómo realizar pruebas unitarias de servicios, manejo de dependencias de servicios en, Cómo realizar pruebas unitarias de servicios

testing, Why Angular, Testing Angular Components-Exercise

benefits of unit tests, Why Unit Test?

component interactions, Testing Component Interactions-Testing Component Interactions

debugging tests, Debugging

frameworks and libraries used for, Testing and Angular

running the tests, Running the Tests

setup for, The Test Setup-test.ts

Karma configuration, Karma Config

test.ts file, test.ts

unit testing services, Unit Testing Services-Exercise

asynchronous code, Unit Testing Async-Unit Testing Async

HTTP calls, Unit Testing HTTP-Conclusion

initial setup, How to Unit Test Services

StockService (example), [How to Unit Test Services](#)

testing components with a service dependency, [Testing Components with a Service Dependency](#)-[Testing Components with a Fake Service](#)

writing Angular-aware unit test, [Writing an Angular-Aware Unit Test](#)

writing isolated unit test, [Writing Unit Tests](#)-[Running the Tests](#)

pruebas, Por qué Angular, Prueba de componentes angulares: beneficios del ejercicio de las pruebas unitarias, ¿Por qué las pruebas unitarias? -Configuración test.tsKarma, archivo Karma Configtest.ts, servicios de prueba test.tsunit, Servicios de pruebas unitarias-Ejercicio código sincrónico, Pruebas unitarias Async-Pruebas unitarias AsyncLlamadas HTTP, Configuración inicial de conclusión HTTP de pruebas unitarias, Cómo realizar pruebas unitarias de ServicesStockService (ejemplo), Cómo realizar pruebas unitarias de servicios: probar componentes con una dependencia de servicio, probar componentes con una dependencia de servicio: probar componentes con un servicio falso: escribir pruebas unitarias con reconocimiento de Angular, escribir una prueba unitaria con reconocimiento de Angular, escribir pruebas unitarias aisladas, escribir pruebas unitarias: ejecutar las pruebas

testing utilities (Angular), [Testing and Angular](#), [Writing an Angular-Aware Unit Test](#)

utilidades de prueba (Angular), Testing y Angular, Escritura de una prueba unitaria compatible con Angular

trackBy option, NgFor directive, [NgFor](#)
opción trackBy, directiva NgFor, NgFor

truthy and falsy in JavaScript, [NgClass](#)
Verdadero y falso en JavaScript, NgClass

ts-loader, [Dependencies](#), [Configuration](#)
ts-loader, Dependencias, Configuración

two-way data binding, [Setting Up Forms](#)
enlace de datos bidireccional, configuración de formularios

TypeScript
Mecanografiado

advantage of using, [TypeScript](#)
ventaja de usar TypeScript

declaring a parameter and property simultaneously, [Digging into the Example](#)
declarar un parámetro y una propiedad simultáneamente, profundizando en el ejemplo

decorators, [Main Module—app.module.ts](#)
decoradores, módulo principal: app.module.ts

installing, [TypeScript](#)
instalación, mecanografiado

translation of code into JavaScript, [Basics of an Angular Application](#)
traducción de código a JavaScript, conceptos básicos de una aplicación angular

tsconfig.server.json file, [Configuration](#)
Archivo tsconfig.server.json, Configuración

U

Ud.

uglification of code in production builds, [Production Build](#)
Fealización del código en compilaciones de producción,
Production Build

unit testing, [Writing Unit Tests](#)

(see also testing)

benefits of, [Why Unit Test?](#)

pruebas unitarias, redacción de pruebas unitarias (ver también
pruebas) beneficios de, ¿por qué pruebas unitarias?

URLs

URL

base path for relative URLs in an application, [Base Href](#)
ruta base para URL relativas en una aplicación, Base Href

templateUrl, [Template](#)

plantillaUrl, Plantilla

V

V

validation of forms
validación de formularios

control state in template-driven forms, [Control State-
Control Validity](#)

estado de control en formularios basados en plantillas,
Estado de control-Validez de control

control validity in reactive forms, **Control State, Validity, and Error Messages**

Control de validez en formas reactivas, estado de control, validez y mensajes de error.

control validity in template-driven forms, **Control Validity- Control Validity**

Controlar la validez en formularios basados en plantillas, Controlar la validez-Controlar la validez.

validators in reactive forms, **Form Groups**

validadores en formularios reactivos, Grupos de formularios

view projection, **View Projection-View Projection**

ver proyección, Ver proyección-Ver proyección

view providers, **Others**

ver proveedores, Otros

ViewEncapsulation.Emulated, **Style Encapsulation**

ViewEncapsulation.Emulated, Encapsulación de estilo

ViewEncapsulation.Native, **Style Encapsulation**

ViewEncapsulation.Native, Encapsulación de estilo

ViewEncapsulation.None, **Style Encapsulation**

ViewEncapsulation.Ninguno, encapsulación de estilo

W

W.

