Jussi Roine, Olli Jääskeläinen

SharePoint Development with the SharePoint Framework

Design and implement state-of-the-art customizations for SharePoint

Packt>

SharePoint Development with the SharePoint Framework

Design and implement state-of-the-art customizations for SharePoint

Jussi Roine
Olli Jääskeläinen

BIRMINGHAM - MUMBAI

SharePoint Development with the SharePoint Framework

First published: September 2017

Production reference: 1260917

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham
B3 2PB, UK.

ISBN 978-1-78712-143-0

www.packtpub.com

Credits

Authors
Jussi Roine
Olli Jääskeläinen

Reviewer
Bill Ayers

Commissioning Editor
Amarabhab Banerjee

Acquisition Editor
Siddharth Mandal

Content Development Editor
Mohammed Yusuf Imaratwale

Technical Editor
Prashant Mishra

Copy Editor
Safis Editing

Project Coordinator
Ritika Manoj

Proofreader
Safis Editing

Indexer
Rekha Nair

Graphics
Jason Monteiro

Production Coordinator
Shantanu Zagade

About the Authors

Jussi Roine started working with SharePoint with the very first version, SharePoint Portal Server 2001, in late 2000. With a strong infrastructure and development background by then, and also having worked quite a bit on Linux-based solutions, SharePoint fascinated him with its seemingly endless potential and easiness for end users. He continued working with SharePoint through all the versions since, including SharePoint Portal Server 2003, MOSS 2007, and SharePoint 2010, 2013, and 2016. He also enjoyed working with customers when they started moving beyond on-premises to the early cloud-based offerings on BPOS and Office 365, later in 2011.

He has been a Microsoft Most Valuable Professional since 2014, and a Microsoft Certified Trainer since 2006. In 2012 and 2014, he had the honor of attending the Microsoft Certified Master (later Microsoft Certified Solutions Master) program, which he passed among the very selected few in the world at the time. In 2017, Jussi became a Microsoft Regional Director, a member of 150 of the world's top technology visionaries chosen for their community leadership and cross-platform expertise.

Having providing his clients with solutions, architectures, and trusted advisor services for 25 years now, he has also had a chance to dive deeply into Microsoft Azure, which connects and integrates with Office 365 robustly. Over the years, he has written several books on SharePoint, the Office clients, productivity, flexwork, and Microsoft platforms. This book is his first international book aimed at a more diverse audience, and he hopes that this book will help current and future developers working on the Office 365 platform to more easily provide superior solutions for their clients using SharePoint Framework.

Olli Jääskeläinen is a Microsoft MVP, **Microsoft Certified Master** (**MCM**), and **Microsoft Certified Trainer** (**MCT**). Olli has been working in the field for over 15 years. He knows SharePoint inside out and is currently focusing on building productivity solutions using Office 365. Olli works as an architect for Sulava and as an organizer for the Office 365 and SharePoint User Group Finland. He has been giving technical presentations since 2013 at events such as TechDays Finland, European SharePoint Conference, TechTalks Finland, SharePoint Saturdays, and Office 365 Engage.

About the Reviewer

Bill Ayers is a consultant developer and solution architect who has been working with SharePoint since the 2003 version of the product. He is a Microsoft Certified Master and Charter MCSM, and a Microsoft Certified Trainer and Office Development MVP. He specializes in Microsoft Office, SharePoint, Office 365, and mobile solutions, with a particular focus on agile software development practices. He has over 25 years of experience in the software industry and speaks regularly at international conferences and user groups. He is also a moderator on `SharePoint.StackExchange.com`, and blogs occasionally at `http://SPDoctor.com/`. He is based in Sheffield, UK.

www.PacktPub.com

For support files and downloads related to your book, please visit `www.PacktPub.com`.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at `www.PacktPub.com` and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at `service@packtpub.com` for more details.

At `www.PacktPub.com`, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.

`https://www.packtpub.com/mapt`

Get the most in-demand software skills with Mapt. Mapt gives you full access to all Packt books and video courses, as well as industry-leading tools to help you plan your personal development and advance your career.

Why subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print, and bookmark content
- On demand and accessible via a web browser

Customer Feedback

Thanks for purchasing this Packt book. At Packt, quality is at the heart of our editorial process. To help us improve, please leave us an honest review on this book's Amazon page at `https://www.amazon.com/dp/1787121437`.

If you'd like to join our team of regular reviewers, you can e-mail us at `customerreviews@packtpub.com`. We award our regular reviewers with free eBooks and videos in exchange for their valuable feedback. Help us be relentless in improving our products!

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Preface

SharePoint Framework is the new, modern, and fresh approach for implementing customizations in SharePoint and SharePoint Online. It's a client-side approach to a server-side problem, allowing developers already familiar with JavaScript, HTML, and CSS to implement lightweight custom functionality for SharePoint.

In the past, SharePoint-based development has been challenging, complex, poorly documented and full of changing directions from Microsoft.

Originally, in early 2000, the guidance, was not to modify or customize SharePoint. Microsoft was saying at the time not to touch anything and simply to use SharePoint as a portal service that hosts your content and documents. With newer versions of SharePoint for on-premises deployments, Microsoft initially introduced a somewhat home-brewed approach to implementing custom web parts (widgets on a page) and UI elements. This included legacy `.bat` files, weird file formats, numerous modifications to XML files with minimal documentation and a cryptic approach to provisioning your solutions between different environments.

Between then and now we've also had different models for developing solutions for SharePoint in the cloud; this included a better approach to implementing features without deploying server-side code, but by the time this was made available developers were already familiar with Microsoft .NET-based customizations on the server-side. Giving a replacement client-side model that had strict rules and security restrictions did not go down easily for seasoned SharePoint developers.

Finally, now with the SharePoint Framework, we have a documented framework for implementing UI elements, custom web parts, and features for SharePoint 2016 in on-premises and SharePoint Online in Office 365. It is still early days, but we're already seeing good uptake from developers who can now finally and confidently migrate their existing code to the cloud while simultaneously switching from .NET-based development languages to TypeScript, which is a superset of JavaScript.

This book will take you from a Hello World-implementation in SharePoint Framework to implementing enterprise-ready functionality with ease. We'll walk you through the essential knowledge you need in order to survive in a modern project that aims to use SharePoint Framework as the core development model. In addition, the necessary tools are explained, as well as debugging and troubleshooting. Obviously, you might still be using other JavaScript frameworks, such as jQuery, AngularJS, and React so we have guidance on that too – because SharePoint Framework does allow and support you to use other frameworks while implementing SharePoint functionality.

This book will provide an invaluable resource for all those seeking to use and learn the SharePoint Framework as their development model for SharePoint.

What this book covers

`Chapter 1`, *Introducing SharePoint Online for Developers*, will give you a short history lesson to understand the limitations of the previous development models for SharePoint. You will begin to see how and where SharePoint Framework differs the most and why it matters.

`Chapter 2`, *Developing Solutions for SharePoint*, will introduce you to SharePoint Framework in more detail. It covers the types of project you can implement and reveals how to code and start using the necessary tools quickly.

`Chapter 3`, *Getting Started with SharePoint Framework*, walks you through installing the toolchain and verifying everything is installed correctly. You'll learn how project files are laid out and where to find what.

`Chapter 4`, *Building Your First Web Part*, you will code your own SharePoint Framework-based web part and deploy it locally for testing and in SharePoint for real use.

`Chapter 5`, *Using Visual Studio Code and Other Editors*, shows you how to get started with Visual Studio Code, the free editor from Microsoft that fully supports SharePoint Framework projects. You will also learn how to use Visual Studio 2015/2017 if you are more familiar with them.

`Chapter 6`, *Packaging and Deploying Solutions*, once development is done, developers need to package and deploy their code in an orderly and managed fashion. You'll learn about the tools to aid you in this task and how to manually deploy your code to SharePoint.

`Chapter 7`, *Working with SharePoint Content*, is the next logical step when implementing solutions with SharePoint Framework. You'll frequently need to access SharePoint-hosted data such as lists and documents. You will also learn using mock data to quickly get your code working before further testing against live data.

`Chapter 8`, *Working with the Web Part Property Pane*, shows you how to create solutions that allow parameters and values from user input, such as settings data for a web part. This way your code can run dynamically regardless of where it is deployed.

`Chapter 9`, *Using React and Office UI Fabric React Components*, explains that Microsoft realized early on that developers prefer other frameworks, such as React, and might need to provide a unified UI that implements Microsoft's design language for SharePoint and Office. With React and Office UI Fabric React you'll learn how to combine these two frameworks with SharePoint Framework projects.

`Chapter 10`, *Working with Other JavaScript Frameworks*, will teach you how to use other popular frameworks, such as jQuery and Angular, to implement your solutions while still using SharePoint Framework

`Chapter 11`, *Troubleshooting and Debugging SharePoint Framework Solutions*, discusses efficient troubleshooting techniques to help you find common problems you might encounter. Debugging a SharePoint Framework solution is slightly different from traditional server-side debugging, and this chapter will teach you how to debug your code.

`Chapter 12`, *SharePoint APIs and Microsoft Graph*, will give you a run-down of the SharePoint APIs and how they differ from each other. You'll learn how to access the APIs, and also how to employ Microsoft Graph, the fabric that provides knowledge for most things in Office 365 and Azure Active Directory.

`Chapter 13`, *The Future of SharePoint Customizations*, provides a glimpse into the future and what SharePoint Framework means for developers from now on. As development models come and go, we give you several ideas how developers should approach this major change.

What you need for this book

- SharePoint 2016 Farm or SharePoint Online tenant (as part of an Office 365 subscription)
- Visual Studio Code or Visual Studio 2015/2017
- Web browser (Chrome or Internet Explorer or Firefox)
- Internet access

Who this book is for

This book is for developers who want to start developing solutions for SharePoint 2016 and/or SharePoint Online using SharePoint Framework. It is also for developers who are already familiar with SharePoint's development models of the past and who are ready to start using SharePoint Framework for their future projects. Though SharePoint experience is not required, you should have a general understanding of the capabilities that SharePoint provides. We'll provide a brief overview of SharePoint as well. It is also helpful to have an understanding of JavaScript-based web development.

Conventions

In this book, you will find a number of text styles that distinguish between different kinds of information. Here are some examples of these styles and an explanation of their meaning.

Code words in the text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "Create another file, called `MockSharePointClient.ts`, in the same folder that the web part is located."

A block of code is set as follows:

```
import { Environment, EnvironmentType } from '@microsoft/sp-core-library';
import { ISPListItem } from "./ISPListItem";
import MockSharePointClient from "./MockSharePointClient";
```

Any command-line input or output is written as follows:

```
gulp serve
```

New terms and **important words** are shown in bold. Words that you see on the screen, for example, in menus or dialog boxes, appear in the text like this: "Write in **Team site name**, for example, SPFX-testing."

Warnings or important notes appear in a box like this.

Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book-what you liked or disliked. Reader feedback is important to us as it helps us develop titles that you will really get the most out of.

To send us general feedback, simply e-mail `feedback@packtpub.com`, and mention the book's title in the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide at `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for this book from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

You can download the code files by following these steps:

1. Log in or register to our website using your e-mail address and password.
2. Hover the mouse pointer on the **SUPPORT** tab at the top.
3. Click on **Code Downloads & Errata**.
4. Enter the name of the book in the **Search** box.

5. Select the book for which you're looking to download the code files.
6. Choose from the drop-down menu where you purchased this book from.
7. Click on **Code Download**.

You can also download the code files by clicking on the **Code Files** button on the book's web page at the Packt Publishing website. This page can be accessed by entering the book's name in the **Search** box. Please note that you need to be logged in to your Packt account.

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

- WinRAR / 7-Zip for Windows
- Zipeg / iZip / UnRarX for Mac
- 7-Zip / PeaZip for Linux

The code bundle for the book is also hosted on GitHub at `https://github.com/PacktPublishing/SharePoint-Development-with-the-SharePoint-Framework`. We also have other code bundles from our rich catalog of books and videos available at `https://github.com/PacktPublishing/`. Check them out!

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books-maybe a mistake in the text or the code-we would be grateful if you could report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/submit-errata`, selecting your book, clicking on the **Errata Submission Form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website or added to any list of existing errata under the Errata section of that title.

To view the previously submitted errata, go to `https://www.packtpub.com/books/content/support` and enter the name of the book in the search field. The required information will appear under the **Errata** section.

Piracy

Piracy of copyrighted material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works in any form on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors and our ability to bring you valuable content.

Questions

If you have a problem with any aspect of this book, you can contact us at `questions@packtpub.com`, and we will do our best to address the problem.

1
Introducing SharePoint Online for Developers

SharePoint Online is a core part of Office 365, which is a **Software as a Service** (**SaaS**) cloud service from Microsoft. **SharePoint Online** (**SPO**) is one of the services in Office 365 that companies and organizations typically purchase when they move parts or all of their infrastructure and services to a public cloud service.

In this chapter, we'll take a lap around Office 365 and SharePoint Online, with a strong focus on development models and features that are relevant to you as a developer. In addition we'll provision a new SharePoint site collection that will aid us in testing code we'll write in upcoming chapters. At the end of this chapter, you'll have a good overview of SharePoint Online and the development models that have been, and some of which still are, available to you in on-premises SharePoint and SharePoint Online.

What is SharePoint Online?

