FORCE.COM SUCCINCTLY

BY **ED FREITAS**

SUCCINCTLY E-BOOK SERIES

Syncfusion®

Force.com Succinctly

By
Ed Freitas

Foreword by Daniel Jebaraj

Technical Reviewer: James McCaffrey

Copy Editor: Courtney Wright

Acquisitions Coordinator: Tres Watkins, content development manager, Syncfusion, Inc.

Proofreader: John Elderkin

Table of Contents

The Story behind the *Succinctly* Series of Books

Daniel Jebaraj, Vice President
Syncfusion, Inc.

Staying on the cutting edge

As many of you may know, Syncfusion is a provider of software components for the Microsoft platform. This puts us in the exciting but challenging position of always being on the cutting edge.

Whenever platforms or tools are shipping out of Microsoft, which seems to be about every other week these days, we have to educate ourselves, quickly.

Information is plentiful but harder to digest

In reality, this translates into a lot of book orders, blog searches, and Twitter scans.

While more information is becoming available on the Internet and more and more books are being published, even on topics that are relatively new, one aspect that continues to inhibit us is the inability to find concise technology overview books.

We are usually faced with two options: read several 500+ page books or scour the web for relevant blog posts and other articles. Just as everyone else who has a job to do and customers to serve, we find this quite frustrating.

The *Succinctly* series

This frustration translated into a deep desire to produce a series of concise technical books that would be targeted at developers working on the Microsoft platform.

We firmly believe, given the background knowledge such developers have, that most topics can be translated into books that are between 50 and 100 pages.

This is exactly what we resolved to accomplish with the *Succinctly* series. Isn't everything wonderful born out of a deep desire to change things for the better?

The best authors, the best content

Each author was carefully chosen from a pool of talented experts who shared our vision. The book you now hold in your hands, and the others available in this series, are a result of the authors' tireless work. You will find original content that is guaranteed to get you up and running in about the time it takes to drink a few cups of coffee.

Free forever

Syncfusion will be working to produce books on several topics. The books will always be free. Any updates we publish will also be free.

About the Author

Ed Freitas works as a consultant on software development applied to customer success, mostly related to financial process automation, accounts payable processing, invoice data extraction, and SAP integration.

He has provided consultancy services, engineered, advised, and supported various projects for global names such as Agfa, Coca-Cola, Domestic & General, EY, Enel, Mango, and the Social Security Agency, among many others.

He's also been invited to various companies such as Shell, Capgemini, Cognizant, and the European Space Agency.

He was recently involved in analyzing 1.6 billion rows of data using Redshift (Amazon Web Services) in order to gather valuable insights on client patterns. He holds an M.S. in computer science.

He enjoys soccer, running, traveling, life hacking, learning, and spending time with his family. You can reach him at http://edfreitas.me.

Acknowledgements

Many thanks to all the people who contributed to this book, and to the Syncfusion team that helped this book become a reality—especially Tres Watkins and Darren West.

The manuscript manager and technical editor—Darren West and James McCaffrey—thoroughly reviewed the book's organization, code quality, and overall accuracy. Thank you all.

Introduction

Salesforce.com is a cloud computing company headquartered in San Francisco, California. Its main product is a Customer Relationship Management (CRM) software that runs in the cloud. It's one of the world's best-known Software-as-a-Service (SaaS) providers.

Force.com is a Platform-as-a-Service (PaaS) that allows developers (and non-developers) to create multitenant (single-instance software that runs on a server and serves multiple tenants) add-on applications that integrate into the main Salesforce application. Force.com applications are hosted on Salesforce's infrastructure.

So, how are they related? Salesforce.com is built on the Force.com platform—from a technical standpoint they are more or less interchangeable. Force.com applies to the infrastructure and codebase that is the foundation for the entire Salesforce solution.

With Force.com, anyone can build powerful enterprise apps without writing too much code—which goes beyond the Salesforce standard CRM app.

Both developers and business users can create powerful apps, workflows, and data schemas easily by using various built-in APIs such as Visualforce, Lightning, and other Salesforce platform components.

Although it is possible to create enterprise apps inside Force.com without writing a lot of code, the platform also provides a very powerful programming language that is syntactically very close to Java and C#—called Apex. There are also powerful third-party libraries such as JSforce, which allow you to write Force.com apps using JavaScript.

Apex is strongly typed and object-oriented. It allows developers to execute flow and transaction control statements on the Force.com platform in conjunction with calls to Force.com APIs. Writing Apex feels like writing Java or C# and acts somewhat like database-stored procedures.

Apex enables developers to add business logic to most system events, including button clicks, related record updates, and pages.

I'm mostly a .NET developer, and when I started exploring Force.com I was a bit skeptical of its capabilities, mostly because of the marketing buzz that surrounds everything to do with Salesforce.

I wondered if it was actually possible to seamlessly create business applications with little or no code. As soon as I started to delve into Force.com, I was pleasantly surprised to discover that the marketing hype was indeed backed by a solid and incredibly flexible platform—which does in fact allow anyone to create apps that require little or no code.

To put this into perspective and in very simple terms, Force.com is a relational database running in the cloud with a fancy user interface. It provides a lot of pre-built components and services ready to be used by simply pointing and clicking.

Force.com abstracts much of the complexity behind a relational database and makes it easy to create apps based on objects that are provided out-of-the-box with a Salesforce subscription.

It is even possible to create custom objects and get them to work seamlessly with the standard objects provided by the platform.

In many ways, I think of Force.com as an evolution to Microsoft Access or Visual Studio LightSwitch, but which is hosted in the cloud.

Force.com is not only a very flexible and interesting platform for developers and business users, it is also an incredible career and business opportunity for anyone willing to jump into enterprise app development and automation.

People with Force.com know-how are being sought all over the world—with incredibly well-paid wages. Businesses are moving into the cloud at a very fast pace and demand for Salesforce solutions keeps accelerating, thus people with Force.com expertise are well positioned to take advantage of this opportunity.

If you are a technical business user or an enterprise developer on any other platform, your value in the market will significantly increase if you spare a bit of time to learn Force.com.

In this e-book, we will explore some of the most important and fundamental aspects of Force.com so that in a relatively short time you can feel comfortable creating business solutions on this platform.

We won't cover all the topics of Force.com development, but those that are more related to developers from other backgrounds, such as those coming from SQL Server, .NET, and JavaScript.

The examples should be easy to follow, fun, and give you a reasonably good coverage and perspective of what is possible to achieve with this amazing platform. Thank you for reading!

Chapter 1 Force.com Overview

Introduction

Nowadays we hear the term Platform-as-a-Service (PaaS) being used more frequently by businesses, IT specialists, and developers. Furthermore, we also hear a lot about Software-as-a-Service (SaaS) and Infrastructure-as-a-Service (IaaS).

But what does this all really mean? Basically, all this fits into cloud computing, which has to do with renting compute resources on demand in a very elastic and scalable way. Although these three terms seem very alike, let's try to differentiate one from the other.

Consider IaaS as the place to run software, SaaS as a way to use software, and PaaS as a place to write software. By establishing these definitions, we can quickly see the differences between each and what binds them together: software—running in the cloud.

By using PaaS platforms, we are able to build and publish applications to a fabric, whether it is .NET, Java, or Force.com—the bottom line is that we know this fabric will run the software we are writing under a controlled and typically multitenant environment in which multiple users share resources together.

In PaaS platforms, scalability and maintainability are already built-in and provided out-of-the-box, which means that there's no exposure to the underlying infrastructure. Some offer additional services like data, caching, messaging, and one or more APIs, which encourages developers to write integrations and also enables automation.

Having said this, the Salesforce Sales Cloud and CRM are ready-to-use sales automation and customer service products, whereas Force.com is the underlying platform that can be used to build a variety of data-driven business applications—hosted on the main Salesforce.com infrastructure.

In this chapter, we'll explore the key concepts behind Force.com as a platform, how it compares to other PaaS offerings, and check its primary services. Following that, we'll start by creating an account and lay the foundation for the app that we will build throughout this book.

Sounds exciting, right? Let's not delay this any further!

Key concepts

First, we'll explore some key concepts that will help us to get started. One fundamental concept to understand is that Force.com does not expose the underlying infrastructure.

On other cloud platforms, you might have a sense of what the infrastructure is and know how many instances your app is running on, but that doesn't happen with Force.com—there's no machine provisioning or awareness of how an application is hosted, provisioned, or distributed.

Another key concept is that Force.com was designed for point-and-click application design, even though coding is supported. It is possible to create objects, fields, and their respective validations and layouts without writing a single line of code.

Force.com is multitenant all the way, which means that it is the same software instance and the same database for all users, running on the same application servers. This all scales out seamlessly, with built-in protection preventing users from taking each other down or hitting collisions on resource usage—because a multitenant architecture doesn't do this automatically.

This is all possible because Force.com is built on top of a relational database, which is the core foundation of this flexible and elastic fabric. There are no indexes to manage or table partitions to optimize—all of this is taken care of automatically.

Furthermore, Force.com uses multifaceted permissions policies, which means that there are different layers of security built in into the platform—basically, application and database-level security regarding who may access certain records and who may not.

Primary services

These are the services that you get out-of-the-box, which you don't have to write yourself. That might not sound like a big deal, but it's a huge factor in getting something out the door as fast as possible. So, let's quickly explore them:

Database: The first and possibly most important service is the database—this is the heart and soul of a Force.com application. This is where all the metadata is stored, all the objects, all your records.

In essence, everything that makes up your application is stored in the database. Objects can be built using relationships, constraints, validations, and auditing—which is all taken care of behind the scenes, and this database can be queried using a very familiar SQL-like syntax.

UI: The second service is the User Interface, which allows a drag-and-drop experience or the possibility to create custom pages using Visualforce or Lightning.

Reporting: This is an out-of-the-box flexible reporting service that allows you to build standard and custom reports very easily.

Application Logic: This service relies on configuration for validation rules and code for more advanced custom logic.

Workflow: This service allows you to model out business process sequences, such as an approval or a collection process, without having to write code, and in an asynchronous way.

Security: This service allows you to enable application-level security and use features like Single Sign-On (SSO).

Integration: This service provides a set of APIs, both SOAP- and REST-based, that allows you to integrate with any of the Force.com services from other applications or external service providers.

All of these primary services give us a good starting point and various options to start working with Force.com. Applications like the Salesforce Sales Cloud or CRM are built on top of these services, which are available to everyone with Force.com—pretty cool!

Force.com and other PaaS offerings

Force.com is probably the most abstracted of all the PaaS offerings that exist nowadays. This is because Salesforce did a very good job of abstracting much of the underlying complexity in order to allow developers and users to focus mostly on the business logic behind their applications.

Force.com is quite similar to Dynamics 365, which is a configurable platform, mostly related to CRM and financial solutions, that leverages .NET.

AWS Elastic Beanstalk is a service from Amazon Web Services that allows you to upload your code and automatically handles deployment, capacity provisioning, load balancing, auto-scaling, and application health monitoring.

Other PaaS services include the Google Cloud Platform, Cloud Foundry, Heroku (owned by Salesforce), and Microsoft Azure.

The main difference between Force.com and these services is that Force.com provides more out-of-the-box primary services that allow business applications to be written using core objects that are part of its underlying relational database, but without necessarily having to write code—at least for the object creation and validation part.

I’m not implying that Force.com is better than any of the other platforms previously mentioned—in several of my other books, I have used services from Azure and the Google Cloud Platform, which have provided a rock-solid foundation for the apps that were developed as examples for those books. Those platforms are amazing.

What I’m trying to convey is that Force.com was developed with the intention that users would be able to get business applications off the ground very quickly without having to worry about database and framework integrations—this is clearly marketed on their main website.

When should you choose Force.com?

I think this is a great question, and ultimately what should drive your decision to use this platform or not.

If you want to build an application that:

- Is data-driven and needs relational data
- Involves a migration of spreadsheet data or a legacy Access- or FileMaker-based application
- Requires a user interface that is data-entry oriented and fits perfectly fine with forms, grids, and wizards styles for reviewing data

- Can benefit from user collaboration
- Includes team-based activities
- Is task based
- Requires fine-grained security accesses and controls
- Requires reports on transactional data
- Requires manually entered data, which is frequently used
- Does not require bulk loads of data, which are CPU intensive
- Does not require high performance computing, such as running Hadoop jobs for big data analysis

If you want to build an application with any of these requirements, Force.com is a great platform for you, as it is focused on applications that use relational data and require collaborative team inputs and tasks.

So now that we've talked about how Force.com stacks up against other PaaS services and when it should be used, let's get started by creating an account.

Creating an account

In order to create an application on Force.com, we must create an account. Let's navigate to the Force.com for Developers main page and click on **Sign Up** at the top.

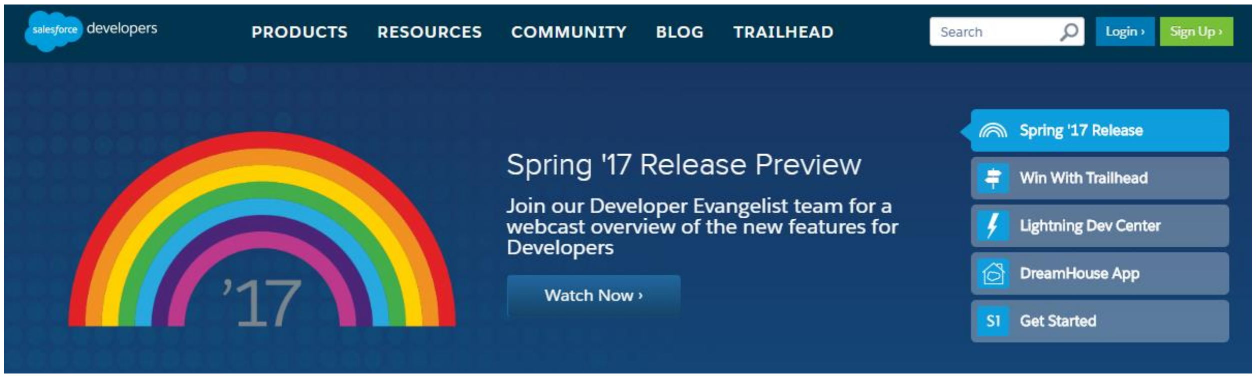


Figure 1-a: The Force.com for Developers Main Page

Keep in mind that Salesforce is a very marketing-oriented company, and, therefore, they are constantly developing their sites, so it's possible that the Force.com for Developers main page might change or look different by the time you read this.

Nevertheless, rest assured that they'll make sure it's easy to navigate, so you'll be able to find your way around.

Once you have clicked on **Sign Up**, you will be redirected to the sign-up page, where you will be asked to enter and submit your personal details.

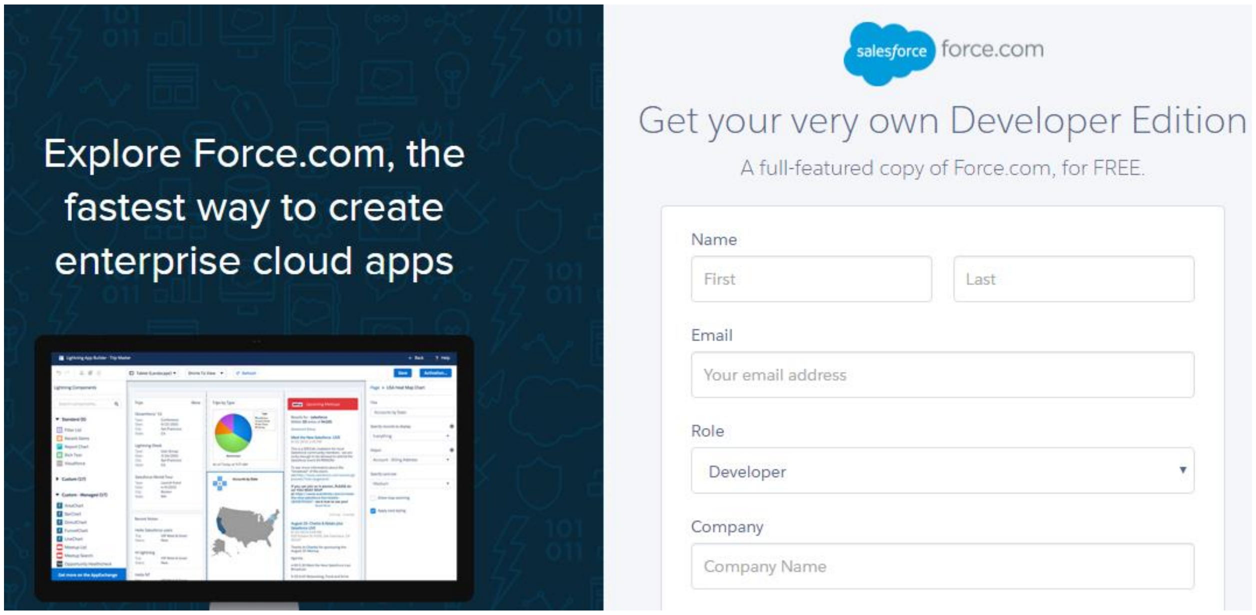

Explore Force.com, the fastest way to create enterprise cloud apps
salesforce force.com
Get your very own Developer Edition
A full-featured copy of Force.com, for FREE.
Name
First
Last
Email
Your email address
Role
Developer
Company
Company Name

Figure 1-b: The Force.com Sign-up Page

There are two very important fields that should have different values. One is your actual email (your real email address), and the other is your **Username**, which you will later use to log in to your Force.com organization.

You'll have to provide your real email address, as this is where you'll receive an email to confirm your account. However, the **Username** field (which needs to be specified in the form of an email address) can be a dummy email address or a secondary email address from your organization. So, the **Email** and the **Username** fields should be different—please keep this in mind.

Once you have entered the values of these required fields, accept the **Master Subscription agreement**, then click **Sign me up**.

Notice that when you sign up for Force.com, you are getting what is known as a Force.com org (a Developer account), which gives you the full functionalities of the Salesforce.com platform, including many pre-built apps such as Sales, Services, Marketing, Community, Content, and Chatter.

In most cases, you will be automatically redirected to your new development org—which loads with the Lightning user interface—and you'll be able to start working. However, in some occasions, that might not be the case. If you are not automatically redirected to your new org, there are two other possibilities.

One is that you receive an email from Salesforce in which you are requested to verify your account. If that is the case, then please follow the steps described in that email to get your account verified.

The other possibility is that you will be logged in but remain on the main Force.com for Developers page, and you will no longer see the Login or Sign Up buttons at the top of the page, but instead a button with your name on it that gives you the ability to go into your account settings and to navigate to your developer org.

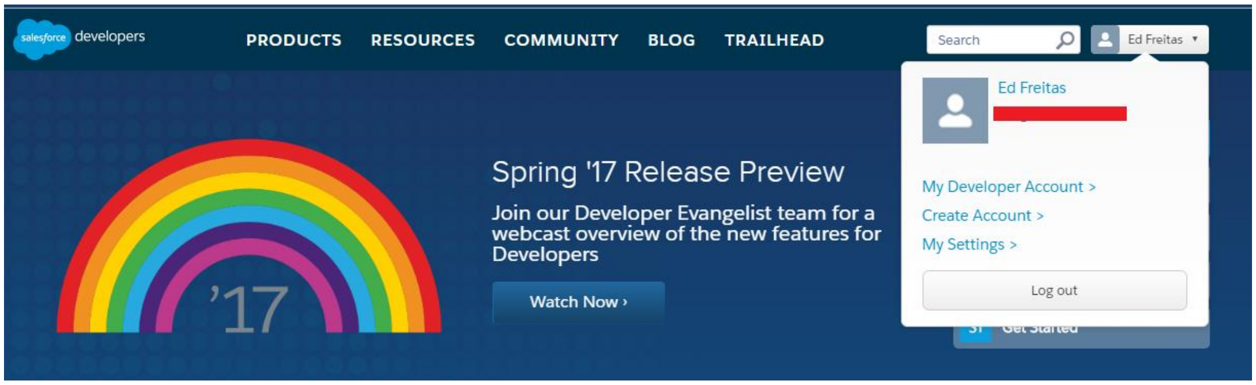


Figure 1-c: Logged in to the Force.com Developers Page

If this is the case, you can reach your developer org by clicking on the **My Developer Account** link from the dropdown menu, under the button with your name on the main page.

Once you click on the **My Developer Account** link, you might be requested to enter your Force.com username and your password, which you should have either received or created through the sign up or verification process. The login screen looks like this:

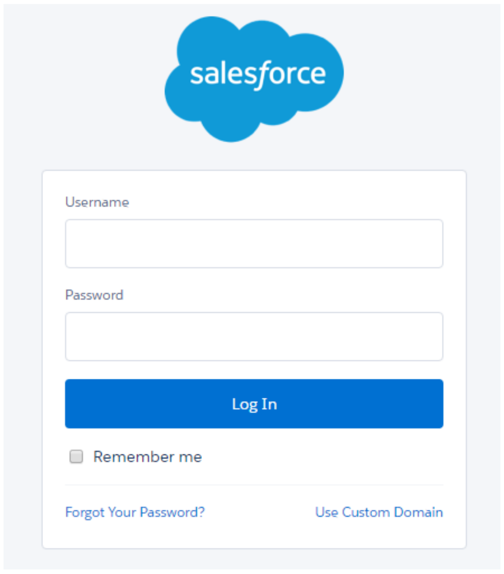
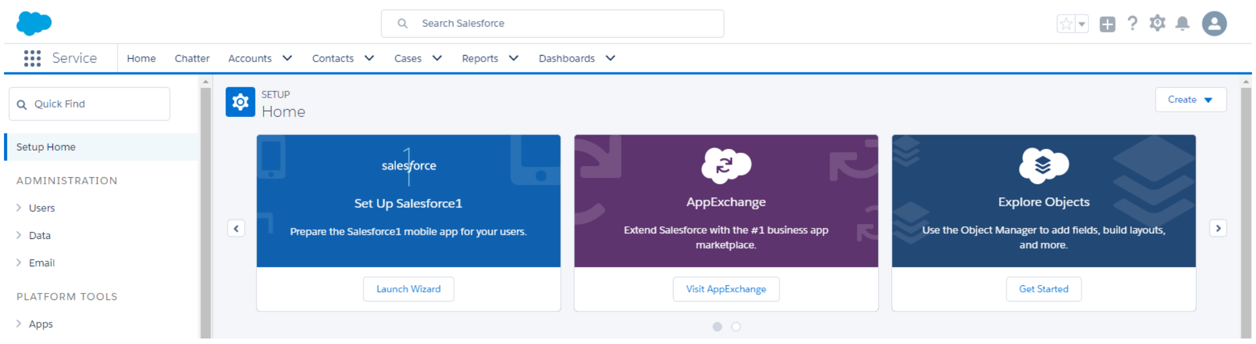


Figure 1-d: Force.com Org Login Page

Enter your details and click **Log In**, and this will take you to your developer org, which should look similar to the following screen.



Prepare the Salesforce1 mobile app for your users.

Extend Salesforce with the #1 business app marketplace.

Use the Object Manager to add fields, build layouts, and more.

Figure 1-e: The Force.com Development Org (Lightning Experience)

I want to highlight that Salesforce is constantly improving their sites and adding extra security verification steps to their login mechanism, so the steps explained might slightly differ by the time you read this. However, you should still find it intuitive enough to set up an account, log in, and start working for the first time on your Force.com org.

Awesome! With this in place, we are ready to start creating our Force.com app.

Creating our app

One of the really cool things I love about Force.com is that immediately after creating an account, we can go ahead and create an application. There's really nothing stopping us from doing that, so that's exactly what we are going to do now.

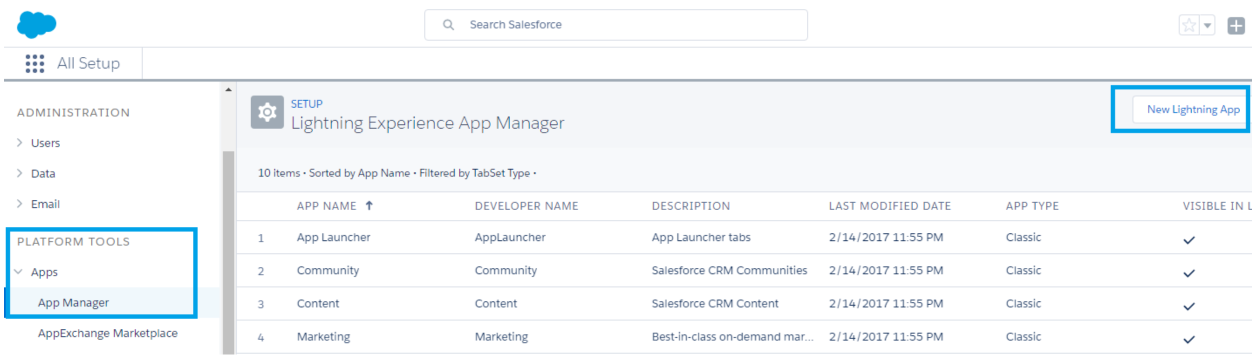
The app we'll create is a basic Customer Success application that will track software maintenance renewals.

In order to customize the app, we'll also need to create some custom objects—we will do this in Chapter 2. We'll create **Client** and **Renewals** custom objects.

This app will also require standard Force.com objects that are used within the support ticketing application that comes out-of-the-box with Salesforce, such as **Cases** and **Accounts**.

Although the **Account** object contains lots of useful fields for managing a customer account, we need extra fields that will be specific for managing clients with software renewals, which we should keep separate from other applications that use the **Account** object, such as the CRM and Sales apps. This is why we'll create a separate **Client** object for our Customer Success app that will be linked to the **Account** object but will still be independent.

In order to get started, on the left side of your developer org, click **PLATFORM TOOLS** > **Apps** > **App Manager**. Once there, click **New Lightning App**.



10 items • Sorted by App Name • Filtered by TabSet Type •

	APP NAME	DEVELOPER NAME	DESCRIPTION	LAST MODIFIED DATE	APP TYPE	VISIBLE IN L
1	App Launcher	AppLauncher	App Launcher tabs	2/14/2017 11:55 PM	Classic	✓
2	Community	Community	Salesforce CRM Communities	2/14/2017 11:55 PM	Classic	✓
3	Content	Content	Salesforce CRM Content	2/14/2017 11:55 PM	Classic	✓
4	Marketing	Marketing	Best-in-class on-demand mar...	2/14/2017 11:55 PM	Classic	✓

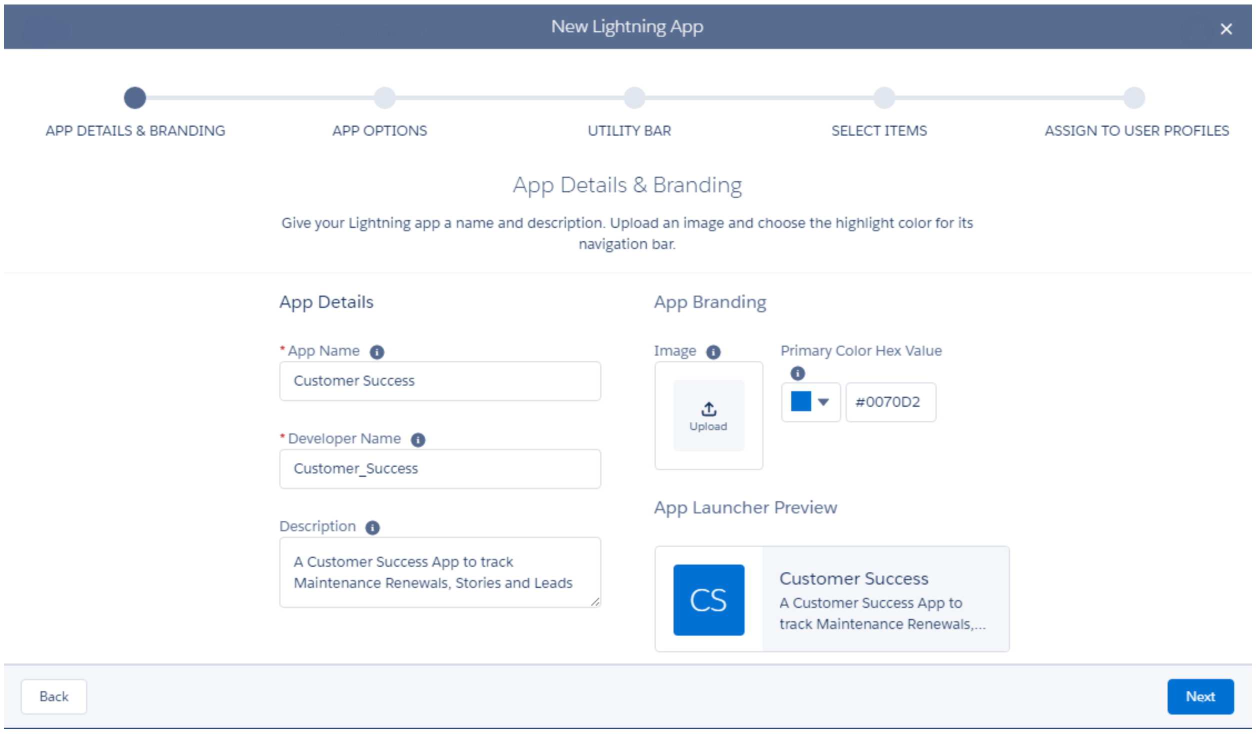
Figure 1-f: Lightning Experience App Manager

Notice that the steps described are specific to Lightning Apps and not the Classic user interface for Force.com apps—however, the process is more or less the same, although the screens, user interface, and layouts are different.

Going forward, Lightning is the newest Force.com user interface experience, and is being actively promoted by Salesforce.

Once we've done that, an app creation wizard will guide us through each of the stages. This will allow us to enter the app details, select which default objects we want to use, and assign any user profiles.

Let's start by entering the **APP DETAILS & BRANDING** panel.



App Details & Branding

Give your Lightning app a name and description. Upload an image and choose the highlight color for its navigation bar.

App Details

*App Name: Customer Success

*Developer Name: Customer_Success

Description: A Customer Success App to track Maintenance Renewals, Stories and Leads

App Branding

Image: Upload

Primary Color Hex Value: #0070D2

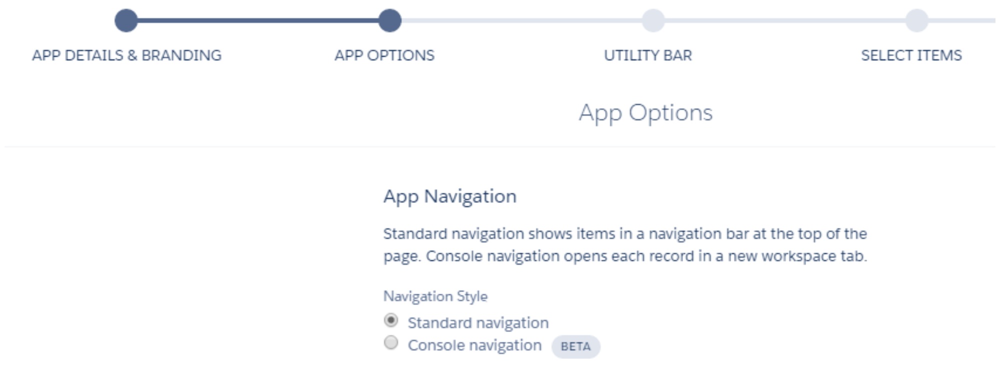
App Launcher Preview

CS Customer Success
A Customer Success App to track Maintenance Renewals,...

Figure 1-g: Lightning Experience App Wizard (APP DETAILS & BRANDING)

Give your app a name—I've called mine Customer Success. You'll notice that Force.com automatically creates the **Developer Name**, which is usually the same name you've chosen with an underscore between words.

Don't forget to enter a meaningful description. Once you have done that, click **Next** in the lower-right corner. In the next screen, the wizard will ask which type of navigation you would like for your app—choose the standard one, and then click **Next**.

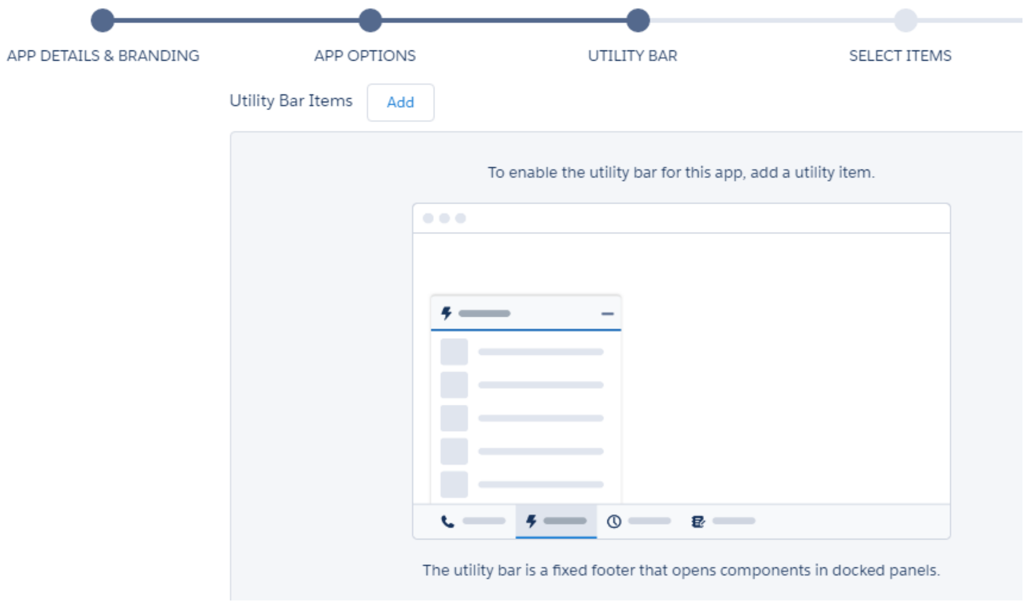


Standard navigation shows items in a navigation bar at the top of the page. Console navigation opens each record in a new workspace tab.

Navigation Style
(●) Standard navigation
( ) Console navigation BETA

Figure 1-h: Lightning Experience App Wizard (APP OPTIONS)

The next screen is quite interesting because it allows us to add **Utility Bar** items to our app. The utility bar is a fixed footer that opens components in docked panels. Let's have a look.



To enable the utility bar for this app, add a utility item.

The utility bar is a fixed footer that opens components in docked panels.

Figure 1-i: Lightning Experience App Wizard (UTILITY BAR)

Personally, I like to have access to recent items that I added or modified when working with a Force.com application, so I'll add a Recent Items component to my app's **Utility Bar** by clicking **Add**.

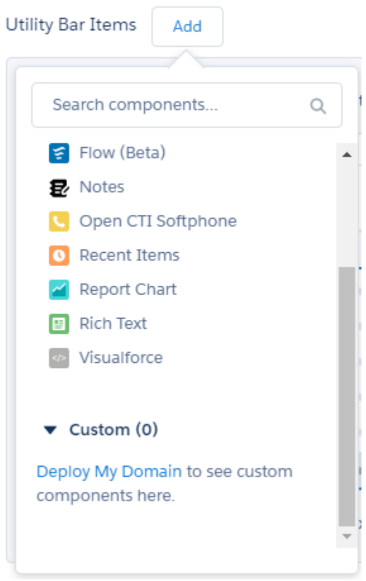
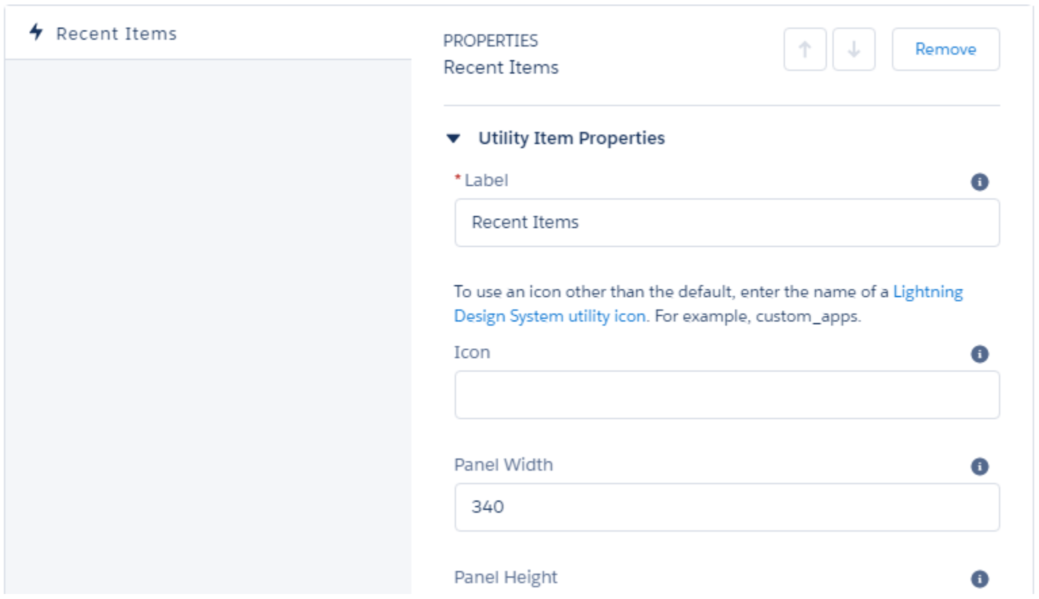


Figure 1-j: Adding a Utility Item (Force.com Component)

From the list, select **Recent Items**. You may add another if you wish. I'll add only this one for my app, by clicking on the **Recent Items** option. Once you have done that, you'll see the following information.



*Label

Recent Items

To use an icon other than the default, enter the name of a Lightning Design System utility icon. For example, custom_apps.

Icon

Panel Width

340

Panel Height

Figure 1-k: Utility Item Properties (Recent Items Component)

These are the properties of the **Recent Items** component. Most of the values you can leave as is; however, there are a couple that I would recommend you customize, which I will do for my app.

In order to do that, scroll down a bit and notice the **Objects** property and the **Number of Records to Display** value.

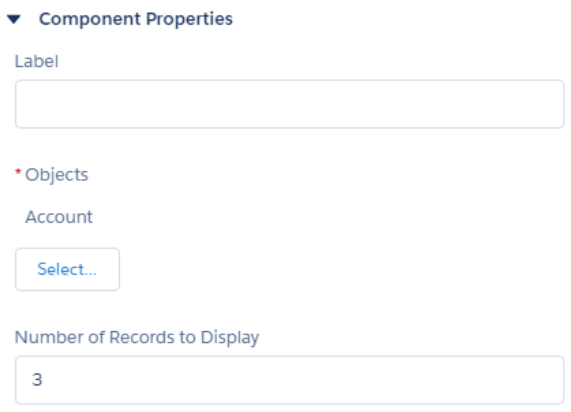


Figure 1-l: Components Properties (Recent Items Component)

Click **Select**. Once you've done that, a popup window will appear that you can use to select which standard Force.com objects will be displayed on the **Recent Items** list. By default, the **Account** object is selected, and I've also added the **Case** and **Contact** objects.