web application development, current state of, **A Word on Web Application Development Today**

desarrollo de aplicaciones web, estado actual, Unas palabras sobre el desarrollo de aplicaciones web hoy

webpack.server.config.js file, [Configuration](#)

Archivo webpack.server.config.js, Configuración

whenStabilize function, [Unit Testing Async](#)

Función WhenStabilize, prueba unitaria asíncrona

whitespaces, stripping or preserving in components, [Others](#)
espacios en blanco, eliminación o conservación de componentes, Otros

wildcards in path matches, [Wildcards and Defaults](#)

comodines en coincidencias de ruta, comodines y valores predeterminados

X

X

XHR calls, testing code with, [Unit Testing HTTP](#)

Llamadas XHR, código de prueba con HTTP de prueba unitaria

About the Author

Sobre el Autor

Shyam Seshadri is the CTO at ReStok Ordering Solutions based out of India. He has previously worked at both Amazon and Google as a software engineer, and headed the engineering for Hopscotch, an ecommerce startup based out of Mumbai. He has written two books on Angular, is a polyglot when it comes to programming languages, and enjoys working full stack and developing innovative solutions.

Shyam Seshadri es el CTO de ReStok Ordering Solutions con sede en India. Anteriormente trabajó en Amazon y Google como ingeniero de software y dirigió la ingeniería de Hopscotch, una startup de comercio electrónico con sede en Mumbai. Ha escrito dos libros sobre Angular, es políglota en lo que respecta a lenguajes de programación y le gusta trabajar full stack y desarrollar soluciones innovadoras.

Colophon

Colofón

The animal on the cover of *Angular: Up and Running* is a tub gurnard (*Chelidonichthys lucerna*). This fish is a bottom-dweller found around the Mediterranean and Atlantic coasts of Europe and Africa. Gurnards are also known as sea robins because the swimming motion of their fins resembles that of a flying bird (and one species, *Prionotus carolinus*, has an orange belly). The gurnard makes a unique croaking or grunting noise, generated by squeezing a muscle against its swim bladder—this sound is commonly heard when they are caught by fishermen.

El animal en la portada de *Angular: Up and Running* es un rubio de tina (*Chelidonichthys lucerna*). Este pez habita en el fondo de las costas mediterráneas y atlánticas de Europa y África. Los rubios también se conocen como petirrojos porque el movimiento de natación de sus aletas se asemeja al de un ave voladora (y una especie, *Prionotus carolinus*, tiene el vientre anaranjado). El rubio emite un croar o gruñido único, generado al apretar un músculo contra su vejiga natatoria; este sonido se escucha comúnmente cuando los pescadores lo capturan.

The tub gurnard is the largest gurnard species. It is primarily red in color, with a bony armored head and large blue pectoral fins. These fins contain sensitive feeler-like spines that help the fish explore the seabed for food, such as smaller fish, crustaceans, and carrion.

El rubio de bañera es la especie de rubio más grande. Es principalmente de color rojo, con una cabeza ósea acorazada y grandes aletas pectorales azules. Estas aletas contienen espinas sensibles en forma de antenas que ayudan a los peces a explorar el fondo marino en busca de alimento, como peces más pequeños, crustáceos y carroña.

In the past, the bony gurnards were often considered bycatch and discarded (or used to bait lobster and crab traps). But as other

marine species have become overfished, gurnards are becoming more popular in the seafood industry. Their flesh is firm and holds together well in soups and stews, such as the French bouillabaisse.

En el pasado, los rubios óseos a menudo se consideraban captura incidental y se descartaban (o se usaban como cebo para trampas para langostas y cangrejos). Pero a medida que otras especies marinas se han visto sobreexplotadas, los rubios se están volviendo más populares en la industria pesquera. Su pulpa es firme y se mantiene bien en sopas y guisos, como la bullabesa francesa.

Many of the animals on O'Reilly covers are endangered; all of them are important to the world. To learn more about how you can help, go to animals.oreilly.com.

Muchos de los animales que aparecen en las portadas de O'Reilly están en peligro de extinción; Todos ellos son importantes para el mundo. Para obtener más información sobre cómo puede ayudar, visite animales.oreilly.com.

The cover image is from *Meyers Kleines Lexicon*. The cover fonts are URW Typewriter and Guardian Sans. The text font is Adobe Minion Pro; the heading font is Adobe Myriad Condensed; and the code font is Dalton Maag's Ubuntu Mono.

La imagen de portada es de Meyers Kleines Lexicon. Las fuentes de portada son URW Typewriter y Guardian Sans. La fuente del texto es Adobe Minion Pro; la fuente del título es Adobe Myriad Condensed; y la fuente del código es Ubuntu Mono de Dalton Maag.