SharePoint Online is, depending, who you ask, a development platform, collaboration service, file management system, and intranet service. It's the logical successor and partially a replacement for SharePoint, the on-premises collaboration and productivity platform. It also (partially or fully depending on the business case) replaces file shares, email attachments (through Outlook's support for OneDrive for Business, which is technically part of SharePoint/SharePoint Online), messaging boards, and similar needs for intra-organization or cross-organizational collaboration.

SharePoint Online is a collection of services bundled together, and these are

- SharePoint team sites
- SharePoint publishing sites
- Search
- User profiles
- InfoPath Forms Service for rich and fillable online forms
- **Business Connectivity Services** (**BCS**) for integrated backend data to SharePoint
- SharePoint add-ins (formerly apps)

In marketing terms, OneDrive for Business is a separate service, but it shares a lot of the same thinking, vision, and in some parts, APIs with SharePoint Online.

Depending on who is accessing SharePoint Online, it can act as a simple team site offering a common storage for documents (typically Office files, such as Word and Excel documents), a messaging board, a blog, and a place to store organizational data such as software licensing information or employee contact information.

SharePoint Online supports accessing content through a web interface, through Office clients and APIs. In some scenarios, content can be accessed through a mapped network drive using WebDAV but this is more or less a legacy way of accessing documents and files stored in SharePoint.

In the following sections, we'll walk you through the essential concepts of SharePoint Online, on a level that we feel is relevant for any developer aiming to work with SharePoint Online.

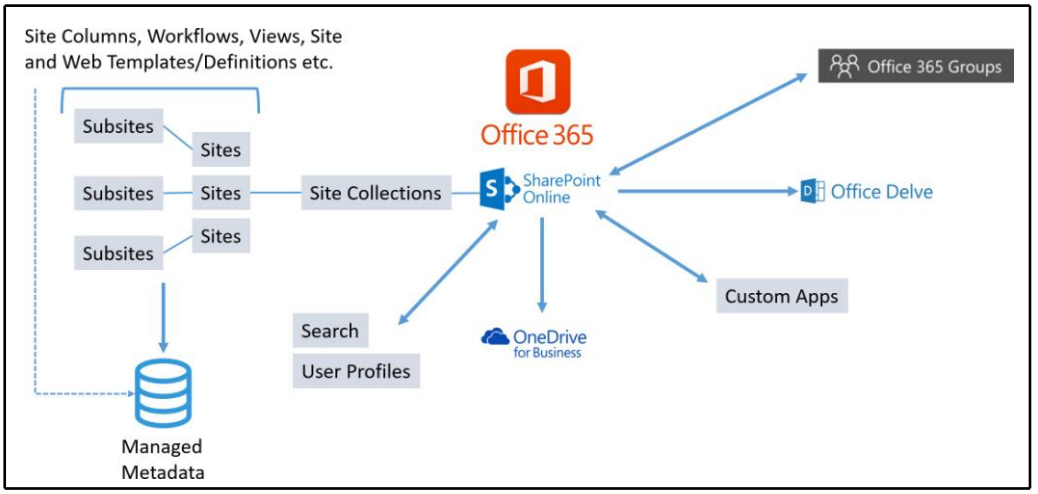
SharePoint sites and site collections

Each SharePoint site is a separate place for storing content. Each site belongs to not more than one site collection. Think of SharePoint sites as folders, and site collections as a drive letter or hard drive partitions. The analogy is slightly lacking, but for now, it's sufficient for relaying the essential concepts of SharePoint Online.

Organizations create one or more site collections and use site collections typically as security boundaries. One site collection could be the intranet, another could be an extranet for specific partner companies or customers. A third one could act as the Board of Directors' secure site collection for storing highly sensitive documents. Each site collection can hold one or more sites, but they always have one site, which will be the root site of a given site collection.

Sites are the common building block for SharePoint Online services, such as intranet and extranet. You could build a very nice intranet using just one site collection, and then using just one site within the site collection. In the real world, however, you would typically need to structure your content and data among multiple sites, and possibly multiple site collections.

In the following illustration, you can see a high-level overview of what SharePoint Online is. On the left-most side, you see site collections, which are made up of sites and possible subsites. Each site can then hold relevant configuration data and SharePoint artefacts such as document libraries and lists.

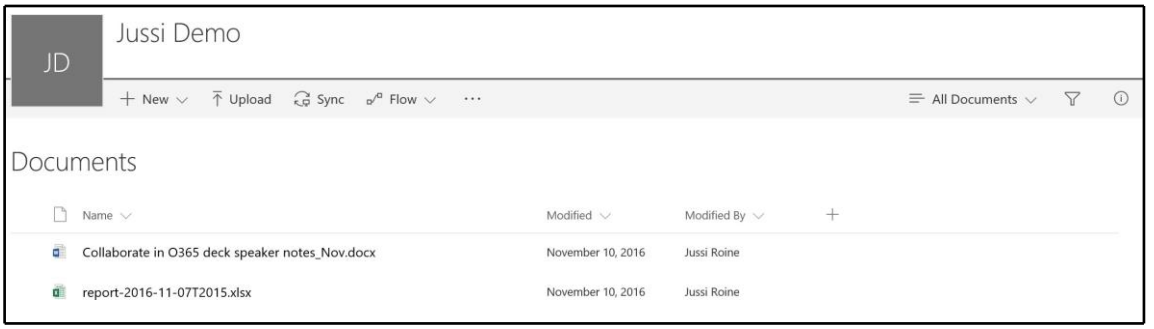


SharePoint document libraries

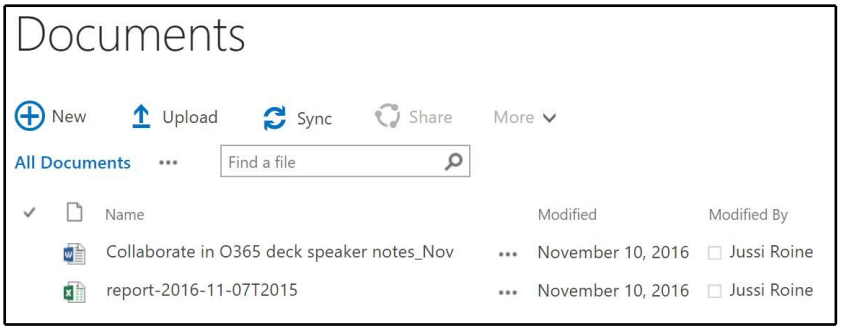
SharePoint sites store information, which requires different types of containers. One type of container is a document library. A document library can be used to store files, including Office documents, text files, media files, and similar content. Most SharePoint sites have, by default, one or more document libraries provisioned and you are free to create more as needed.

Document libraries can either have a modern user experience or a classic user experience. This is due to the changes Microsoft is rolling out over several months (or sometimes even years), which in turn allows for customers to decide when to deploy new and drastic changes, such as a new user experience for something as central as a document library.

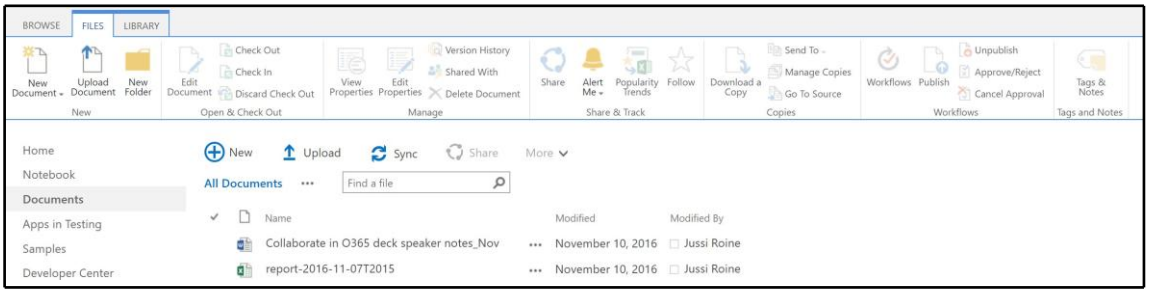
The modern user experience is fully responsive and has a fresh look. All essential tooling is available in a simple toolbar and the ribbon interface from the Office 2007/2010 era is gone:



The classic experience is still very much in use; it is supported and, in certain scenarios, works better. This holds especially true for situations where users have used SharePoint Online for several years and are very accustomed to the classic experience:



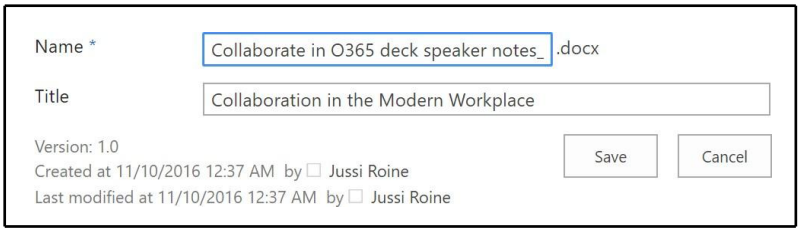
Both views relay almost identical information, but the classic experience also exposes the ribbon UI, which in turns hides (or shows, depending on context) a lot of additional tooling:



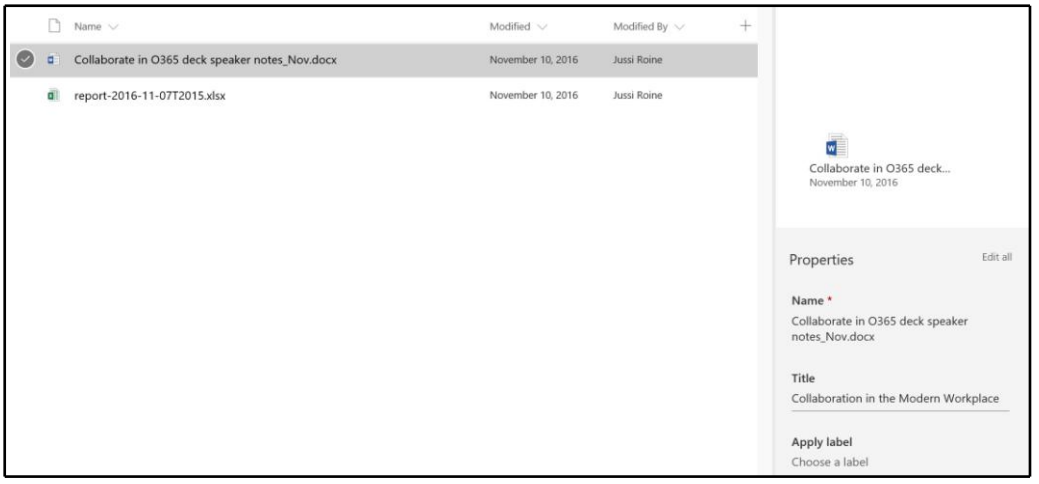
One of the challenges for developers with the ribbon UI has always been the cumbersome approach to modifying, hiding, or adding custom buttons in the ribbon. While it's possible, it's certainly not a task customers would request initially. Users are familiar with the default buttons, and changing or disabling essential buttons for something else will most certainly confuse users.

Each document in a document library holds metadata. Metadata is a larger topic, and not in the scope of this book, suffice to say that metadata in SharePoint Online document libraries is used to reveal and store information about files, and data in general. Metadata can be used to search files and different views on data can be shown, based on metadata filtering, sorting, and selections.

The classic view for modifying metadata is very bare bones but also easy to use:



The modern view for modifying metadata is more modern, but might involve a little learning curve for users accustomed to the classic view:

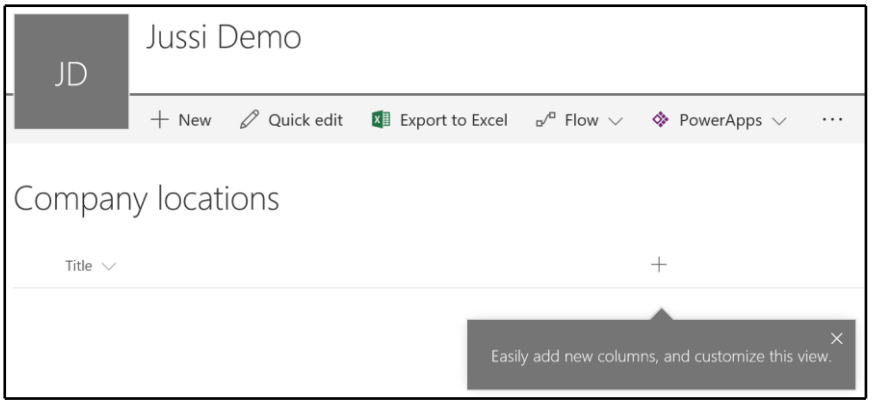


Notice how metadata in the modern view is shown upon selection of a file, while in the classic view the view is completely different and the context of other files is hidden.

SharePoint lists

Besides document libraries, SharePoint sites can hold lists. Technically, lists are identical to document libraries but rather than storing binary files, they store other data such as text, numbers, selections, or even attachment files. Lists are typically used to create ad-hoc data structures to hold data that can be sorted, filtered, and queried. Document libraries often act as a file store and lists act as a data or information store without really storing physical files.