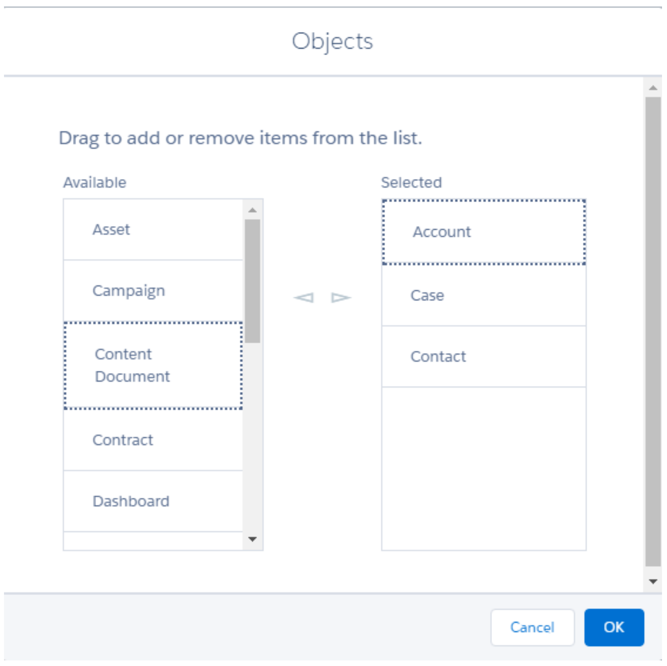
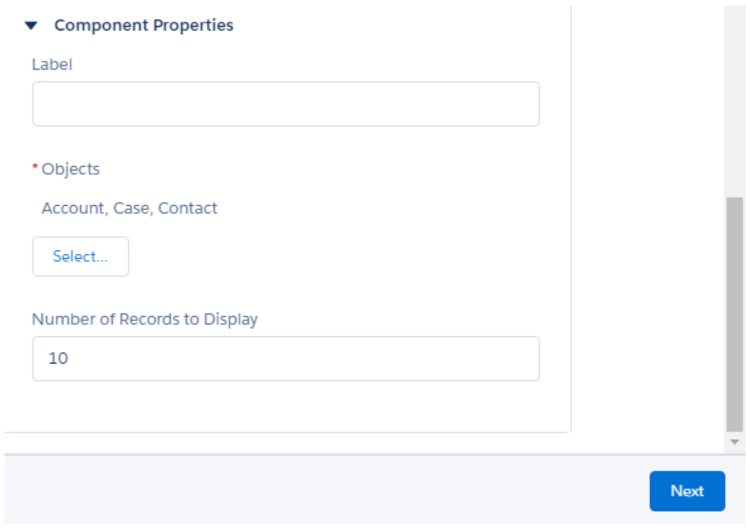


Figure 1-m: Selecting Objects for the Recent Items List

Once you have selected the components, click **OK**. Personally, I like to see at least the 10 most recent items I've worked with, so I'll modify the **Number of Records to Display** value of **10**.


Component Properties

Label

*Objects

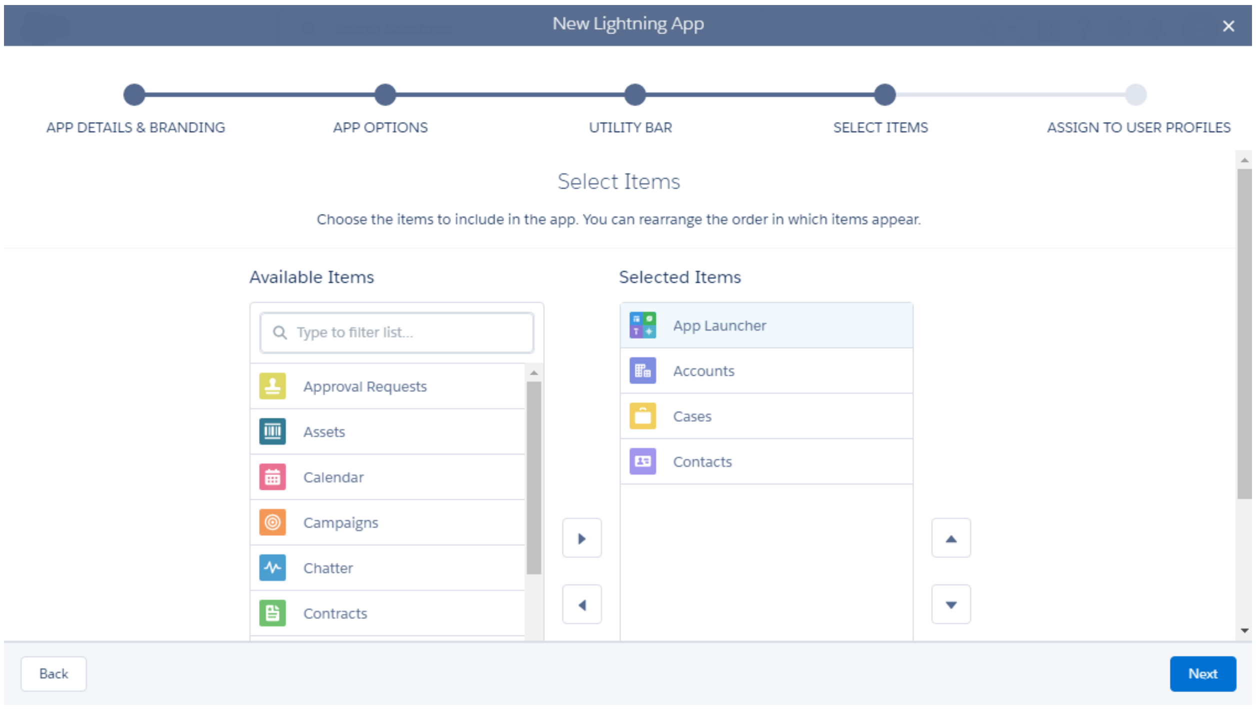
Account, Case, Contact

Number of Records to Display

10

Figure 1-n: Recent Items Components Properties Set

Once this has been done, in order to continue the wizard process, click **Next**. At this point, we've moved past **ADD OPTIONS** stage, and we are now on the **SELECT ITEMS** stage of the wizard.


New Lightning App

APP DETAILS & BRANDING — APP OPTIONS — UTILITY BAR — SELECT ITEMS — ASSIGN TO USER PROFILES

Select Items

Choose the items to include in the app. You can rearrange the order in which items appear.

Available Items	Selected Items
Approval Requests	App Launcher
Assets	Accounts
Calendar	Cases
Campaigns	Contacts
Chatter	
Contracts	

Figure 1-o: The Select Items Stage

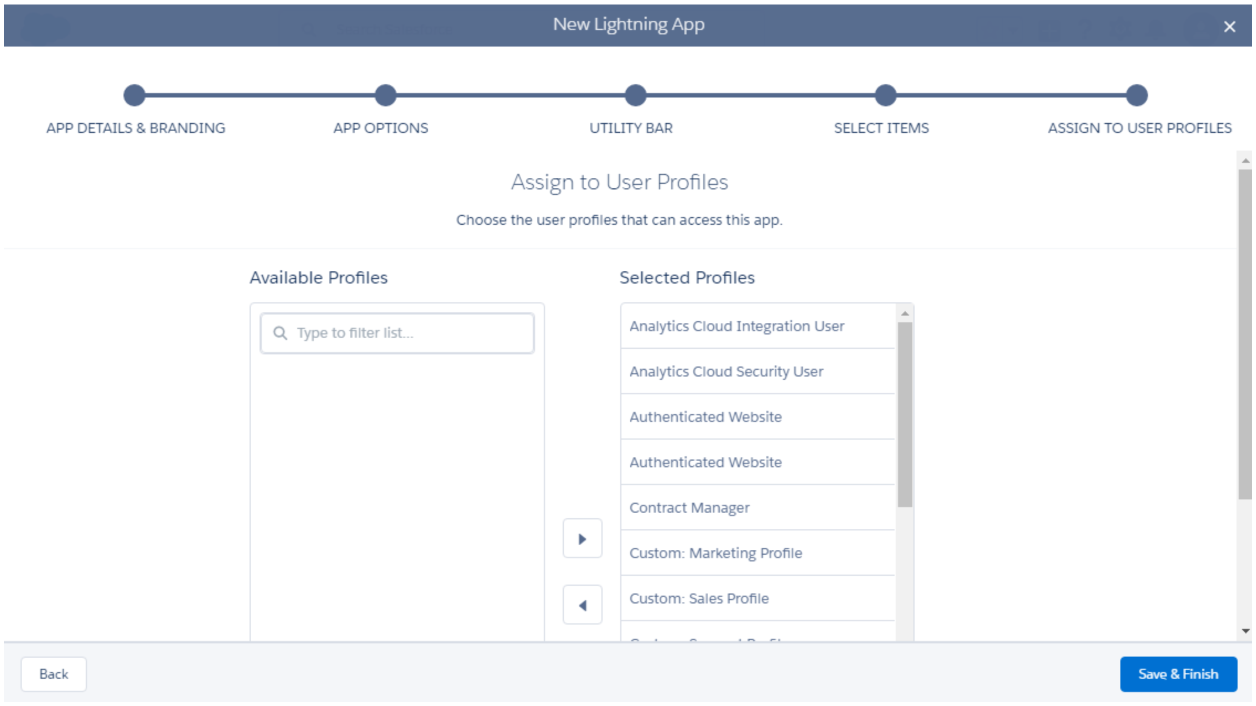
In this stage, you can choose which objects you would like to see (include) when your Customer Success app loads.

These will correspond to tabs that will allow you to access the data associated with each of those objects. Think of objects as database tables. In my case, I'm interested in having data from **Accounts**, **Cases**, and **Contacts**, which are all related to a Customer Success application—so I have selected them. In order to continue, click **Next**.

We are almost done with the app creation wizard, but there's one last step, which is to assign user profiles to our app. This step is important, as it will determine which Salesforce user groups will have access to see our Customer Success app when they log in.

Keep in mind that we are Force.com developers, and we are creating a Customer Success application that will be used by regular Salesforce users within our organization, so this is why assigning user profiles is a required step in the app creation process.

For now, let's select all available user profiles and assign them to our app—we can always later modify this in the app's setup. Scroll all the way down to the bottom, and while holding the **Shift** key, select them all, and add them to the **Selected Profiles** list.



Assign to User Profiles

Choose the user profiles that can access this app.

Available Profiles

Type to filter list...

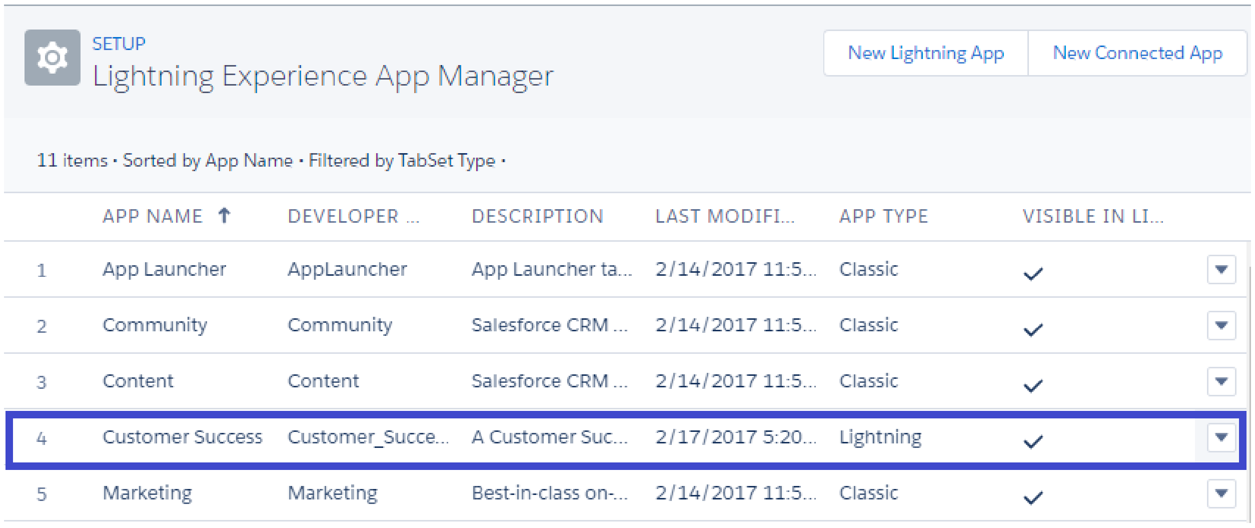
Selected Profiles

Analytics Cloud Integration User
Analytics Cloud Security User
Authenticated Website
Authenticated Website
Contract Manager
Custom: Marketing Profile
Custom: Sales Profile

Figure 1-p: The Assign to User Profiles Stage

Finally, click **Save & Finish** in order to finalize the wizard and create our app. We have now created our Customer Success app template!

We can see it on the **Lightning Experience App Manager** list as follows.



SETUP
Lightning Experience App Manager

New Lightning App | New Connected App

11 items • Sorted by App Name • Filtered by TabSet Type •

	APP NAME	DEVELOPER ...	DESCRIPTION	LAST MODIFI...	APP TYPE	VISIBLE IN LI...
1	App Launcher	AppLauncher	App Launcher ta...	2/14/2017 11:5...	Classic	✓
2	Community	Community	Salesforce CRM ...	2/14/2017 11:5...	Classic	✓
3	Content	Content	Salesforce CRM ...	2/14/2017 11:5...	Classic	✓
4	Customer Success	Customer_Succe...	A Customer Suc...	2/17/2017 5:20...	Lightning	✓
5	Marketing	Marketing	Best-in-class on-...	2/14/2017 11:5...	Classic	✓

Figure 1-q: The Customer Success App Shown in the Lightning Experience App Manager

The fun is just about to start. The wizard has given us a starting point, and it's created a basic app with out-of-the-box and ready-to-use Force.com objects we can already interact with, such as **Accounts**, **Cases**, and **Contacts**, which we added to our app in the SELECT ITEMS stage.

However, in order to make our Customer Success app really useful, we'll need to create custom objects with specific custom fields and all the applicable validation logic—this is what we'll do in the next chapter.

Summary

In this chapter, we've quickly explored what Force.com is, what primary services and capabilities it includes out-of-the-box, and how it compares to other PaaS offerings that exist in the market.

We also got started by signing up for a Developer account and then created the foundation of the Customer Success application we'll be writing throughout this book.

The really cool thing (in my opinion) is that we've already had a sense of how intuitive the platform is—it provides quite a lot of functionality. Furthermore, the new Lightning Experience gives the user interface a modern and responsive look and feel, which is appealing to work with.

In the next chapter, we'll dig deeper into the app by adding custom objects and fields, and we'll create relationships and custom validations, which will be the building blocks of our Customer Success application.

Let's explore what exciting things might lie ahead.

Chapter 2 Client Custom Object

Introduction

Force.com comes with a few objects out-the-box that provide many of the features that are included with some of the built-in Salesforce applications, such as CRM and Sales. I've already briefly mentioned some of these objects, such as **Accounts**, **Cases**, and **Contacts**.

These default objects are amazing because they come with a lot of pre-built fields, validations, and layouts—in essence, a lot of useful functionality to build any new custom app.

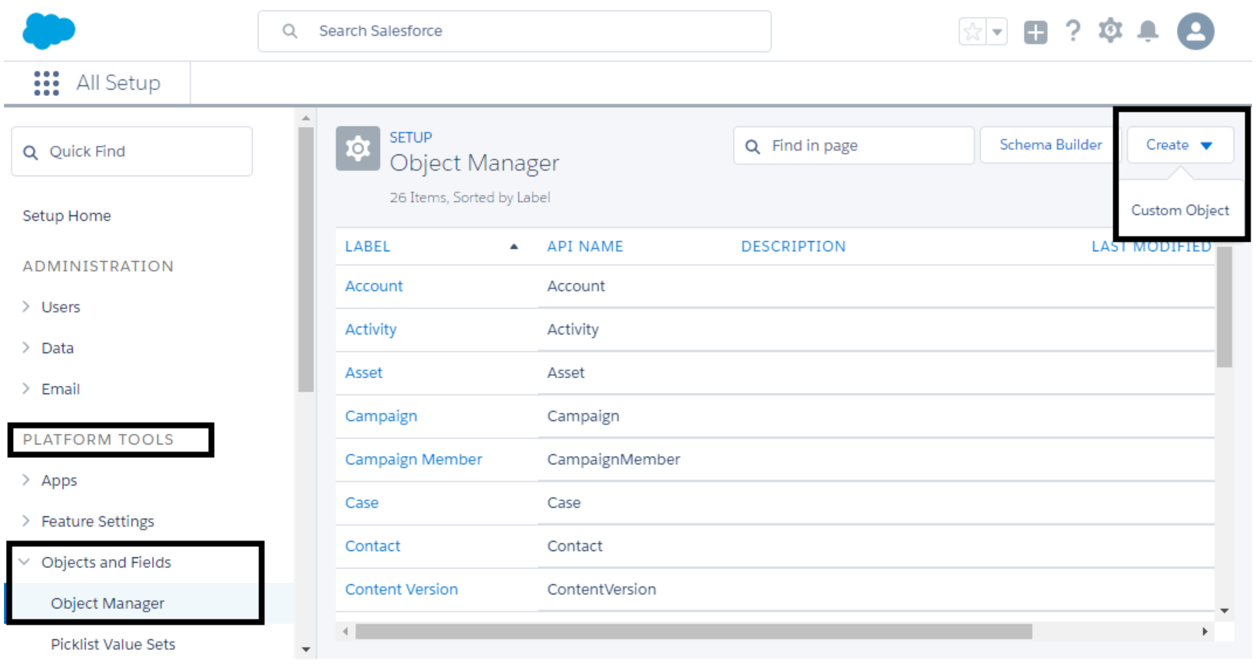
Besides this, Force.com allows us to create our own custom objects, which is very handy when it comes to creating an app. Although it is possible to create an app with only default objects, you'll see that being able to create and use custom objects is a quite a powerful feature.

You can think of objects as database tables, but with extra features like predefined field types, built-in validations, and layout capabilities. Besides this, default and custom objects can be related to one another by master-detail and lookup relationships, just like tables in a relational database.

Objects are the heart and soul of a Force.com app, and custom objects are like clean canvases on which we can sketch out our app. In this chapter, we'll dive into this topic in detail, and we'll be creating the **Client** object for our Customer Success app. Ready, set, go!

Creating custom objects

Within the main **Lightning Experience** screen, on the left side, click **PLATFORM TOOLS** > **Objects and Fields** > **Object Manager**. Once you do this, you'll see the list of available objects that already exist, which should look similar to Figure 2-a.



26 Items, Sorted by Label

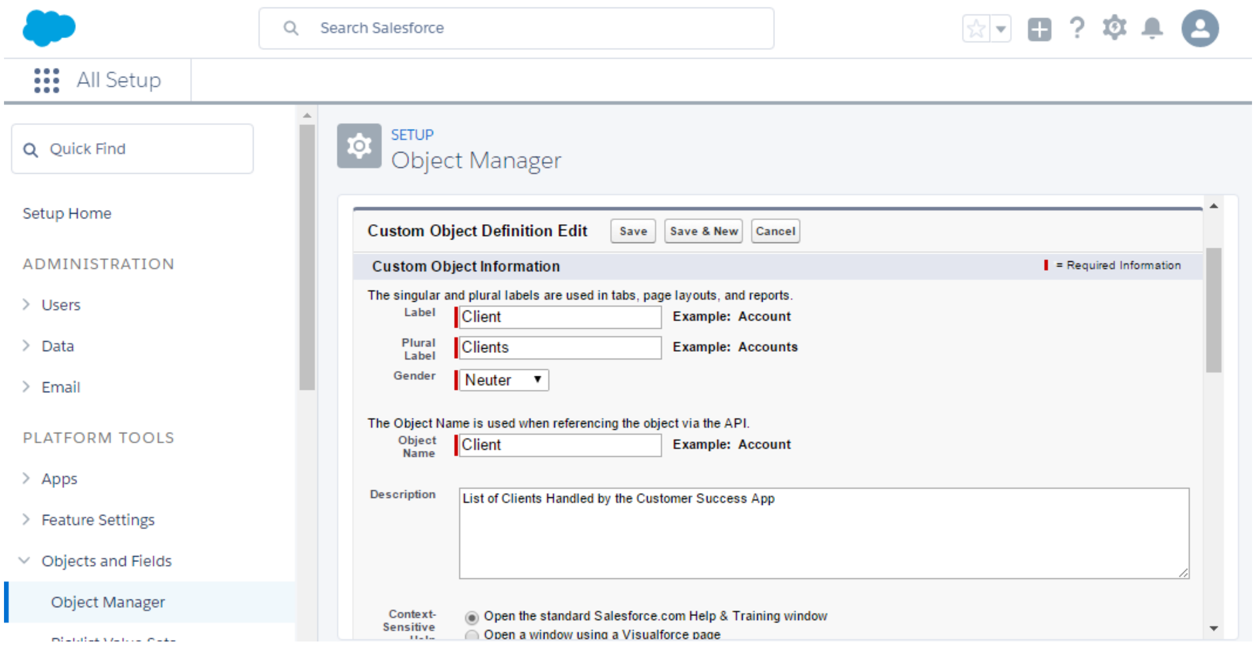
LABEL	API NAME	DESCRIPTION	LAST MODIFIED
Account	Account		
Activity	Activity		
Asset	Asset		
Campaign	Campaign		
Campaign Member	CampaignMember		
Case	Case		
Contact	Contact		
Content Version	ContentVersion		

Figure 2-a: The Object Manager

In order to create a new custom object, click **Create** on the top-right side, and then the **Custom Object** option.

The first custom object we will need for our Customer Success app is the **Client** object. This will be the main object our app will use.

Once we have clicked on the **Custom Object** option, we'll see a screen similar to the following one.



Custom Object Information | = Required Information

The singular and plural labels are used in tabs, page layouts, and reports.

Label	Client	Example: Account
Plural Label	Clients	Example: Accounts
Gender	Neuter	

The Object Name is used when referencing the object via the API.

Object Name	Client	Example: Account
Description	List of Clients Handled by the Customer Success App	

Context-Sensitive Help: [✓ Open the standard Salesforce.com Help & Training window] Open a window using a Visualforce page

Figure 2-b: Creating a New Custom Object: Client

When creating a new object, Force.com will ask us to enter some details about the object we are interested in creating.

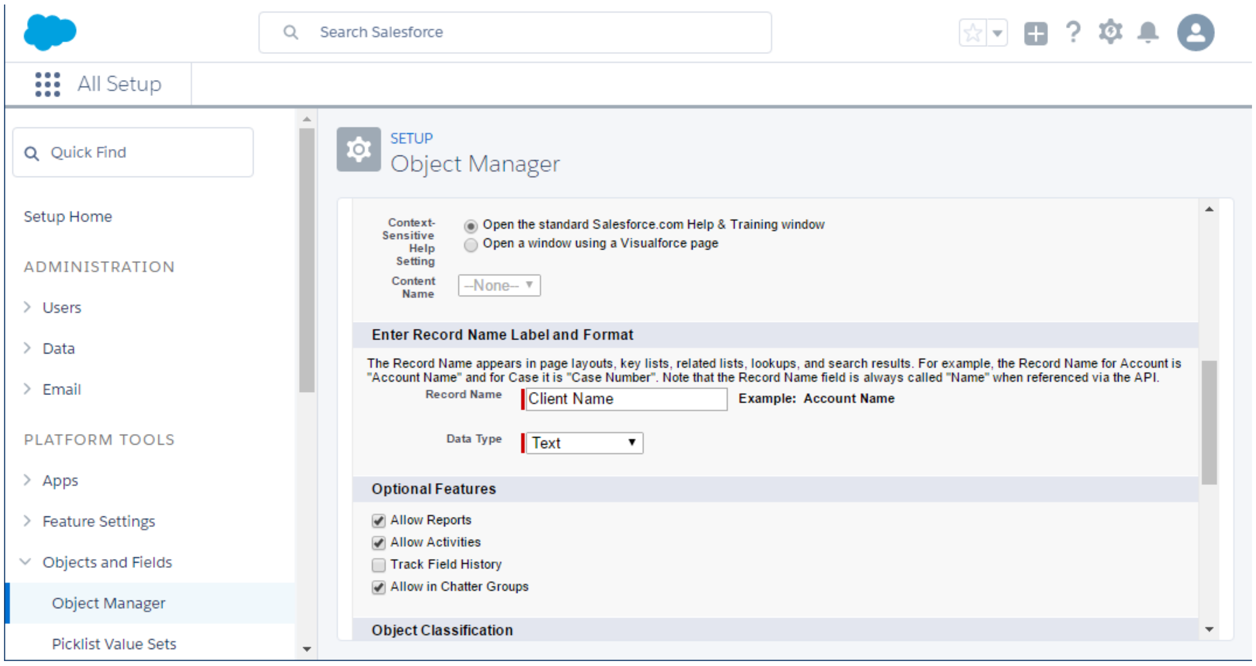
The first detail we are asked to enter is the Label of the object, which is just the caption or text that will visually identify the object on one of the Force.com tabs.

We'll label our object as **Client**. Interestingly, Force.com will always ask us for the Plural Label, which in this case will be **Clients**.

There's a **Gender** field, which we can leave set to the default option and the name of the object itself, which we will also designate as **Client**.

Further to that, it is recommended to enter a **Description**; it is not a required field, but it helps to give some basic information about our object in case we need to come back in the future and make some adjustments, or, alternatively, if someone else needs to revise the app structure later on.

If we scroll down, there are other fields that are both required and optional. Let's explore them.



Context-Sensitive Help Setting: [✓ Open the standard Salesforce.com Help & Training window] Open a window using a Visualforce page

Content Name: --None--

Enter Record Name Label and Format

The Record Name appears in page layouts, key lists, related lists, lookups, and search results. For example, the Record Name for Account is "Account Name" and for Case it is "Case Number". Note that the Record Name field is always called "Name" when referenced via the API.

Record Name: Client Name **Example: Account Name**

Data Type: Text

Optional Features

- [✓] Allow Reports
- [✓] Allow Activities
- [ ] Track Field History
- [✓] Allow in Chatter Groups

Object Classification

Figure 2-c: Creating a New Custom Object: Client (Continued)

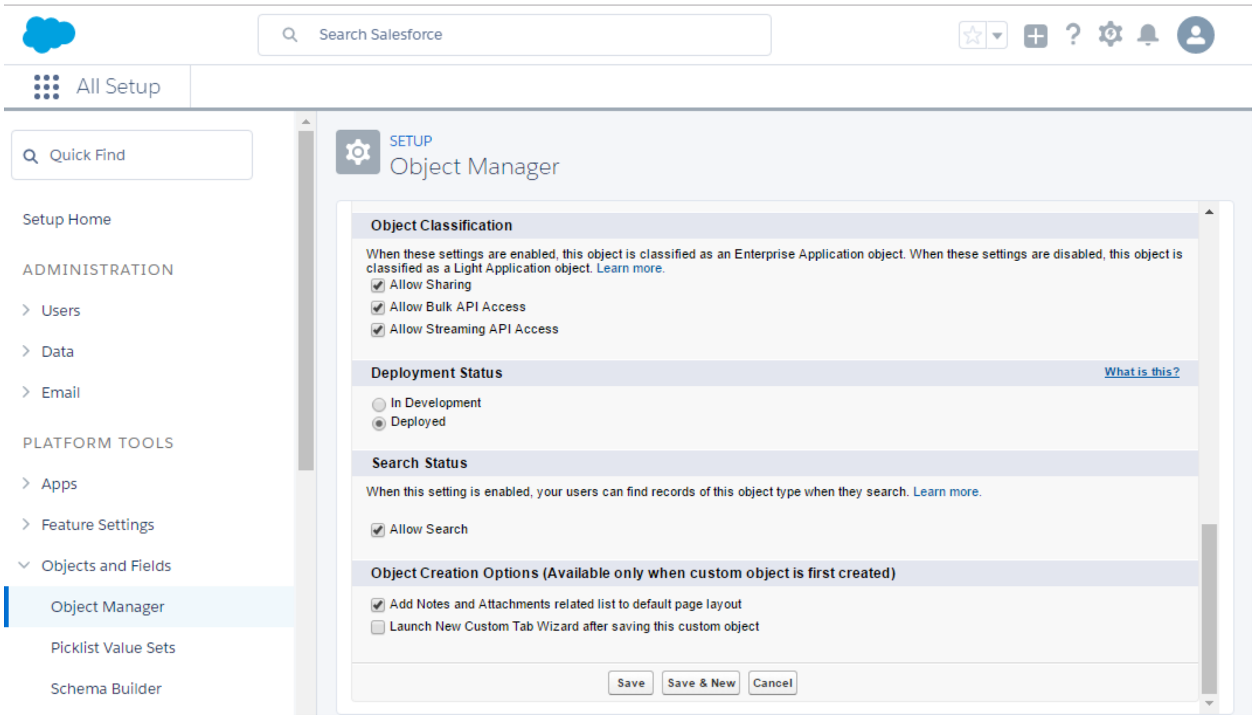
We can leave the **Context Sensitive Help** option as is and notice that for the **Record Name** field, Force.com has chosen for us by default **Client Name** as the main field for our object, and has also set the **Data Type** as **Text**. We can leave this as is.

The **Optional Features** are usually unchecked by default, so I have explicitly selected **Allow Reports**, **Allow Activities,** and **Allow in Chatter Groups**.

I have done this because it is useful to have **Client** object data show up in reports. Also, activities can be added to any **Client** record, and **Client** records can also be fetched when users interact with each other using the pre-built **Chatter** functionality within Force.com.

We don't really need to keep track of any field changes, so we can leave the option **Track Field History** unchecked.

We're not done yet, so let's scroll a bit further down and finish setting up the rest of our **Client** object properties.



Object Classification

When these settings are enabled, this object is classified as an Enterprise Application object. When these settings are disabled, this object is classified as a Light Application object. Learn more.

- [✓] Allow Sharing
- [✓] Allow Bulk API Access
- [✓] Allow Streaming API Access

Deployment Status What is this?

- ○ In Development
- [✓ ●] Deployed

Search Status

When this setting is enabled, your users can find records of this object type when they search. Learn more.

- [✓] Allow Search

Object Creation Options (Available only when custom object is first created)

- [✓] Add Notes and Attachments related list to default page layout
- [ ] Launch New Custom Tab Wizard after saving this custom object

Save | Save & New | Cancel

Figure 2-d: Creating a New Custom Object: Client (Last Part)

The **Object Classification** properties allow the object to be exposed to other Force.com APIs that can be used when creating enterprise applications. So by default, **Allow Sharing**, **Allow Bulk API Access**, and **Allow Streaming API Access** are checked; we can leave this as is—which is recommended, anyway.

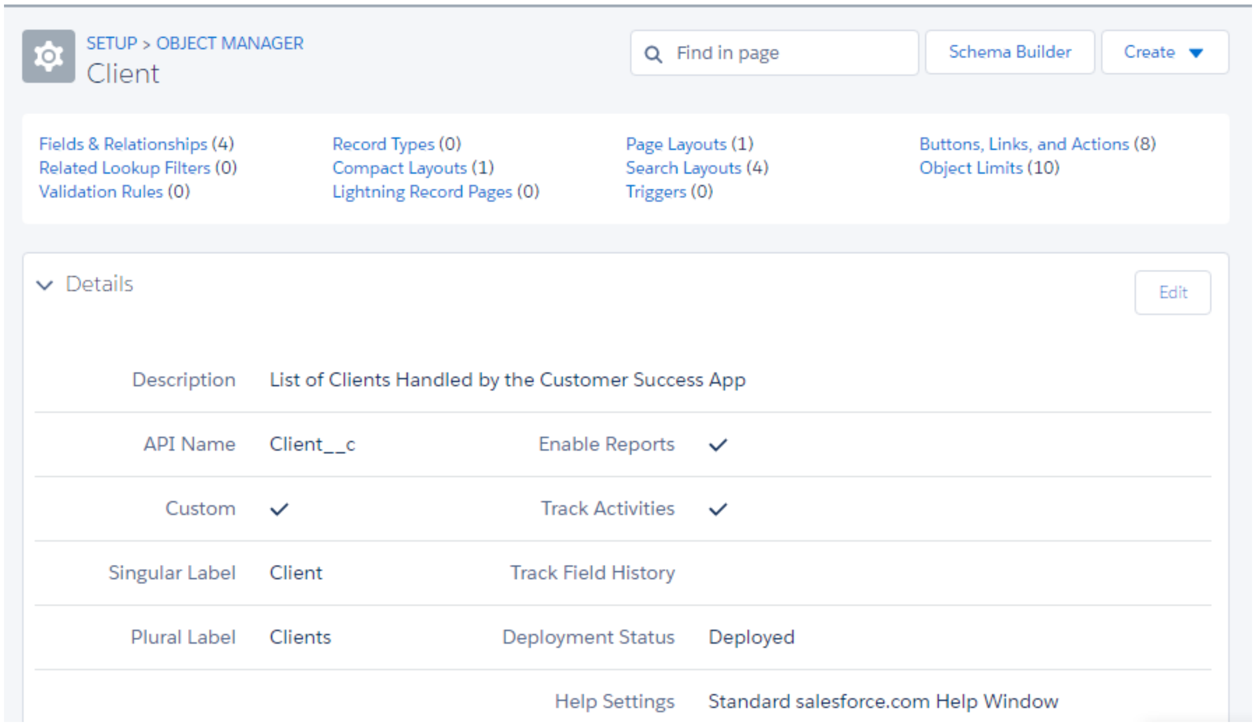
These APIs are beyond the scope of this e-book, but you can read more about them here.

As for the **Deployment Status**, this is set to **Deployed** by default. **Deployment Status** allows us to control when a custom object and its associated custom tab, related lists, and reports become visible to non-admin users.

As for the **Search Status**, we are definitely interested that the data this object will contain can be fully text searchable. By default, this option is unchecked, so it is highly recommended that you select it.

Finally, in regard to the **Object Creation Options**, which is only available when a custom object is first created (not available when a custom object is modified), I usually like to select the **Add Notes and Attachments related list to default layout** option (which is unchecked by default), as it is handy to have a placeholder for adding text notes and attachments (files) to an object.

In order to create the **Client** object, all we have to do is to click **Save**. Once we do that, we'll see the following screen.


SETUP > OBJECT MANAGER
Client

Description	List of Clients Handled by the Customer Success App		
API Name	Client__c	Enable Reports	✓
Custom	✓	Track Activities	✓
Singular Label	Client	Track Field History	
Plural Label	Clients	Deployment Status	Deployed
		Help Settings	Standard salesforce.com Help Window

Figure 2-e: The Newly Created Client Custom Object

This contains the list of properties and all the details that our **Client** object has at this point. From here, we'll add custom fields that define the object (*Client__c*) layout, which is what users will see and interact with.

Now that we've created our **Client** object, we'll need to define what fields we'll use to store data inside of it, so let's go ahead and do that.

Creating custom fields

Creating custom fields in Force.com for any object is like creating fields in a table within a relational database, but on steroids—there's a lot of functionality already baked into the product, mostly predefined field types.

Let's first look at the fields we'll be adding to our **Client** object. The **Field** column on Table 2-a represents the name of the custom field, and **Type** indicates the actual Force.com data type.

Don't worry if some of these data types sound a bit unfamiliar, as they are actually very easy to understand—we'll explore each as we go along.

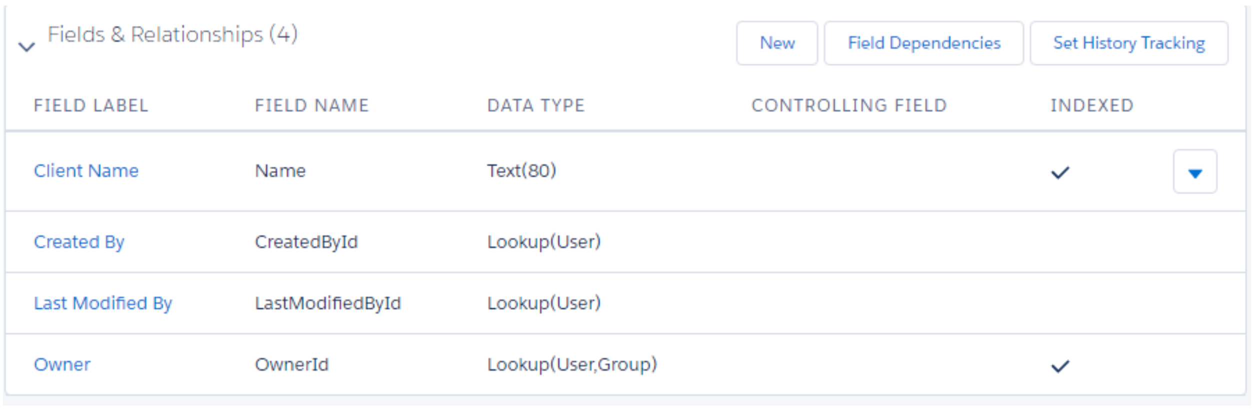
Table 2-a: Fields for Our Client Custom Object

Field	Type
Account	Lookup (on the Accounts object)
Client Name	Text (80) Unique Case Insensitive

Field	Type
CS Manager	Picklist
Versions	Picklist (Multi-Select)
Latest Renewal	Roll-Up Summary (MAX Renewal)
Region	Picklist
Status	Picklist
Upgrade	Formula (Checkbox)

Now that we know which fields we want to add to our **Client** object, let's add one at a time.

We carry on where we left on Figure 2-e, on the **SETUP** > **OBJECT MANAGER** > **Client** object. Scroll a bit down to the **Fields & Relationships** section and notice that by default, four fields have been automatically created—one of them being the **Client Name** that we examined earlier, which is an index field. You can think of it like a Primary Key in a relational database table.



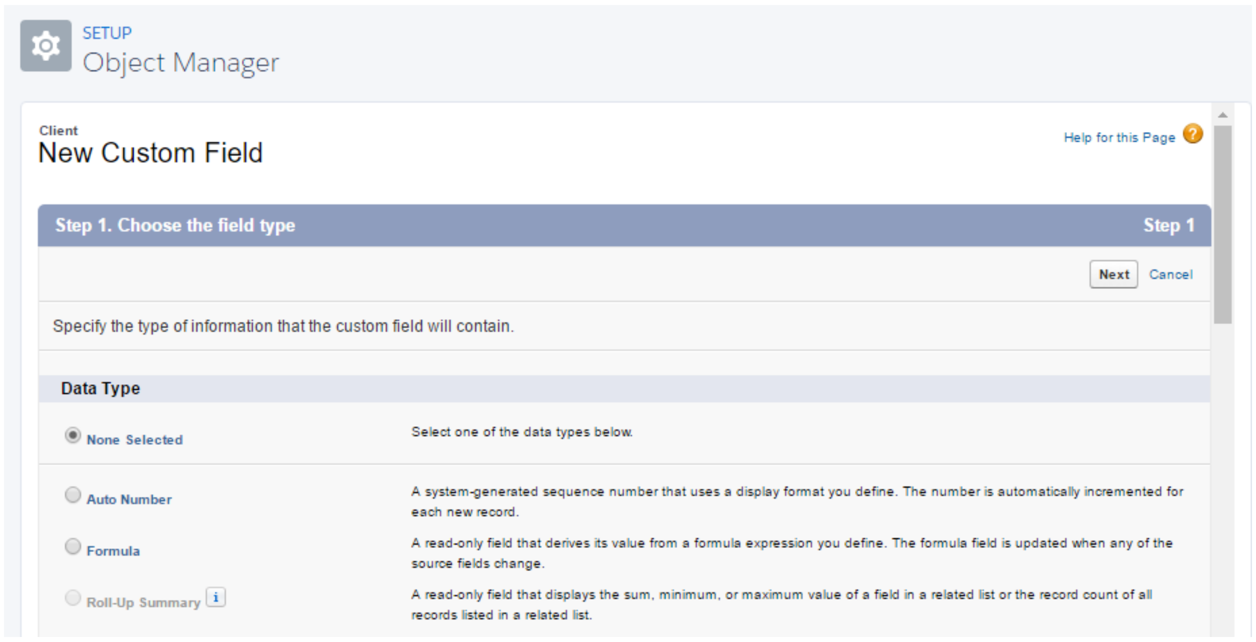
FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
Client Name	Name	Text(80)		✓
Created By	CreatedById	Lookup(User)		
Last Modified By	LastModifiedById	Lookup(User)		
Owner	OwnerId	Lookup(User,Group)		✓

Figure 2-f: Default Fields & Relationships (Client Object)

The other three default fields are: **Created By**, which stores the name of the user that created a **Client** record, **Last Modified By**, which indicates which user last updated a **Client** record, and **Owner**, which indicates which user or group owns the record.

These fields are automatically added when any new object is created. Only the **Client Name** field can be edited, and this can be done by clicking on the arrow button and then on the **Edit** option—however, we won't do this.

Now, let's create our first custom field. In order to do this, click **New** inside the **Fields & Relationships** region. Once you have done that, a screen similar to the following one will appear.


SETUP
Object Manager

Client
New Custom Field

Help for this Page

Step 1. Choose the field type Step 1

Next Cancel

Specify the type of information that the custom field will contain.

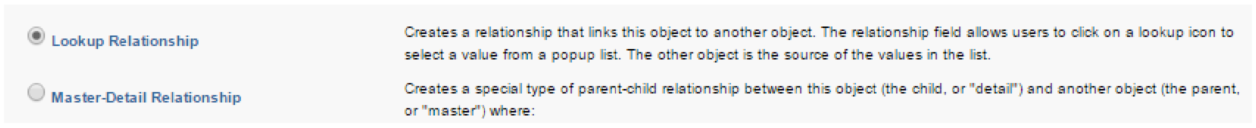
Data Type

● None Selected	Select one of the data types below.
○ Auto Number	A system-generated sequence number that uses a display format you define. The number is automatically incremented for each new record.
○ Formula	A read-only field that derives its value from a formula expression you define. The formula field is updated when any of the source fields change.
○ Roll-Up Summary	A read-only field that displays the sum, minimum, or maximum value of a field in a related list or the record count of all records listed in a related list.

Figure 2-g: Creating a New Custom Field

Notice that Force.com requests us to specify what type of information this field will contain. We'll be creating the **Account** field, which will be a **Lookup** on the **Account** standard object.

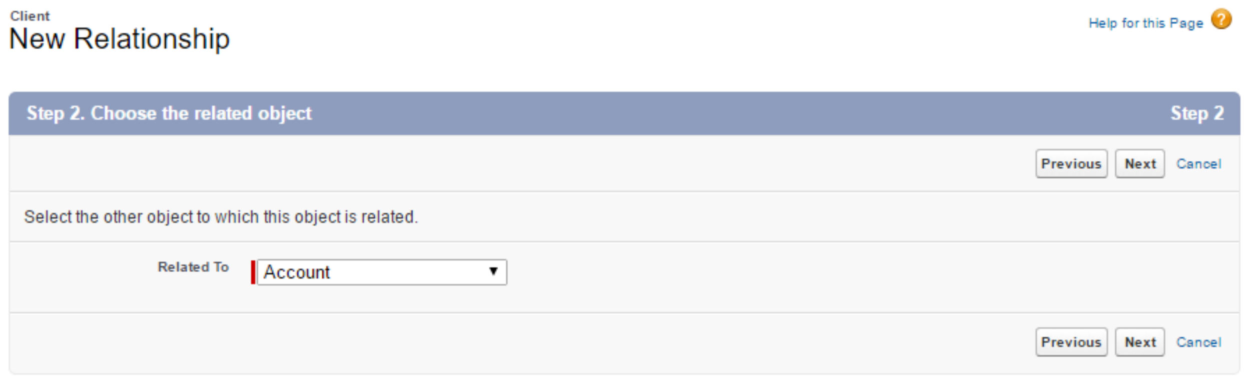
Let's scroll down and select the **Lookup Relationship** option.



● Lookup Relationship	Creates a relationship that links this object to another object. The relationship field allows users to click on a lookup icon to select a value from a popup list. The other object is the source of the values in the list.
○ Master-Detail Relationship	Creates a special type of parent-child relationship between this object (the child, or "detail") and another object (the parent, or "master") where:

Figure 2-h: The Lookup Relationship Option for the Account Name Field

Once you've selected this option, click **Next**. At this stage, Force.com will ask us which object this **Lookup Relationship** is related to—choose the **Account** object from the list.


Client
New Relationship

Help for this Page

Step 2. Choose the related object Step 2

Previous Next Cancel

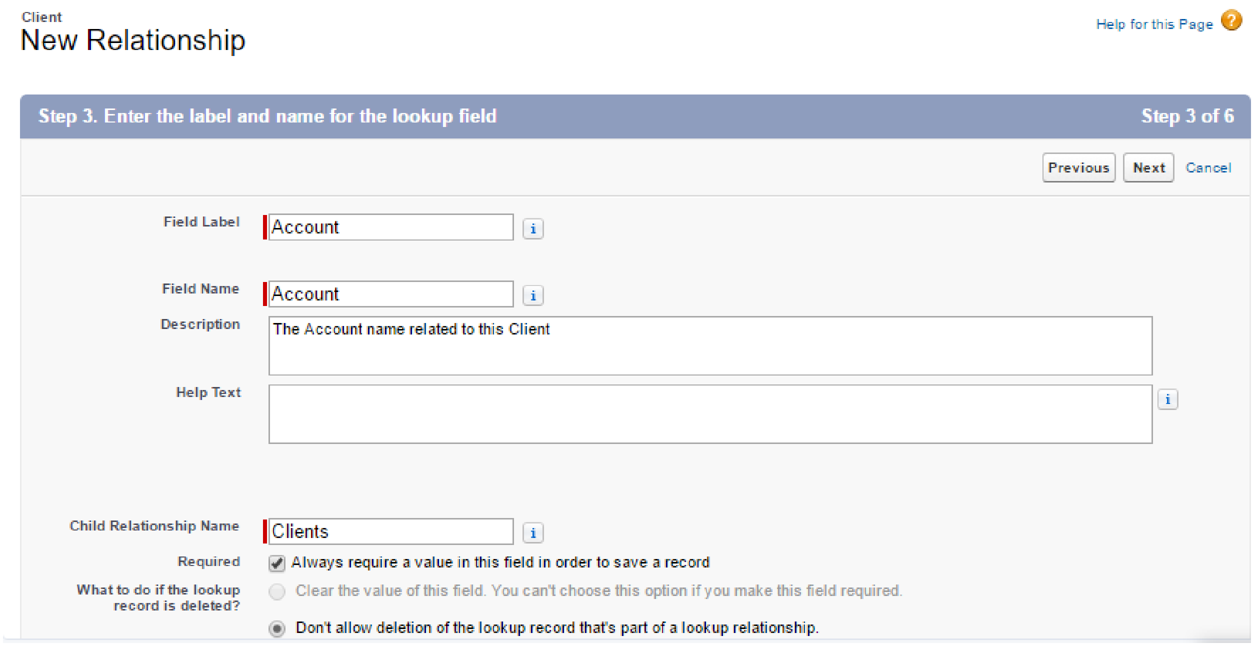
Select the other object to which this object is related.

Related To: Account

Previous Next Cancel

Figure 2-i: Choosing the Related Object for the Lookup Relationship

Click **Next** once again to continue the process. The next stage is where we define the actual field itself and give it a name, so let's do that.



Client
New Relationship

Help for this Page

Step 3. Enter the label and name for the lookup field — Step 3 of 6

Previous | Next | Cancel

Field Label	Account
Field Name	Account
Description	The Account name related to this Client
Help Text	
Child Relationship Name	Clients
Required	[✓] Always require a value in this field in order to save a record
What to do if the lookup record is deleted?	Clear the value of this field. You can't choose this option if you make this field required. [✓ Don't allow deletion of the lookup record that's part of a lookup relationship.]

Figure 2-j: Giving the Custom Field an Identity

Enter the **Field Label** and **Field Name** values—in our case, we'll enter the same for both and call it **Account**. It's always recommended that we enter a **Description**, so let's do that as well.

Once we've entered these basic details, it is important to notice that the **Child Relationship Name** is automatically provided by Force.com, and it's recommended that we leave this value as is, unless you chose to name the relationship differently.

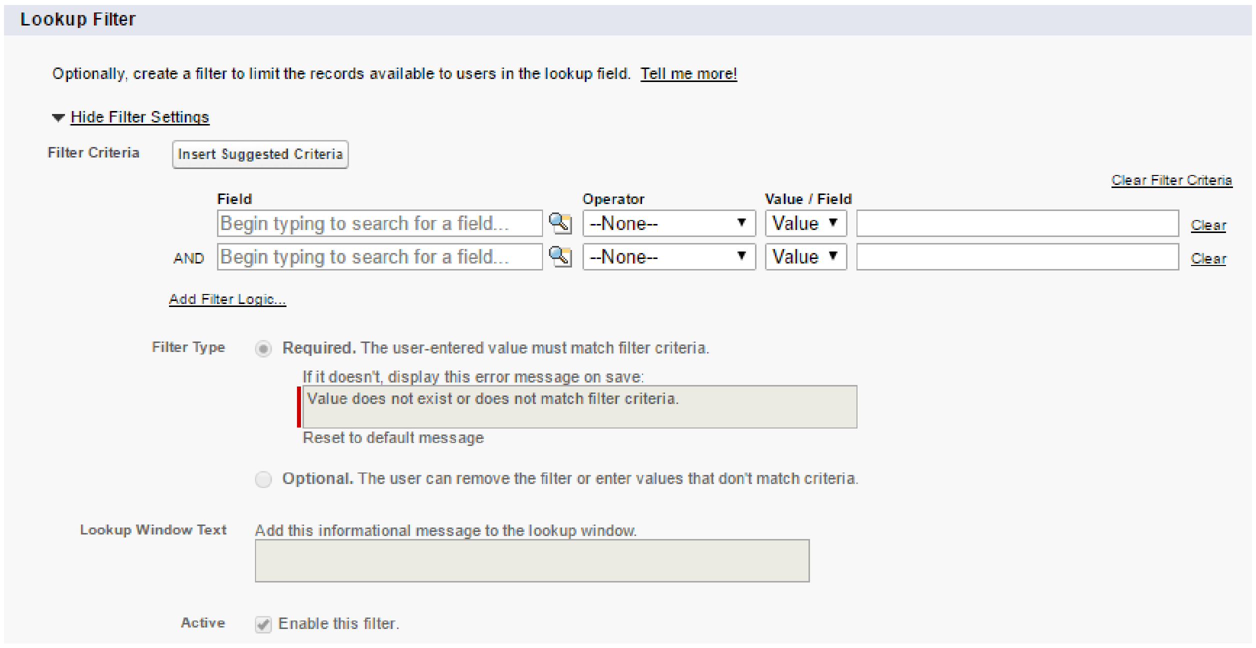
The **Required** option is not selected by default, which means that if we keep it this way, we will be able to leave the field empty when creating a new **Client** object. In this case, we want to enforce the condition that the **Account** field is not empty when a new **Client** object is saved, so let's set it as required by clicking this option.

The other option we want to have selected is the one called **Don't allow deletion of the lookup record that's part of a lookup relationship**.

This means that if the **Client** record is removed, the **Account** linked to that **Client** record won't be deleted from the Force.com database. This is useful and important, as the **Account** object is also commonly used by other Force.com apps, such as CRM and Sales.

What is happening here is that by adding this **Lookup** field, Force.com is establishing, behind the scenes, a database relationship between our **Client** custom object and the out-of-the-box **Account** standard object.

If we scroll down a bit, there's an optional **Lookup Filter** section, which can be used to limit the number of records available to the users in the **Lookup** field. This can come in handy if we would like to limit the results, but we won't be using this option.



Lookup Filter

Optionally, create a filter to limit the records available to users in the lookup field. Tell me more!

Hide Filter Settings

Filter Criteria: Insert Suggested Criteria

Clear Filter Criteria

	Field	Operator	Value / Field		
	Begin typing to search for a field...	--None--	Value		Clear
AND	Begin typing to search for a field...	--None--	Value		Clear

Add Filter Logic...

Filter Type: [✓ **Required.** The user-entered value must match filter criteria.]

If it doesn't, display this error message on save:

Value does not exist or does not match filter criteria.

Reset to default message

Optional. The user can remove the filter or enter values that don't match criteria.

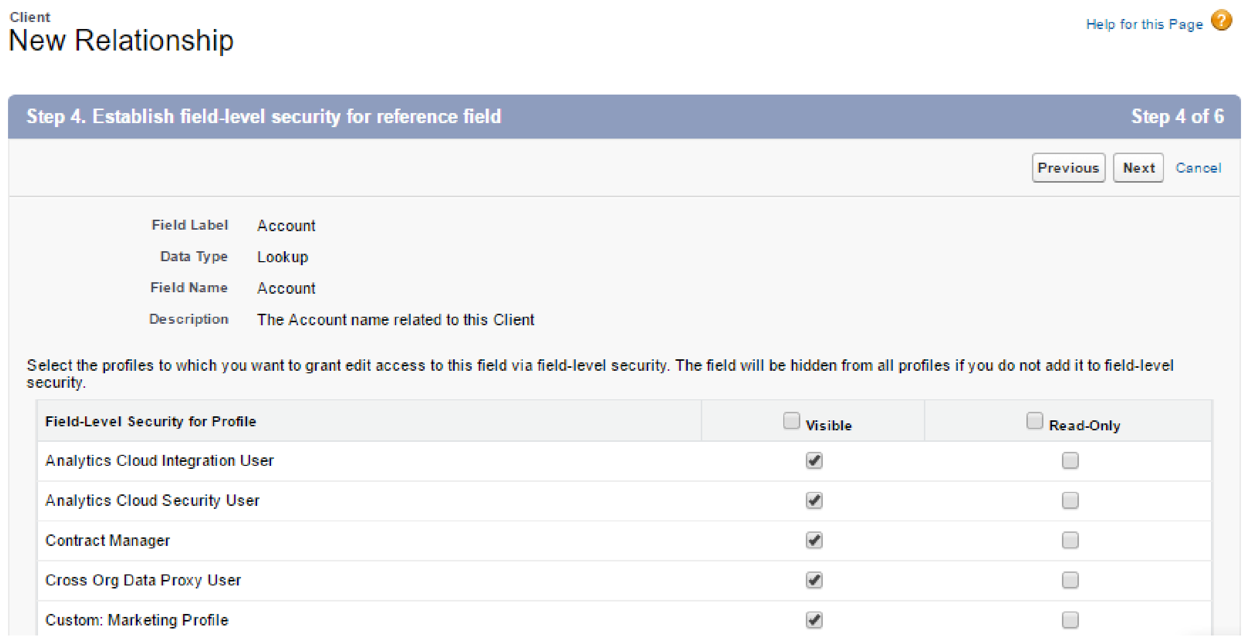
Lookup Window Text: Add this informational message to the lookup window.

Active: [✓] Enable this filter.

Figure 2-k: The Lookup Filter Optional Section

In order to continue, scroll a bit further down and click **Next** again. Once you have done that, you'll be asked to establish the **field-level security** for the field being created.

Here you can select which profiles can have access to this field. By default, the field is set to have access to all profiles (the **Visible** checkbox is selected for all profiles). In my case, I have left these default options selected.



Client

New Relationship

Help for this Page

Step 4. Establish field-level security for reference field — Step 4 of 6

Previous | Next | Cancel

Field Label	Account
Data Type	Lookup
Field Name	Account
Description	The Account name related to this Client

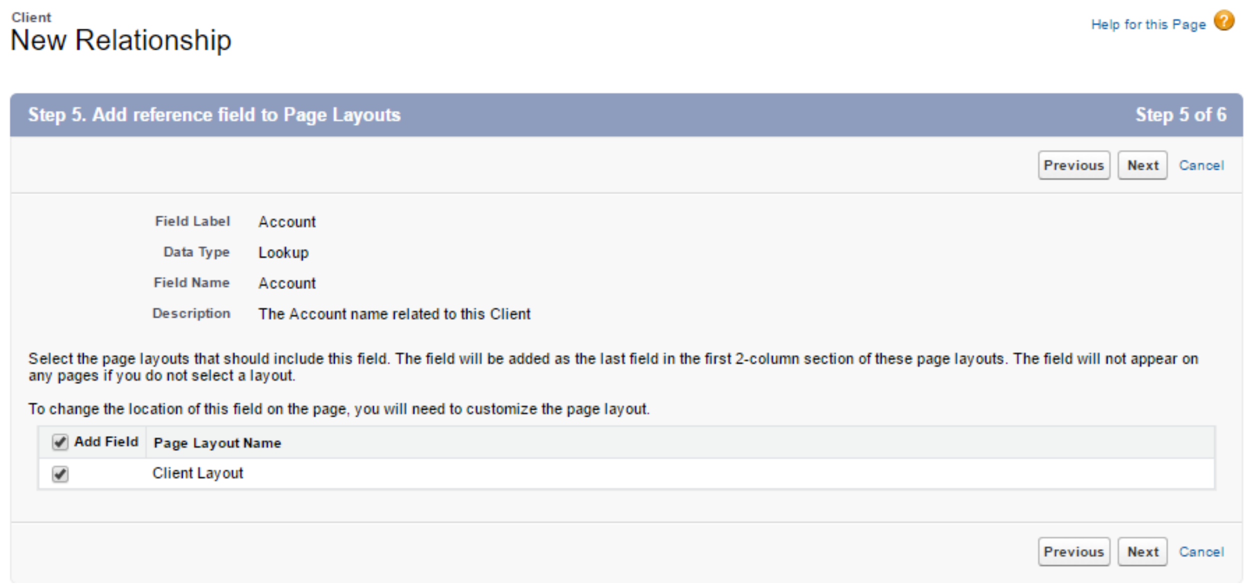
Select the profiles to which you want to grant edit access to this field via field-level security. The field will be hidden from all profiles if you do not add it to field-level security.

Field-Level Security for Profile	Visible	Read-Only
Analytics Cloud Integration User	✓	
Analytics Cloud Security User	✓	
Contract Manager	✓	
Cross Org Data Proxy User	✓	
Custom: Marketing Profile	✓	

Figure 2-l: Field-Level Security for the Field that is Being Created

In order to continue, click **Next**. Once that has been done, you'll be presented with a screen where you'll be asked to select the page layout(s) that should include this field.

By default, the field will be included within the layout of our object—in our case, this corresponds to the layout of the **Client** object.



Field Label	Account
Data Type	Lookup
Field Name	Account
Description	The Account name related to this Client

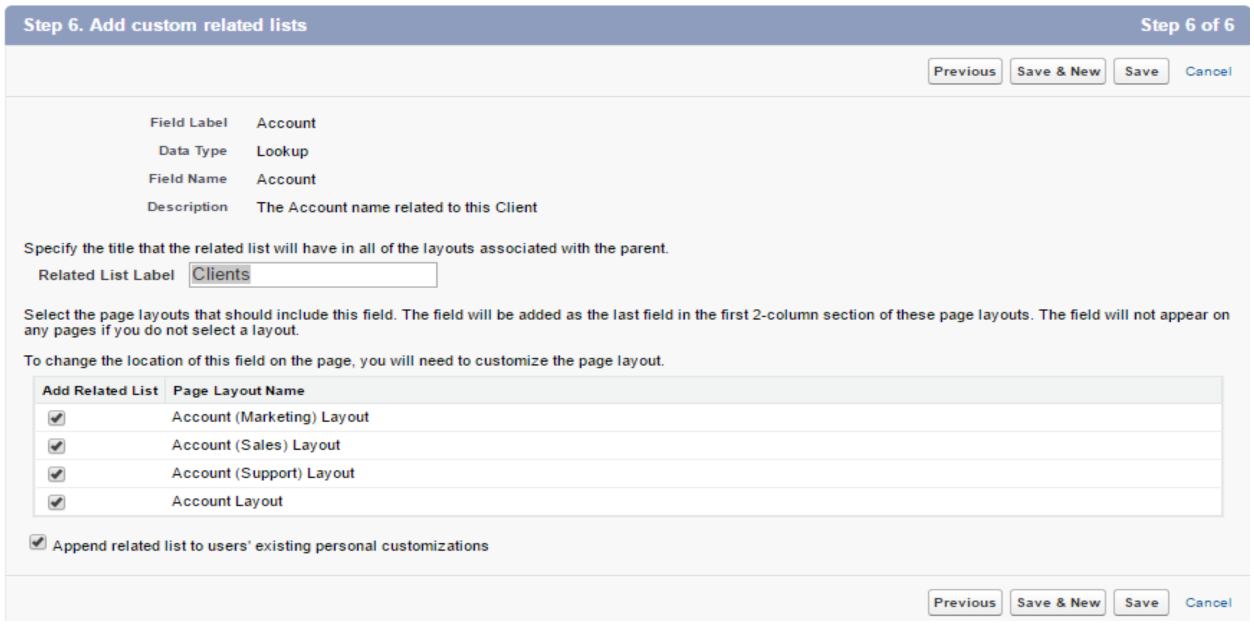
Select the page layouts that should include this field. The field will be added as the last field in the first 2-column section of these page layouts. The field will not appear on any pages if you do not select a layout.

To change the location of this field on the page, you will need to customize the page layout.

✓ Add Field	Page Layout Name
✓	Client Layout

Figure 2-m: Add Reference Field to Page Layouts

To continue, click **Next**. You'll be presented with the final field creation screen. Here you'll be asked to specify the title that this field will have in all the layouts associated with the **Account** object, and also in which of those layouts you would like the field to be shown. This is because this field is linked to the **Account** object.



Field Label	Account
Data Type	Lookup
Field Name	Account
Description	The Account name related to this Client

Specify the title that the related list will have in all of the layouts associated with the parent.

Related List Label: Clients

Select the page layouts that should include this field. The field will be added as the last field in the first 2-column section of these page layouts. The field will not appear on any pages if you do not select a layout.

To change the location of this field on the page, you will need to customize the page layout.

Add Related List	Page Layout Name
✓	Account (Marketing) Layout
✓	Account (Sales) Layout
✓	Account (Support) Layout
✓	Account Layout

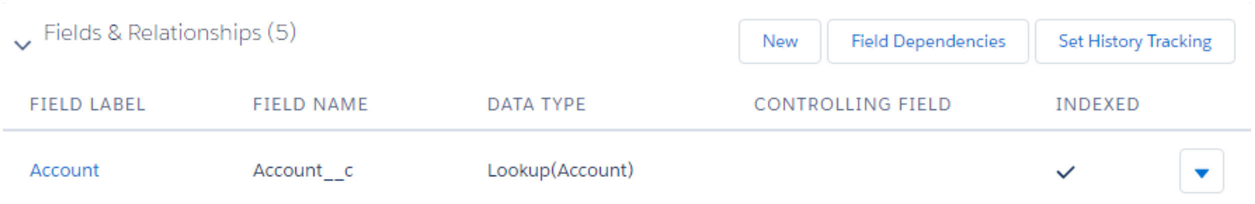
✓ Append related list to users' existing personal customizations

Figure 2-n: Add Custom Related Lists

You may unselect one or more of the associated **Account** object layouts—in our case we can leave it as is, so this field can be shown in all those layouts.

Finally, click **Save** to finalize the creation of the field, or the **Save & New** button in order to save this field and create a new one.

I've clicked on the **Save** button, and our field now appears under the **Fields & Relationships** section of the **Client** object.



FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
Account	Account__c	Lookup(Account)		✓

Figure 2-o: The Newly Created Account Field

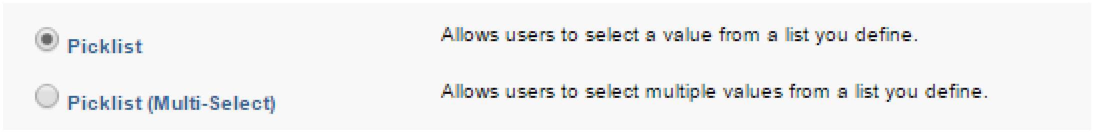
Awesome—we have created our first custom field. This is a great achievement, but our **Client** object is far from done. Let's create the rest of the fields it needs, as mentioned on Table 2-a.

The CS Manager field

By default, the **Client Name** field was created when the **Client** object was set up, so we'll skip this one.

The next field on our list is **CS Manager**. This field is basically a **Picklist** that will contain the names of the customer success managers that will be assigned to various customer accounts. Let's create it.

Under the **Fields & Relationships** section, click **New**. You'll be presented with the **New Custom Field** screen. Here, scroll down and select the **Picklist** data type, and then click **Next**.



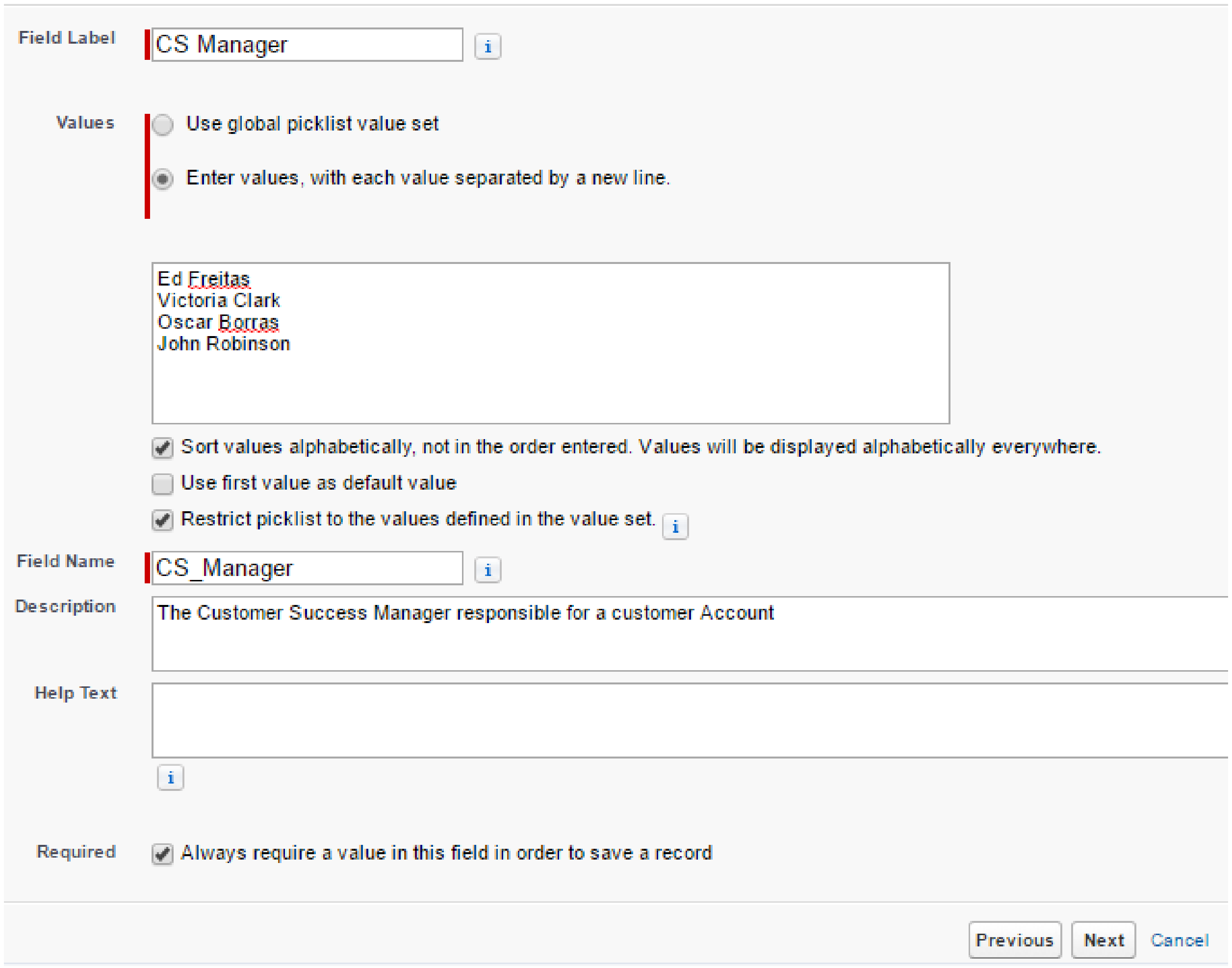
● Picklist	Allows users to select a value from a list you define.
○ Picklist (Multi-Select)	Allows users to select multiple values from a list you define.

Figure 2-p: Selecting the Text Field Type for the CS Manager Field

Notice that there's also a **Picklist (Multi-Select)** data type, which allows us to select more than one value.

For our app, we are only interested in having one **CS Manager** per **Account**, and this is why we have chosen **Picklist** instead of **Picklist (Multi-Select)**.

Once we click **Next**, we'll see the following screen. I've already filled in the details to save some time.



Field Label	CS Manager
Values	○ Use global picklist value set ● Enter values, with each value separated by a new line.
	Ed Freitas Victoria Clark Oscar Borras John Robinson
	✓ Sort values alphabetically, not in the order entered. Values will be displayed alphabetically everywhere.
	☐ Use first value as default value
	✓ Restrict picklist to the values defined in the value set.
Field Name	CS_Manager
Description	The Customer Success Manager responsible for a customer Account
Help Text	
Required	✓ Always require a value in this field in order to save a record

Figure 2-q: Properties of the CS Manager Field

The first property we need to fill in is the **Field Label**—I've entered the name **CS Manager** as the field's label, and Force.com has automatically assigned to the **Field Name** property the value **CS_Manager**.

As you've probably noticed, Force.com simply added an underscore, as field names cannot contain spaces. It is possible to change the **Field Name** to something else, but it's really not a must—so we can leave it as is.

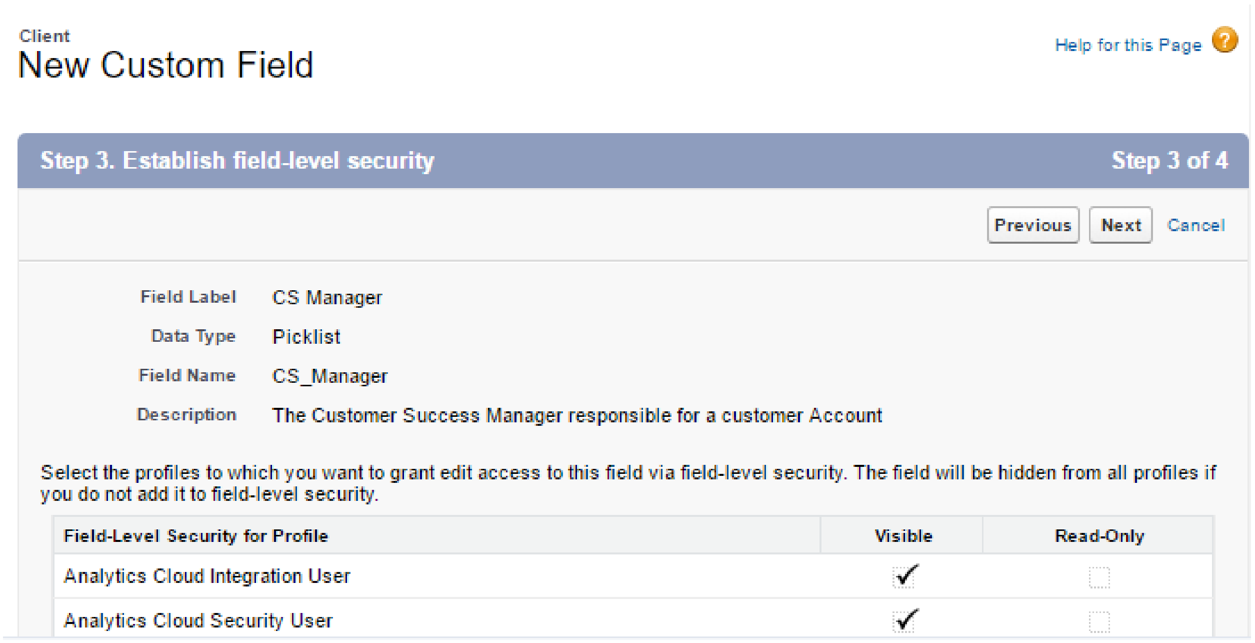
By default, the **Values** property is set to **Use global picklist value set**; however, I chose the option **Enter values, with each value separated by a new line**. This is because I have a list of customer success managers that I want to be able to select, so I've manually entered them.

I've also selected the option **Sort values alphabetically**—which is not selected by default. The option **Restrict picklist to the values defined in the value set** is selected by default, which I have left as is.

Last but not least, I've entered a meaningful **Description** so that we don't forget what this field is for—it's always useful to have later on.

The **Required** property is also not selected by default, so I've selected it because I'm interested in always having a **CS Manager** assigned when a **Client** record is added. Once this has been done, click **Next** to continue.

On the screen that follows, you'll be shown the **field-level** security screen. There's nothing really to do here, as all the **field-level** security settings are set to **Visible** by default, so simply click **Next**.



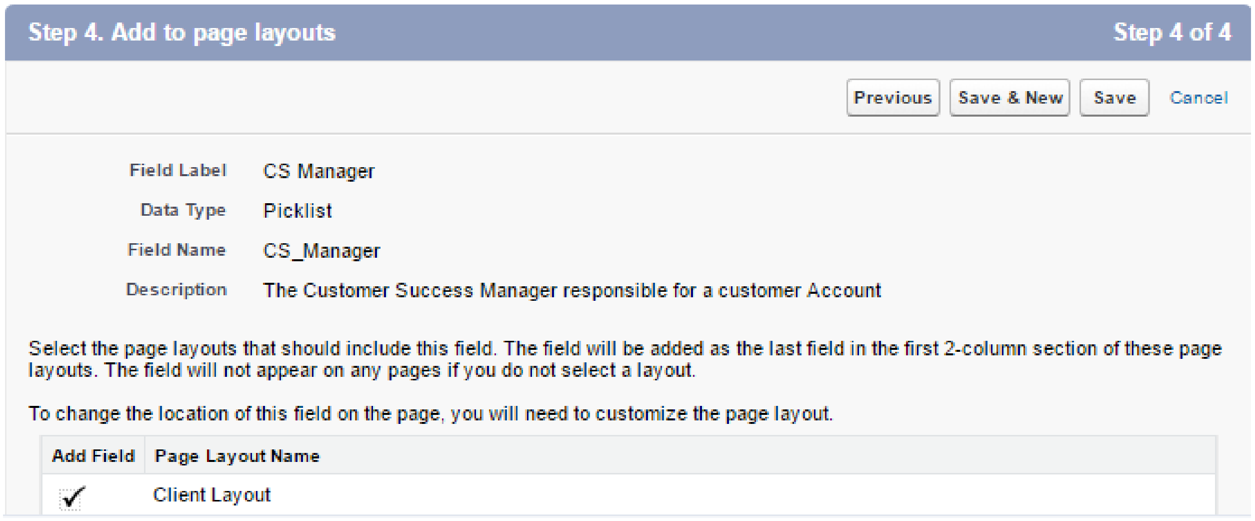
Field Label	CS Manager
Data Type	Picklist
Field Name	CS_Manager
Description	The Customer Success Manager responsible for a customer Account

Select the profiles to which you want to grant edit access to this field via field-level security. The field will be hidden from all profiles if you do not add it to field-level security.

Field-Level Security for Profile	Visible	Read-Only
Analytics Cloud Integration User	✓	☐
Analytics Cloud Security User	✓	☐

Figure 2-r: Field-Level Security for the CS Manager Field

The following screen is **Add to page layouts**. Nothing to do here either, as the field is automatically selected to be added to the layout of the **Client** object. Simply click **Save** in order to finalize the creation of the field.