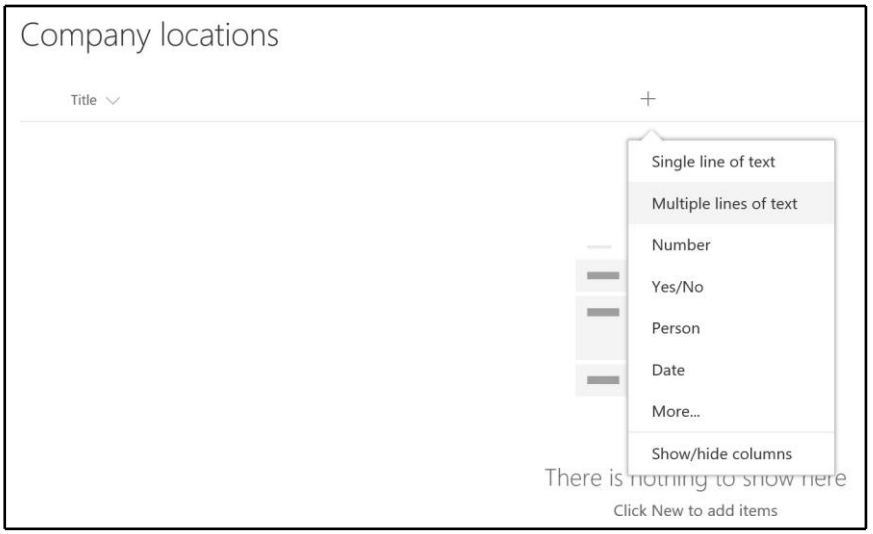
While document libraries are just that: document libraries, lists come in all sizes and varieties. The common base list is a custom list, which is essentially an empty list with just a few data columns as a template. Other list templates include contact lists (predefined person data columns), links (URLs and descriptive text columns), and announcements (news-style items with headlines and summary text), to mention a few:



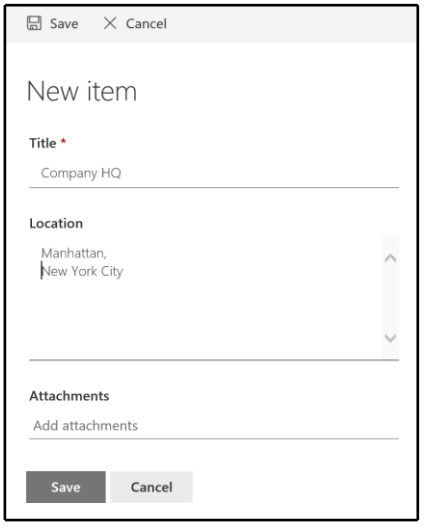
By creating a custom list you can easily build a data store for any kind of data your application might require at a later time. The list can be queried similarly to how you would query a relational database table. It's a graphical representation of an SQL table, with the addition of built-in functionality such as graphical views, sorting, filtering, and integration to other Office 365 services.

By adding new columns (metadata) to a list, you effectively provide additional ways to store data. The **Title** column is the base column, which you can later rename if you choose to.

To add a new column, simply define the datatype and other properties and it will automatically be added to the default view:



When you add a new column of the datatype **Multiple lines of text**, you effectively create a new text box that can be filled out when adding a new line to your custom list:



This data can now be saved and is immediately visible and accessible on the list:

It's crucial to understand that you have a built-in, scalable, secure, and ready-to-use platform for storing data within SharePoint lists and document libraries. Instead of always designing, provisioning, and maintaining a separate SQL database with custom-created tables, you can always provision your application's data to a SharePoint list or library, depending on which works best for you.

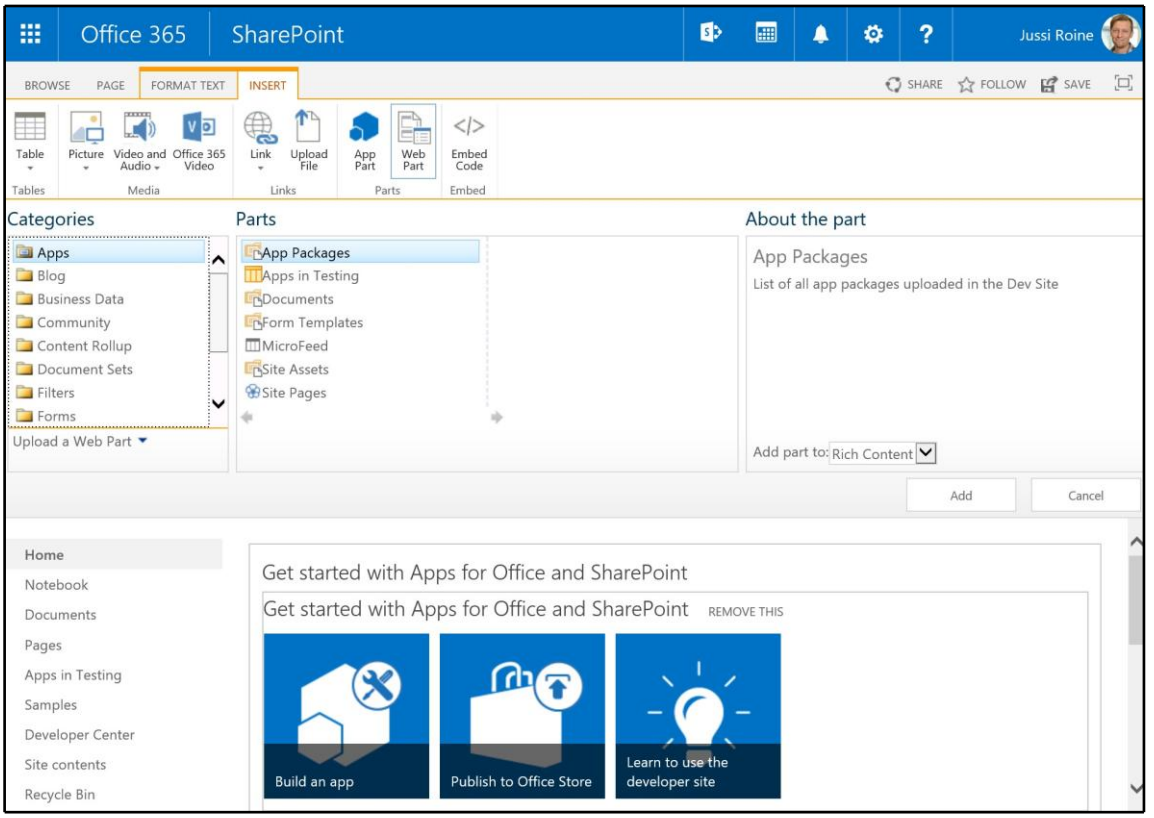
SharePoint web parts

Web parts are the building blocks for SharePoint pages. Pages sit in a document library and provide a graphical view that can be customized or hand-built depending on need.

Each document library and list, upon provisioning, create a web part representation of itself that can be embedded on a SharePoint page. As such, web parts are an easy way to create a clickable portal page to quickly access document libraries and lists or for simply resurfacing information from within SharePoint structures.

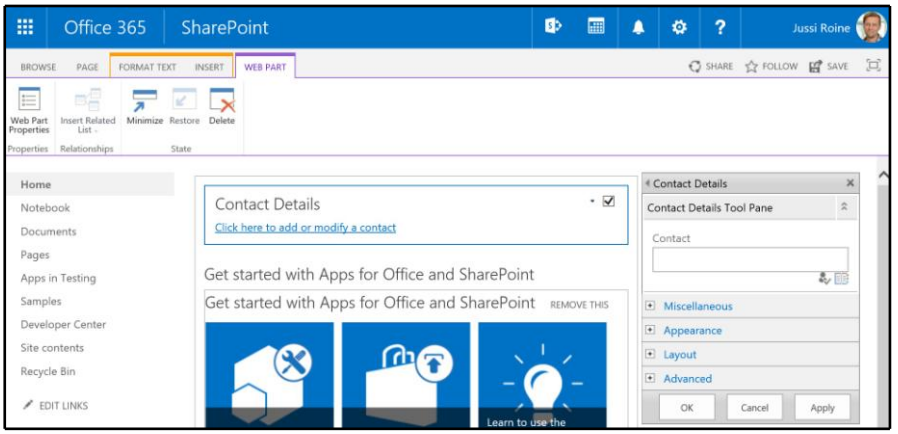
There are also a few dozen pre-created web parts that provide additional functionality, such as embedding comments on a page or retrieving an RSS feed and rendering a nice formatted list out of the feed items. Developers often create custom web parts that do the kind of actions that default web parts lack. This could include custom buttons, views, fields, and using data stored within SharePoint and providing a nicer interface for modifying said data.

By editing a SharePoint page, users can rearrange existing web parts on a page or add new ones and remove existing ones:



Adding a web part by editing the page and then selecting **Insert** | **Web Part** provides a list of all supported and installed web parts on the service.

The interface is a bit retro in the sense that it involves a lot of dragging, dropping, and popup menus to finalize the web part settings:



When editing a web part, the **WEB PART** tool pane (on the right) becomes visible, allowing manipulation of the **Web Part Properties**. This includes the title text and possible parameters to instruct the web part to act differently.

Why SharePoint Online?

So far, we've taken a whirlwind tour of SharePoint Online, barely scratching the surface of the default functionality within a SharePoint site. Considering the first version of SharePoint was introduced in 2001, a lot of has evolved since then. But at the same a lot has surprisingly stayed the same: document libraries, lists, and sites more or less share the same concepts as they did over 15 years ago.

Today, companies choose SharePoint Online for several reasons. In our opinion, the three main reasons companies choose SharePoint Online are:

- Compatibility and cooperation with Office suite applications, including cross-platform support on macOS and Office Online browser users
- A readily available intranet and extranet platform that works with single sign-on (using Azure Active Directory as the identity provider)
- It works natively with Exchange, Skype for Business, and OneDrive for Business, which are the default choices of productivity tools for many organizations

Most probably the development aspects of SharePoint Online are not the primary reason for companies to choose SharePoint Online. Interoperability, common standards, and well-documented APIs are critical, but often not the main driver for choosing SharePoint Online to start with.

Microsoft claims* that over 90 percent of Fortune 500 users are using Microsoft's cloud services; this, of course, includes both Microsoft Azure and Office 365. While the claim is very impressive, it does not mean that *all* 90 percent of Fortune 500 companies are solely using Office 365 and/or Microsoft Azure; they might be using Amazon's AWS or any number of other hosting or public cloud offerings available, together with Office 365. The reality, though, is that over 100 million users are accessing Office 365 workloads every month. This is an amazing number of users who have enrolled to a paid license and are actively using any number of the multiple services Office 365 has to offer. As such, development for SharePoint Online is something that is in demand from both small and large organizations and enterprises, as well ISVs who need to implement their own products to fit with Office 365.

*
Microsoft 2017 earnings call; see `https://www.microsoft.com/en-us/Investor/earnings/FY-2017-Q3/press-release-webcast`.

Another reason that organizations choose SharePoint Online is that they might already be paying for it. When they initially purchased a license for Office 365 to maybe use Exchange Online for their emails, they typically also purchased a SharePoint Online license for users. Nowadays this includes 1 terabyte of storage to hold all SharePoint Online content, as well as the majority of SharePoint Online functionality. Comparing this *free* offering to an on-premises SharePoint 2013 or SharePoint 2016 deployment often means that SPO becomes a compelling option. While the on-premises deployment of SharePoint typically requires at least 3-6 months for full installation, roll-out for users, customization, configuration, and managing--the equivalent in Office 365-SharePoint Online is already there and ready to use. This is not to say that SharePoint Online and SharePoint on-premises are identical; however, they do share more or less the same set of features that most users tend to need.

Office 365 licensing

While licensing is out of the scope for this book, we feel it is important for developers to understand the rough edges and limitations of a given service in order to be able to build performant and robust cloud solutions.

Office 365 licensing is a fixed per-user license that is billed monthly (or yearly if you choose to pre-pay for a full year). You can have one or more different types of licenses, but only one license type at any given time for a specific user. You can, of course, purchase a higher tier license, assign it to a user, and later change that to a lower tier license when the needs of the user change.

When writing about public cloud services such as Office 365, it is an assumption nowadays that prices fluctuate. You can always verify the current prices at `http://www.office365.com`. Make sure to change your country at the bottom of the page to get pricing in your preferred currency.

Office 365 licenses are sold for home use, business use, and enterprise use. When researching suitable licenses for yourself or your users, make sure you choose the correct category of licenses in order to find the best one for your needs.

Choosing an Office 365 license for development use

When you choose developer solutions for SharePoint Online, you have to choose one of the organization licenses for Office 365. The available licenses are in two license families; the **B** (**Business**) and **E** (**Enterprise**) license families. Most developers working on SharePoint Online commonly use E-licenses. This is mostly for minimizing any impending issues when deploying your code in the future, as E-licenses have most of the features one might need within SharePoint Online.

Over the years, many have complained about the complex licensing models Microsoft imposes on some of their products. SharePoint on-premises tended to have all kinds of small limitations, connector licenses, and similar things, to look out for that plagued many development projects in the early 2000 and 2010's. Office 365 is a different breed as licensing is very opaque and easier to understand. As stated before, you will need at least one license for your development use and additional licenses for any users who might be accessing your solutions that you aim to deploy to SharePoint Online.

We recommend you use a license from the E-family of plans, such as E1, E3 or even E5, which, at the time of writing, is the license offering the most features for Office 365 services. It is also the most expensive license.

You can see a detailed listing of all E-family plans and their respective features at `https://products.office.com/en-us/business/compare-more-office-365-for-business-plans`.