Field Label	CS Manager
Data Type	Picklist
Field Name	CS_Manager
Description	The Customer Success Manager responsible for a customer Account

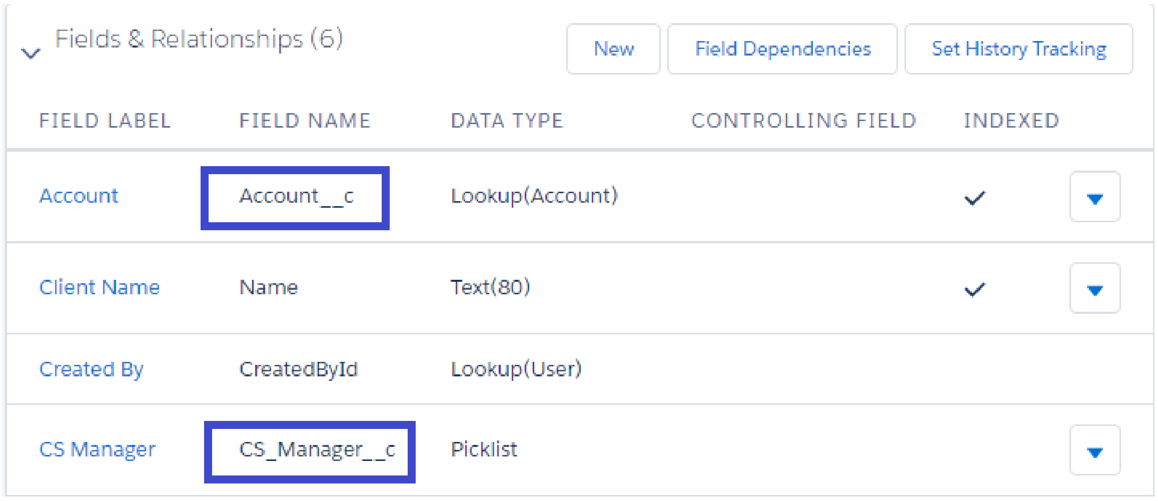
Select the page layouts that should include this field. The field will be added as the last field in the first 2-column section of these page layouts. The field will not appear on any pages if you do not select a layout.

To change the location of this field on the page, you will need to customize the page layout.

Add Field	Page Layout Name
✓	Client Layout

Figure 2-s: Add to Page Layouts for the CS Manager Field

Our newly created field will show up under the **Fields & Relationships** section of the **Client** object.



FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
Account	Account__c	Lookup(Account)		✓
Client Name	Name	Text(80)		✓
Created By	CreatedById	Lookup(User)		
CS Manager	CS_Manager__c	Picklist		

Figure 2-t: Fields of the Client Object

Important: Notice how some fields are always appended the *__c* suffix, which indicates that these are custom fields that we have created. Fields that are created by Force.com do not have the *__c* suffix appended to them.

So now we've now created another field. Let's carry on and finalize our **Client** object.

For the next fields, in order to save some time, I'll skip the **Establish field-level security** and **Add to page layouts** screens, by simply clicking **Next** and **Save**.

Let's fast forward and immediately create the **Region** and **Status** fields first, as they are also **Picklist** types.

The Region & Status Picklists

The **Region** field will indicate which geographical region a **Client** record belongs to, and the **Status** field will indicate the customer's position within our organization—if the customer is in active support or is being upgraded, for instance.

Let's create the **Region** field first. Click **New** under the **Fields & Relationships** section of the **Client** object.

Chose **Picklist** as the data type and continue. Then, enter **Region** as the **Field Label**, and on the **Values** property, type in the following entries: *North America, Latin America, Europe, Middle East, Africa, Asia,* and *Pacific*—each of them on separate lines.

Make sure that **Sort values alphabetically** is selected and **Restrict picklist** is checked. Enter a meaningful **Description** and make sure that the **Required** property is also ticked.

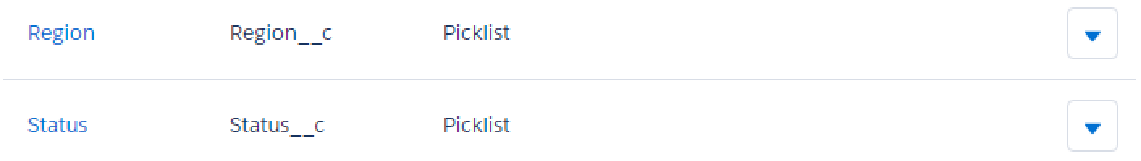
Click **Next** to continue—you'll then be presented with the **Field-level** security screen, so simply click **Next** to carry on. Finally, on the last screen click **Save & New**.

Awesome—our **Region** field has been created, and now we can create the **Status** field. Choose again the **Picklist** data type when prompted, and continue. Enter **Status** as the **Field Label**, and on the **Values** property, type in the following entries: *In Active Support, Upgrading, End of Life,* and *Inactive*.

Just as we did a moment ago for the **Region** field, make sure that **Sort values alphabetically** is selected and **Restrict picklist** is checked.

Don't forget to enter a meaningful **Description**, and make sure that the **Required** property is selected.

Next is the **Field-Level** security screen—you know what to do. On the last one, click **Save**, and we're done! We've now created the **Status** field.



Region	Region__c	Picklist
Status	Status__c	Picklist

Figure 2-u: The Region & Status Fields

Now that we've created all our **Picklist** fields for our **Client** object, let's create the rest of the fields—some of them include other interesting data types that we haven't explored yet.

The Versions Multi-Select Picklist

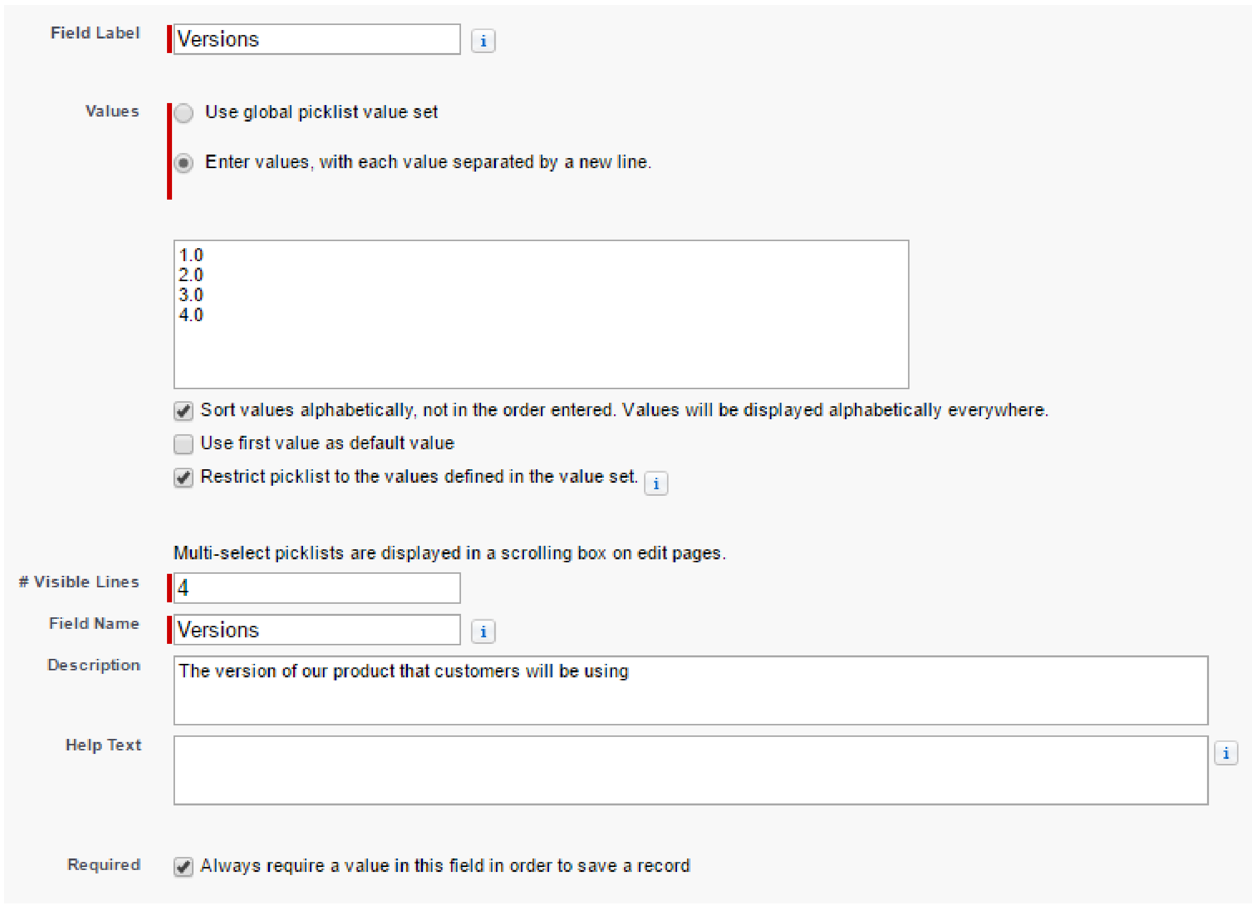
The **Versions** field is an interesting one, as it should allow us to select multiple product versions that customers of our organization might be using. It's quite similar to the **Picklist** data type, but with a subtle difference. Let's explore that.

In order to create our field, click **New** under the **Fields & Relationships** section of the **Client** object.

Select **Picklist (Multi-Select)** as the field data type, and then click **Next**, which will display the screen to enter the field details.

Within the field properties, notice that there's a **Visible Lines** property that is not present on regular **Picklist** fields, which indicates how many lines will be displayed—**Multi-Select Picklists** are shown in a scrolling box, and not a dropdown menu like regular **Picklists**.

I've entered the following details for the **Versions** field.



Field Label	Versions
Values	○ Use global picklist value set ● Enter values, with each value separated by a new line. 1.0 2.0 3.0 4.0 ✓ Sort values alphabetically, not in the order entered. Values will be displayed alphabetically everywhere. ☐ Use first value as default value ✓ Restrict picklist to the values defined in the value set.
	Multi-select picklists are displayed in a scrolling box on edit pages.
# Visible Lines	4
Field Name	Versions
Description	The version of our product that customers will be using
Help Text	
Required	✓ Always require a value in this field in order to save a record

Figure 2-v: Details of the Versions Field

With these details in place, click **Next** and then **Save** in order to finish creating this field. We now have our **Versions** field created.

The Upgrade Formula Checkbox field

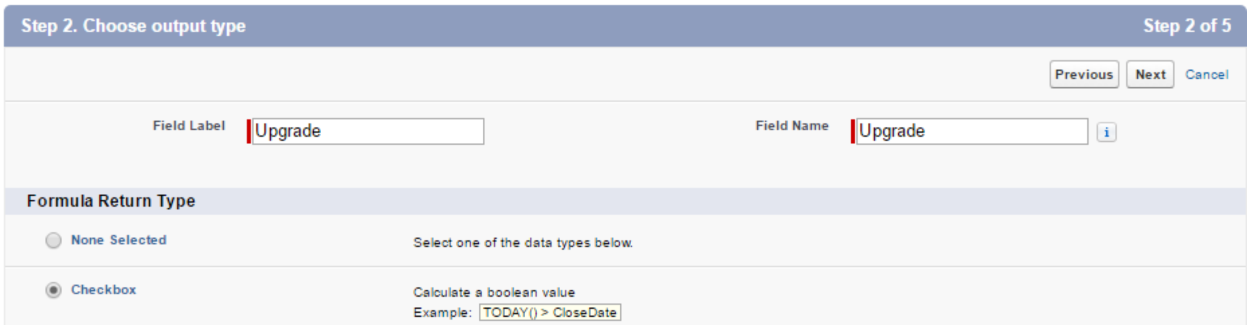
Because our customers could be using multiple versions of our product, it would be quite useful if with a quick glimpse, we could easily see which ones would require an upgrade to our product's latest version. A quick way to achieve this is with a **Formula Checkbox** field.

The **Upgrade** field's purpose is to precisely determine this. If the field is checked, it indicates that a customer needs an upgrade. But how can we determine if the customer needs an upgrade?

The answer is easy: if the customer is using the latest (highest) version of our product, they will not require an upgrade—even though they might still be using older versions in parallel. On the contrary, if they are not using the latest version, then they will require an upgrade—the **Upgrade** checkbox will be ticked. This can be determined by using a formula.

So, let's go ahead and create our **Upgrade** field. Click **New** under the **Fields & Relationships** section.

Select **Formula** as the field data type, and then click **Next**. Specify the **Field Label**, and for the **Formula Return Type**, select **Checkbox**.



Field Label: Upgrade

Field Name: Upgrade

Formula Return Type

None Selected — Select one of the data types below.

Checkbox — Calculate a boolean value
Example: TODAY() > CloseDate

Figure 2-w: The Output Type of the Upgrade Field

Then, click **Next**—this will take us to a screen where we'll be able to enter the formula for our field.

The formula editor is easy to understand and navigate. I usually use the **Advanced Formula** tab, as it has a handy listbox that allows us to quickly view the available set of functions we can use.

Code Listing 2-a shows the formula we'll use in order to determine if a customer requires an upgrade.

Code Listing 2-a: The Upgrade Field Formula

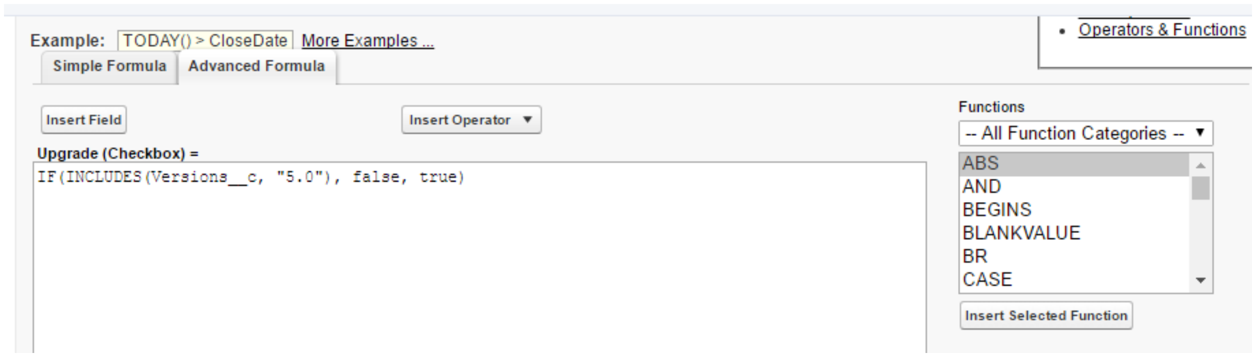
```
IF(INCLUDES(Versions__c, "5.0"), false, true)
```

The formula is very straightforward, and it basically checks whether or not the field **Versions** (*Versions__c*) includes the string **`5.0`**—if it does, then **`false`** is returned, which means that no upgrade is required. On the contrary, if **`5.0`** is not found, then **`true`** is returned—meaning that an upgrade would be needed.

Say, for example, that in the future, version 6.0 of our software is released, and we need to update the **Versions** field and, subsequently, the **Upgrade Field Formula**. So what do we do?

Easy—we can update this at any time by going to the **Fields & Relationships** section of the **Client** object and clicking on the arrow button, and then on the **Edit** option—this is available for each field.

So, let's add our formula and use the **Advanced Formula** editor.


Example: TODAY() > CloseDate More Examples ...

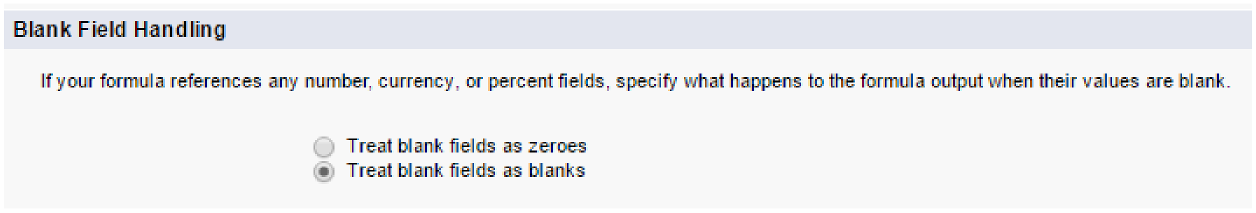
Upgrade (Checkbox) =

```
IF(INCLUDES(Versions__c, "5.0"), false, true)
```

Figure 2-x: The Advanced Formula Editor

It's always useful to verify that the formula we have typed in is actually accurate. There's a **Check Syntax** button just below the formula editor. Click it in order to verify that the formula indeed does have a valid syntax.

If you scroll down a bit further, there's an important section called **Blank Field Handling**. By default, this is set to **Treat blank fields as zeroes**; however, for this formula, we are interested in **Treat blank fields as blanks**.


Blank Field Handling

If your formula references any number, currency, or percent fields, specify what happens to the formula output when their values are blank.

Treat blank fields as zeroes [✓ Treat blank fields as blanks]

Figure 2-y: Blank Field Handling

Because our **Versions** field is actually just a **List** of **Strings**, it makes more sense to treat blank fields as blanks rather than zeroes.

Let's click on **Next** in order to continue—we'll be shown the **field-level** security screen. We can skip this and accept the default settings by clicking **Next**.

Finally, the **Add to page layouts** screen will be shown. Leave the default settings and click **Save**—this will create our **Upgrade** field.

Summary

Wait, aren't we missing a field for our **Client** object? Good point! We haven't yet created the **Latest Renewals** field, as shown in Table 2-a. We haven't created this field yet because it depends on another object that we haven't created—the **Renewal** object.

The **Latest Renewals** field will be a **Roll-Up Summary** field that will retrieve information from the most recent **Renewal** record.

First, we need to create our **Renewal** object, then we can come back to the **Client** object and add this **Latest Renewals** summary field.

Customer success for most organizations means that we want to keep our customers engaged and using our products, so renewals are an important part of a Customer Success application—this is why, in the chapter that follows, we'll create the **Renewal** object and tie it to the **Client** object we've created.

Our app is starting to take shape. Let's keep exploring what lies ahead.

Chapter 3 Renewal Custom Object

Introduction

For SaaS businesses, getting customers onboard is vital, but keeping them hooked to a product and paying subscription or maintenance—also known as renewals—is critical for long-term cash flow.

In recent years, a new profession has emerged, one focused on reducing churn and keeping customers engaged with an organization’s products—this role is most commonly known as the Customer Success Manager.

Throughout this chapter, we’ll create a **Renewal** object that will be closely related to the **Client** object we previously created.

The idea behind renewals in general is to keep track of when the customer’s support contract expires and how much they pay, and to keep notes and activities related to issues they raise.

This should be more than enough to have some sort of basic Customer Success functionality within our application—from a database perspective. So, in this chapter we’ll focus on creating this object and linking it to functionality we have previously created.

In later chapters, we’ll start adding some logic and business rules in order to bring the app to life. There’s still a lot of interesting things to build and learn. Let’s keep having fun.

Renewal custom fields

Just as we did with the **Client** object, let’s define which fields we need in order to keep track of **Renewal** records. Here’s the list of fields we will create.

Table 3-a: Fields for Our Renewal Custom Object

Field	Type
Client	Master-Detail (on the Client object)
Renewal Name	Text (80) Unique Case Insensitive
Start Date	Date
End Date	Date
Expired	Formula (Checkbox)
Invoiced	Checkbox

Field	Type
Amount	Number (16, 2)
Currency	Picklist
Cases	Master-Detail (on the Case object)

We'll have to create each of these fields. We can see that our **Renewal** object will have two **Master-Detail** fields, which we have not covered yet—so this will be a great asset to learn, as it is an important part of any Force.com application. But before we can create any field, we need to create the **Renewal** object itself—so let's do that.

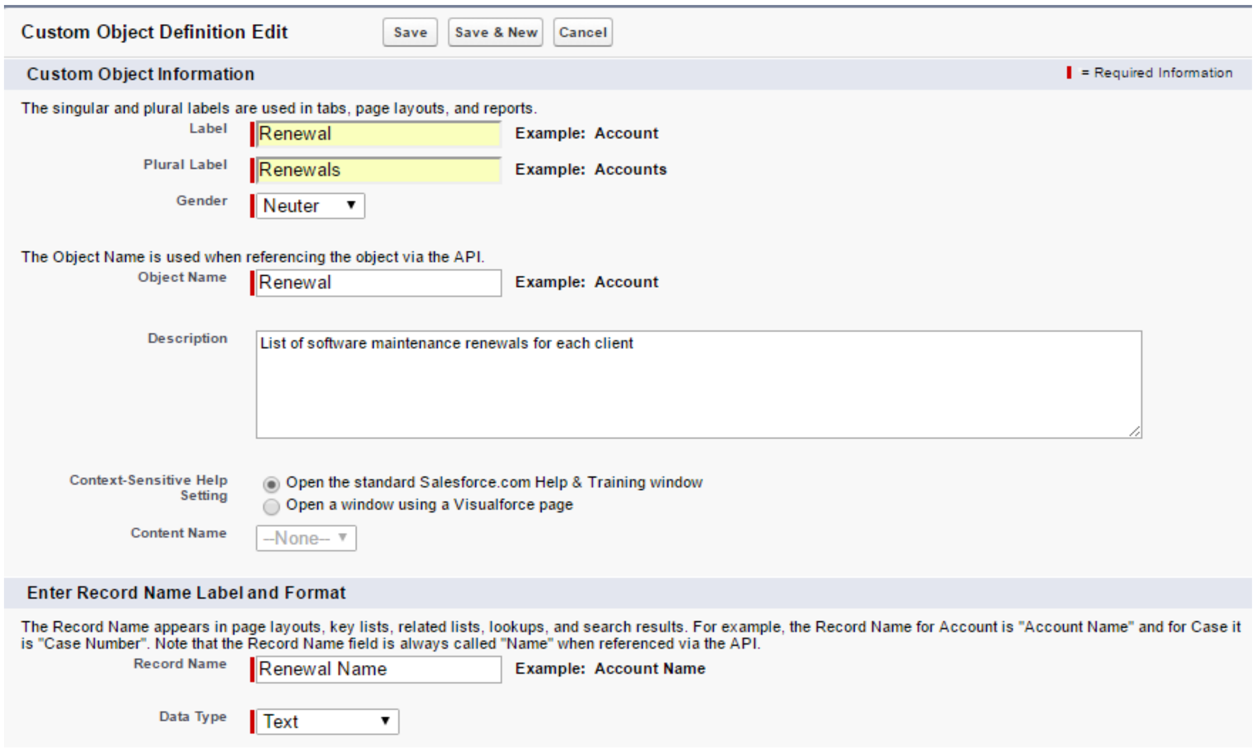
Creating the Renewal custom object

In order to create the **Renewal** object, go to the main **Lightning Experience** screen. On the left side, click **PLATFORM TOOLS** > **Objects and Fields** > **Object Manager**. Once you do this, you'll see the list of available objects that already exist—including our custom **Client** object we previously created.

In order to create a new custom object, click **Create**, and then click the **Custom Object** option. Refer back to Figure 2-a.

Let's specify the object's basic information—we'll need to initially type in the **Label** and **Plural Label**. The **Object Name** is automatically generated, and so is the **Record Name**.

As previously suggested, it is advisable to enter a meaningful **Description** for our object.



Custom Object Definition Edit

Custom Object Information

The singular and plural labels are used in tabs, page layouts, and reports.

Label	Renewal	Example: Account
Plural Label	Renewals	Example: Accounts
Gender	Neuter	

The Object Name is used when referencing the object via the API.

Object Name	Renewal	Example: Account
Description	List of software maintenance renewals for each client	
Context-Sensitive Help Setting	[● Open the standard Salesforce.com Help & Training window] ○ Open a window using a Visualforce page	
Content Name	--None--	

Enter Record Name Label and Format

The Record Name appears in page layouts, key lists, related lists, lookups, and search results. For example, the Record Name for Account is "Account Name" and for Case it is "Case Number". Note that the Record Name field is always called "Name" when referenced via the API.

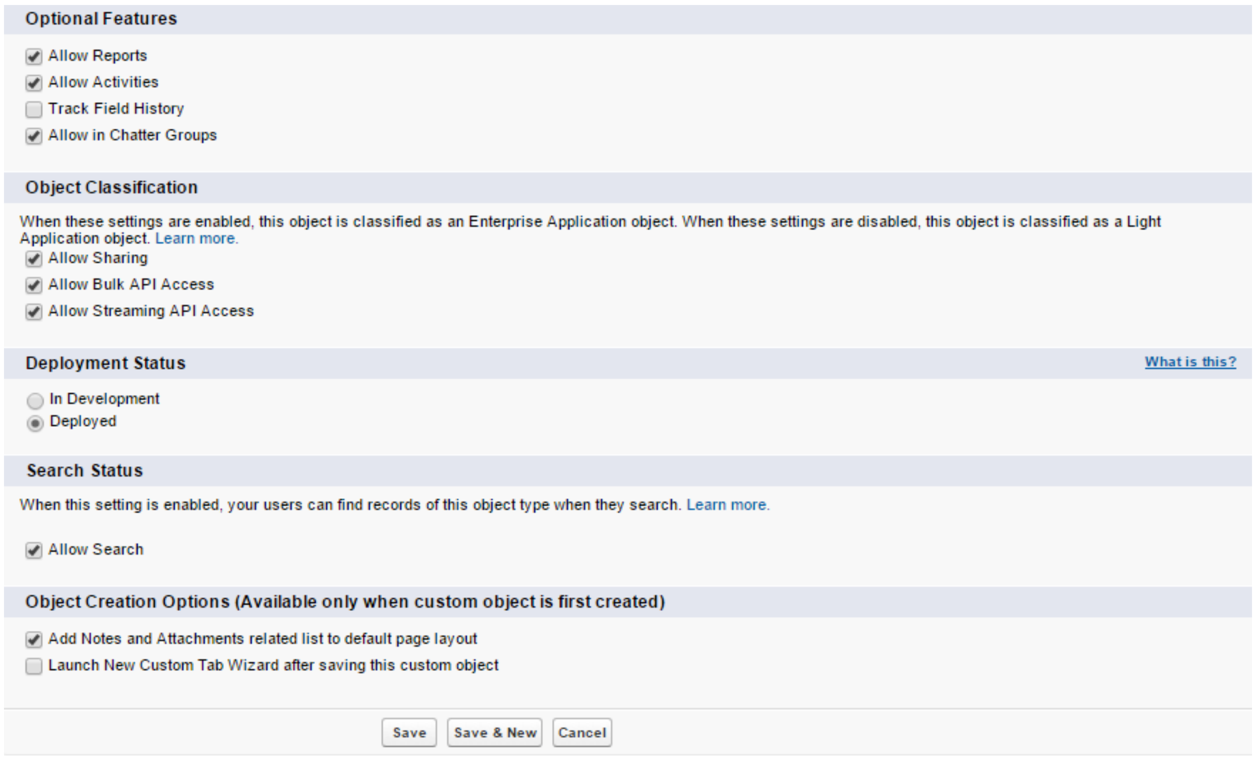
Record Name	Renewal Name	Example: Account Name
Data Type	Text	

Figure 3-a: The Renewal Object (Basic Details)

With these details entered, let's scroll a bit further down in order to finish creating this object.

Just as we did when we created the **Client** object, make sure **Allow Reports**, **Allow Activities**, **Allow in Chatter Groups**, **Allow Sharing**, **Allow Bulk API Access**, and **Allow Streaming API Access** are all ticked.

Make sure the **Deployment Status** is set to **Deployed**. I also recommend that you specifically select the **Allow Search** and **Add Notes and Attachment related list to default page layout** options, too.



Optional Features

- [✓] Allow Reports
- [✓] Allow Activities
- [ ] Track Field History
- [✓] Allow in Chatter Groups

Object Classification

When these settings are enabled, this object is classified as an Enterprise Application object. When these settings are disabled, this object is classified as a Light Application object. Learn more.

- [✓] Allow Sharing
- [✓] Allow Bulk API Access
- [✓] Allow Streaming API Access

Deployment Status What is this?

- ○ In Development
- ● Deployed

Search Status

When this setting is enabled, your users can find records of this object type when they search. Learn more.

- [✓] Allow Search

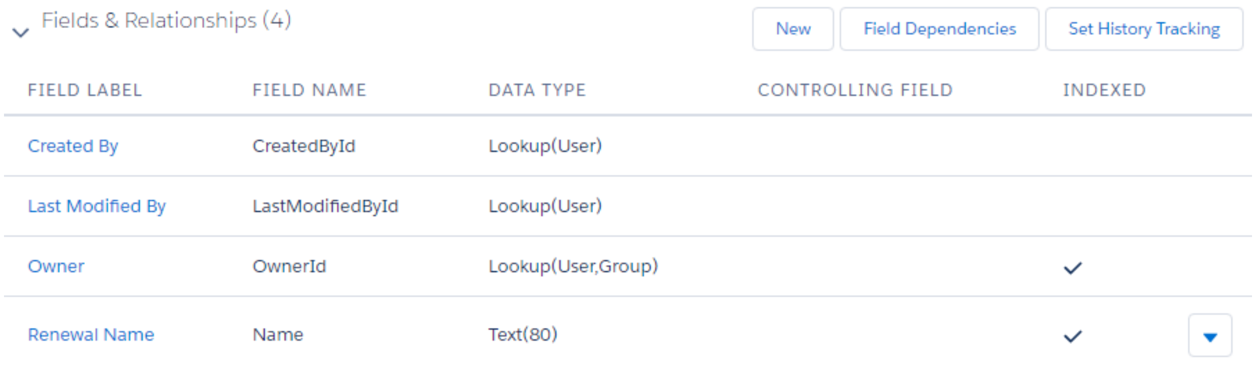
Object Creation Options (Available only when custom object is first created)

- [✓] Add Notes and Attachments related list to default page layout
- [ ] Launch New Custom Tab Wizard after saving this custom object

Figure 3-b: The Renewal Object (Extended Details)

Great—with all these properties defined, let's click **Save**. This will create our **Renewal** object.

We can see that the following fields have been automatically added by Force.com to our **Renewal** object.



FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
Created By	CreatedById	Lookup(User)		
Last Modified By	LastModifiedById	Lookup(User)		
Owner	OwnerId	Lookup(User,Group)		✓
Renewal Name	Name	Text(80)		✓

Figure 3-c: Automatically Created Fields (Renewal Object)

We are ready to start creating our custom fields. The first field we need to create is the **Client** field, which will be linked to the **Client** object. Let's go and do that.

The Client Master-Detail field

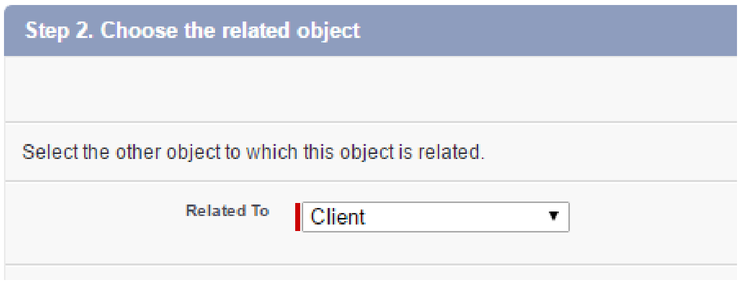
The **Client** field will contain the name of the **Client** object that is related to a **Renewal** record. A **Client** record can have multiple **Renewal** records, each during different periods of time.

This means that there will be a **Master-Detail** relationship between the **Client** object (Master) and the **Renewal** object (Detail).

Under **Fields & Relationships**, click **New**, then choose the **Master-Detail Relationship** as the data type for the new field.

Figure 3-d: Selecting the Master-Detail Relationship

After selecting this data type, click **Next** in order to choose the object to which this field will be related.



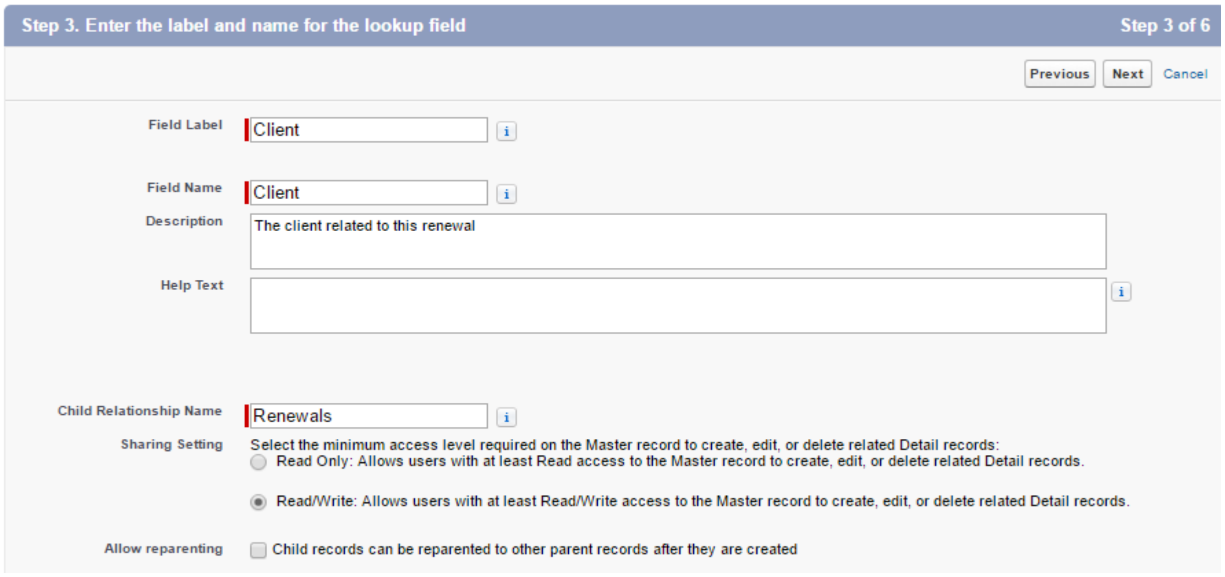
Step 2. Choose the related object

Select the other object to which this object is related.

Related To: Client

Figure 3-e: Selecting the Related Object for the Master-Detail Field

Select the **Client** object and click **Next**. Then type in the **Field Label** and add a **Description**. The **Field Name** is automatically assigned by Force.com.



Step 3. Enter the label and name for the lookup field — Step 3 of 6

Field Label: Client

Field Name: Client

Description: The client related to this renewal

Help Text:

Child Relationship Name: Renewals

Sharing Setting: Select the minimum access level required on the Master record to create, edit, or delete related Detail records:

- ○ Read Only: Allows users with at least Read access to the Master record to create, edit, or delete related Detail records.
- ● Read/Write: Allows users with at least Read/Write access to the Master record to create, edit, or delete related Detail records.

Allow reparenting: ☐ Child records can be reparented to other parent records after they are created

Figure 3-f: The Lookup Relationship Details

The **Child Relationship Name** is also automatically assigned by Force.com. The **Sharing Setting** is set by default to **Read/Write**.

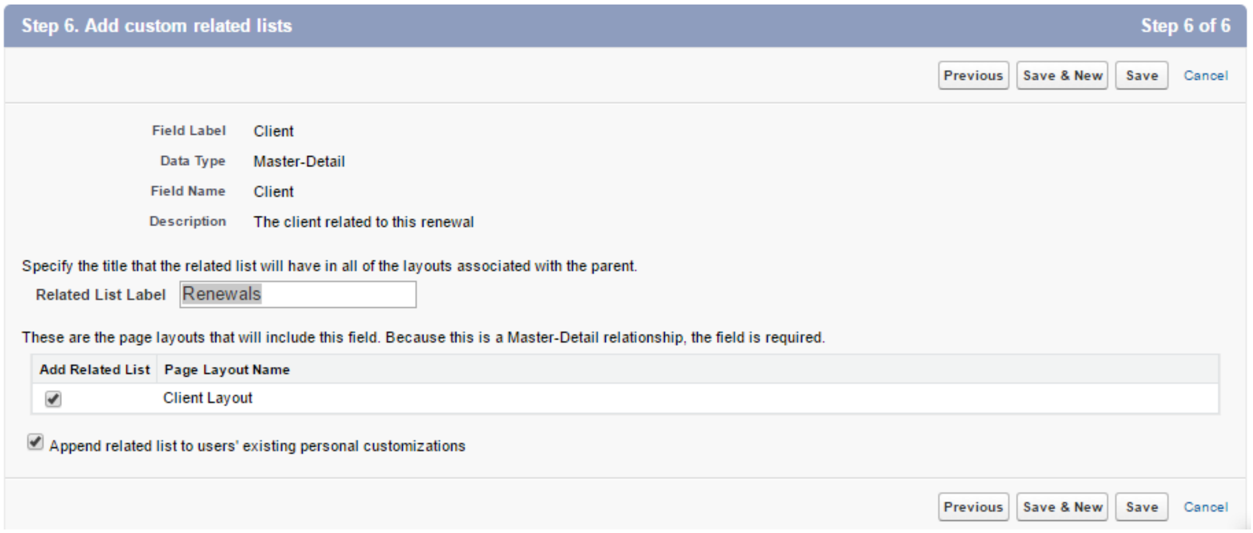
This allows users with read and write access to the **Master** record to create, edit, or delete any related **Detail** records—this is the option we are interested in, as we want users to be able to both read and write **Client** and **Renewal** records.

Also by default, the **Allow reparenting** option is not selected—which we can leave as is, as we want every **Renewal** record to be parented (associated) to its corresponding **Client** record.

If you scroll down a bit, there are some **Lookup Filter** options that we are also not interested in—which we can skip. So, click **Next**, which will display the **field-level** security screen, which has nothing really for us to do—so click **Next** again.

Afterward, we arrive at the **Add reference field to Page Layouts** screen, which also has nothing for us to do, so again, let's click **Next**.

Finally, we are at the **Add custom related lists** screen, as we can see in Figure 3-g.



Field Label	Client
Data Type	Master-Detail
Field Name	Client
Description	The client related to this renewal

Specify the title that the related list will have in all of the layouts associated with the parent.

Related List Label: Renewals

These are the page layouts that will include this field. Because this is a Master-Detail relationship, the field is required.

Add Related List	Page Layout Name
✓	Client Layout

✓ Append related list to users' existing personal customizations

Figure 3-g: The Add Custom Related Lists Screen

Force.com has automatically assigned a name to the **Related List Label**, which is the same as the plural of our **Detail** object—**Renewals**. We can leave this as is.

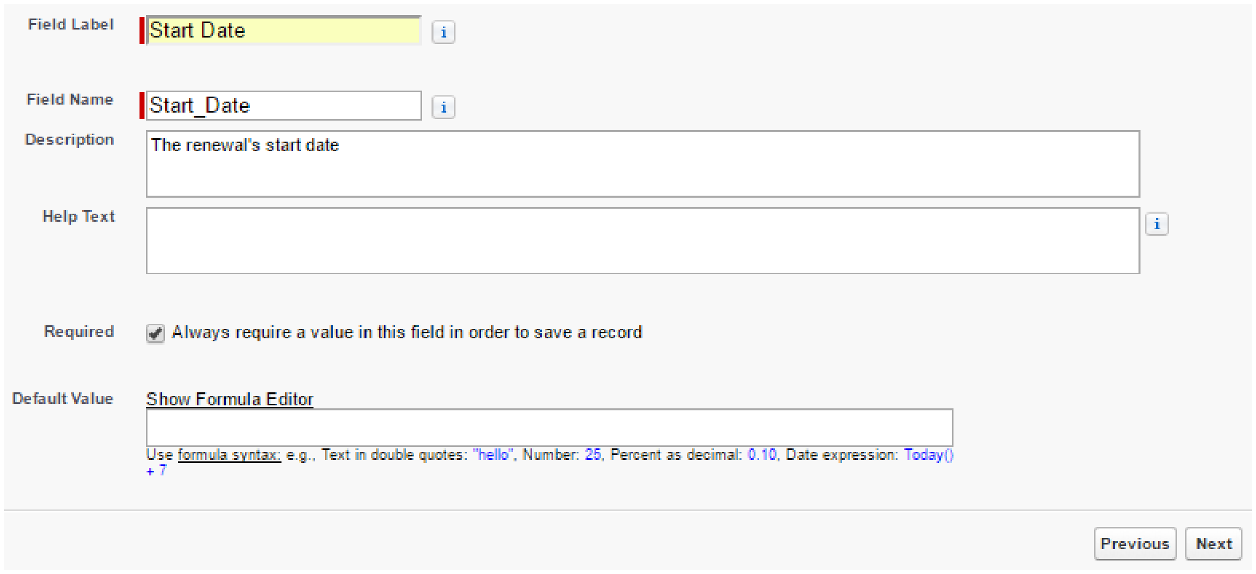
In order to create the **Client** field, simply click **Save**. We've now created our first **Master-Detail** field.

The Dates fields

The **Start** and **End** dates are key for tracking renewals. Both are, as you might have guessed, **Date** data type fields.

Let's create them. We'll start by creating the **Start Date** field. Click on the **New** button under **Fields & Relationships**.

When asked, choose **Date** as the field type, then click **Next**.



Field Label	Start Date
Field Name	Start_Date
Description	The renewal's start date
Help Text	
Required	✓ Always require a value in this field in order to save a record
Default Value	Show Formula Editor

Use formula syntax: e.g., Text in double quotes: "hello", Number: 25, Percent as decimal: 0.10, Date expression: Today() + 7

Figure 3-h: Start Date Field Details

Enter the **Field Label**—Force.com will automatically assign the **Field Name** property. Also enter a meaningful **Description**. Set the **Required** property, and then click **Next**.

On the **field-level security** screen, click **Next**, and on the **Add to page layouts** screen, click **Save & New**—this way we can immediately create the **End Date** field also.

Follow the same steps as for the **Start Date** field, and make sure you set the **Field Label** as **End Date**.

The Expired Formula checkbox

Now that we've created the **Dates** fields, let's create a **Formula Checkbox** field that will basically tell us if a **Renewal** record has expired or not. Expired means that the support maintenance contract is no longer valid—it has not been renewed to this date. This is a handy field to have, so let's create it.

Click **New** under **Fields & Relationships**. When asked, choose **Formula** as the field type, and then click **Next**.

On the **Field Label**, type in the name **Expired**. Force.com will automatically assign the **Field Name** based on the **Field Label**. Then, as the **Formula Return Type**, select **Checkbox** and click **Next**.

We'll use the following formula in order to determine if a **Renewal** record has expired.

Code Listing 3-a: The Expired Field Formula

```
IF(End_Date__c < TODAY(), true, false)
```

`End_Date__c` is the internal name of the **End Date** field. Remember that internally, all custom fields within Force.com are appended the **`__c`** suffix.

The formula is quite simple. Essentially, if the value of **`End Date`** is less than today's date, then **`true`** is returned—which means that the **Renewal** record has expired. On the contrary, if **`false`** is returned, it means that the record has not expired.

Scroll a bit further down and set the **Blank Field Handling** property as **Treat blank fields as blanks**, then click **Next**.

On the **Establish field-level security** screen, leave all the default options as is, and click **Next**.

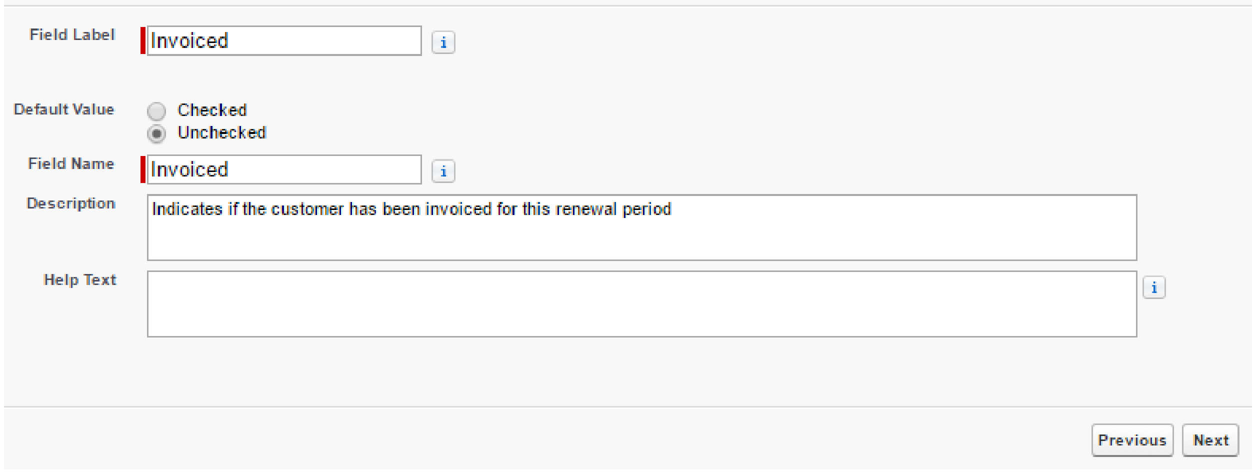
Do the same on the **Add to page layouts** screen—leave the default option and click **Save** in order to create the field.

The Invoiced checkbox

The **Invoiced** field is simply a **Checkbox** that will indicate if the customer has been invoiced for the applicable renewal period. It's just a flag—a **Boolean**, and not a calculated field. So, let's create it now.

Click on the **New** button under **Fields & Relationships**. When asked, choose **Checkbox** as the field type, and then click **Next**.

Now we type in the **Field Label**—the **Field Name** will automatically be assigned. Also enter a **Description** and click **Next**.



Field Label	Invoiced
Default Value	Checked [✓ Unchecked]
Field Name	Invoiced
Description	Indicates if the customer has been invoiced for this renewal period
Help Text	

Figure 3-i: Start Date Field Details

On the **Establish field-level security** screen, leave the defaults and click **Next**. On the **Add to page layouts** screen, leave the default option and click **Save** in order to create the field.

Great, we have created another field. Let's move on.

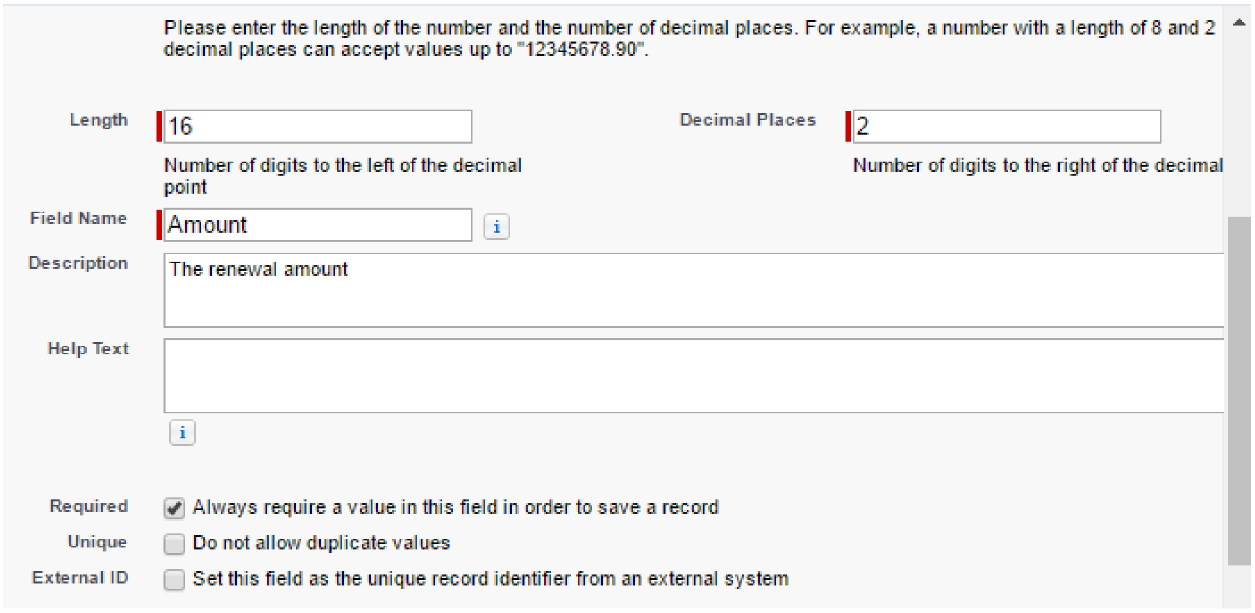
The Amount field

The **Amount** field, as its name implies, corresponds to the amount that the customer will be paying for maintenance—basically, the **Renewal** cost. Let's create it.

Click on the **New** button under **Fields & Relationships**. When asked, choose **Number** as the field type, and then click **Next**.

Now type in the **Field Label**, using **Amount** as the name. Set the **Length** to **16** and the **Decimal Places** to **2**.

Enter a **Description**, select the **Required** property, and then click **Next**.



Please enter the length of the number and the number of decimal places. For example, a number with a length of 8 and 2 decimal places can accept values up to "12345678.90".

Length	16	Decimal Places	2
	Number of digits to the left of the decimal point		Number of digits to the right of the decimal
Field Name	Amount		
Description	The renewal amount		
Help Text			
Required	[✓] Always require a value in this field in order to save a record		
Unique	[ ] Do not allow duplicate values		
External ID	[ ] Set this field as the unique record identifier from an external system		

Figure 3-j: The Amount Field Details

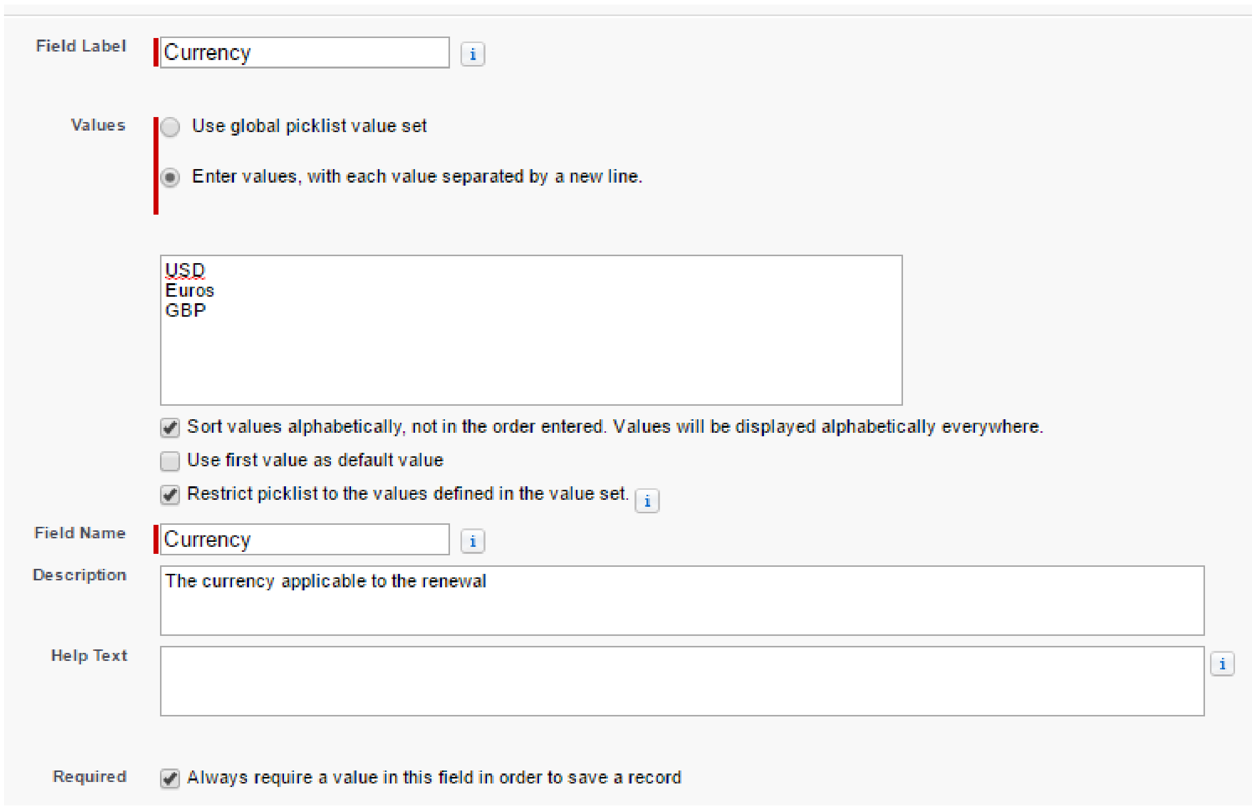
On **field-level** security there's nothing to do, so click **Next**, and on the **Add to page layouts** screen, click **Save** in order to create the field.

The Currency field

With the **Amount** field created, we can now create the **Currency** field. This will be used to indicate which currency applies to the **Renewal**. We've created **Picklists** before, so this should be familiar.

To create this field, click **New**, under **Fields & Relationships**. When asked, choose **Picklist** as the field type, and then click **Next**.

Type in **Currency** as the **Field Label,** and for the **Values** property, select the option **Enter values, with each value separated by a new line**. Then type in the values as shown in the following screenshot.


Field Label: Currency

Values:
( ) Use global picklist value set
(●) Enter values, with each value separated by a new line.

```
USD
Euros
GBP
```

[✓] Sort values alphabetically, not in the order entered. Values will be displayed alphabetically everywhere.
[ ] Use first value as default value
[✓] Restrict picklist to the values defined in the value set.

Field Name: Currency

Description: The currency applicable to the renewal

Help Text:

Required: [✓] Always require a value in this field in order to save a record

Figure 3-k: The Currency Field Details

Select the options **Sort values alphabetically** and **Always require a value in this field in order to save a record**. By default, the option **Restrict picklist to the values defined in the value set** is selected.

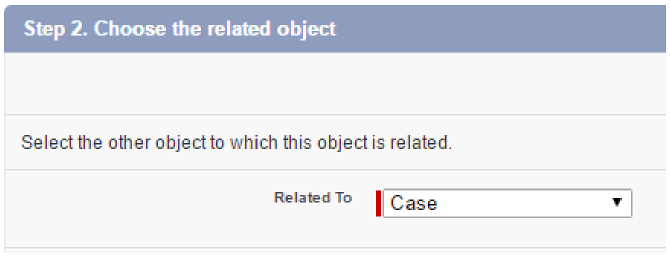
Following that, click **Next**—this will display the **Establish field-level security** screen, on which there's nothing to do, so click **Next** again. Finally, on the **Add to page layouts** screen, click **Save** in order to create the field.

The Cases Master-Detail field

We've now reached the last field of the **Renewal** object—exciting! The **Cases** field is a **Master-Detail** field linked to the **Case** standard object. In other words, once we create this field, Force.com will establish a one-to-many relationship between the **Renewal** and the **Case** object. This means that any given **Renewal** record could have multiple **Case** records.

Let's create this field. Click on **New** under **Fields & Relationships**. For the field-type, select the **Master-Detail Relationship** option, and click **Next**.

Immediately afterwards, you'll be asked to choose the related object. Select the **Case** object from the dropdown list.

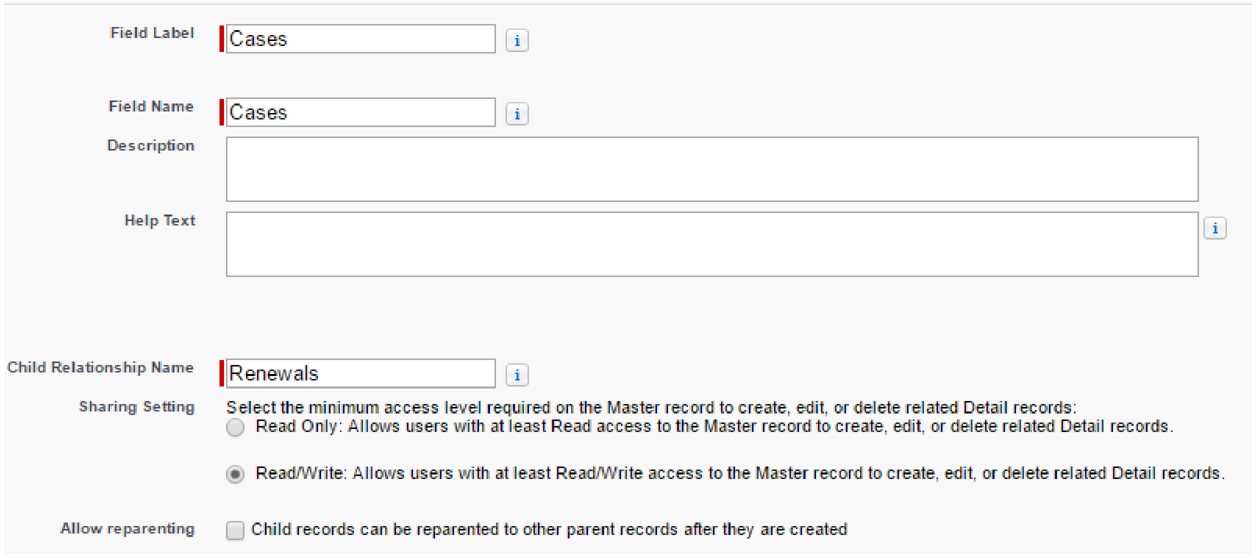


Select the other object to which this object is related.

Related To: Case

Figure 3-l: The Related Object for the Cases Field

Once you've selected this object, click **Next**. On the screen that follows, you'll be asked to specify the **Field Label** and other properties. Type in **Cases** as the **Field Label**, enter a **Description**, and then click **Next**.



Field Label: Cases

Field Name: Cases

Description:

Help Text:

Child Relationship Name: Renewals

Sharing Setting: Select the minimum access level required on the Master record to create, edit, or delete related Detail records:
- Read Only: Allows users with at least Read access to the Master record to create, edit, or delete related Detail records.
- [✓ Read/Write: Allows users with at least Read/Write access to the Master record to create, edit, or delete related Detail records.]

Allow reparenting: Child records can be reparented to other parent records after they are created

Figure 3-m: Details of the Cases Field

As you might have guessed, the following screen is the **field-level security**. Nothing to do really, so simply click **Next**.

On the **Add reference field to page Layouts** screen, click **Next**, and, finally, on the **Add custom related lists** screen, accept the default options and click **Save—**this will create the field.

Awesome—we have now created all the fields of the **Renewal** object. However, do you remember that there is one field that is part of the **Client** object that we couldn't create because it was dependent on having the **Renewal** object ready?

The field that we have yet to create is the **Latest Renewal Roll-Up Summary** field. So, let's go ahead and create it.

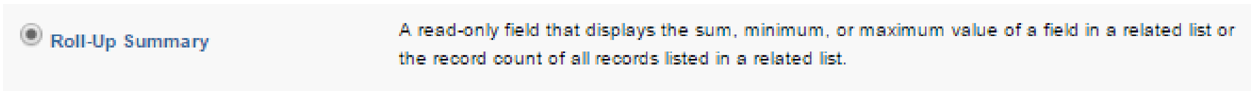
The Latest Renewal Roll-Up Summary field

The purpose of this field is to know which is the most recent **Renewal** object for any given **Client** record—this is what is known in Force.com as a **Roll-Up Summary** field.

It is an aggregation field—in this case we'll be aggregating by the **Creation Date** field of the **Renewal** object.

Under **PLATFORM TOOLS** > **Object and Fields** > **Object Manager**, locate the **Client** object previously created, and click it. Under **Fields & Relationships**, click **New**.

Select **Roll-Up Summary** as the field-type and click **Next**.



Roll-Up Summary — A read-only field that displays the sum, minimum, or maximum value of a field in a related list or the record count of all records listed in a related list.

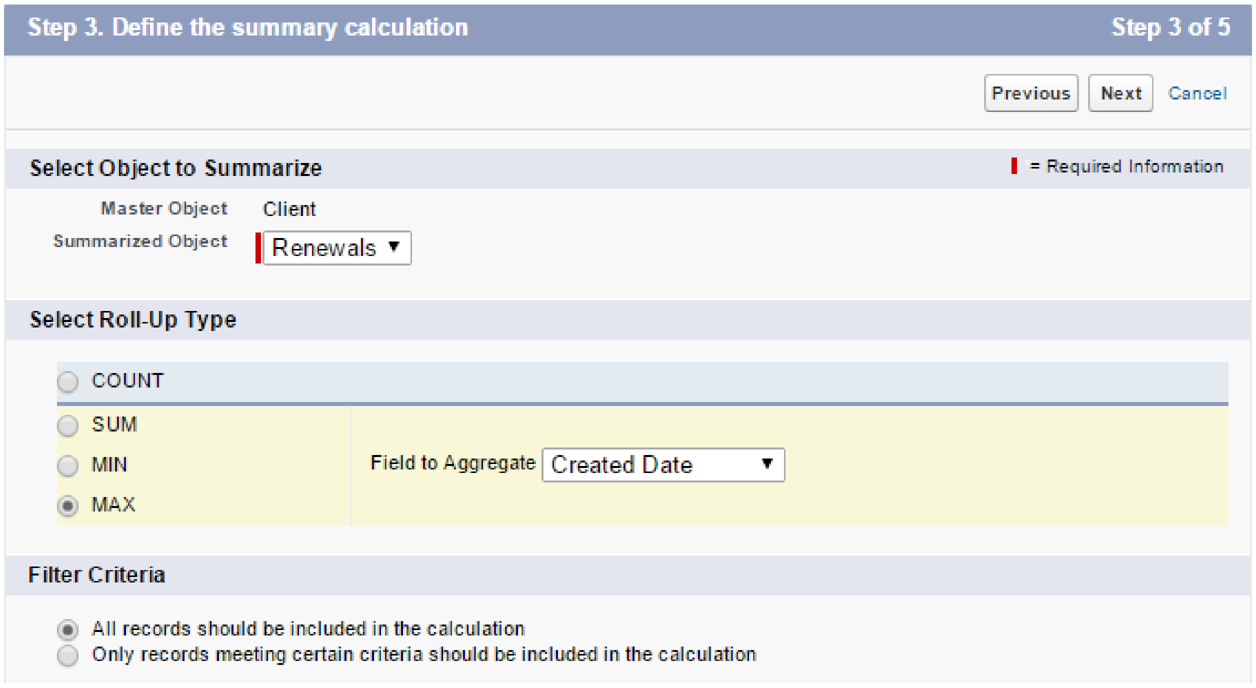
Figure 3-n: The Roll-Up Summary Field Type

Type in the **Field Label** (Latest Renewal) and a **Description**, and click **Next**. Now comes the interesting part, something we have not covered before.

In this step, we need to **Define the summary calculation**. Under the **Summarized Object** property, select **Renewals** from the dropdown list.

Under **Select Roll-Up Type**, select **MAX**, and on the **Field to Aggregate** property, choose the **Created Date** field from the list.

By default, the **Filter Criteria** is set to **All records should be included in the calculation.** Click **Next**.



Step 3. Define the summary calculation — Step 3 of 5

Select Object to Summarize — = Required Information

Master Object: Client
Summarized Object: Renewals

Select Roll-Up Type

( ) COUNT
( ) SUM
( ) MIN
(●) MAX

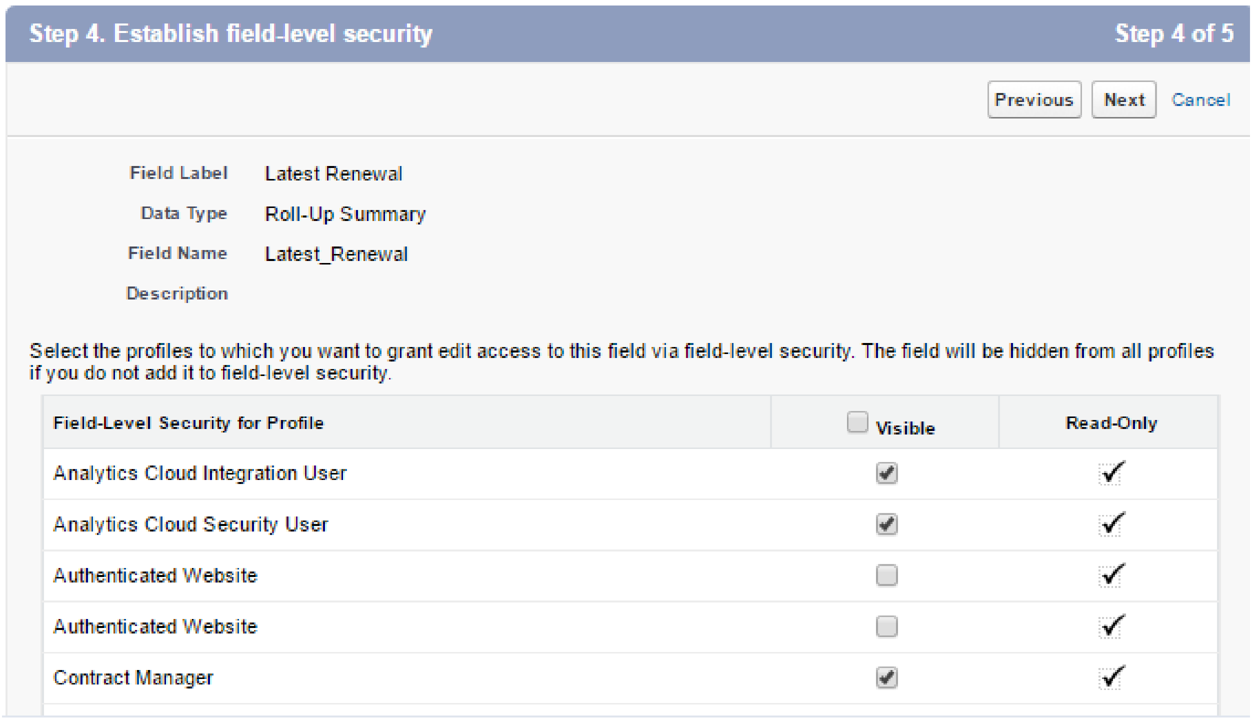
Field to Aggregate: Created Date

Filter Criteria

(●) All records should be included in the calculation
( ) Only records meeting certain criteria should be included in the calculation

Figure 3-o: The Define Summary Calculation Screen

On the **Establish field-level security** screen, leave the default options as shown in the following figure, and click **Next**.



Step 4. Establish field-level security — Step 4 of 5

Field Label	Latest Renewal
Data Type	Roll-Up Summary
Field Name	Latest_Renewal
Description	

Select the profiles to which you want to grant edit access to this field via field-level security. The field will be hidden from all profiles if you do not add it to field-level security.

Field-Level Security for Profile	Visible	Read-Only
Analytics Cloud Integration User	✓	✓
Analytics Cloud Security User	✓	✓
Authenticated Website	☐	✓
Authenticated Website	☐	✓
Contract Manager	✓	✓

Figure 3-p: The Establish Field-Level Security Screen

Finally, on the **Add to page layouts** screen, leave the default option as is, and click **Save**. We now have all the fields from both the **Client** and **Renewal** objects created.

Summary

Throughout this chapter, we explored step-by-step how to create each of the fields of the **Renewal** object—which is at the very core of our Customer Success app.

Furthermore, we've explored various field types and their relationship with the main **Client** object.

In the next chapter, we'll explore the Lightning Experience user interface, how to add, edit, and remove records, and also how add tabs for our custom objects.

Up to this point, we've been immersed in the design of relational objects and their fields, without which we would not be able to have an app on the Force.com platform. Going forward, we'll learn how bring these objects to life.

Chapter 4 UI Basics

Introduction

When you create an object in Force.com, you instantly get a user interface for it. Think about that for a moment—you automatically get **List Views** and **Object Views** with inline and full editing capabilities without having to add any code.

It is even possible to customize the **Views** by changing **Page Layouts** using a drag-and-drop, intuitive **Layout Properties** configuration interface. It is also possible to have different kinds of layouts, even for mobile devices. All this without writing a single line of code.

If you need more flexibility, there's Visualforce. It's a bit more complex than the drag-and-drop configuration interface, as you need to know some basic HTML markup along with some Apex tags; however, there's no need to know anything about styling or CSS. With Visualforce, a lot of functionality can be done with standard Apex controllers, which is also an advantage.

After Visualforce came Lightning—which is the user interface we have been seeing so far throughout this book. It's modern and responsive—it adapts nicely to different device formats and looks cool.

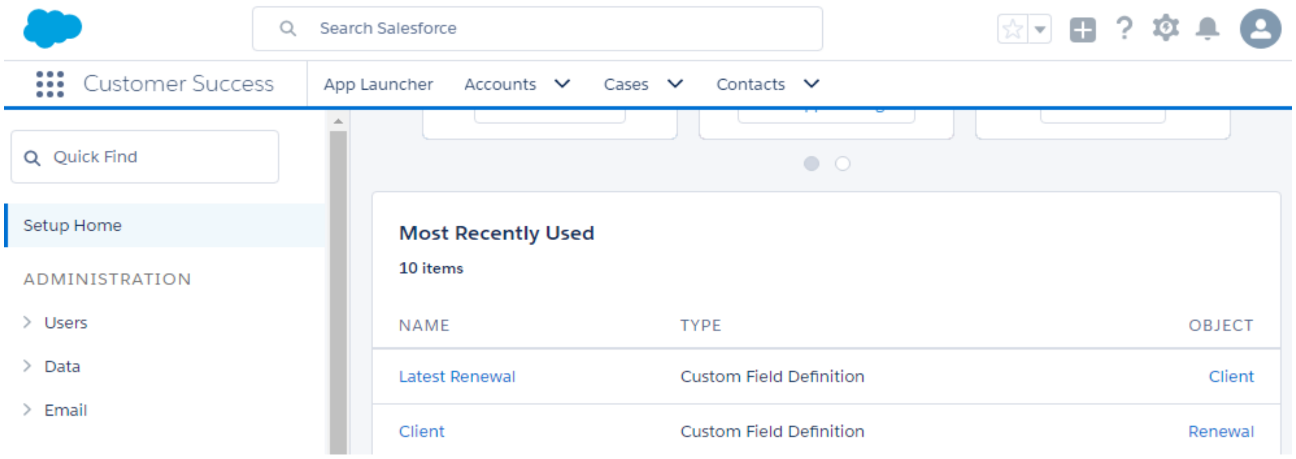
In this chapter, we'll explore the basics of user interfaces in Force.com, focusing on the Lightning Experience.

We'll also look at the layouts that were "auto-magically" built when we created our custom objects. Sounds interesting, so let's find out more.

Built-in layouts

In the previous chapters, when we created the **Client** and **Renewal** objects, Force.com automatically and behind the scenes created some layouts for us. Layouts are simply a way for us to see and manipulate fields within objects.

In the **Setup Home** page of our Lightning Experience, we should see a list of the **Most Recently Used** items. There you will surely find the **Client** and **Renewal** objects.



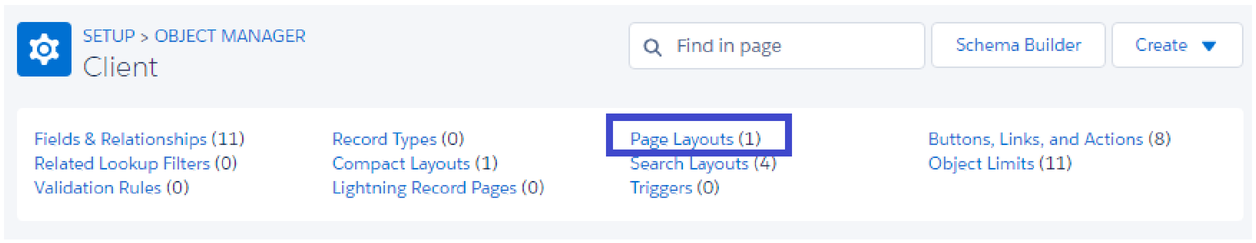
Most Recently Used

10 items

NAME	TYPE	OBJECT
Latest Renewal	Custom Field Definition	Client
Client	Custom Field Definition	Renewal

Figure 4-a: The List of Most Recently Used Items

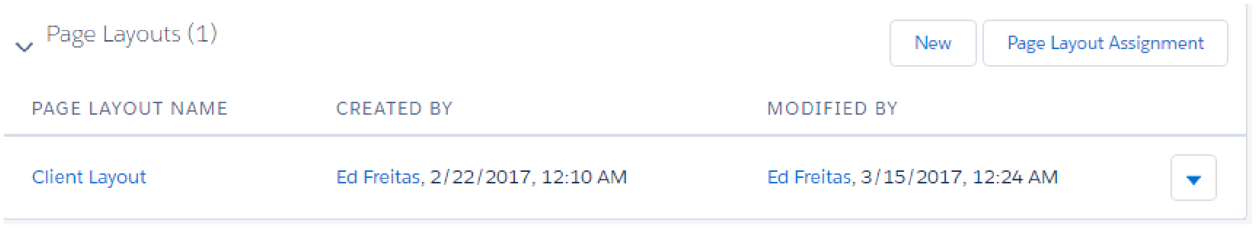
So, let's click on the **Client** object in order to explore the layouts that were automatically added when we created this object. Then, click **Page Layouts**.



Fields & Relationships (11)
Related Lookup Filters (0)
Validation Rules (0)
Record Types (0)
Compact Layouts (1)
Lightning Record Pages (0)
Page Layouts (1)
Search Layouts (4)
Triggers (0)
Buttons, Links, and Actions (8)
Object Limits (11)

Figure 4-b: The Client Object's Definition

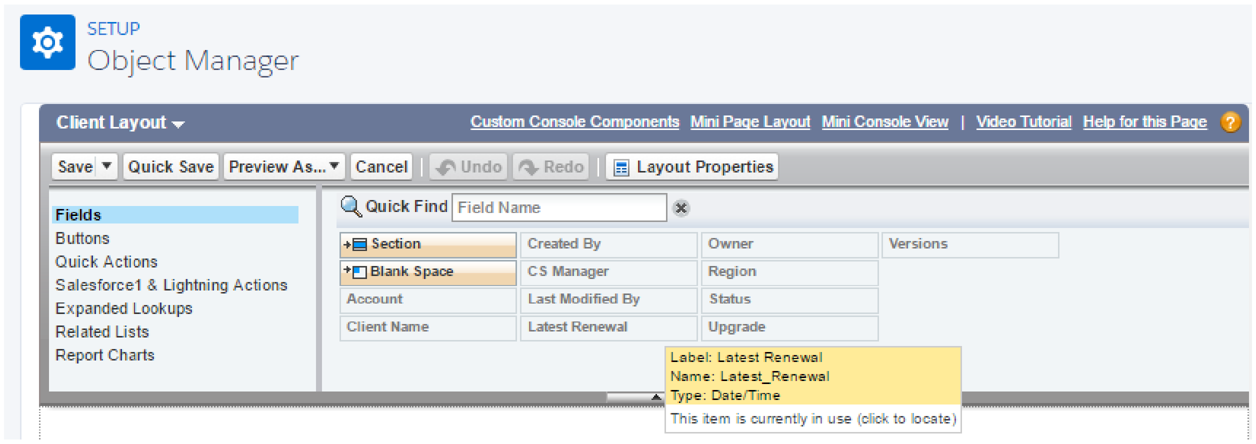
Click **Client Layout,** as seen in the following figure.



PAGE LAYOUT NAME	CREATED BY	MODIFIED BY
Client Layout	Ed Freitas, 2/22/2017, 12:10 AM	Ed Freitas, 3/15/2017, 12:24 AM

Figure 4-c: The Client Layout Item

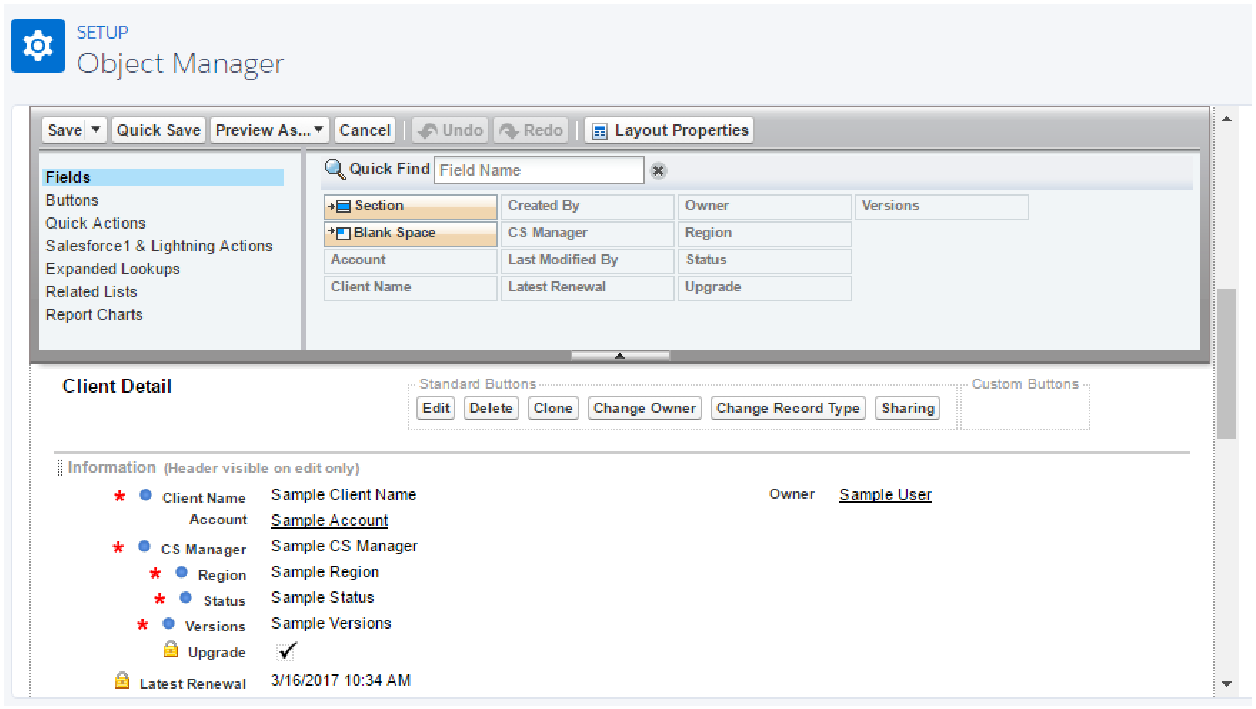
When you have clicked on that link, you'll be presented with the **Client Layout** configuration interface, which looks like Figure 4-d.