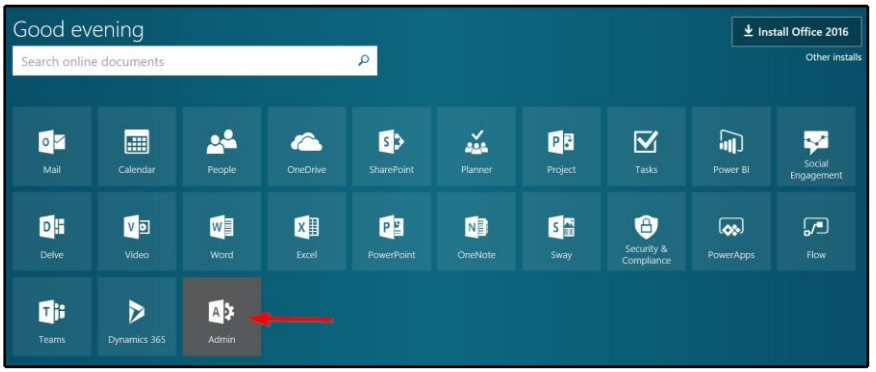
SharePoint Online is available for E1, E3, and E5, so from that perspective, there is little difference when starting development on SharePoint Online. Keep in mind that certain minuscule features might differ or might not be available in E1 if they are available in E3 and E5. These special situations might arise at a later time, so this is something to keep in mind if you choose to go with the E1 license. Remember that you can upgrade from E1 to E3/E5 at any time in the future (unless you pre-pay for the yearly license).

This book assumes you will be using an E3 or E5 capable license, as they provide the most functionality and are generally the license model organizations choose to use for their information workers.

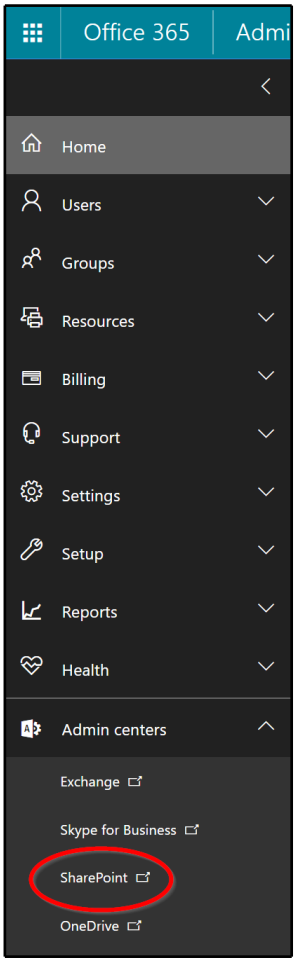
To enroll to Office 365, you can purchase a license at `https://products.office.com/en-us/business/office-365-enterprise-e3-business-software`.

Getting started with SharePoint Online

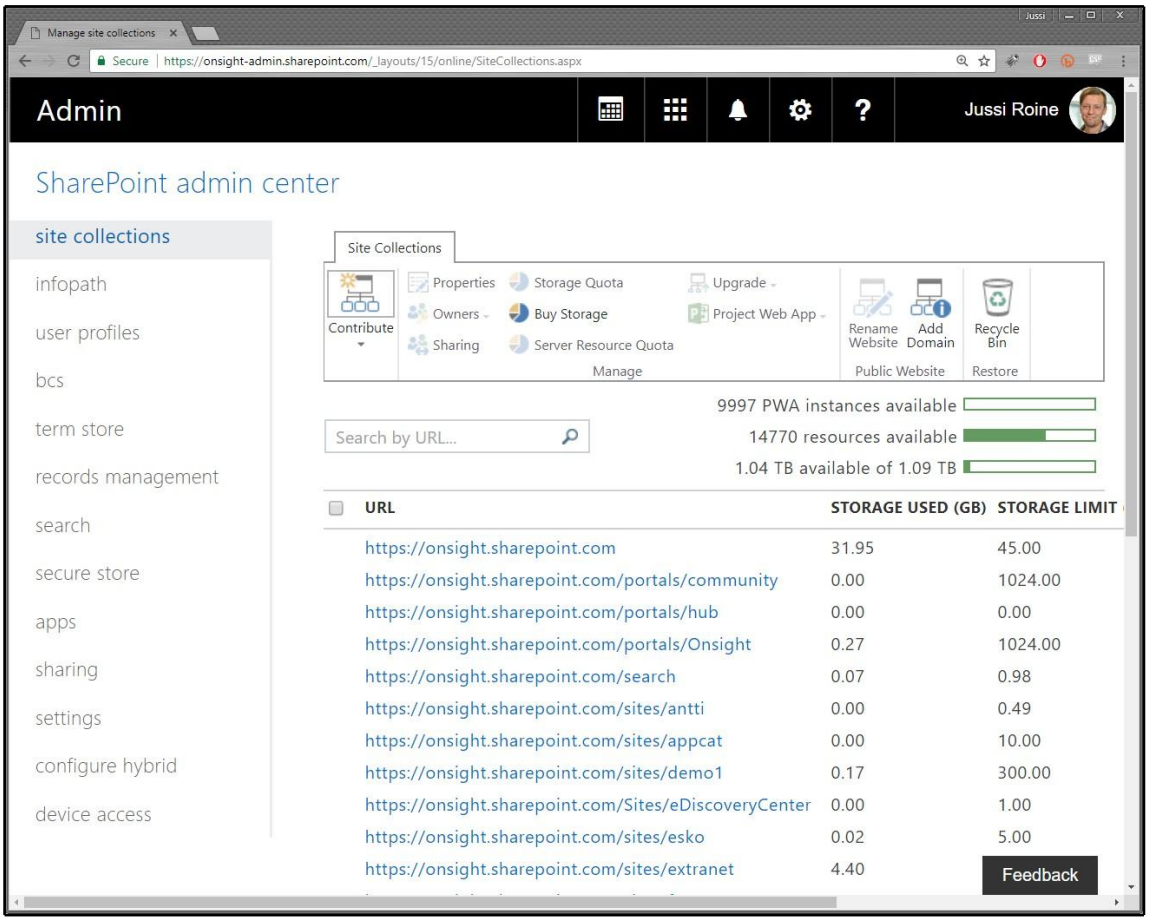
SharePoint Online has a separate administration and management interface called the SharePoint Admin Center. To access the page, click **Admin** on the Office 365, landing page (`https://portal.office.com`). This takes you to **Office 365 Admin Center** and requires you to be a SharePoint Admin or a Global Admin for your organization:



Under **Office 365 Admin center**, on the left-hand side, you'll get a navigation pane with all Office 365 management pages. Scroll down and click **Admin centers** and then click **SharePoint:**



This takes you to **SharePoint admin center**, with a direct address of the format `https://{tenant}-admin.sharepoint.com/`, where `{tenant}` is your Office 365 tenant name that you specified during Office 365 provisioning and sign-up phase:



In **SharePoint admin center**, you can create new collections of SharePoint sites, called **site collections**, configure any of the built-in services, and access any existing site collections within your tenant.

The interface is somewhat different from the more modern Office 365 admin center, but certainly still very usable. Many times, developers need to access this interface mostly to check on something, rather than spending time doing daily management tasks such as provisioning new site collections or fiddling with quotas. We anticipate the SharePoint admin center to get a refresh sometime in the future.

Your tenant should have at least the root (home) site collections provisioned at `https://{tenant}.sharepoint.com`. We recommend that, in the following chapters, when you build your solutions and try out the sample code, you provision new site collections and do not deploy any code in the root site collections. This way, other users in your tenant do not get confused if you leave around sample code or code that is still being worked on and might not work as intended.

Creating new site collections

A site collection is a core concept in SharePoint Online and works more or less the same as it has worked in SharePoint on-premises versions for years. You can create one or multiple site collections and each site collections can hold one or more sites. You cannot, however, create a site collection that holds another site collection underneath. You can only have site collections that consist of sites and subsites.

To create a new site collection, you can use **SharePoint admin center**, which is typically the easiest way for one-off site collections you might need for development use. If you need multiple site collections, it might be easier to use a scripting language, such as PowerShell, to create multiple site collections.

A word or two on SharePoint site templates

Each SharePoint site is based on a template. This makes sense as SharePoint (on-premises) and SharePoint Online typically provision tens or hundreds of sites for any given tenant, and most sites share a common set of functionality and features.

Site definitions

Templates used to be XML-based definitions, called Site Definitions. These would reside on the SharePoint servers in an on-premises deployment, but one couldn't directly modify these. The reason was that all SharePoint default files, that reside in the so-called SharePoint Hive (or SharePoint Root) on the file system, might be updated in any upcoming service pack, cumulative update, hotfix, or public update. Thus, if you edited even one XML file just slightly, you would run into unforeseen provisioning troubles, or even render yourself in a state that your SharePoint deployment would be unsupported.

Site templates

Site templates could then be used to provision new sites, with modified templates. Site templates were database-only copies of Site Definitions, and they could include content such as SharePoint document libraries, lists, and even files within document libraries. The challenge with Site templates was (and still is) that they are only supported in non-publishing sites, meaning sites that do not use SharePoint Publishing Features capabilities. This would include site templates (not Site Templates, note the capitalization difference here) such as Developer Site, Team Site, and Blank Site. A Site Template would, upon creation, save itself, and a reference to the original site definition it was based on, and create a `.STP` file within the SharePoint content database. As you might guess, this proved to be problematic as well, since content in the database was strictly only accessible through a set of (then) very-limited APIs or through the user interface of SharePoint. For developers, this was not adequate.

Web templates

Microsoft made a third type of template, called the Web Template. This was an evolution on Site Definitions but also took the better parts from Site Templates. A Web Template was a single XML file that only contained the deltas (changes) from a Site Definition without the need to first create a custom Site Definition. A developer could then have a peek at all the available Site Definitions as they were on the disk of the SharePoint server, figure out which was the best *base* template for the intended need, and provides a simple XML file that would provide the instruction set for a custom Web Template. This template would then show on the template selection list upon creating a new SharePoint site (or site collection, for that matter), and would not differ for the end user but, at the same time, could provide additional or customized functionality from the original Site Definition. The XML file for the Web Template could also be packaged in a manageable package and distributed as fit.

SharePoint Online and site templates

So far, we've covered Site Definitions, Site Templates, and Web Templates. One would think that with these three template types, developers could create any style of SharePoint site with full freedom and flexibility to customize sites. There is a slight roadblock here that subsequently forces us to move *away* from all these template types in SharePoint Online.

In SharePoint Online, developers do not have direct disk access. As such, developers are unable to take a peek at the SharePoint Hive and select from the list of available Site Definitions what kind of Web Template they are about to create. Of course, Microsoft has published the full list of supported Site Definitions, but the limitations of Web Templates is often a major challenge. For this purpose, and other smaller but persistent issues, developers needed something more manageable and something that could better be molded into whatever needs a project might have for a custom SharePoint site.

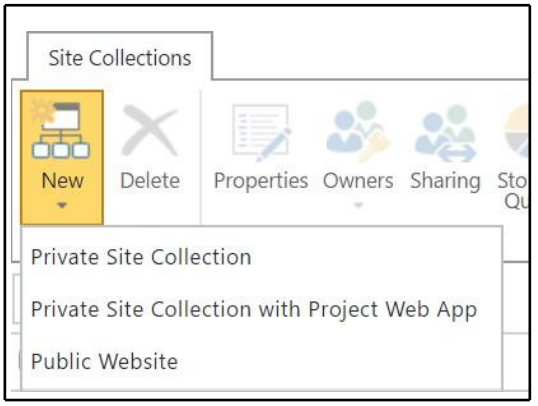
The solution is *remote provisioning*. We will cover remote provisioning in much more detail later in this book, but suffice to say that this approach allows for most freedom for developers while also maintaining full fidelity and supportability to any of the existing Site Definitions that SharePoint Online currently supports. With remote provisioning, developers either scan through newly-created sites or get an event (a sort of trigger) that a new site is being provisioned. Upon finding one, they can programmatically verify the origins of the site and the base template it was provisioned from--typically this would be a Team Site, Developer Site, or a Blank Site. Armed with this information, developers can then use the APIs to instruct the new site to re-configure itself based on another template--which would be a templating system the developer has coined.

It's not important to understand all the intricacies of the cumbersome SharePoint site template system as it has a 15 year history and most of that history is irrelevant for SharePoint Online developers. But it is essential to understand the basics of the templating system, especially the limitations developers might face when initially creating new SharePoint deployments.

When developing SharePoint Online-based solutions, you always need to base your solutions on the assumption that sites are based on a pre-defined site definition. You could also force the selection of the base template and discourage the user from selecting a template that might not be suitable for your needs. If you scan through the templates in the site collection creation dialog, you'll see there are a lot of templates that might not make sense today, such as the Visio Process Repository template that has a very specific usage.