Label: Latest Renewal
Name: Latest_Renewal
Type: Date/Time
This item is currently in use (click to locate)

Figure 4-d: The Layout Configuration Interface

Scroll a bit further down to the **Client Detail** section. This shows exactly how Force.com has set up the **Client** layout for us—that is, in which order the fields will appear when a new **Client** record is created, or when an existing **Client** record is modified.

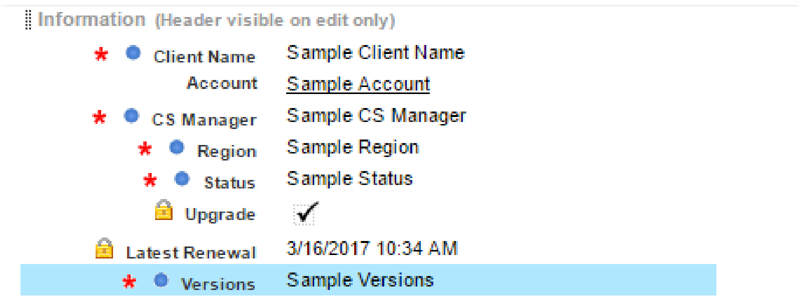


* Client Name	Sample Client Name	Owner	Sample User
Account	Sample Account		
* CS Manager	Sample CS Manager		
* Region	Sample Region		
* Status	Sample Status		
* Versions	Sample Versions		
Upgrade	✓		
Latest Renewal	3/16/2017 10:34 AM		

Figure 4-e: The Details of the Client Layout (Field Order)

Here we see the order in which the **Client** fields will appear on the screen, as seen under the **Client** section. If we want to rearrange the order, we can simply drag and drop any of the fields to the desired location.

For example, let's move down the **Versions** field, so it appears as the last field of the **Client** layout. In order to do this, simply click on the field and drag it down below the **Latest Renewal** field.



Information (Header visible on edit only)	
* ● Client Name	Sample Client Name
Account	Sample Account
* ● CS Manager	Sample CS Manager
* ● Region	Sample Region
* ● Status	Sample Status
Upgrade	✓
Latest Renewal	3/16/2017 10:34 AM
* ● Versions	Sample Versions

Figure 4-f: The Re-Ordered Client Layout

After you've done that, click **Save** in order keep this layout change.

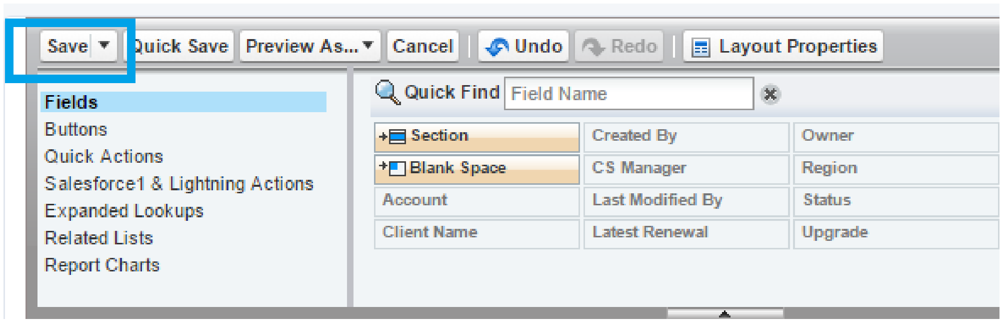


Figure 4-g: The Save Button on the Configuration Interface

Once you have clicked on the **Save** button, you will be redirected to the main setup screen of the **Client** object (under Setup **Home** > **PLATFORM TOOLS** > **Object Manager**).

We've now seen how to quickly alter one of the built-in layouts that Force.com has automatically created for us.

There's another visual tool called **Schema Builder** that also provides a drag-and-drop experience in order to customize objects and layouts (which we won't be covering).

Now, let's explore how these layouts actually look at runtime—in data-entry mode.

Accessing our custom objects—data-entry mode

Now that we've seen how easy it is to customize our object layouts, it's time to see how they look and feel during runtime—so that we can use them to add and edit data.

I want to take the opportunity to mention that in Force.com, there is really no actual concept of runtime versus design-time. I'd thought I'd use the word though, in order to distinguish between being in setup mode versus being in data-entry mode.

So far in our Lightning Experience, we've been in setup mode. In order to go into data-entry mode, which would be the equivalent of running our custom application, we have to explicitly click on the **App Launcher** button (the 3-by-3 dots icon in the upper-left side), and then select our **Customer Success** app.

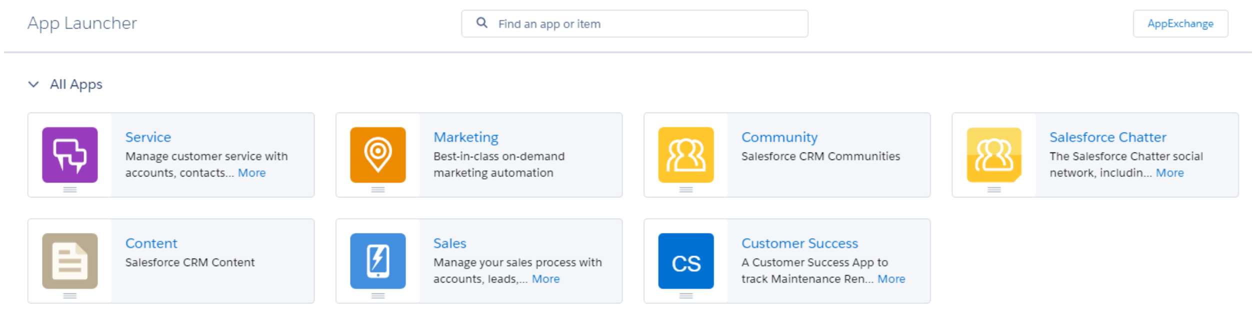

App Launcher
Find an app or item
AppExchange
All Apps
Service
Manage customer service with accounts, contacts... More
Marketing
Best-in-class on-demand marketing automation
Community
Salesforce CRM Communities
Salesforce Chatter
The Salesforce Chatter social network, includin... More
Content
Salesforce CRM Content
Sales
Manage your sales process with accounts, leads,... More
CS
Customer Success
A Customer Success App to track Maintenance Ren... More

Figure 4-h: App Launcher

Once you've clicked on the **Customer Success** App option, you'll see the following.

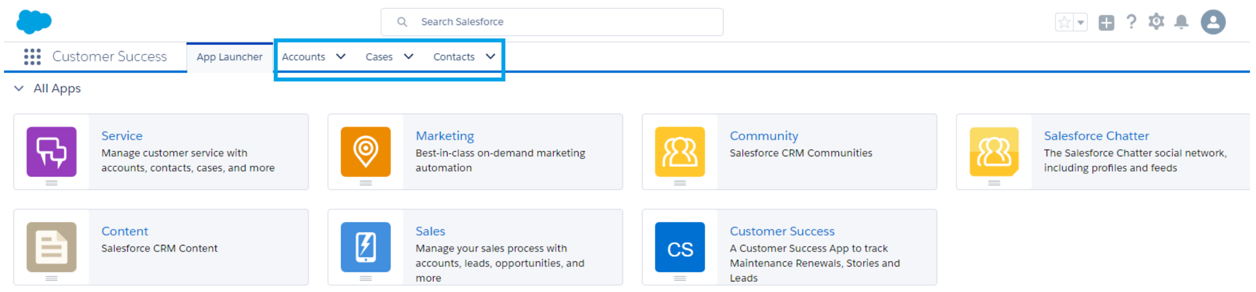

Search Salesforce
Customer Success
App Launcher
Accounts
Cases
Contacts
All Apps
Service
Manage customer service with accounts, contacts, cases, and more
Marketing
Best-in-class on-demand marketing automation
Community
Salesforce CRM Communities
Salesforce Chatter
The Salesforce Chatter social network, including profiles and feeds
Content
Salesforce CRM Content
Sales
Manage your sales process with accounts, leads, opportunities, and more
CS
Customer Success
A Customer Success App to track Maintenance Renewals, Stories and Leads

Figure 4-i: The Customer Success App

Notice how, by default, the **App Launcher** tab is displayed, and only the **Accounts**, **Cases**, and **Contacts** tabs are visible. These correspond to the **Account**, **Case**, and **Contact** objects, respectively.

But where are the **Client** and **Renewal** custom objects we created? That's a good question.

Essentially, in order for us to add and edit data for both the **Client** and **Renewal** objects, we must first add a tab to the **Customer Success** user interface for each object.

Because neither the **Client** nor **Renewal** objects have a corresponding tab defined, we cannot access these objects yet in data-entry mode. So, let's add these tabs.

Adding application tabs

On the top part of the screen, click **Gear** and then on the **Setup** option—but open it in a new browser tab. In the **Quick Find** search box, type in the word **tabs**, and you'll see the following.

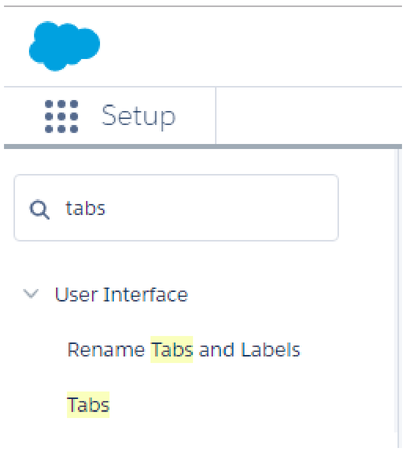
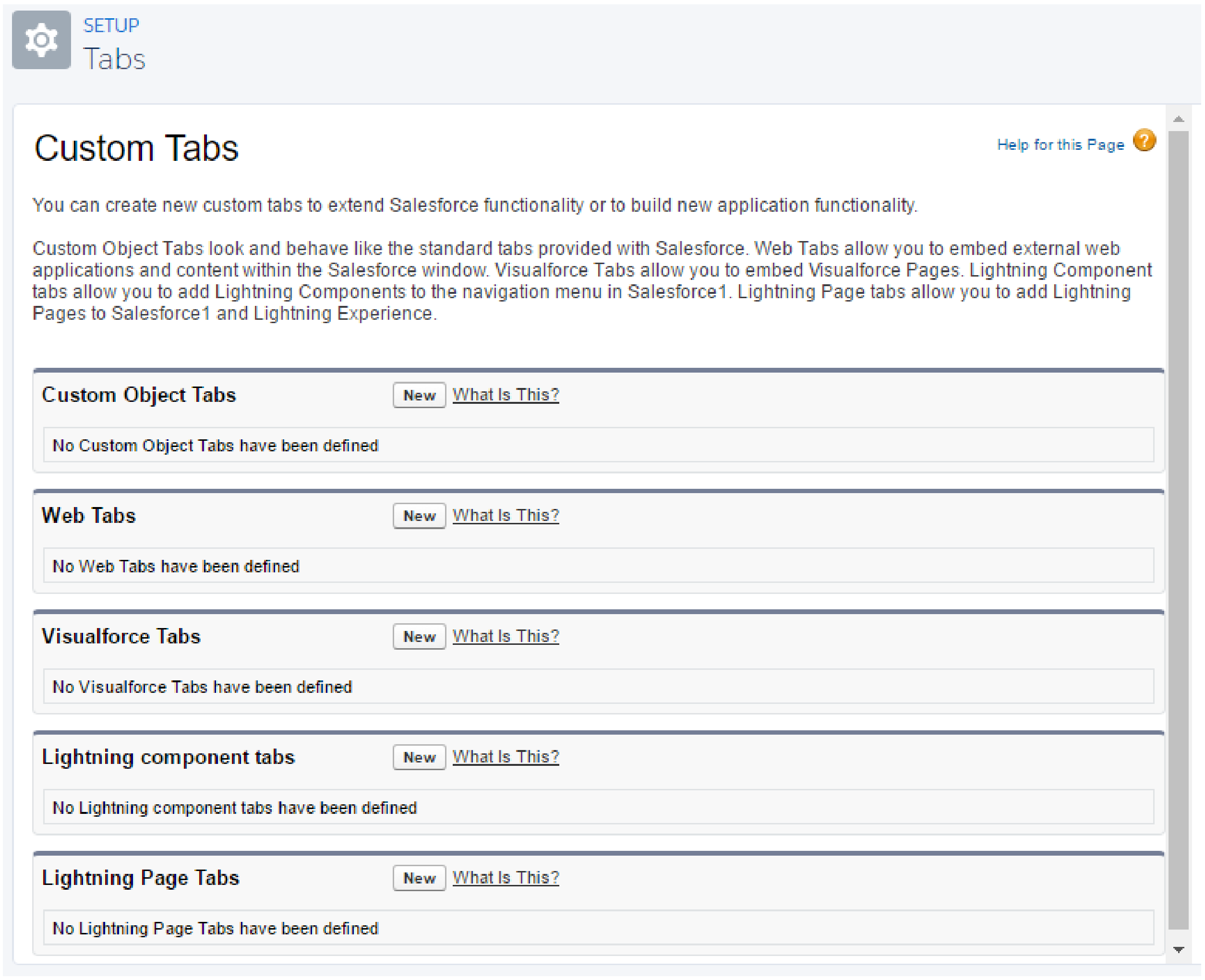


Figure 4-j: Searching for the Tabs Option within the Setup

Then, click on the **Tabs** option, which will take you to the following screen.



Custom Tabs

You can create new custom tabs to extend Salesforce functionality or to build new application functionality.

Custom Object Tabs look and behave like the standard tabs provided with Salesforce. Web Tabs allow you to embed external web applications and content within the Salesforce window. Visualforce Tabs allow you to embed Visualforce Pages. Lightning Component tabs allow you to add Lightning Components to the navigation menu in Salesforce1. Lightning Page tabs allow you to add Lightning Pages to Salesforce1 and Lightning Experience.

Custom Object Tabs

No Custom Object Tabs have been defined

Web Tabs

No Web Tabs have been defined

Visualforce Tabs

No Visualforce Tabs have been defined

Lightning component tabs

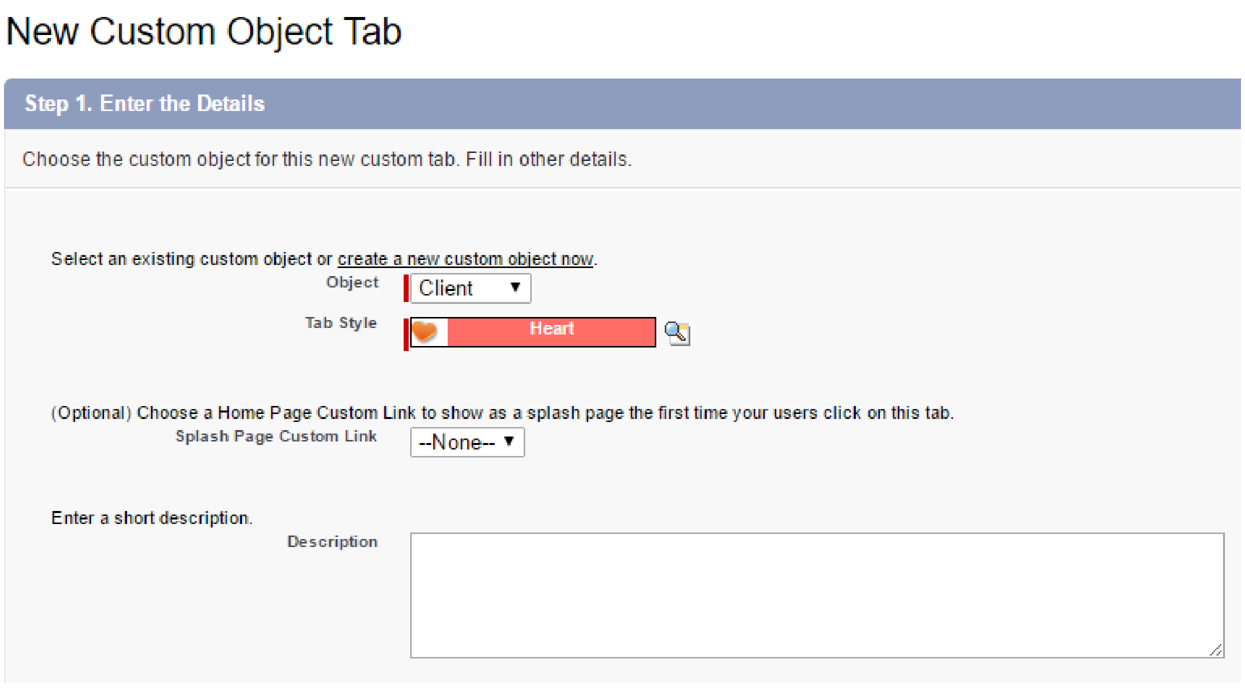
No Lightning component tabs have been defined

Lightning Page Tabs

No Lightning Page Tabs have been defined

Figure 4-k: The Tabs Setup Screen

Let's first create a **Tab** for our **Client** object. Click **New** right next to **Custom Object Tabs**.


New Custom Object Tab

Step 1. Enter the Details

Choose the custom object for this new custom tab. Fill in other details.

Select an existing custom object or create a new custom object now.

Object: Client

Tab Style: Heart

(Optional) Choose a Home Page Custom Link to show as a splash page the first time your users click on this tab.

Splash Page Custom Link: --None--

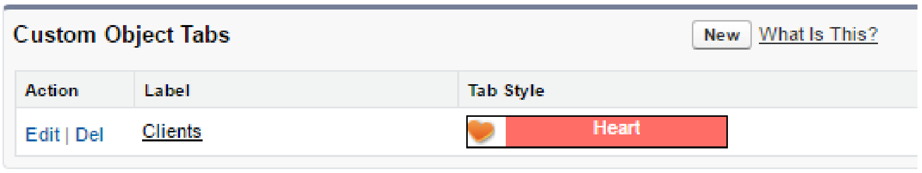
Enter a short description.

Description:

Figure 4-l: The Clients Tab Setup

Select **Client** as the object, and select any option available for the **Tab Style**—in my case, I have chosen **Heart**. Then, click **Next**.

In **Step 2—Add to Profiles**, simply leave the default options and click **Next**. For **Step 3—Add to Custom Apps**, do the same: leave the default options and click **Save**. You should now see the **Clients** tab created.


Custom Object Tabs

Action	Label	Tab Style
Edit | Del	Clients	Heart

Figure 4-m: The Clients Tab Created

Let’s now do the same for the **Renewal** object, and create a tab for it. Click **New** button, which is next to **Custom Object Tabs**, and follow the same steps, but this time select **Renewal** as the **Object**, and also a different **Tab Style—**in my case, I’ll select **Sack** as the style.

If you now switch back to the **Customer Success** app, which you left open on your other browser tab, and refresh the page, you will see two new tabs.

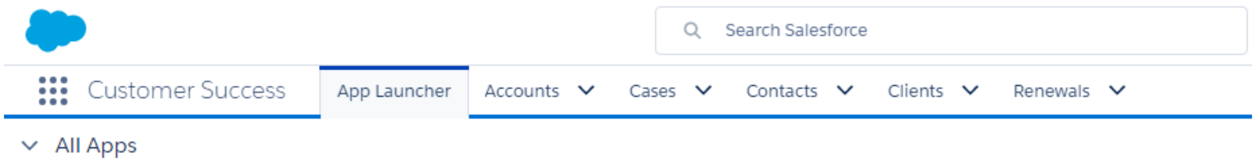


Figure 4-n: The Customer Success App with the Clients & Renewals Tabs

Adding records

Let’s now add some data to our application. Before we can add any data to the **Client** object, we’ll create a couple of **Accounts**. This is because our **Client** object has a dependency on the **Account** object. Once we’ve created at least one **Account** record, we can then create a **Client** record.

Before we can create a **Renewal** record, we’ll need to have a **Client** record created—as the **Renewal** object has a dependency on the **Client** object. At least one **Case** record will have to be added as well, because the **Renewal** object also depends on it—so keep this in mind. Now let’s add some data.

Click on the **Accounts** tab, and then **New**, in order to create a new **Account** record.

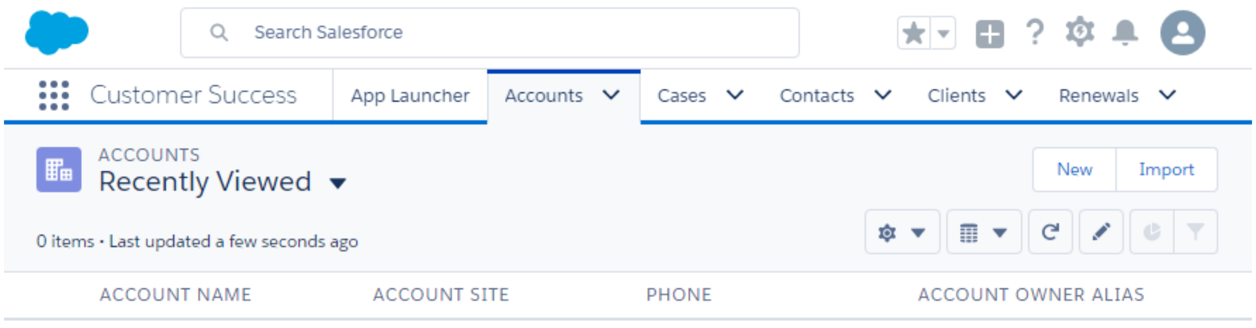
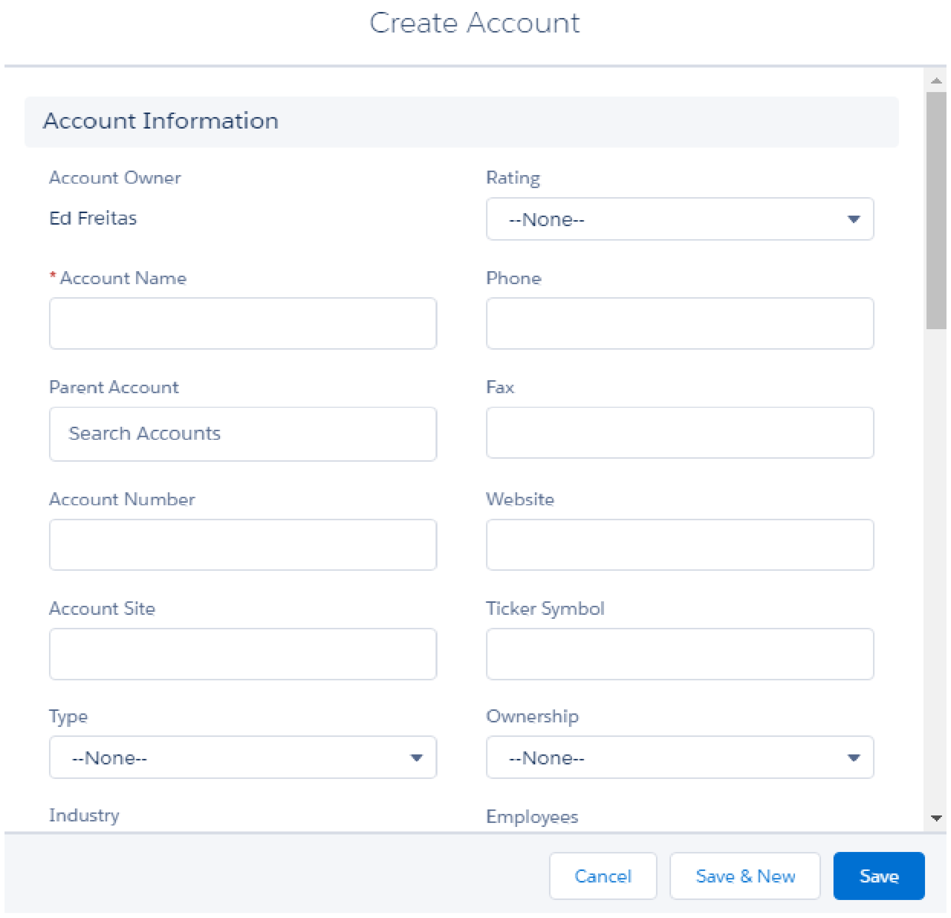


Figure 4-o: The Accounts Tab

Once you’ve clicked on the **New** button, you’ll see the data-entry form shown in Figure 4-p. Fill in the required fields and some other non-required fields you might find useful.

An **Account** record refers to the very high-level details of a customer’s organization that represents your **Client**. You can enter details related to their **Billing Address** and also **SLA** info. Once done, click **Save**.


Create Account

Account Information

Account Owner	Rating
Ed Freitas	--None--
* Account Name	Phone
Parent Account	Fax
Search Accounts	
Account Number	Website
Account Site	Ticker Symbol
Type	Ownership
--None--	--None--
Industry	Employees

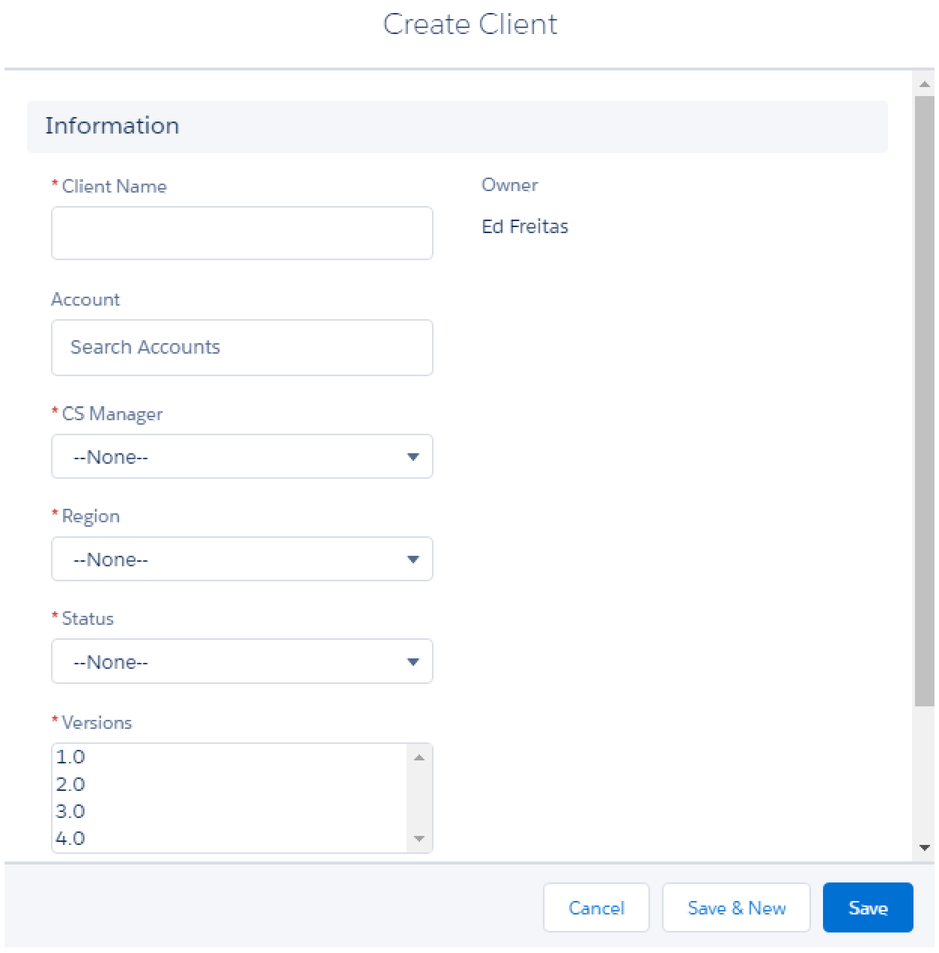
Figure 4-p: The Create Account Form

Feel free to create a few extra **Account** records if you wish. When you have done that, click on the **Client** tab in order to create some **Client** records for each of those **Accounts** created.

It might seem at first that we are duplicating information by adding **Account** and **Client** records. Although the relationship between **Account** and **Client** is usually a one-to-one relationship, there are cases when we might want to have more than one **Client** for the same **Account**—for instance, for having two separate business units.

The other reason is that the **Account** object is used by other Force.com apps that are provided out-of-the-box, so it's convenient not to extend this object with custom properties that are specific to the Customer Success app, but instead to add those specific properties to a new custom object—this is another reason why the **Client** object was created.

With that said, let's add the **Client** records. Assuming you are already on the **Clients** tab, click **New**. The **Client** record data-entry form will appear as follows.


Create Client

Information

* Client Name

Owner

Ed Freitas

Account

Search Accounts

* CS Manager

--None--

* Region

--None--

* Status

--None--

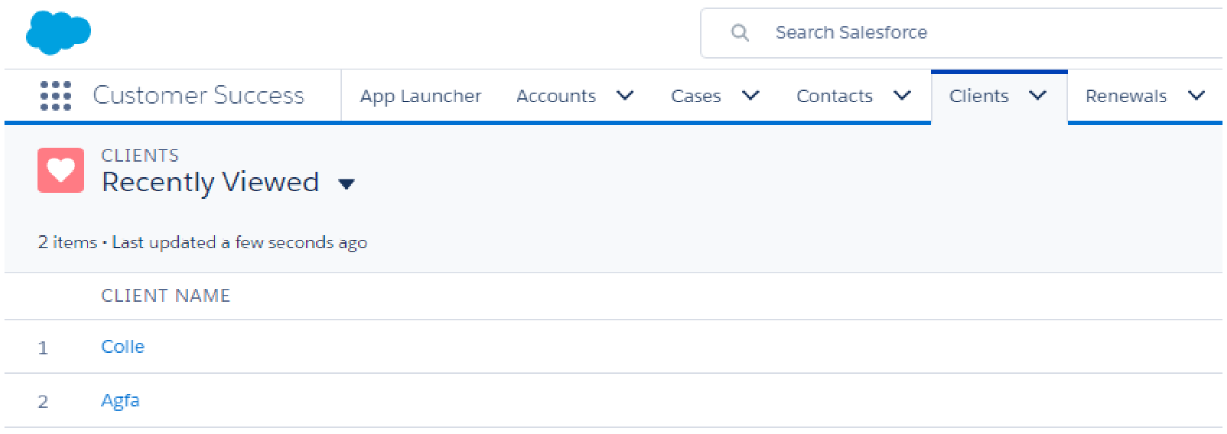
* Versions

1.0
2.0
3.0
4.0

Figure 4-q: The Create Client Form

Enter the required fields, then click **Save**, or **Save & New** if you want to create more **Client** records.

By now you should have created one or more **Client** records. I created a couple of them, and what I see under my **Clients** tab is shown in Figure 4-r.



CLIENTS

Recently Viewed

2 items • Last updated a few seconds ago

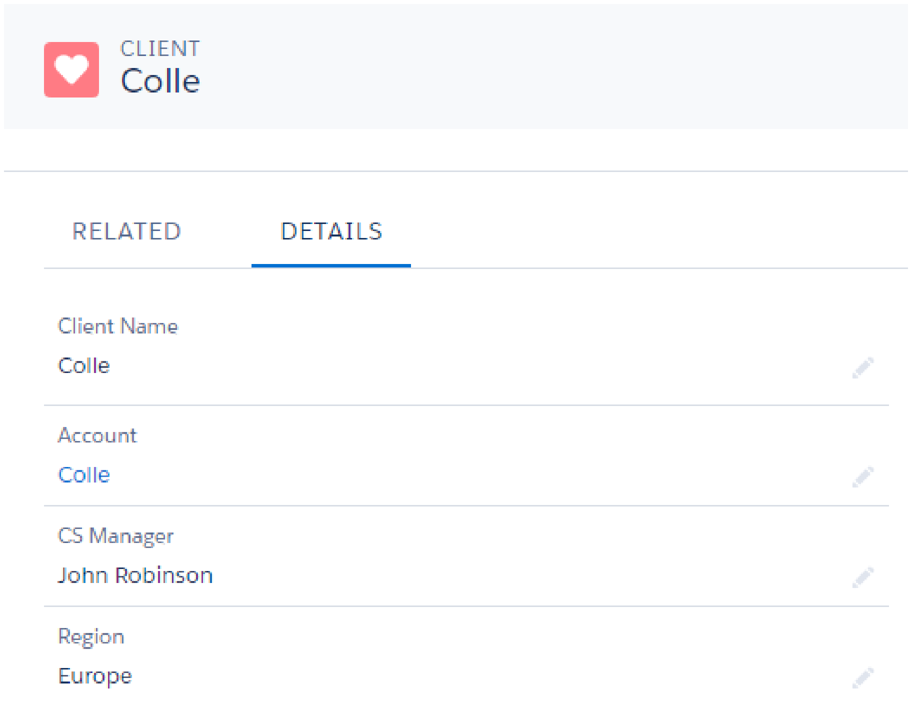
	CLIENT NAME
1	Colle
2	Agfa

Figure 4-r: Client Records

Feel free to add some **Renewal** records as well. This will come in handy later.

Editing data

Now that we have some records and data to work with, we can easily edit any record by clicking on the record name from the list—this will bring up the data associated with the record as follows.



Client Name
Colle

Account
Colle

CS Manager
John Robinson

Region
Europe

Figure 4-s: Details of a Client Record

If you would like to make any changes to this record, all you need to do is to double-click on any part of the record—when you do that, the screen will look as follows.

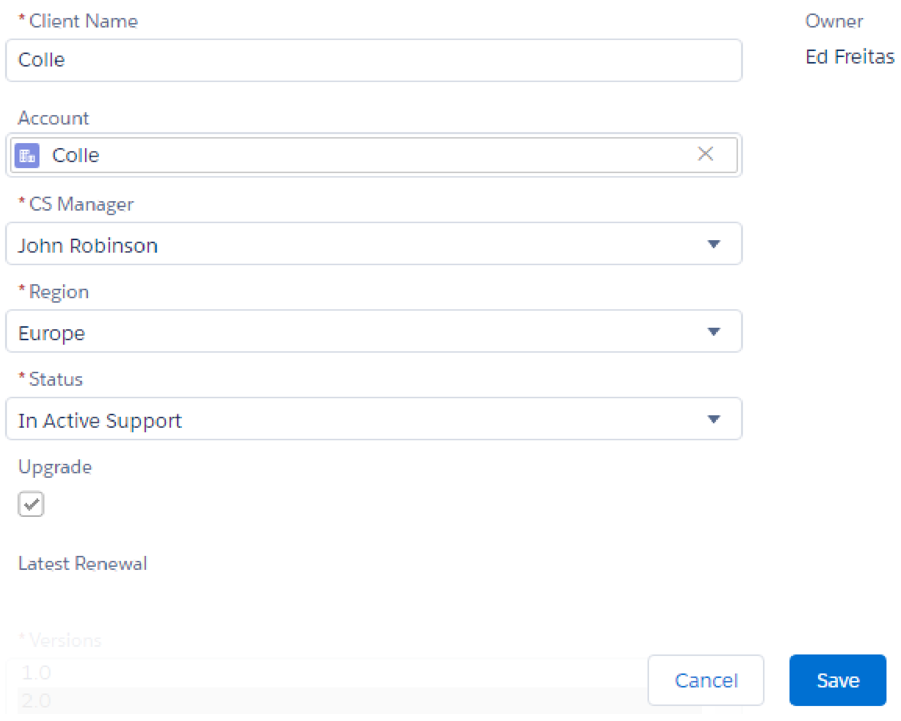

* Client Name: Colle
Owner: Ed Freitas
Account: Colle
* CS Manager: John Robinson
* Region: Europe
* Status: In Active Support
Upgrade: ✓
Latest Renewal
* Versions
1.0
2.0

Figure 4-t: Client Record—in Edit Mode

Notice that when you double-click on a record, it goes into Edit Mode, which means that you can change any of the values for any of the fields. Once you are done making changes, simply click **Save**.

Deleting a record

Removing a record is also very simple. From the record list, there's an arrow button for each record that has a **Delete** option.

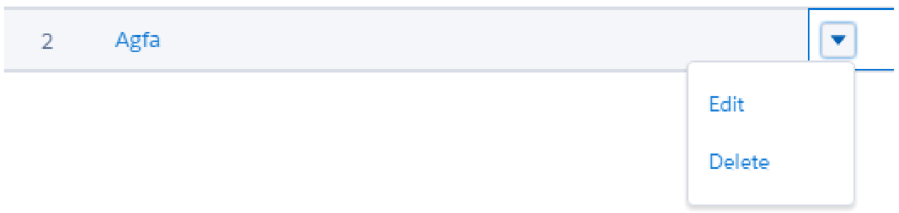


Figure 4-u: The Delete Record Option

Clicking **Delete** will remove that specific record.

Summary

Throughout this chapter, we've explored the Lightning Experience **Layouts** that are provided out-of-the-box when objects are created, and also how to modify them.

We’ve had a look at how to add, edit, and remove record data, and how to add custom objects to the user interface by adding **Tabs**.

In essence, without any programming required—except the use of a couple of formula fields—we have created a data-entry application with a clean and modern looking UI and that is powered by a strong relational object database. We did this simply by following a series of easy steps and intuitive point-and-click wizards. I personally think this is quite powerful, and part of the beauty of Force.com.

In the following chapters, we’ll move our attention to the programming side of the platform and how to add specific custom logic.

Chapter 5 Lightning App Basics

Two web architectures

When the Salesforce CRM application was originally developed, Visualforce was the primary (and only) user interface web architecture powering the system. It still is important today, and a lot of customers on the platform use it every day.

Visualforce works this way: the HTML for the DOM is built on the server using Visualforce markup, and no JavaScript is required. So, the server is doing the hard work of rendering the UI. Most user interaction results in a request to the server and requires the server to return all or part of a page.

With Lightning—which is based on the open-source Aura framework, the DOM is built and modified directly on the client using JavaScript. With Aura, the user interaction processed by JavaScript only requests the data it needs from the server (if any), which is much more efficient, as it reduces the load on the server and consumes much less data.

One of the cool things about Lightning is that you need to use JavaScript in order to go beyond trivial applications, which is something that most modern full stack and frontend developers appreciate, whereas with Visualforce, no JavaScript is required (although it can still be used).

As the Lightning Experience is definitely the way Salesforce and Force.com apps are going, this is what we'll be exploring. So, let's see how we can create a very basic Lightning App manually.

Hello World in Lightning

In order to understand the basics of Lightning, it's best to create a small and simple Hello World example. In order to do that, we'll need to open the **Developer Console**.

To open this up, click the **Gear** icon at the top of the Lightning UI. This will display a pop-up menu with **Setup** and **Developer Console** options.