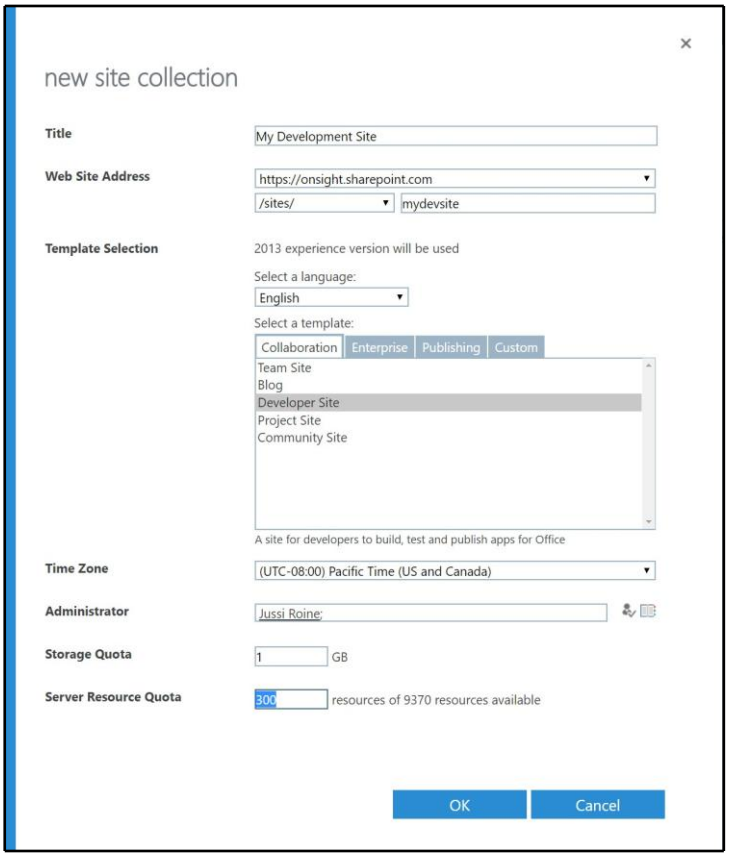
Creating a new site collection

To start creating a new site collection, click **New** in the toolbar and select **Private Site Collection**. This is the preferred choice for intranet-style deployments or sites you aim to share later for external users. **Private Site Collection with Project Web App** is a special type of site collection that also enables the use of **Project Online**, a project management and sharing platform that can be purchased with a separate license for Office 365. A public website might still be visible in your tenant, depending on when and how your Office 365 was initially provisioned. This is a functionality that will soon fade away and should not be used as it was originally intended for public-facing and anonymous SharePoint Online sites:



Next, fill out the provisioning form for a new **Site Collection**. For **Title**, you can choose whatever name you feel is suitable for your needs. For the **Web Site Address**, your tenant address is pre-filled and cannot be changed. The second part of the upcoming site collection URL will be a pre-defined managed path, either **/sites/** or **/teams/**. We prefer using **/sites/** as it is commonly used for all kinds of SharePoint sites regardless of their usage. For the last part of the URL, type in a name (no whitespace or special characters) for the site, such as `MyDevSite` and there is less risk to accidentally confuse these as Microsoft Teams-based sites.

Thus, your new site collection can then be accessed through `https://{tenant}.sharepoint.com/sites/mydevsite`:



For **Template Selection**, you should always choose **English** as the primary language. The reason for this is that if at any later time you run into errors or issues with your code and need to do troubleshooting, this language dictates the language used for error pages and exceptions. We've sometimes seen customers who originally provisioned their site collections in a language other than English, and since, this is a setting that cannot be changed (without destroying the whole site collection), troubleshooting code-specific or SharePoint Online specific gets very problematic. Template language does not dictate the language for SharePoint page content or any other setting for users later on.

For the actual template, the same rule applies--it's a one-time setting. It's best to spend a bit of time here, as the template selection is a crucial selection and developers should understand why this is as it is.

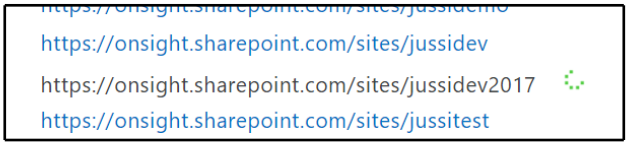
As explained earlier, there are a lot of different templates available in SharePoint Online. When starting your development adventures in SharePoint Online, it's always good to go with the **Developer Site** template, as it provides some advantages over the other templates.

For **Time Zone**, choose whichever timezone is most fitting for you. For **Administrator**, use your own credentials to ensure you have full access to your new site collection.

For **Storage Quota**, while being important, it's not really relevant for development sites so 1 GB should be more than enough for any development needs you might require for now.

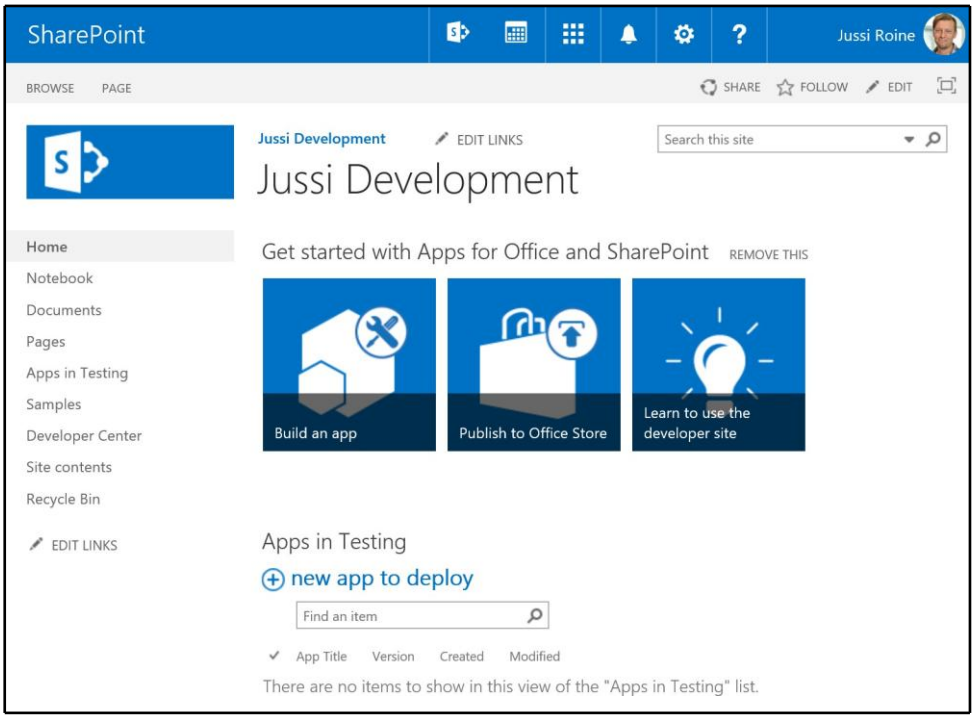
The **Server Resource Quota** is reminiscent of the time when development mostly occurred in sandbox development models and it is not in active use anymore. You can leave the default value of 300 and then commit the provisioning of a new site collection by clicking **OK**.

Creating a new site collection takes a little bit of time, anywhere from a minute to more than 15 minutes depending on the load on the service. In normal circumstances, a new site collection should be provisioned in no more than five minutes.



When the site has been provisioned, you can access the site at `https://{tenant}.sharepoint.com/sites/{sitename}`.

When you access the new site collection, you initially land on the root site of the given site collection. Sometimes this is confusing since you're accessing both the site collection and the root site at the same time. Just keep in mind that you are most often working with individual sites not specifically with individual site collections:



Note that the developer site, which we chose as the site template, is not updated for modern experience yet. As such, we're mostly receiving the classic experience when working on a developer site.

Developer sites versus team sites

A developer site has a few differences from regular team sites that you would typically provision for users. First and foremost, developer sites allow add-in sideloading. This is a method for developers to quickly deploy a custom-developed add-in for their site for testing, thus bypassing the safety procedures and possible governance models SharePoint Online admins tend to prefer in production environments.

As such, if you choose to develop your own add-in, you could deploy it directly to your own developer site. Later, when you are certain your add-in behaves as it should and is ready for production, you can submit the add-in to a separate site, called the Application Catalog, that admins use to provision the custom add-in for wider consumption.

Second, besides the add-in sideloading functionality, developer sites have a few quick links in the left-most navigation bar to aid in quickly navigating between core features within a developer site. The **Apps in Testing** link takes you to a list of add-ins you currently have sideloaded on your site so that you can quickly hop back and forth between multiple add-ins. **Developer Center** is a static link in Microsoft's own public Office Developer Portal at `https://dev.office.com/docs`. **Samples** are also a static link to Microsoft's public repository of sample code.

Don't worry about the navigation bar items or the overall look and feel of your developer site. The purpose of the site is to allow you to kickstart your add-ins for further testing, not to act as a landing site for your users to access your custom add-ins in the future. Typically developers re-create new developer sites when the need arises and might not use the same developer site eternally.

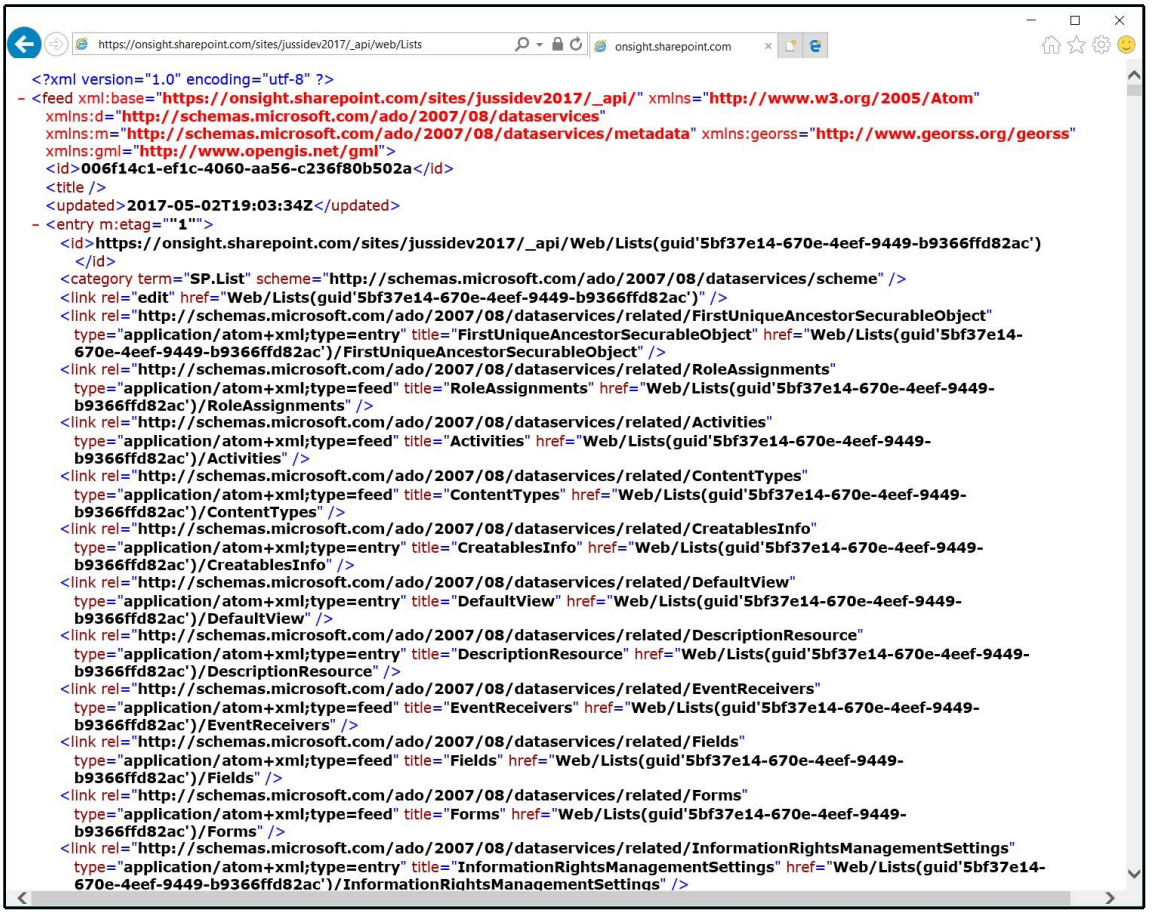
SharePoint Online APIs

Developers need APIs in order to successfully integrate their solutions with other workloads and services. SharePoint Online exposes a good set of APIs, which we'll introduce in this chapter. In later chapters we'll walk through how to best employ the APIs, including authentication and authorization, as well as proper use cases for different kinds of APIs.

Currently, SharePoint Online exposes three types of APIs.

REST APIs, that can be found under the `/_api/` URI, for example, `https://{tenant}.sharepoint.com/_api/{commands}/{parameters}`. This is the most common use case for accessing SharePoint Online structures, data, and information. A sample API call would be a call to the Lists API, which exposes all site lists. The URL for this would be `https://{tenant}.sharepoint.com/_api/web/lists`.

Note, that we need to first call `/web/` underneath `/_api/`, in order to instruct the API to look at the current SharePoint site, and then call the `Lists API`:



The output, by default, is XML. This sample call exposes all SharePoint lists in the given SharePoint site.

The REST APIs available for SharePoint Online are listed here: `https://msdn.microsoft.com/en-us/library/office/jj860569.aspx?f=255MSPPError=-2147217396#Reference`.

In addition to REST APIs, SharePoint Online still exposes a smaller subset of **SOAP Web Services**. These Web Services are from the SharePoint 2007 era, but still, work and can sometimes turn out to be useful in specific situations where the REST APIs can't be used or where they might not provide the needed information. In truth, the SOAP Web Services are randomly used and not something SharePoint Online developers should resort to unless absolutely needed.

The SOAP Web Services available for SharePoint Online are listed here: `https://msdn.microsoft.com/en-us/library/office/gg454740%28v=office.14%29.aspx?f=255MSPPError=-2147217396`

Last, but definitely not least, are the **Microsoft Graph APIs** or, more specifically, endpoints. As the name implies, these are not specific to SharePoint Online but more of an overall collection of APIs that Microsoft Graph provides.

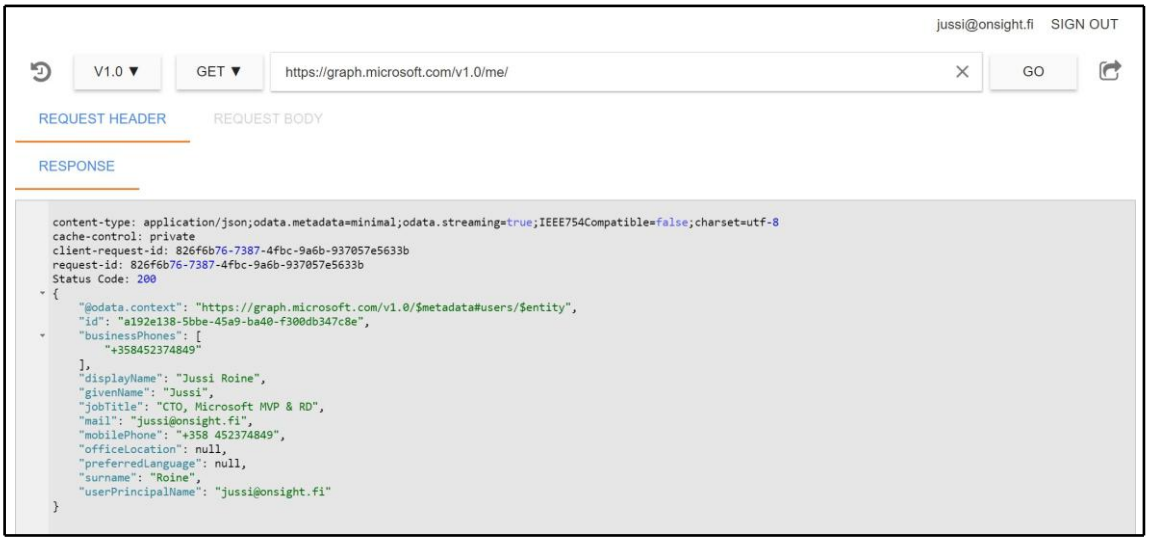
A quick primer on Microsoft Graph

Microsoft Graph is an ever-growing collection of APIs that can be uniformly accessed through a common contract. The idea with Microsoft Graph is that developers would only need to learn one set of authentication, authorization, and access models to access any kind of cloud-based service within Office 365. For now, Microsoft Graph does not provide parity for all SharePoint Online REST APIs, but it has a fairly good collection of modernized APIs developers may wish to use for their applications.

Using Microsoft Graph is essentially more helpful with Graph Explorer, which is a web-based tool for figuring out what data there is and where it lies. It can be accessed through the Microsoft Graph marketing page at `https://graph.microsoft.io/`, by clicking **Graph Explorer** on the top navigation:

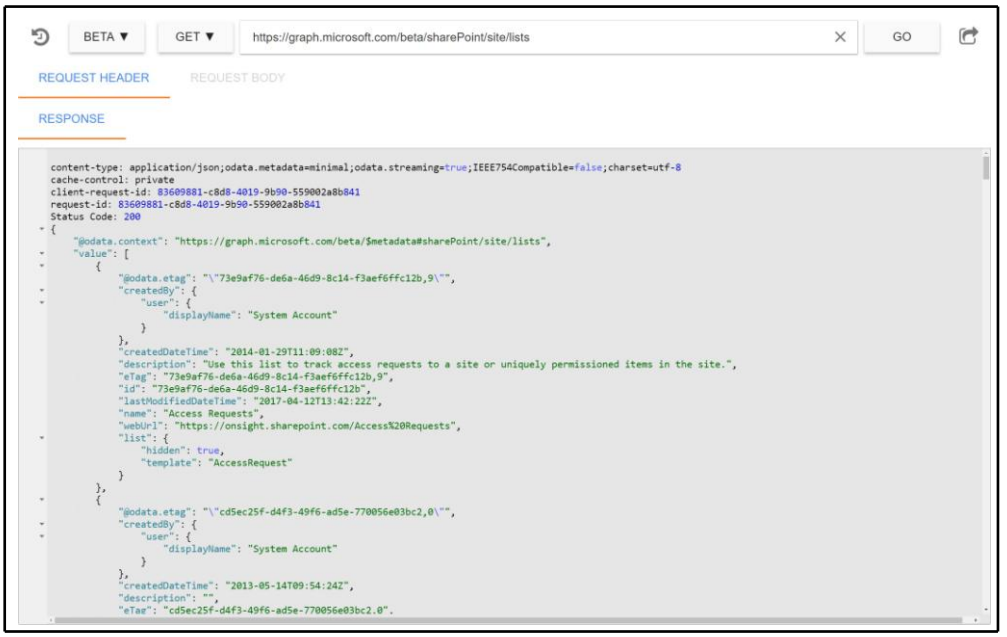
You can try out the APIs in a demo tenant, or by clicking **Sign in**, you can access your real data. Keep in mind that the Graph Explorer is built and operated by Microsoft, but it's still a good idea to avoid logging in with your global admin account unless you absolutely and unconditionally trust the service.

After logging in and running the default query against Microsoft Graph, the `/me/` object resolves to your current account's metadata:



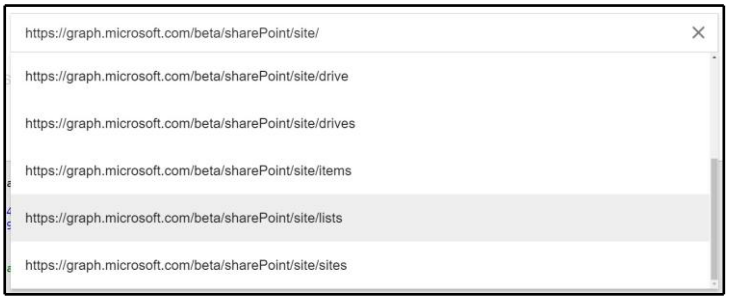
To target the queries against my SharePoint Online site, I need to change queries from `/v1.0/me/` to `/beta/`, as SharePoint Online APIs are still in beta for Microsoft Graph.

Under `/beta`, I can point my queries to `/beta/sharePoint/` (note the lowercase 's', and capital 'P') and then to `/beta/sharePoint/site/lists`:



This returns a clean list formatted in JSON for us to use within our code. The purpose of the Graph Explorer is to allow the developer to rapidly test against live data and figure out the correct API queries.

Graph Explorer has an intellisense-style helper built-in, so when you're constructing your queries, you should see a small window popup and provide suggestions for the current query:



All Microsoft Graph SharePoint Online APIs are referenced and documented here: `https://developer.microsoft.com/en-us/graph/docs/api-reference/beta/resources/sharepoint`.

You are free to combine your calls to different APIs between SharePoint Online REST APIs, Microsoft Graph calls, and perhaps also the SOAP Web Services. In addition, you could also build your own custom APIs using Microsoft Azure API Apps, Azure Functions, and similar services. A good practice is to first check Microsoft Graph for your needs. If you find it lacking for your specific requirements, check the SharePoint Online REST APIs, and only if these do not provide you with the necessary functionality, consider building your own custom APIs.

Developing solutions for SharePoint Online

By now, you should have a basic understanding of what SharePoint Online is, how information is structured within SPO, and how you can provision your own developer site that can be used as a playground for your upcoming custom deployments.

Next, we'll look at what kind of solutions developers can build for SPO. We'll start with the classic and oldest models and move up the ladder to the add-in approach. This excludes the latest addition to the family, the SharePoint Framework (SPFx), which the book will cover in deep detail in the following chapters.

We feel it's crucial to understand how things were built for previous (and sometimes current) versions of SharePoint, even if at a pretty high level, in order to respect and understand the decision and options we have at our disposal today.

Solutions for SharePoint and SharePoint Online

Solutions for SharePoint (primarily versions from 2007 to 2016) and SharePoint Online can be built in numerous ways. Next, we'll go through a bit of history because we feel it is important to have a glimpse of how (bad) things were in the beginning. It is often helpful to understand why things work in a certain way with SharePoint, as many technical functionalities in SharePoint are based on the pre-SharePoint Online era.

SharePoint 2001-2003: direct modification of files

As the title of this section states, this relates to the early days of SharePoint, between SharePoint Portal Server 2001 and SharePoint Portal Server 2003. These two versions did not have a way to formally package, deploy, or retract custom code. Developers resorted to all kinds of hacks and kludges that kind of work, but did not really respect any solid development approaches, such as application lifecycle management or version control.

Developers would simply open a remote connection (typically via Remote Desktop) to a SharePoint 2001/2003 server, find the file they needed to modify, and use either Notepad on the server or copy the file for additional editing on their own workstation. Files would reside in the SharePoint Hive, under such directories as `C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\<version>\`. Each SharePoint server, if you had several in a farm configuration, would have identical files. Thus, changing one file on a single server would require you to change the same file identically on other servers as well.

In a way, this was a quick, fast, and reasonably well-understood approach. If you needed a change within SharePoint, you could simply figure out which file was responsible for the change and inject your changes to that file only. The result would often be so-so and kind of acceptable--until the next Service Pack was released and installed on the very same SharePoint server. At best, it would simply wipe customizations and force you to re-do all your changes once more. At worst, it would break something horribly as some files would now be patched with the Service Pack's version, while others would be customized and there would be no knowing what had gone wrong and where. Pages wouldn't load and random errors would occur for users.

One inventive, although certainly unsupported approach, was to tinker with Windows' **Internet Information Services** (**IIS**) web server settings to fool SharePoint just a tiny bit. SharePoint typically, at the time, mapped the infamous `/_layouts/` virtual directory to the SharePoint Hive under `C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\<version>\` subdirectories. Clever system administrators and developers would copy the hive directory structure to a sibling directory structure, and map the virtual directory from `/_layouts/` in IIS to this new copy of the original files. In practical terms, this allowed one to tinker with the files without fear that original files would be destroyed. Upon new Service Pack deployments, you could point the virtual directory back to the original hive, allow patching to complete, and then do a file diff comparison between the original (patched) hive and your customized hive. Talk about spending a lot of time to get to your results!

Luckily, SharePoint versions 2001 and 2003 are not supported and we haven't seen these versions in production in years. But this is where it all started back in the day.

SharePoint 2007--Full-trust code

Microsoft Office SharePoint Server 2007, or MOSS as it was known then, introduced the first true and quite proper development approach to SharePoint. This vehicle became known as SharePoint solution packages, that packaged our native .NET code to files with a `.WSP` extension.

This was also known as the full-trust code, as all code you wrote, packaged, deployed and used would run as fully trusted within SharePoint's memory space. A poor developer could easily write horribly sloppy code that iterated through hundreds of SharePoint sites recursively, then iterated through each list and document library and changed something in one or two of the document libraries. This would then be embedded in a web part, that is a building block that acts as a widget on a SharePoint page. Initially, the developer might have tested this on his or her laptop, and everything could have executed flawlessly. Change this to a production environment, with possibly hundreds or thousands of people loading the very same page, and you quickly run into all sorts of unforeseen performance issues.

Full-trust code initially did not have proper support within Visual Studio at the time, so developers would build custom tooling to compile, define, and package the deployment packages. Deployment would also be part of custom scripts and tooling executed locally on one or more SharePoint servers. Debugging would then occur on a separate server and would often be time-consuming and error-prone.

Solutions would be crafted around Features, which acted as a kind of lights on/lights off containment model for SharePoint artifacts. One or more Features would be packaged within a SharePoint solution package (the `.WSP` file) and uploaded to SharePoint's database for deployment. Each Feature would consist of at least one XML file introducing the deployment logic, and each solution package would consist of exactly one manifest file that held everything together. Combine this with possibly multiple solution packages per the overall solution and you often had a time-sensitive and highly detailed deployment script that needed to be executed just so, in the hope that everything would work.

What's important to realize is that MOSS 2007 gave us the Feature framework and the Solution framework. Much of the development between SharePoint versions 2007 and 2010 was tinkering and fiddling with XML file structures, essentially describing in pre-defined XML structures how a certain feature might act within SharePoint. Here and there developers could inject a little bit of HTML, combined with JavaScript, CSS and, if lucky, possibly also some C# code. We still remember countless meetings and review discussions with talented developers who swore never to work with SharePoint again, as it was not real coding, but would rather modify cumbersome XML structures that at times were poorly documented and lacked a strongly typed approach.

You wouldn't have compile-time error handling, and deployment to SharePoint might have taken 10 or 15 minutes for each try.

SharePoint solution packages (and Features) are still supported for all SharePoint versions from 2007, 2010 and 2013 to 2016. It's not advisable to continue building solution packages but, at times, a transformation from a legacy solution might be required. It should also be clear without saying, but just to be clear solution packages *are not* supported in SharePoint Online. As such, developers aiming to build cloud-supported solutions for SharePoint Online, should not build solution packages or solutions using the Feature framework.

SharePoint 2010 and SharePoint Online: sandbox solutions

With SharePoint 2010, Microsoft introduced a subset of the full-trust code development model. This was called sandbox solutions and, on paper, the idea was great. Not fabulous, but great.

The thinking at the time was that since organizations were starting to move more and more of their on-premises workloads to hosting environments or even public clouds (pre-Office 365 era), something needed to be done so that SharePoint deployments littered with full-trust code could safely be lifted and shifted or migrated to a multitenant platform. Since full-trust code was, well, fully trusted, it wouldn't take more than a few hours for someone to deploy bad-behaving code that would crash the whole application pool of a given IIS, and possibly even consume all available virtual memory within the SharePoint server.

Sandbox solutions were introduced for both SharePoint 2010 and SharePoint Online. These were initially meant for SharePoint 2010 and were offered as a multitenant platform for organizations so that they could share a common platform, and were later ported to SharePoint Online as a means to govern what kind of custom code would be deployed to a shared platform.