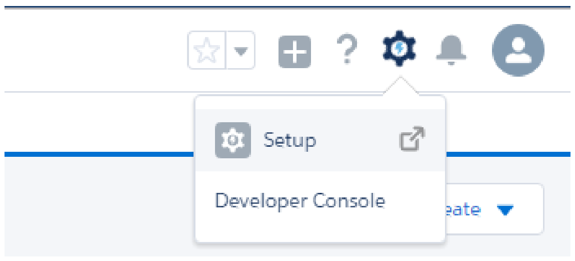


Figure 5-a: The Developer Console Option

Click the **Developer Console** option in order to open it up. By default, it opens as a new pop-up browser window. However, I recommend you open it as a new browser tab instead, which is a bit handier to work with.

Once opened, the **Developer Console** looks like Figure 5-b. As you can see, it resembles a small development or debugging environment, although it is running in the browser.

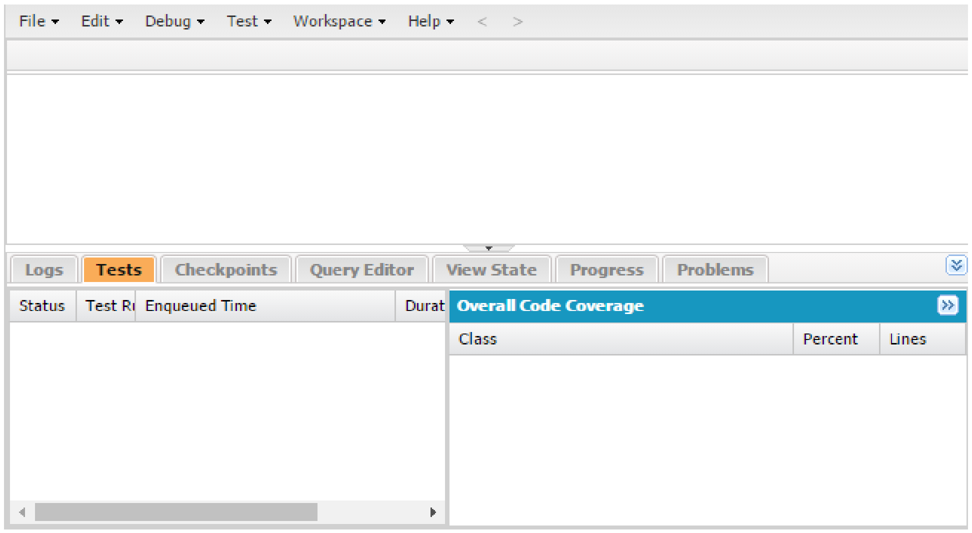


Figure 5-b: The Developer Console

To create our Hello World application, click **File** > **New** > **Lightning Application**.

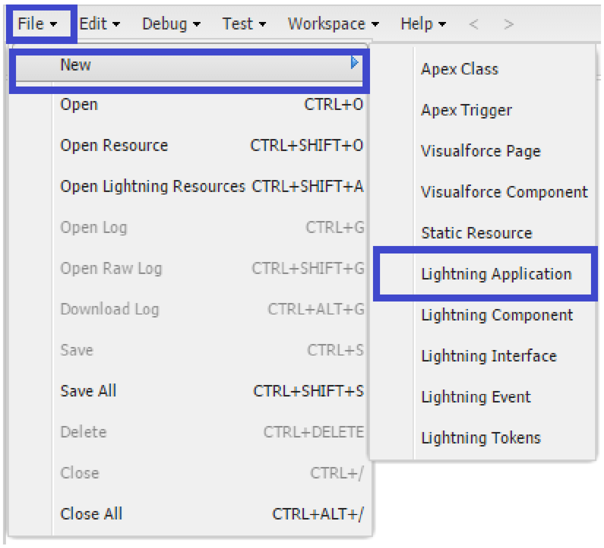
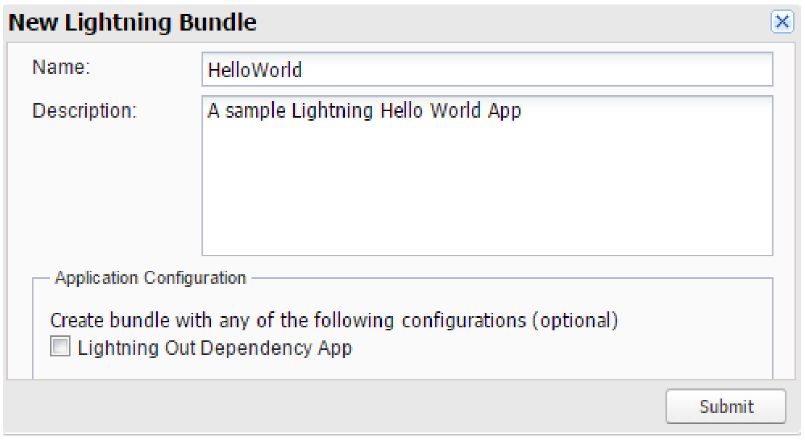


Figure 5-c: The Developer Console Menu

Once you have done that, an in-browser pop-up window will appear, which looks like Figure 5-d.



HelloWorld

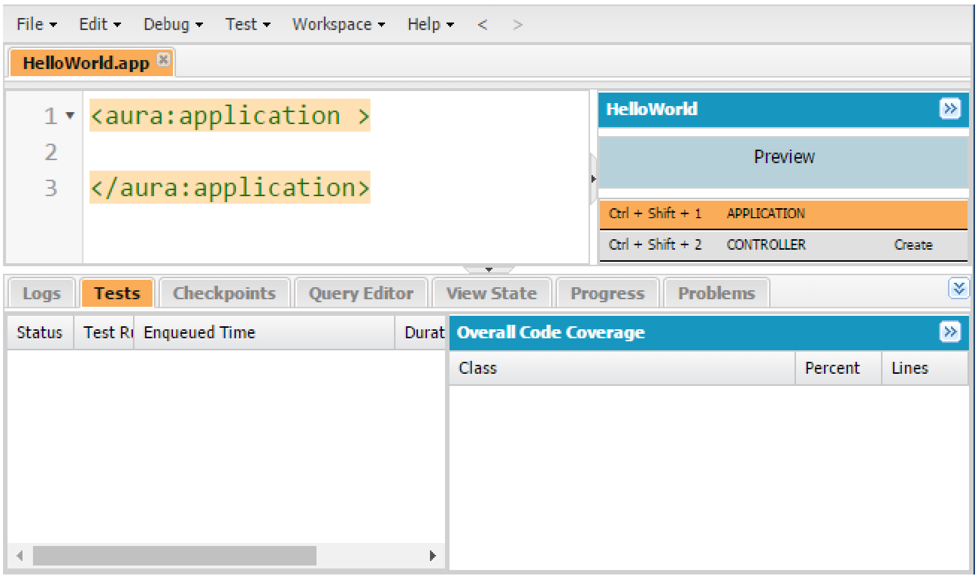
A sample Lightning Hello World App

Create bundle with any of the following configurations (optional)

☐ Lightning Out Dependency App

Figure 5-d: Creating the New Lightning Sample App

Enter the **Name** and **Description**, then click **Submit** to get started. Once that has been done, you will see the following Aura application tags on the **Developer Console**.



```
<aura:application >

</aura:application>
```

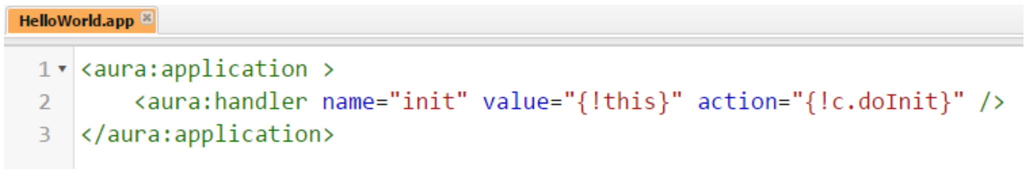
Figure 5-e: The Newly Created Lightning Sample App

Let’s create a **`handler`** tag that will wait for the **`init`** event, which occurs when a page is fully initialized. When that event occurs, we’ll call the controller’s **`doInit`** method.

Code Listing 5-a: The Aura Handler for the Init Event

```
<aura:handler name="init" value="{!this}" action="{!c.doInit}" />
```

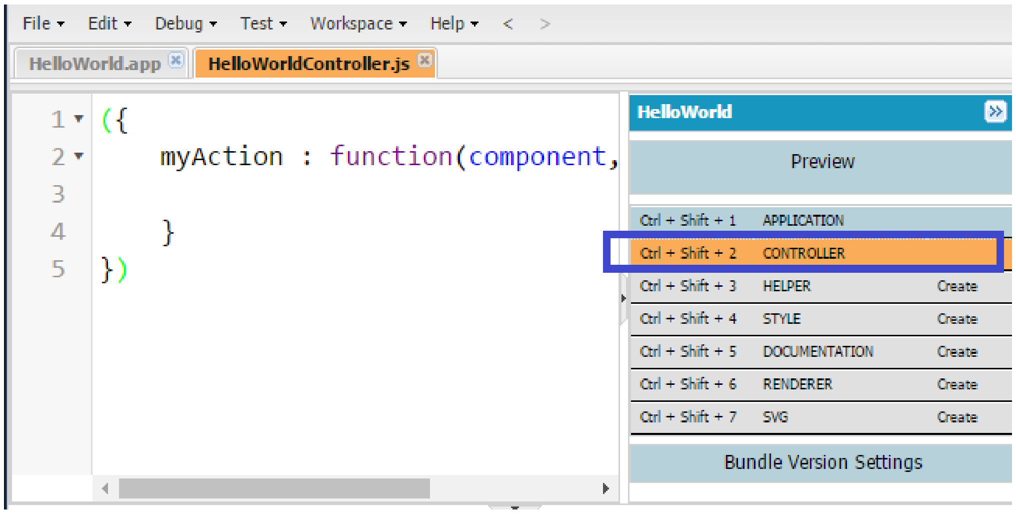
Let’s paste this code within the Aura application tags. Then under the **File** menu, click **Save**.



```
<aura:application >
    <aura:handler name="init" value="{!this}" action="{!c.doInit}" />
</aura:application>
```

Figure 5-f: The Aura Handler Tag

Next, we need to edit the controller. In order to do that, go to the right side of the **Developer Console** screen and click the **CONTROLLER** tab, as seen in Figure 5-g.


File
Edit
Debug
Test
Workspace
Help
HelloWorld.app
HelloWorldController.js
HelloWorld
Preview
Ctrl + Shift + 1 APPLICATION
Ctrl + Shift + 2 CONTROLLER
Ctrl + Shift + 3 HELPER Create
Ctrl + Shift + 4 STYLE Create
Ctrl + Shift + 5 DOCUMENTATION Create
Ctrl + Shift + 6 RENDERER Create
Ctrl + Shift + 7 SVG Create
Bundle Version Settings

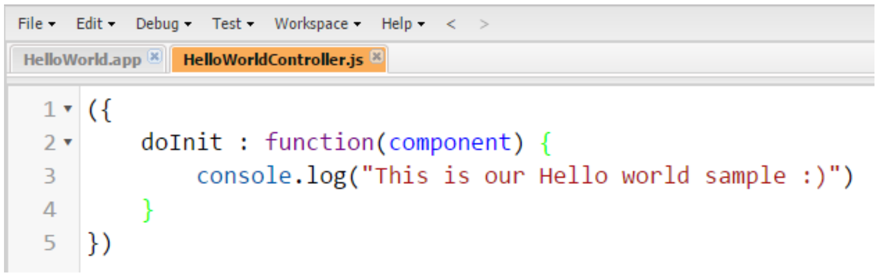
```
({
    myAction : function(component,

    }
})
```

Figure 5-g: The HelloWorldController.js File

This creates a controller JavaScript file with a default **`myAction`** function. For now, just replace the **`myAction`** function name with **`doInit`**. Also, we do not need the **`event`** and **`helper`** parameters, so they can be removed from the function.

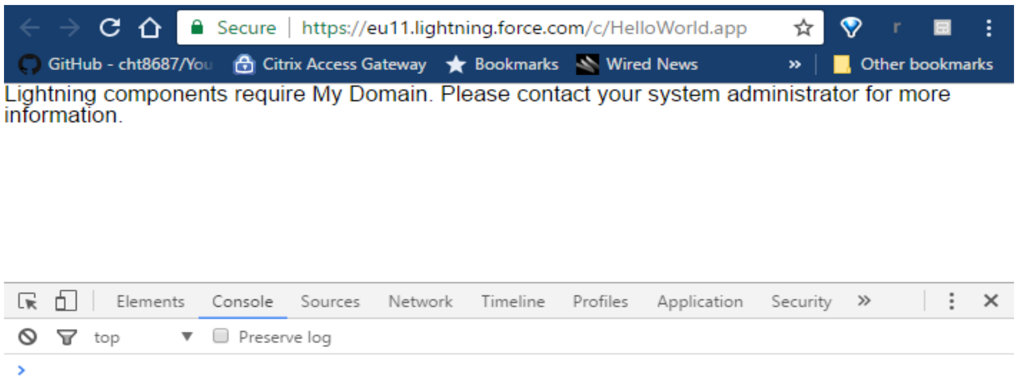
Inside the body of the **`doInit`** function, add a **`console.log`** statement, and then click the **File** menu, and then **Save**. You should have something like Figure 5-h.



```
({
    doInit : function(component) {
        console.log("This is our Hello world sample :)")
    }
})
```

Figure 5-h: The Modified HelloWorldController.js File

In order to view what we have just done, click on the **Preview** tab seen on the right side of the screen, just above the CONTROLLER tab.



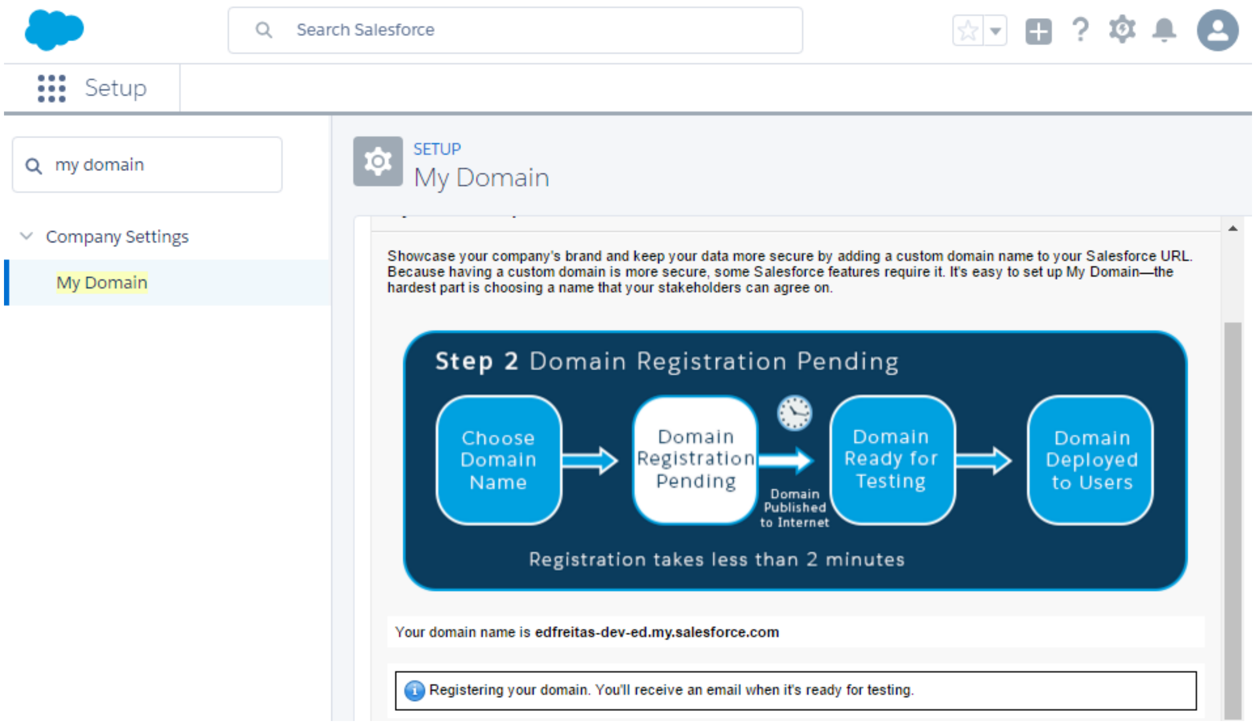
Lightning components require My Domain. Please contact your system administrator for more information.

Figure 5-i: Previewing the HelloWorld Lightning App

Oops! What happened here? When previewing our app, we have received a message saying that Lightning components require **My Domain**. What does this mean?

Well, **My Domain** is a setting that is available within the Lightning Experience, under **Company Settings**, and it is accessible by typing in the words **My Domain** on the **Quick Find** search box.

In my case, I entered my domain name (*edfreitas*) and submitted it. Force.com also checks for the domain's availability—i.e. that it has not been taken by someone else. It looks like Figure 5-j.



Showcase your company's brand and keep your data more secure by adding a custom domain name to your Salesforce URL. Because having a custom domain is more secure, some Salesforce features require it. It's easy to set up My Domain—the hardest part is choosing a name that your stakeholders can agree on.

Step 2 Domain Registration Pending
Choose Domain Name
Domain Registration Pending
Domain Published to Internet
Domain Ready for Testing
Domain Deployed to Users
Registration takes less than 2 minutes

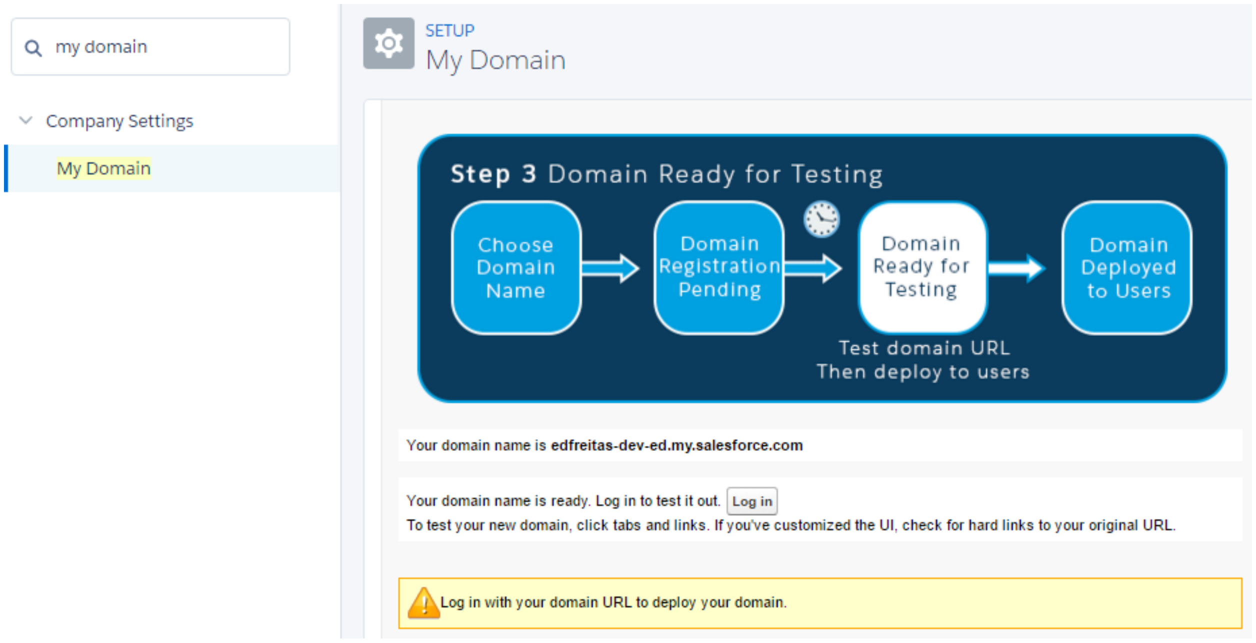
Your domain name is **edfreitas-dev-ed.my.salesforce.com**

Registering your domain. You'll receive an email when it's ready for testing.

Figure 5-j: My Domain Settings

Notice that domains can take several minutes to get registered. Once the domain is ready, you will receive an email stating that it is available to be used.

Before we can preview our Hello World Lightning app again, we'll need to log in with our new domain. We can do this if we refresh the **My Domain** settings page and click **Log in**.



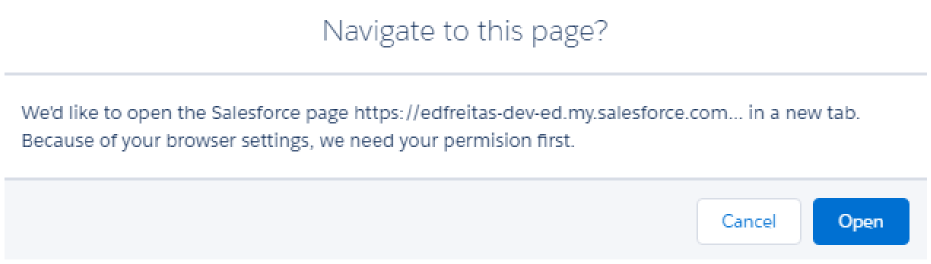
Your domain name is **edfreitas-dev-ed.my.salesforce.com**

Your domain name is ready. Log in to test it out. Log in
To test your new domain, click tabs and links. If you've customized the UI, check for hard links to your original URL.

Log in with your domain URL to deploy your domain.

Figure 5-k: My Domain Settings with the Log in Button

Once you click **Log in**, a pop-up message will appear asking if you want to navigate to that domain's specific URL.

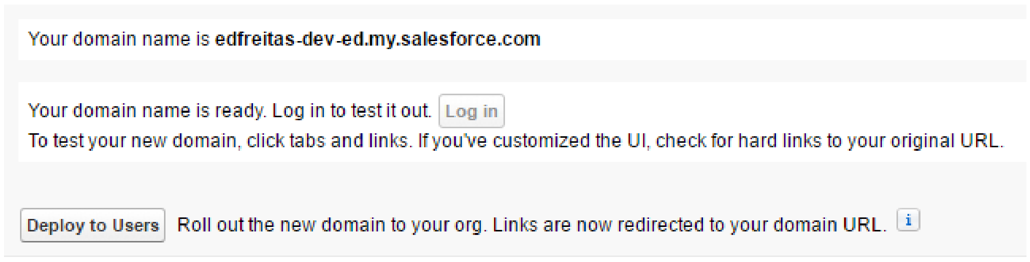

Navigate to this page?

We'd like to open the Salesforce page https://edfreitas-dev-ed.my.salesforce.com... in a new tab. Because of your browser settings, we need your permision first.

Figure 5-l: Navigate to the Domain's URL

By clicking on the **Open** button, you'll be redirected to the Force.com login page, where you'll be asked to sign in with your username and password. Then click **Log in**.

Once logged in, you'll be redirected to the **My Domain** page again, where you'll be requested to deploy the domain to users. This is done by clicking **Deploy to Users**.


Your domain name is **edfreitas-dev-ed.my.salesforce.com**

Your domain name is ready. Log in to test it out. Log in
To test your new domain, click tabs and links. If you've customized the UI, check for hard links to your original URL.

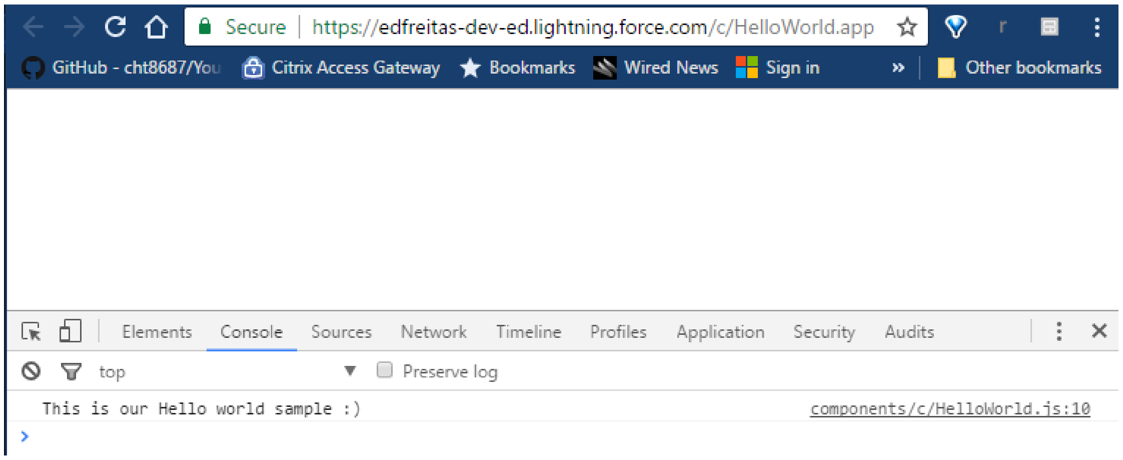
Deploy to Users Roll out the new domain to your org. Links are now redirected to your domain URL.

Figure 5-m: Deploying the Domain to Users

When you have clicked **Deploy to Users**, your domain will be deployed, and we are finally ready to preview our Hello World application.

That's quite a lot of steps in order to preview a simple sample application. However, this is a one-time setup process that all Lightning applications must go through. Instead of highlighting this way back in the beginning of the book or previous chapters, I thought it would make more sense to do it now.

If we go back to the **Developer Console** browser tab or window and click the **Preview** tab, we'll now see the result of what we have done.



```
This is our Hello world sample :)                    components/c/HelloWorld.js:10
```

Figure 5-n: Our HelloWorld App Running

The app has executed the **console.log** statement. Notice how it runs under the domain I registered, which is why we went through all those previous steps.

Summary

In this chapter, we've explored how to create a Hello World Lightning app manually by using the **Developer Console**. We've also registered the domain that will host any of the further Lightning developments we'll be doing.

We have yet to dive deeper to understand how Lightning components work, and how they interact with Apex controllers and custom objects. This is what we'll do next.

Chapter 6 Diving Deeper into Lightning

Introduction

In the previous two chapters, we've laid the groundwork for understanding how UI works in Force.com, specifically with the Lightning Experience.

In this chapter, we'll build upon that previous foundation and start to really explore all the facets of JavaScript development within Force.com by building a Lightning app that we can use with one of the custom objects we have created in previous chapters.

The full source code of the project can be downloaded here. So, let's step right in and start exploring.

A developer's perspective

Lightning in the Force.com world can mean many different things to different people. For users, it means a totally new user interface and working experience, but for a developer, it might mean something totally different. So, what is Lightning, really?

There are Lightning Apps, Lightning Data Services, Lightning Processes, Lightning Sync, Lightning Actions, Lightning Experiences, Lightning Connect, Lightning Components, and possibly even more services.

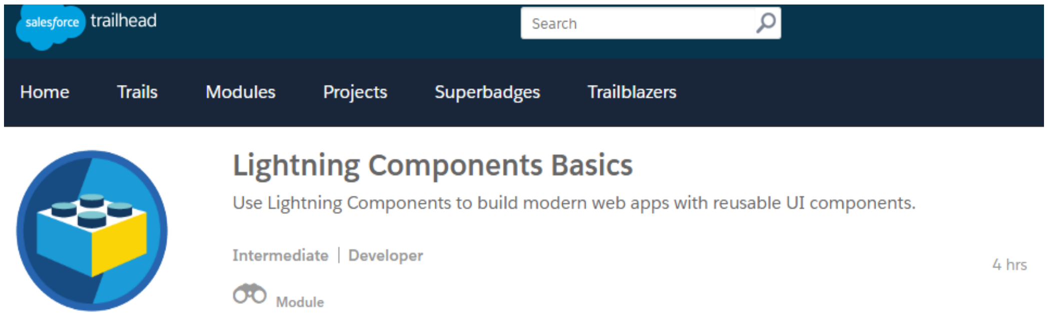
Everything in Force.com nowadays revolves around the "Lightning" buzzword, even if the technologies might not be exactly related, and despite confusion that could result. In any case, this buzzword is quite catchy and seems to be selling like hotcakes.

With that said, for developers and advanced business users, it is important to be able to separate what is a marketing term and what is an actual technology.

The parts that we are concerned with are actual **Lightning Components**, which are based on the Aura UI framework. These are the fundamental building blocks of Lightning Apps and constitute the overall Lightning look and feel—The Experience.

In essence, Lightning is a modern and responsive Singe Page Application (SPA) framework centered around Aura that runs in the multi-tenant environment of Force.com.

For a full understanding of what Lightning components are, I highly encourage you to follow the Lightning Components Basics Trailhead course, which can be found here.



Lightning Components Basics

Use Lightning Components to build modern web apps with reusable UI components.

Intermediate | Developer

4 hrs

Module

Figure 6-a: The Lightning Components Basics Trailhead Course

Lightning bundle

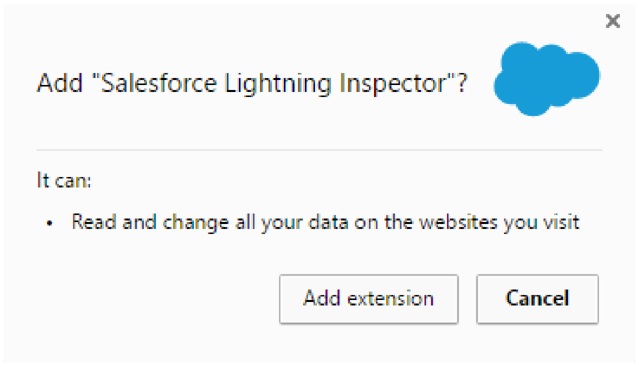
We are going to create a Lightning App manually with the **Developer Console** that we are going to call QuickRenewals.

This app will make use of the **Renewal** custom object that we previously created, but we'll create the UI from scratch, adding styles and JavaScript controllers, and we'll also use an Apex language server-side controller. This is what is known as a bundle.

The Lightning Inspector

The Lightning Inspector is a Chrome extension that is helpful when we are debugging. Go to the Chrome App Store and search for Salesforce Lightning Extension and install it by clicking **ADD TO CHROME**. It's not a must, but a nice-to-have.

When prompted, click **Add extension**. This will install the extension, which might come in handy later.



Add "Salesforce Lightning Inspector"?

It can:

- Read and change all your data on the websites you visit

Figure 6-b: Adding the Salesforce Lightning Inspector

QuickRenewals overview

We are now going to use Lightning Components in order to build a small app called QuickRenewals, which is going to be based on the **Renewal** custom object we previously created.

The overall goal of the application is to provide quick access to **Renewal** records, and it will use two Lightning Components that will communicate through events. These components will be: **`Search`** and **`RenewalsList`**.

Throughout the creation of this app, you will learn how to:

- Create an Apex Controller that exposes data to a Lightning app
- Create a Lighting app, which we briefly looked at in the previous chapter
- Create Lightning components
- Create and work with Lightning events
- Use some static resources within a Lightning app
- Exchange info between Lightning components using events

Sounds awesome, right? Let's start building this right away.

Static resources

I think most people would agree that anyone creating an app nowadays wants the app to look good. Not only does an app need to be able to do its job properly and be functional, but it also needs to be appealing to the eye, and this is why using static resources comes in handy.

One the most widely known and used front-end frameworks for building awesome-looking and responsive web apps is Bootstrap.

However, we'll need to use a version of Bootstrap that is customized to match the Lightning Design System guidelines. This customized version of Bootstrap can be found here.

Download and unzip this customized version of Bootstrap to a local folder on your machine. On your Force.com org, go to **Setup**, then on the **Quick Find** search box, type in the word **Static**.

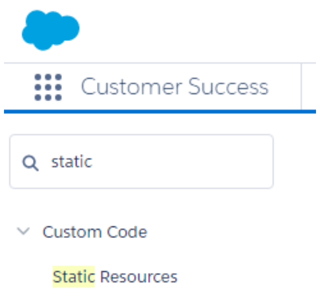
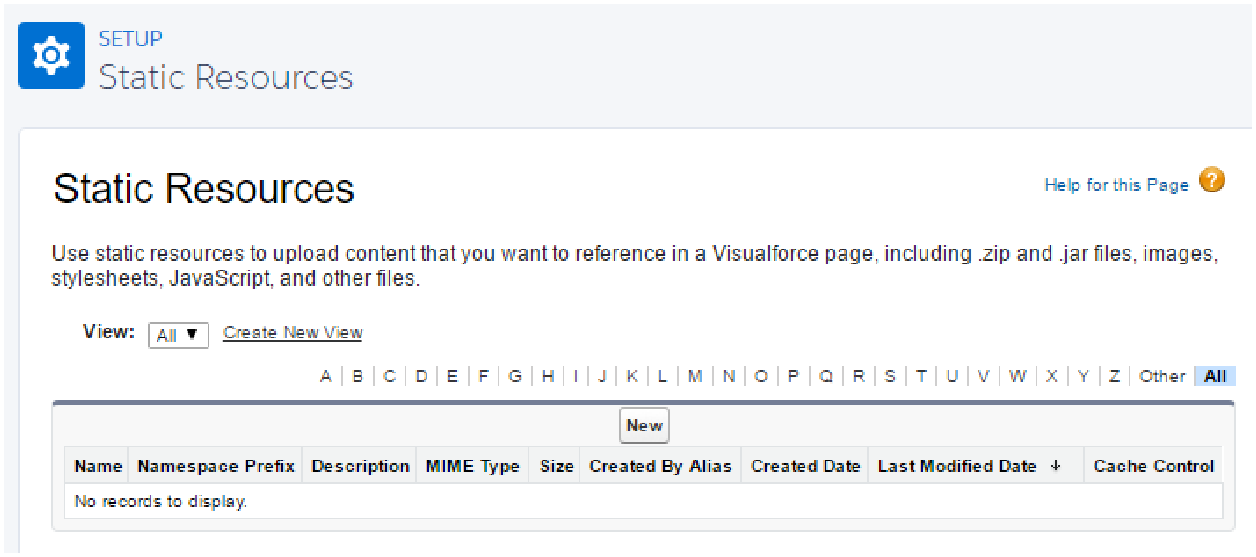


Figure 6-c: The Static Resources Option

Next, click on the **Static Resources** option. You'll then see the following screen.


SETUP

Static Resources

Static Resources

Use static resources to upload content that you want to reference in a Visualforce page, including .zip and .jar files, images, stylesheets, JavaScript, and other files.

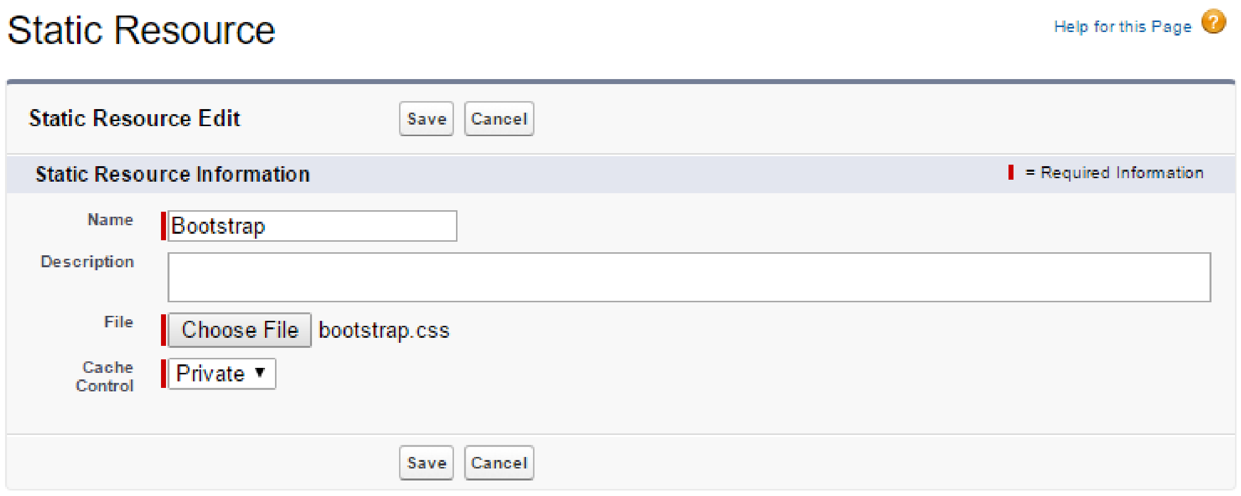
View: All

Name	Namespace Prefix	Description	MIME Type	Size	Created By Alias	Created Date	Last Modified Date	Cache Control
No records to display.								

Help for this Page
Create New View
A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | Other | All
New

Figure 6-d: The Static Resources Main Setup Page

Click **New**. You'll then see the following details screen.


Static Resource

Static Resource Edit

Static Resource Information

= Required Information

Name	Bootstrap
Description	
File	bootstrap.css
Cache Control	Private

Figure 6-e: The Static Resources Detail Page

Specify **Bootstrap** as the name of the **Static Resource** and click **Choose File**—select the **bootstrap.css** file found under the folder **dist/css** of the unzipped folder. Finally, click **Save**.

We can now use the customized Bootstrap framework in our Lightning app. The next thing we need to do is create our Apex controller.

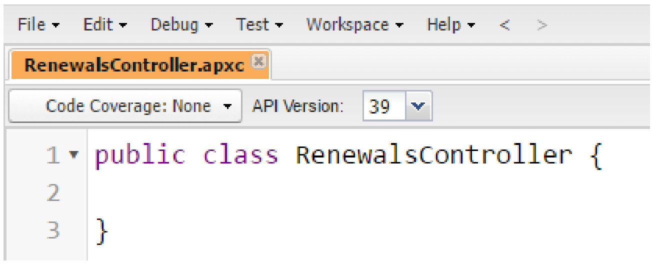
The Renewals controller

One of the things I love most about Force.com is how easy it is to work with data. They have done a great job in making it extremely easy and convenient to access data contained within objects.

We are now going to create an Apex Controller that will allow us to fetch **Renewal** records and search them by using the **Renewal Name** field. An Apex controller is basically an Apex class.

In order to create this class, from the **Gear** option at the top of your Lightning Experience screen, open the **Developer Console** in a new browser tab, so it is easier to work with.

Within the **Developer Console**, click on **File** > **New** > **Apex Class**. Name it RenewalsController when prompted and click **OK**. By default, you will see what is shown in Figure 6-f.



```
public class RenewalsController {

}
```

Figure 6-f: The Default Class Code

Let’s make some changes to this code in order to make it Lightning compatible.

Code Listing 6-a: The Modified Class Code

```
public with sharing class RenewalsController {
    @AuraEnabled
    public static List<Renewal__c> FindAll() {
        return [SELECT Name, Client__c, Amount__c, Currency__c,
                Expired__c, LastModifiedById from Renewal__c LIMIT 50];
    }
}
```

First, notice that the **`RenewalsController`** class has now been set as **`public with sharing`**, and we’ve also added a **`static`** method called **`FindAll`** that will return all **`Renewal`** records.