The idea with sandbox solutions was that running code within a package could only access a very limited set of APIs and namespaces within the SharePoint Object Model. One limitation that was set, meant that all calls to methods within the SharePoint development models could only access elements under a single site collection by using the `SPSite()` object. This way developers couldn't access any structures or data above the site collection that was running the code. Access to direct file I/O was also prohibited, as well as access to directly call HTTP interfaces through SharePoint.

This approach worked well, on paper. But in reality, porting existing code from full-trust packages proved to be troublesome, time-consuming, and sometimes just impossible. Existing code often resorted to using certain methods and features of the SharePoint Object Model, so that a replacement within the `SPSite()` object was not simply available. Therefore, either big pieces of custom code had to be cut in order to move the platform to the cloud, or fully rewritten without having proper APIs.

As part of this ideology, Microsoft introduced the idea of resources. Each call from the code would consume a small piece of resources from a pool of resources, and administrators could designate pre-allocations for each site collection. When the customization exceeded its allotted resources, it would simply be shut down. Imagine having a graphical component running on your SharePoint site that exposes a tool for your users. Suddenly, the tool simply disappears from the page because it exceeded a hidden resource allocation that end users weren't aware of. Administrators would have to reallocate more resources or start digging through code to understand where and how the resources were spent and how to best optimize this piece of code for a small subset of users complaining loudly.

As such, sandbox solutions never really took off. A couple of years after introducing sandbox solutions in SharePoint Online, Microsoft quietly deprecated the ability to run custom .NET code within a sandbox solution package. This effectively rendered the whole model to a vehicle for provisioning XML-based artifacts for a given SharePoint site or site collection. Not much changed, as developers were now complaining (again) that developing solutions for SharePoint were merely pointless exercises in understanding archaic XML structures in Notepad.

From time to time we still see customer deployment in SharePoint Online happily running sandbox solutions. This is fine but in no way advisable for the future. A good approach is to export the sandbox solution files, rename the packages from `.WSP` to `.ZIP` files, and explore what's in the package. This way you can learn and understand what the solution truly implement, and how to best replace the solution.

SharePoint 2013, SharePoint 2016, and SharePoint Online: add-ins

With SharePoint 2013, Microsoft felt it was time to introduce a new approach to partially (or fully) replace the now aging Feature and Solution framework development model. This model was initially launched to market with a catchy name, the **Cloud App Model** (**CAM**). As its name implies, CAM was crafted on purpose to provide a cloud-enabled development approach that fully supported SharePoint Online without the missteps of sandbox solutions. Later, CAM was renamed the App Model (AM) for a brief moment, and a while later it was renamed the Add-in model. For now, the Add-in model seems to have stuck, so in the context of development, we'll be referencing all types of apps as add-ins when they relate to SharePoint add-ins.

The winning ticket within the Add-in model was the de-coupling of server-side code from SharePoint. Developers were now able to code in their preferred programming language, the server-side implementation of their customization, and simply run that outside SharePoint. As such, it was never similar to full-trust code from the SharePoint 2007-2010 era, but more a model that allowed developers to execute code elsewhere.

This was also fairly close to the time when Microsoft Azure started gaining more and more ground and popularity within organizations. Ideally, companies could implement custom add-ins that would run in Microsoft Azure's **Platform as a Service** (**PaaS**) environment, and the surface on SharePoint with the help of the Add-in model's **Client-Side Object Model** (**CSOM**).

Add-ins initially came in three flavors:

- **SharePoint-hosted Add-ins**, which are add-ins without any server-side code or additional logic that would need to run outside SharePoint. They would use SharePoint's data structure, including a separate subsite within the current SharePoint site where the add-in was being provisioned. The subsite (called the AppWeb) would be used as a fixed storage for everything the add-in might need to record. The AppWeb could, obviously, store only data that fit the artifacts available for a SharePoint site: lists, document libraries, and similar quite limited elements. All user interface elements had to be built with CSS, HTML, and JavaScript, and all assets had to be hosted within the AppWeb to provide better security and in order to avoid cross-site scripting attacks.

- **Provider-hosted add-ins**, which are true add-ins in the sense that server-side code can be included as part of the add-in. The development consists of a separate website, that can be implemented with several supported frameworks and languages--even in non-Microsoft programming languages such as Python or Ruby on Rails. The idea is to implement a separate website and then call upon that site and embed the output of the site to a SharePoint site using a specifically crafted IFRame. For the most part, this solution turned out to be a fairly reasonable approach to running custom code outside SharePoint. For many scenarios though, performance was subpar, and security was problematic to configure and understand. The whole framework relied on carefully created public-key cryptography certificates that had to be deployed just in the right order in the hope of getting the platform to securely call the external website within the user's security realm. Provider-hosted add-ins are still built today, but they are not as common as one might expect. For several larger deployments with tens of thousands of users, we are seeing one or maybe two provider-hosted add-ins, and they tend to be rather complex implementations of internal applications that must play nicely with SharePoint's interface and look and feel.
- **Auto-hosted add-ins** were a novel idea to run add-ins in the cloud and if code required additional resources, they would be automatically provisioned from Microsoft Azure. There were too many questions on this approach--especially from customers who would ask which credit card or billing contract was used for the Azure-provisioned assets. Since then, Auto-hosted Add-ins have been retired and they have not supported or in use anymore.

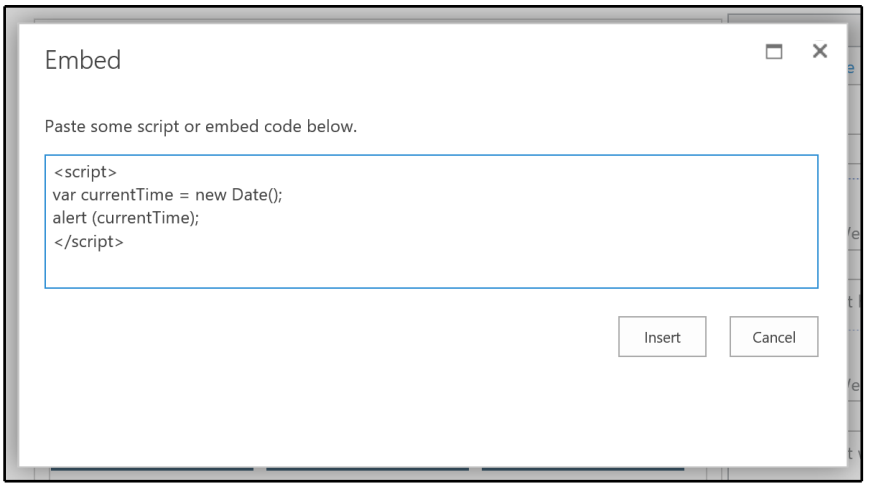
Today, the Add-in model is still supported and alive. Not much progress has occurred with the model and it's mostly been stale since 2015.

SharePoint Online--add-ins and client-side scripts

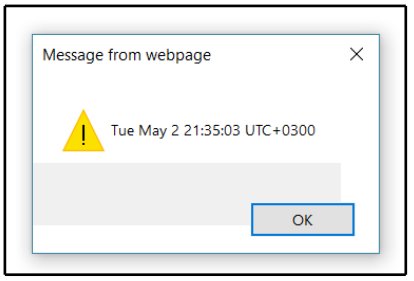
In addition to add-ins, developers often need to drop small functionality and bits of features to individual pages within SharePoint sites. A common approach is to add a Script Editor Web Part on a page and drop a piece of JavaScript and/or HTML within the page. This is a very simple approach but can be both powerful and troublesome in the long run.

The benefits of using a **Script Editor Web Part** (**SEWP**) is that it's very easy to add as an ad-hoc solution when in a meeting with the site owner and simply code on-the-fly whatever is needed. This is assuming the developer is quite capable and fluent with JavaScript and the SharePoint APIs.

Adding a SEWP on a page allows developers to write JavaScript through the browser and save it into SharePoint Online's own database:



Upon page load, the script is executed and the user gets the result of the preceding code in a popup since the sample is using the classic JavaScript `alert()` message box:



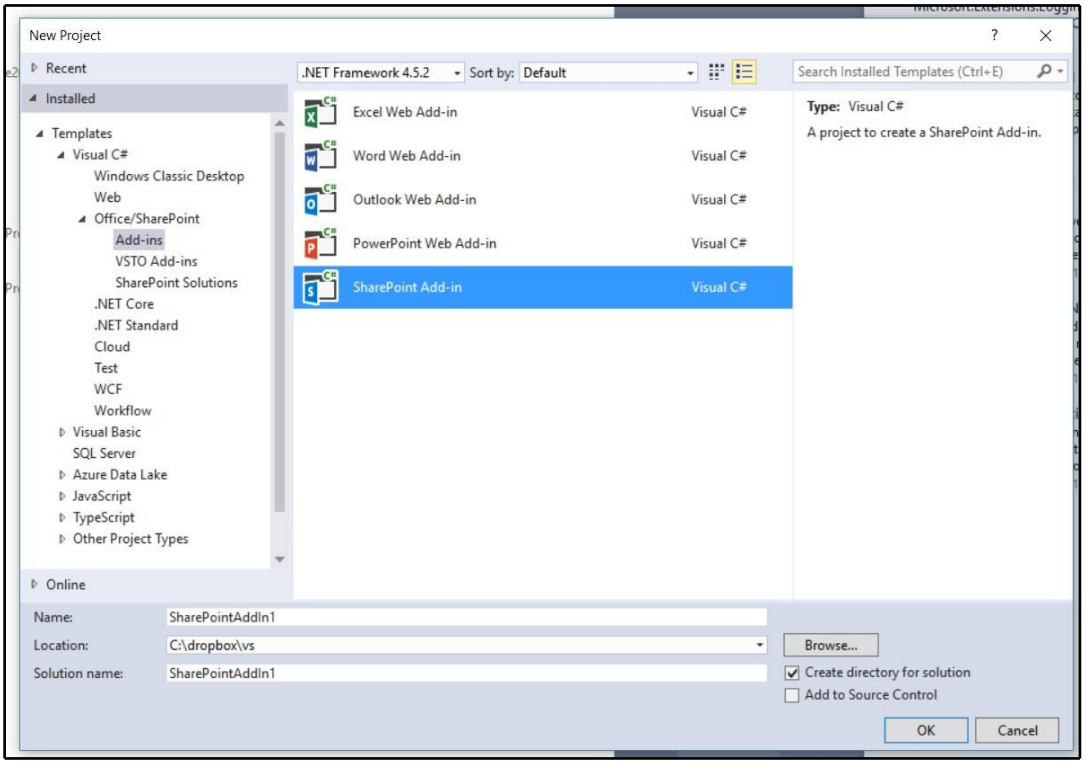
The downside is that pages tend to become littered with all sorts of small JavaScript tidbits that no-one can keep track of. One piece of code could reference an external library that has a locally stored copy of a SharePoint document library, while another piece of code (even on the same page!) might reference the very same external library but a different version through a public CDN URI. Thus, the payload for a user loading the page would be at least double, as the client browser would need to resolve both external frameworks, load the payload, and figure out which version to use and when. Imagine having hundreds of sites, thousands of pages, with customizations using the SEWP-approach on five percent of all pages. You very quickly run into performance and supportability issues, as well as troubleshooting errors that are not obvious as frameworks are not referenced in a proper manner.

Development tooling for SharePoint Online

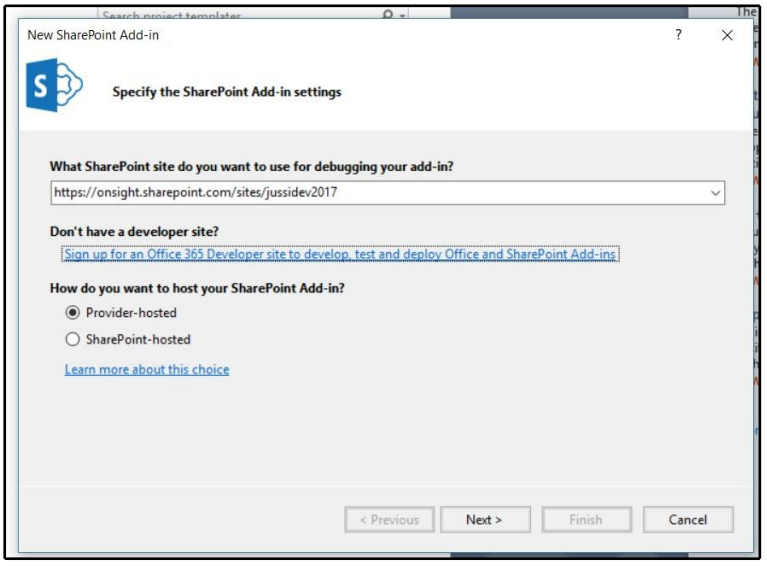
So far we've established that solutions built for SharePoint Online historically have been mostly about sandboxed solutions, custom scripts embedded on pages with the Script Editor Web Part, and custom add-ins that are either hosted within SharePoint or separately in Microsoft Azure (or a similar environment).

Developers use a reasonably new version of Visual Studio development tooling to build, package, and deploy their code to SharePoint Online. This can be Visual Studio 2015 or 2017, or newer when one becomes available. By installing Office Development Tools during Visual Studio installation, developers get to enjoy the pre-built templates for new projects that target SharePoint Online.

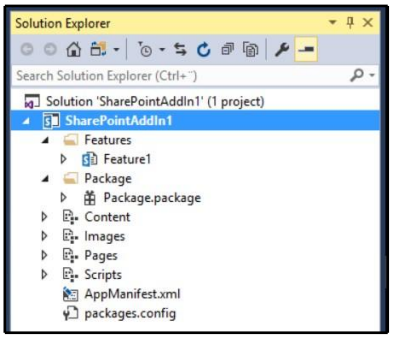
The only template that can be reasonably used today is the SharePoint Add-in template:



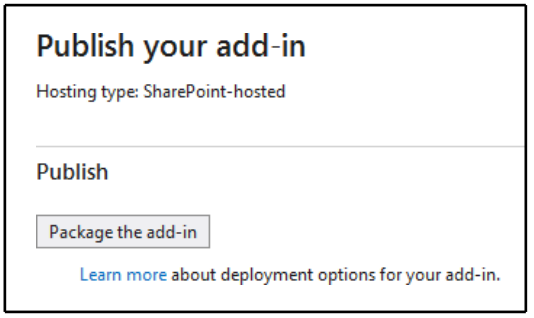
1. This template allows for the creation of a SharePoint add-in that can be either a SharePoint-hosted add-in or a Provider-hosted add-in. During the initial creation of the project, you can specify the target site for development, and this can be the developer site you created earlier within your SharePoint Online tenant:



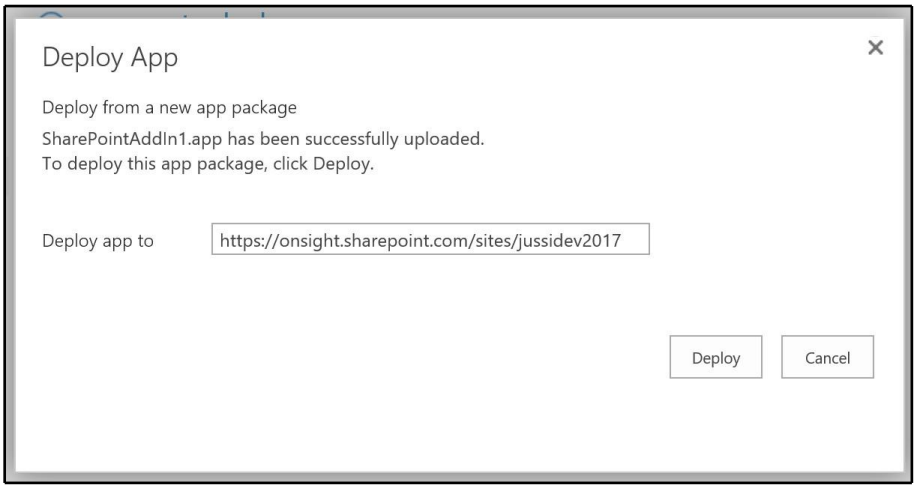
2. Note that if you choose to create a provider-hosted add-in, you'll need to do additional configuration with certificates in order to secure the deployment. For a SharePoint-hosted add-in, no additional configuration is needed and you can start building your solution right away.
3. The solution consists of a single Feature, the Add-in package, and necessary support files and assets:



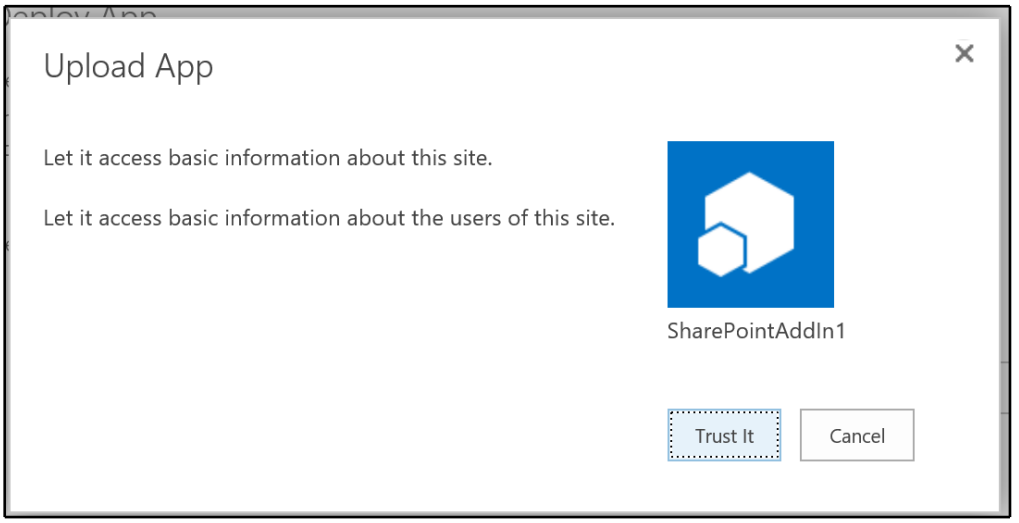
4. When deploying the solution, you can right-click on the **project**, and select **Publish**.



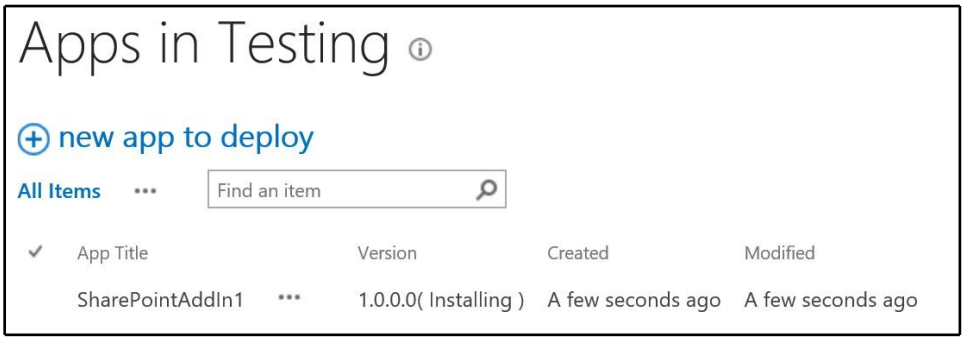
5. This allows you to publish the add-in as a package for manual deployment, or deploy the package to Office Store. Office Store is a marketplace run by Microsoft, allowing for reselling and marketing of your add-in to other organizations.
6. The packaged add-in can now be manually deployed to your developer site. First, simply upload the add-in package file to your developer site's **Apps in Testing** document library. Upon completion of the upload, you can simply deploy the app to the same site:



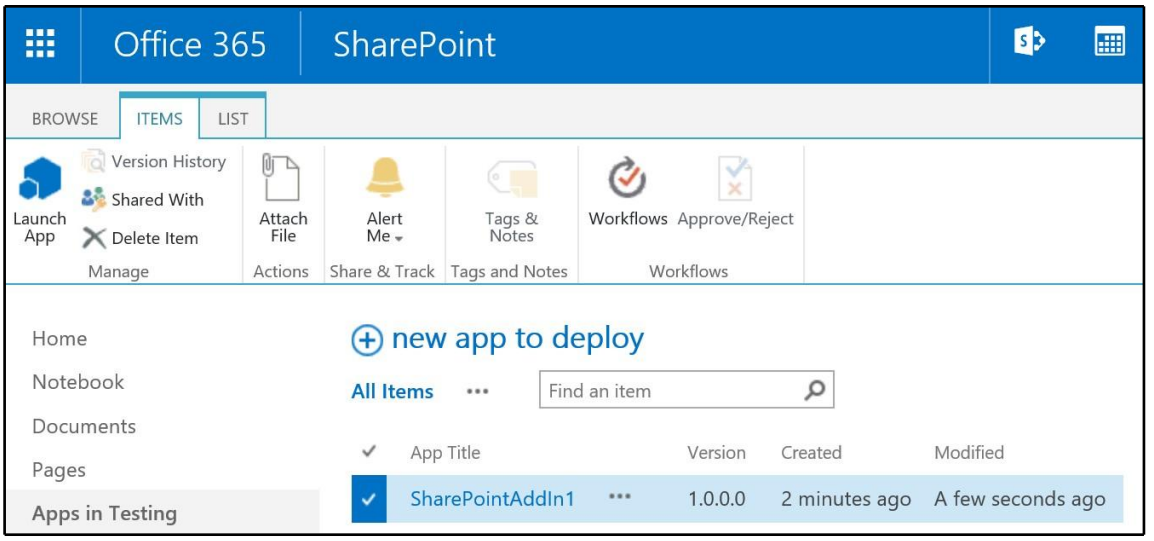
7. As the add-in is merely a sample add-in, we haven't introduced any additional user permissions we might be needing while impersonating the logged-in user (executing the add-in). As such, we can simply trust the add-in:



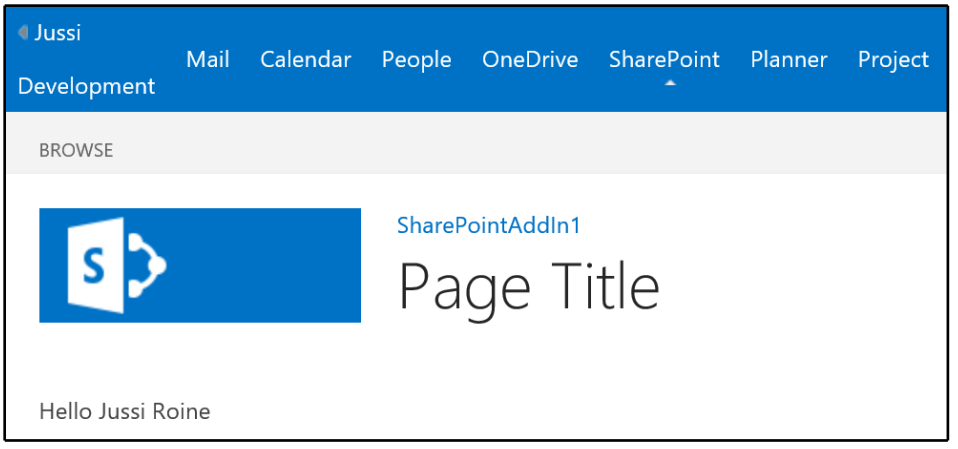
8. The add-in will now install and self-provision within the site:



9. Installation might take a minute or two. After it completes, the ribbon has a **Launch AppButton** that becomes enabled when the app is selected in the document library. Clicking the button will call the add-in:



10. Since we didn't configure the add-in at all and just compiled the default template, the add-in is a full page app. It inherits the SharePoint Online look and feel, but is running in a different base URL to the SharePoint Online site it was launched from:

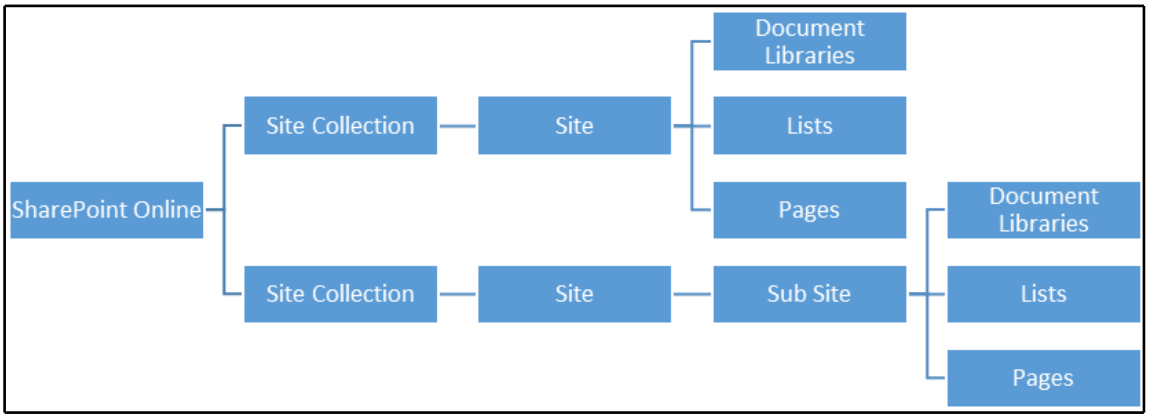


11. The add-in calls a small piece of JavaScript to resolve the logged-in user's full name. Everything else is static on the page, including the top-most navigation. The same add-in could also be embedded in a SharePoint page if we add an App Part instruction to tell SharePoint Online it can embed the add-in safely.

Summary

In this initial chapter, we had a tour of the core SharePoint Online building blocks, including document libraries, lists, and web parts. Additionally, these are always stored within SharePoint sites, which are then constructed within a given SharePoint site collection.

A somewhat simplified logical view of the structure of SharePoint Online is shown in the following graphic:



Typically, you would create one or more site collection and one or more site within these site collections. One site collection could be an intranet and another could be a dedicated site collection for project sites. Each site could then host your customizations and custom add-ins.

Development for on-premises versions of SharePoint differs quite a bit from SharePoint Online, as SharePoint Online only supports either client-side code with the use of JavaScript, CSS, and HTML, or code executed outside SharePoint--typically within a Microsoft Azure service such as Azure App Service.

For development, developers tend to use Visual Studio when building add-ins, as the templates are immensely helpful in managing the necessary file structures and packaging. Deployment can be done manually against a SharePoint Online developer site for real-life testing before production deployment through a special Application Catalog site.

In `Chapter 2`, *Developing Solutions for SharePoint Online*, we will start our journey of developing solutions for SharePoint Online. We will first have a look at SharePoint Framework, which partially replaces but also augments some of the existing development models in SharePoint Online. In addition, we'll dip our toes the usual development aspects of any type of project, including application lifecycle management and the best ways to manage your development efforts in SharePoint Online.