Querying the database (object records) with Apex is very simple, and **Salesforce Object Query Language** (SOQL) statements can be embedded into the Apex code seamlessly. As you can see, if you are familiar with SQL, you’ll feel right at home with SOQL. Notice how the **`Renewal`** object and its fields are accessed, using the **`__c`** suffix.

In the .NET world, it is not possible to embed directly fully functional SQL statements in C# code as native language instructions—it is necessary to use ADO.NET, LINQ, Entity Framework, or other libraries in order to interact with relational databases.

With Force.com, SOQL is executed directly within the Apex language context itself—which is very convenient, expressive, and powerful.

Another interesting thing to notice is that the **`FindAll`** method is marked with the **`@AuraEnabled`** annotation, which indicates that it can be used within a Lightning app. This code should seem very straightforward for any developer with a C# or Java background.

Now, let’s add another method to this class in order to find a **`Renewal`** record by its name. The class would now look like this:

Code Listing 6-b: The Expanded Class Code

```
public with sharing class RenewalsController {
    @AuraEnabled
    public static List<Renewal__c> FindAll() {
        return [SELECT Name, Client__c, Amount__c, Currency__c,
                Expired__c, LastModifiedById from Renewal__c LIMIT 50];
    }

    @AuraEnabled
    public static List<Renewal__c> FindByName(String sKey) {
        String n = '%' + sKey + '%';
        return [SELECT Name, Client__c, Amount__c, Currency__c,
                Expired__c, LastModifiedById FROM Renewal__c WHERE Name
                LIKE :n LIMIT 50];
    }
}
```

Notice that we’ve added a new static method, called **`FindByName`**, that filters out records based on the field **`Name`** that represents the **Renewal Name**. Notice the unusual **`:`** syntax in order reference the **`n`** variable inside the SOQL statement.

Also notice how both queries have been limited to return only the first 50 records, as this improves performance. Don’t forget to click on the **File** > **Save** option in order to save these code changes.

Creating the Lightning app & main layout

Now that we have created the Apex controller that will be responsible for returning the data using SOQL, we now need to actually create the Lightning app and define the basic layout using Bootstrap.

From the **Developer Console**, click **File** > **New** > **Lightning Application**. Use QuickRenewals as the bundle name. Let’s now implement the following markup code.

Code Listing 6-c: The Layout Markup Code

```
<aura:application >
    <link href='/resource/Bootstrap/' rel="stylesheet"/>

    <div class="navbar navbar-default navbar-static-top"
        role="navigation">
        <div class="container">
```

```
            <div class="navbar-header">
                <a href="#" class="navbar-brand">Quick Renewals</a>
            </div>
        </div>
    </div>

    <div class="container">
      <div class="row">
        <div class="col-sm-12">
            Renewals List Goes Here
        </div>
      </div>
    </div>
</aura:application>
```

Let's have a quick look at what we've just done here. The link tag references the **Bootstrap** style sheet we previously uploaded as a **Static Resource**.

Lightning App can include Lightning components and regular HTML markup. The code above contains regular HTML markup that refer to **Bootstrap** classes, contained within the Aura application tags.

In order to preview what we have accomplished, click on the **Preview** (or **Update Preview**) option on the right side of the **Developer Console**.

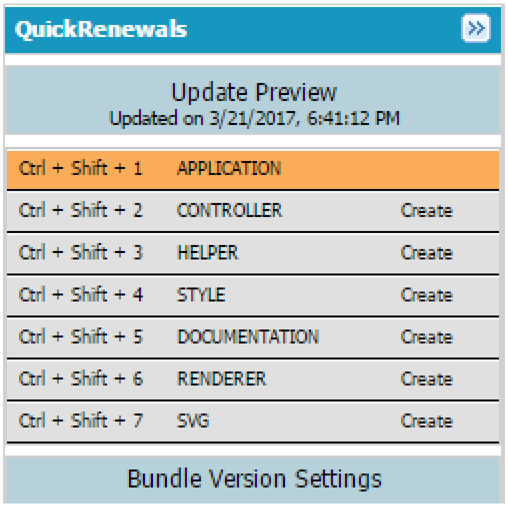
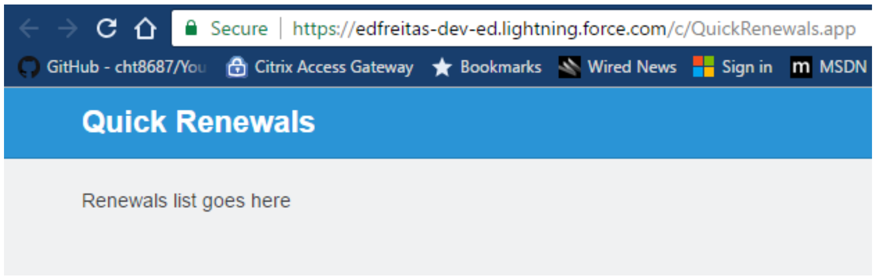

QuickRenewals
Update Preview
Updated on 3/21/2017, 6:41:12 PM
Ctrl + Shift + 1 APPLICATION
Ctrl + Shift + 2 CONTROLLER Create
Ctrl + Shift + 3 HELPER Create
Ctrl + Shift + 4 STYLE Create
Ctrl + Shift + 5 DOCUMENTATION Create
Ctrl + Shift + 6 RENDERER Create
Ctrl + Shift + 7 SVG Create
Bundle Version Settings

Figure 6-g: QuickRenewals Options

When you have clicked on that, you'll see the following in a new browser tab (see Figure 6-h). Notice that the URL ends with the **.app** extension.

The URL's domain is the same one we defined in the previous chapter when we created the Hello World example.



Quick Renewals

Renewals list goes here

Figure 6-h: QuickRenewals Preview

It seems like we are making progress and our app is taking shape. Let's carry on.

Creating the RenewalsList component

With the main layout created, let's now move our attention to creating a Lightning component that will be responsible for the list of **Renewals** that we will then add to our QuickRenewals app.

We'll do this by using the **Developer Console**, and we'll explore how to use component attributes and event handlers in order to add this component to our application. So, let's go ahead and create our component.

Click on **File** > **New** > **Lightning Component** and specify RenewalsList as the component name. Implement the code as follows. Leave the optional configuration checkboxes as unchecked.

Code Listing 6-d: The RenewalsList Markup Code

```
<aura:component controller="edfreitas.RenewalsController">
    <aura:attribute name="renewals" type="Renewal__c[]"/>
    <aura:handler name="init" value="{!this}" action="{!c.doInit}" />

    <ul class="list-group">
        <aura:iteration items="{!v.renewals}" var="renewal">
            <li class="list-group-item">
                <a href="{! '#renewal/' + renewal.Name }">
                    <p>{!renewal.Name }</p>
                </a>
            </li>
        </aura:iteration>
    </ul>
</aura:component>
```

For now, let's pay attention to the first line, specifically the **`controller`** attribute of the **`aura:component`** tag. **`RenewalsController`** must belong to a namespace, which in my case is: *edfreitas*.

A namespace in Force.com is a globally unique identifier across all Salesforce organizations. Namespaces are case insensitive and have a maximum length of 15 alphanumeric characters.

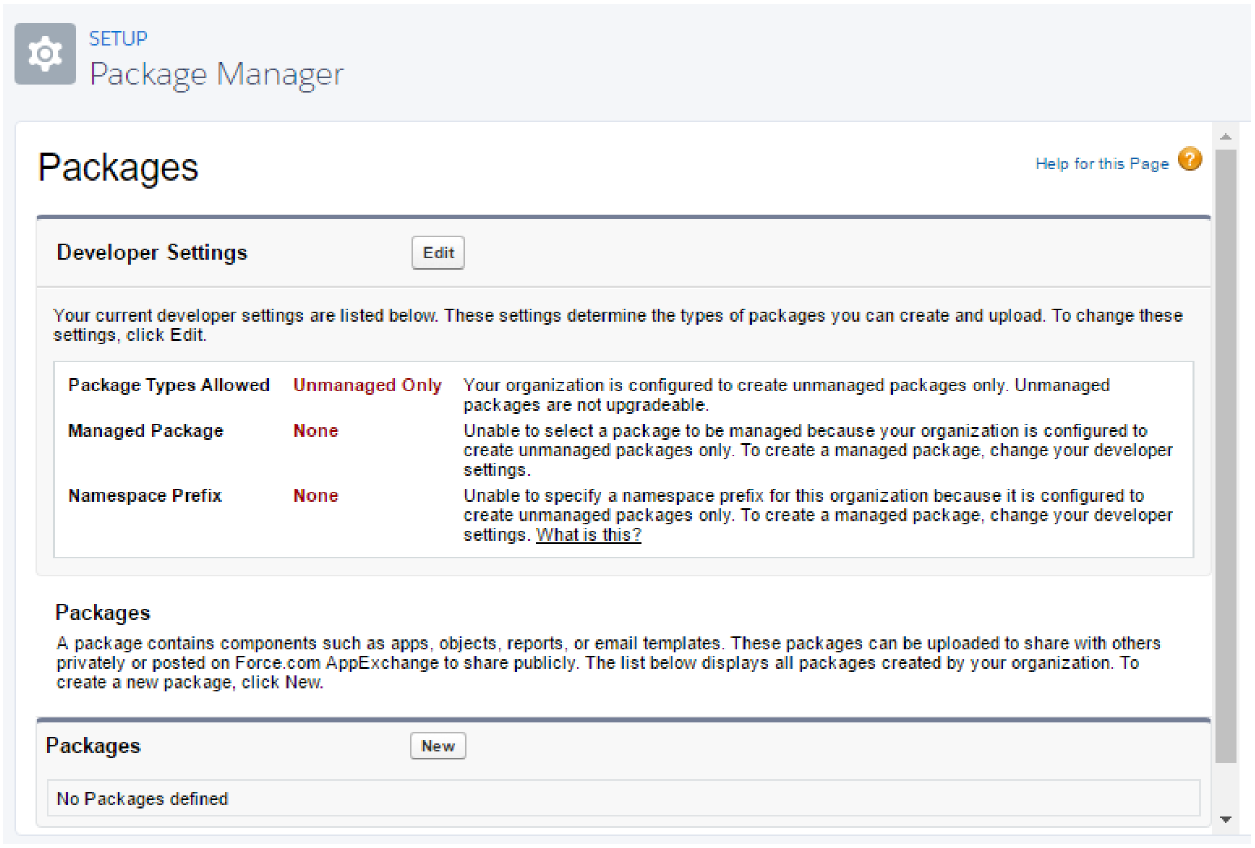
We'll need to create a namespace for our app—let's do that right away, and then come back to our component.

Creating a namespace

Leave your **Developer Console** browser tab open so we can come back to our component shortly.

If your Lightning Experience main screen is not opened on one of your other browser tabs, open it up. Then click on the **Gear** option, then on **Setup**.

On the **Quick Find** search box, type in the word **Package**. From the results, click on **Package Manager**. This will show you the following screen.



Packages

Developer Settings

Your current developer settings are listed below. These settings determine the types of packages you can create and upload. To change these settings, click Edit.

Package Types Allowed	Unmanaged Only	Your organization is configured to create unmanaged packages only. Unmanaged packages are not upgradeable.
Managed Package	None	Unable to select a package to be managed because your organization is configured to create unmanaged packages only. To create a managed package, change your developer settings.
Namespace Prefix	None	Unable to specify a namespace prefix for this organization because it is configured to create unmanaged packages only. To create a managed package, change your developer settings. What is this?

Packages

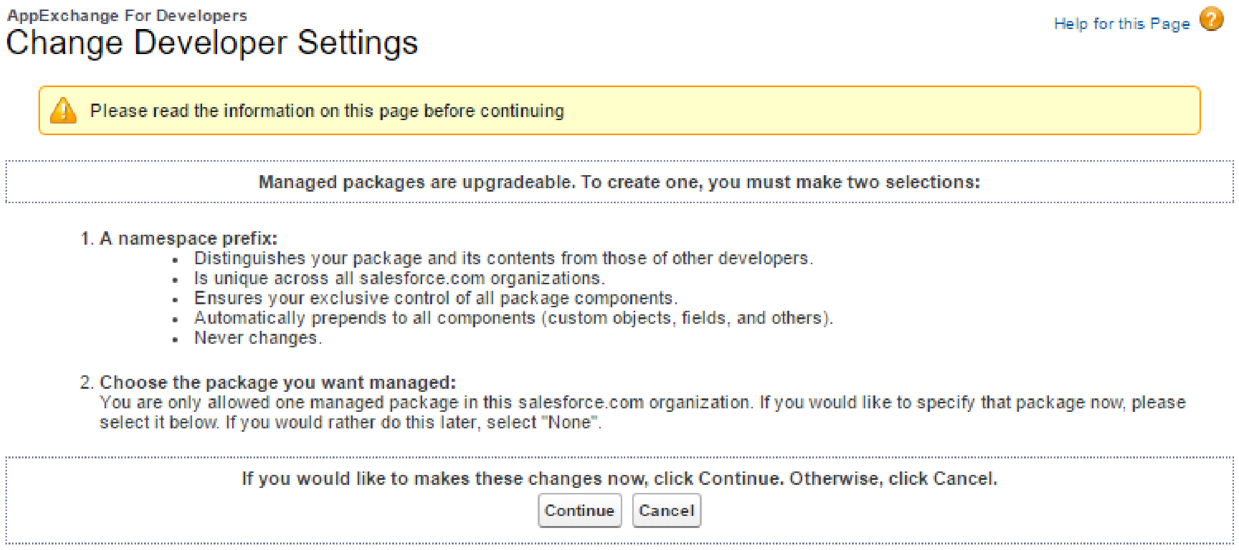
A package contains components such as apps, objects, reports, or email templates. These packages can be uploaded to share with others privately or posted on Force.com AppExchange to share publicly. The list below displays all packages created by your organization. To create a new package, click New.

Packages

No Packages defined

Figure 6-i: The Package Manager

Click **Edit**. You'll then be presented with the following information.


AppExchange For Developers

Change Developer Settings

Please read the information on this page before continuing

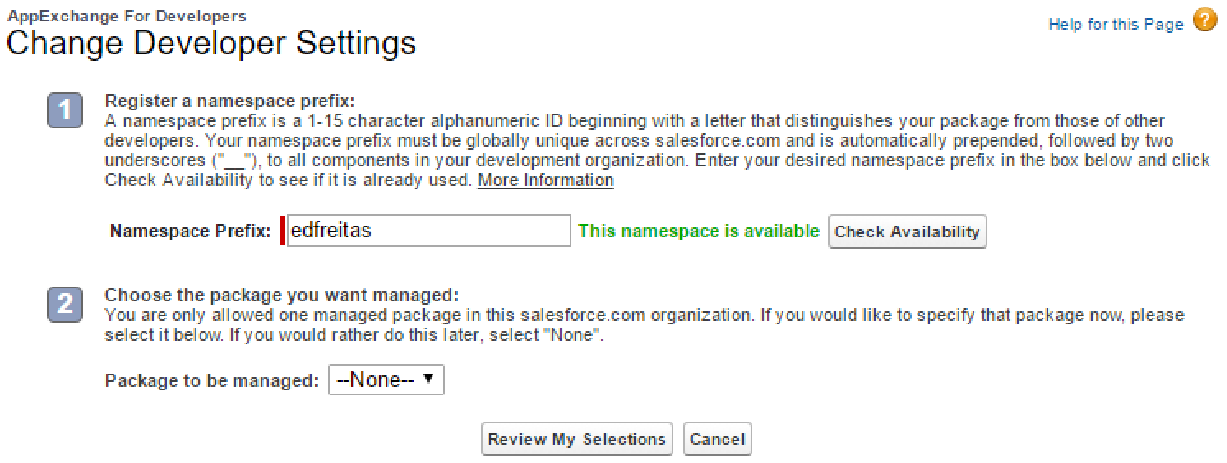
Managed packages are upgradeable. To create one, you must make two selections:

1. A namespace prefix:
   - Distinguishes your package and its contents from those of other developers.
   - Is unique across all salesforce.com organizations.
   - Ensures your exclusive control of all package components.
   - Automatically prepends to all components (custom objects, fields, and others).
   - Never changes.
2. Choose the package you want managed:
   You are only allowed one managed package in this salesforce.com organization. If you would like to specify that package now, please select it below. If you would rather do this later, select "None".

If you would like to makes these changes now, click Continue. Otherwise, click Cancel.

Figure 6-j: Change Developer Settings

Click **Continue**. Once you've done that, you'll be asked to enter a **Namespace Prefix,** and check whether it is available, by clicking on the **Check Availability** button.


AppExchange For Developers

Change Developer Settings

1 Register a namespace prefix:
A namespace prefix is a 1-15 character alphanumeric ID beginning with a letter that distinguishes your package from those of other developers. Your namespace prefix must be globally unique across salesforce.com and is automatically prepended, followed by two underscores ("__"), to all components in your development organization. Enter your desired namespace prefix in the box below and click Check Availability to see if it is already used. More Information

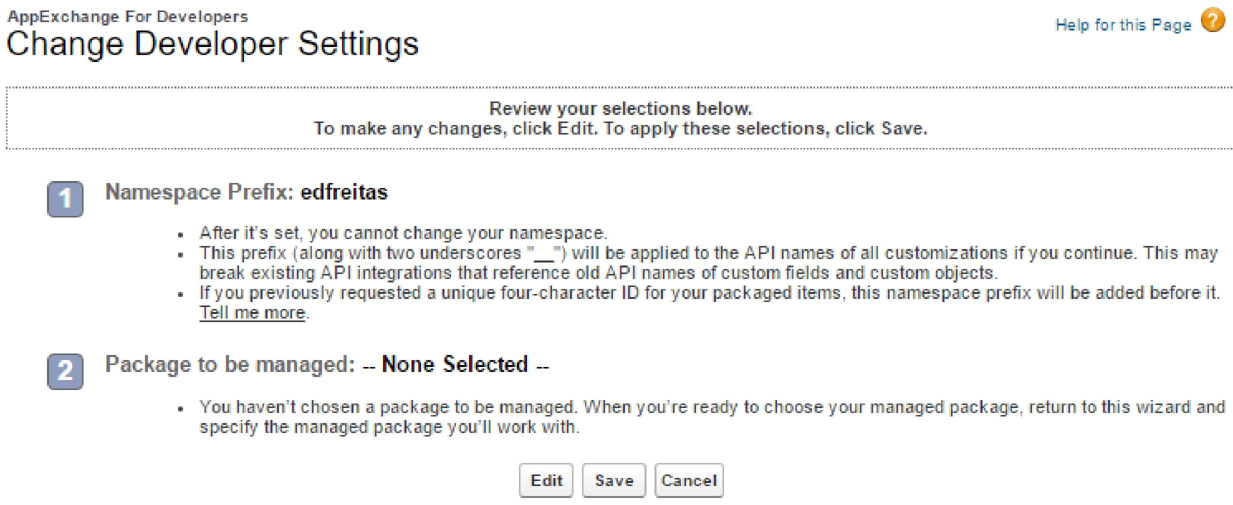
Namespace Prefix: edfreitas This namespace is available

2 Choose the package you want managed:
You are only allowed one managed package in this salesforce.com organization. If you would like to specify that package now, please select it below. If you would rather do this later, select "None".

Package to be managed: --None--

Figure 6-k: Change Developer Settings (Namespace Prefix)

When the namespace is available, simply click **Review My Selections**. Finally, you'll be presented with the following screen.


AppExchange For Developers

Change Developer Settings

Review your selections below.
To make any changes, click Edit. To apply these selections, click Save.

1 Namespace Prefix: edfreitas

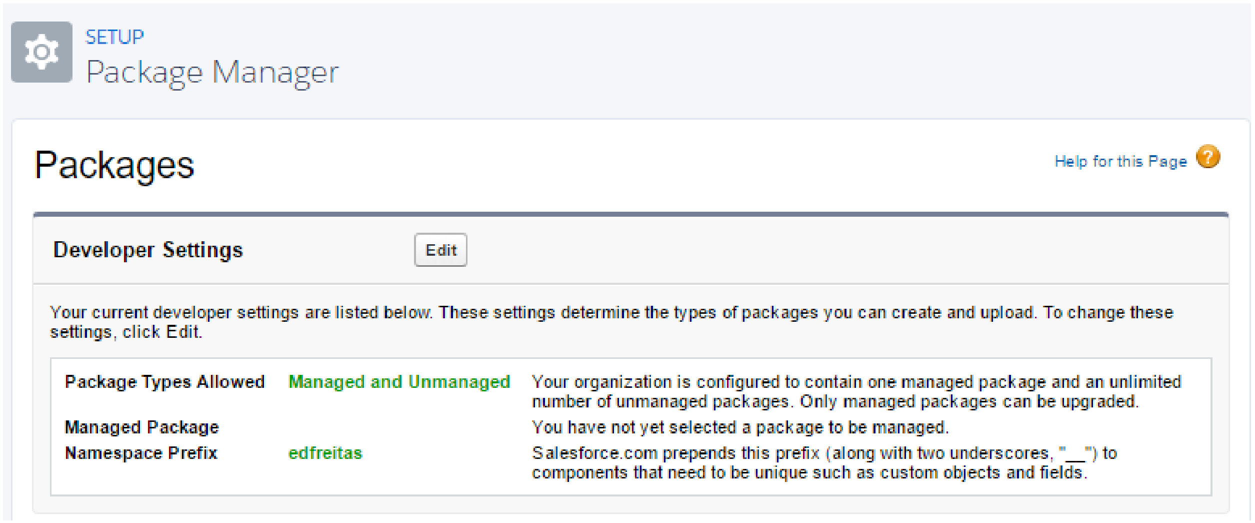
- After it's set, you cannot change your namespace.
- This prefix (along with two underscores "__") will be applied to the API names of all customizations if you continue. This may break existing API integrations that reference old API names of custom fields and custom objects.
- If you previously requested a unique four-character ID for your packaged items, this namespace prefix will be added before it. Tell me more.

2 Package to be managed: -- None Selected --

- You haven't chosen a package to be managed. When you're ready to choose your managed package, return to this wizard and specify the managed package you'll work with.

Figure 6-l: Change Developer Settings (Review Selections)

Click **Save** button in order to finalize the namespace creation. The Package Manager will now look like Figure 6-m.


SETUP

Package Manager

Packages

Developer Settings

Your current developer settings are listed below. These settings determine the types of packages you can create and upload. To change these settings, click Edit.

Package Types Allowed	Managed and Unmanaged	Your organization is configured to contain one managed package and an unlimited number of unmanaged packages. Only managed packages can be upgraded.
Managed Package		You have not yet selected a package to be managed.
Namespace Prefix	edfreitas	Salesforce.com prepends this prefix (along with two underscores, "__") to components that need to be unique such as custom objects and fields.

Figure 6-m: Updated Package Manager Settings

Notice that when you add a namespace**,** now all custom objects and fields will have the have the namespace name prepended to their actual names. In my case, mine will have the word *edfreitas* as a prefix.

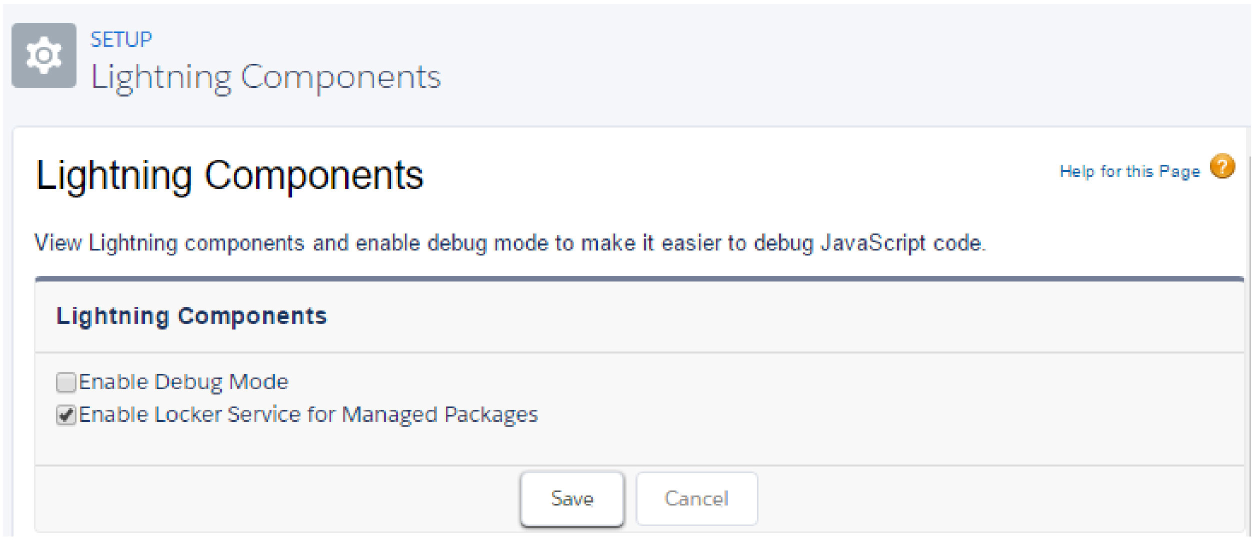
For example, this is how the **`Renewal`** object will now be internally called: **`edfreitas__Renewal__c`**.

The **`Amount__c`** custom field will now be **`edfreitas__Amount__c`**. This is only applicable to custom objects and fields.

Nevertheless, we can continue to use the initial names in our Apex code—without the need to use the namespace name when invoking the objects or fields.

Enabling Lightning components for Debug mode

On the Lightning Experience screen, on the **Quick Find** search box, type in the words **Lightning Components** and click on the resulting option. This will bring you to the following screen.


SETUP
Lightning Components

Lightning Components

Help for this Page

View Lightning components and enable debug mode to make it easier to debug JavaScript code.

Lightning Components

☐ Enable Debug Mode
☑ Enable Locker Service for Managed Packages

Figure 6-n: Lightning Components Setup Options

Select the **Enable Debug Mode** checkbox and then click **Save**. This option could come in handy when debugging JavaScript code on the Developer Console.

Continuing with the RenewalsList component

Now that we've created the namespace and have selected the **Enable Debug Mode** option, we can now carry on with the development of the **`RenewalsList`** component.

But before we add any extra functionality, let's examine the code from Code Listing 6-d by breaking it into small chunks and understanding each part.

Code Listing 6-e: The RenewalsList Markup Code (Part 1)

```
<aura:component controller="edfreitas.RenewalsController">
```

The controller assigned to the component refers to the server-side Apex controller we previously wrote. Here we must specify the namespace.

Code Listing 6-f: The RenewalsList Markup Code (Part 2)

```
<aura:attribute name="renewals" type="Renewal__c[]"/>
```

The **`renewals`** attribute is defined to hold a list of renewal objects **`(Renewal__c)`** returned from the server.

Code Listing 6-g: The RenewalsList Markup Code (Part 3)

```
<aura:handler name="init" value="{!this}" action="{!c.doInit}" />
```

The **`init`** handler is defined to execute some code when the component is initialized. That code (**`doInit`**) is defined in the component's client-side controller, which you'll implement in the next step.

Code Listing 6-h: The RenewalsList Markup Code (Part 4)

```
<aura:iteration items="{!v.renewals}" var="renewal">
```

The **`aura:iteration`** tag is used to iterate through the list of **`renewal`** objects returned from the server and create a **`<li>`** for each **`renewal`** record. Notice the **`<aura:iteration>`** and **`</aura:iteration>`** tags.

Code Listing 6-i: The RenewalsList Markup Code (Part 5)

```
<a href="{! '#renewal/' + renewal.Name }">
```

The **`<a href="{! '#renewal/' + renewal.Name }">`** anchor tag around the renewal data is defined to set the page hashtag to **`#renewal/`** followed by the name of the renewal record.

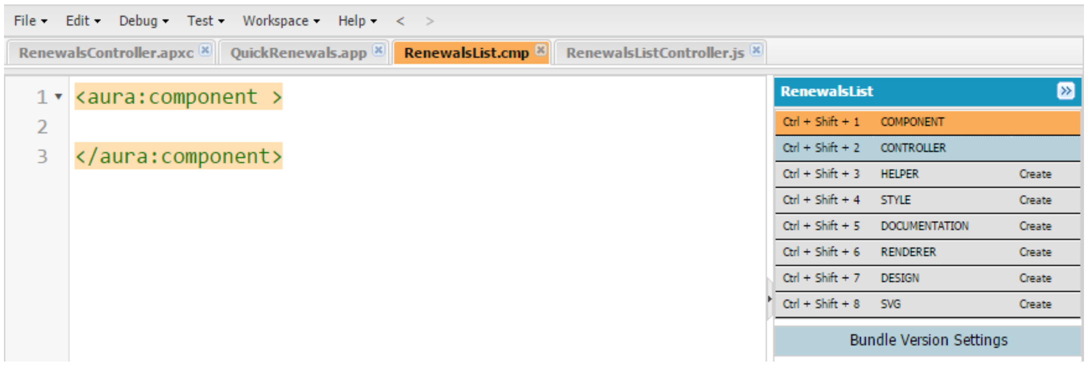
Later, you may want to create a **`RenewalDetails`** component that will use that hashtag to display detailed information about that particular renewal record.

We won't create this **`RenewalDetails`** component, but that might be something fun for you do to later on.

Just in case you haven't saved the code yet, click on **File** > **Save** in order to save your latest changes.

The client-side controller

The next thing we need to do is to implement the client-side controller. On the right side of the **Developer Console**, click on the **CONTROLLER** option.



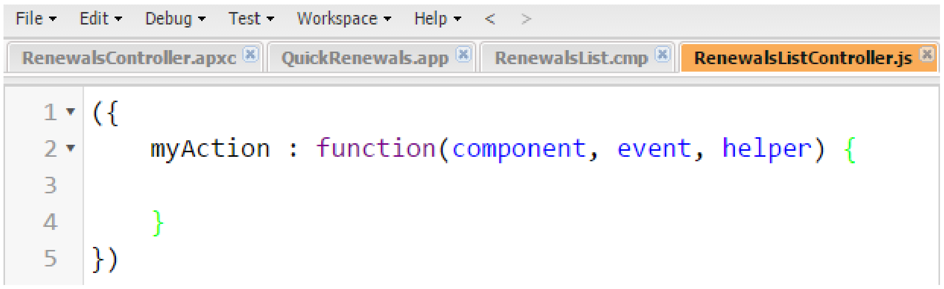
```
<aura:component >

</aura:component>
```

Ctrl + Shift + 1	COMPONENT	
Ctrl + Shift + 2	CONTROLLER	
Ctrl + Shift + 3	HELPER	Create
Ctrl + Shift + 4	STYLE	Create
Ctrl + Shift + 5	DOCUMENTATION	Create
Ctrl + Shift + 6	RENDERER	Create
Ctrl + Shift + 7	DESIGN	Create
Ctrl + Shift + 8	SVG	Create

Figure 6-o: The RenewalsList Component Bundle Options

When you do that, the RenewalsListController.js file will be automatically created with the following default code.


File Edit Debug Test Workspace Help
RenewalsController.apxc
QuickRenewals.app
RenewalsList.cmp
RenewalsListController.js

```
({
    myAction : function(component, event, helper) {

    }
})
```

Figure 6-p: The RenewalsListController.js File

Let’s implement our client-side controller code as follows.

Code Listing 6-j: The RenewalsList Markup Code (Part 5)

```
({
    doInit : function(component, event) {
        var action = component.get("c.FindAll");

        action.setCallback(this, function(a) {
            component.set("v.renewals", a.getReturnValue());
        });

        $A.enqueueAction(action);
    }
})
```

This controller has a single function called **`doInit`**. This is the function the component calls when it is initialized.

The first thing the code does is to get a reference to the **`FindAll`** method in the component’s server-side controller (Apex code we previously wrote) and store this in the **`action`** variable.

Because the call to the server's **`FindAll`** method is asynchronous, we then register a callback function that is executed when the call returns. In the callback function, we simply assign the list of renewal records to the component's **`renewals`** attribute.

The **`$A.enqueueAction(action)`** sends the request to the server. More precisely, it adds the call to the queue of asynchronous server calls. This is an optimization feature of Lightning.

Don't forget to click on **File** > **Save** in order to save any code changes.

Adding RenewalsList to the application

In the **Developer Console**, let's go back to the QuickRenewals application.

If the tab is not visible any more in the **Developer Console**, go to **File** > **Open Lightning Resources,** and then select **QuickRenewals** > **APPLICATION**. Finally, click **Open Selected**.

Now that the QuickRenewals.app file is opened, let's modify the container **`div`** and replace the old code with the following one. Notice how now we have to reference the namespace in order to call the **`RenewalsList`** component.

Code Listing 6-k: The Container Div Markup Code

```
<div class="container">
    <div class="row">
        <div class="col-sm-12">
            <edfreitas:RenewalsList />
        </div>
    </div>
</div>
```

Now, click on **Preview** or **Update Preview** (visible on the right side of the screen). If you have added a least one **`Renewal`** record previously, you should see something like this.

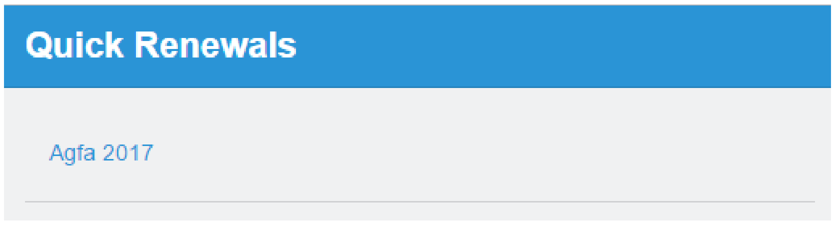
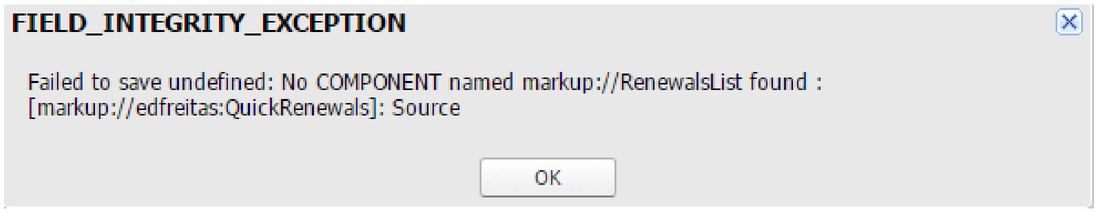


Figure 6-q: Preview of the QuickRenewals App

You might be asking yourself if we really do need the namespace when calling our RenewalsList component—the answer is yes. If the tag was **<RenewalsList />**, we would get the following error when previewing.


FIELD_INTEGRITY_EXCEPTION

Failed to save undefined: No COMPONENT named markup://RenewalsList found : [markup://edfreitas:QuickRenewals]: Source

Figure 6-r: Field Integrity Exception

So yes, we do need to use the namespace. Now that we've added the **`RenewalsList`** component to our QuickRenewals app, let's create the **`Search`** component.

The search component

It's time to give our app some basic search capabilities. Let's now create a component that can allow us to search renewals by their name.

We might have added the component to the **`RenewalsList`** component, but that would have limited the reusability of the component—so it's best to have them as individual components, as it makes the code more modular and manageable.

So, if both components are going to be independent, how can we make sure they communicate with each other? Good question—this can be achieved by using events.

The Key Change event

Lightning events enable communication between components. We'll need to implement an event that will be responsible for raising notifications when the search key changes. So, let's go ahead and do that.

In the **Developer Console**, click **File** > **New** > **Lightning Event** and type in KeyChange as the bundle name. Let's implement the code as follows.

Code Listing 6-l: The KeyChange Event

```
<aura:event type="APPLICATION">
    <aura:attribute name="sKey" type="String"/>
</aura:event>
```

The event has one argument, called **`sKey`**. Don't forget to click **File** > **Save** in order to save any changes.

Creating the Search component

In the **Developer Console**, click **File** > **New** > **Lightning Component** and indicate Search as the bundle name. Let's implement the component code as follows.

Code Listing 6-m: The Search Component

```
<aura:component>
    <input type="text" class="form-control"
           onkeyup="{!c.KeyChange}" placeholder="Search"/>
</aura:component>
```

This is a component with a single input field. When the user types in a character, the **`onkeyup`** event triggers the **`KeyChange`** function, which is executed in the component's client-side controller.

Doing it this way, the search is refined every time the user types in a character. Click **File** > **Save** in order to save any changes.

Implementing the Search controller

On the right side of the screen, click on **CONTROLLER** and implement the search controller code as follows. Click **File** > **Save** in order to save any changes.

Code Listing 6-n: The Controller Client-Side Code

```
({
    KeyChange: function(component, event, helper) {
        var evt = $A.get("e.edfreitas:KeyChange");
        evt.setParams({"sKey": event.target.value});
        evt.fire();
    }
})
```

Listening for the KeyChange event

Now let's go back to the **`RenewalsList`** component and add an event handler for the **`KeyChange`** event. This goes right after the **`init`** handler.

Code Listing 6-o: The Event Handler for the KeyChange Event

```
<aura:handler event="edfreitas:KeyChange" action="{!c.KeyChange}"/>
```

The **RenewalsList** component code should now look like this.

Code Listing 6-p: The Updated RenewalsList Component Code

```
<aura:component controller="edfreitas.RenewalsController">
    <aura:attribute name="renewals" type="Renewal__c[]"/>
    <aura:handler name="init" value="{!this}" action="{!c.doInit}" />
    <aura:handler event="edfreitas:KeyChange" action="{!c.KeyChange}"/>

    <ul class="list-group">
        <aura:iteration items="{!v.renewals}" var="renewal">
            <li class="list-group-item">
                <a href="{! '#renewal/' + renewal.Name }">
                    <p>{!renewal.Name }</p>
                </a>
            </li>
        </aura:iteration>
    </ul>
</aura:component>
```

Now click on **CONTROLLER** on the right side of the screen in order to add the following code to the RenewalsListController.js file.

Code Listing 6-q: KeyChange Function—RenewalsListController.js File

```
KeyChange : function(component, event) {
    var sKey = event.getParam("sKey");
    var action = component.get("c.FindByName");

    action.setParams({ "sKey": sKey });
    action.setCallback(this, function(a) {
        component.set("v.renewals", a.getReturnValue());
    });

    $A.enqueueAction(action);
}
```

Code Listing 6-r shows how the updated RenewalsListController.js file should look.

Code Listing 6-r: The Updated RenewalsListController.js File

```
({
    doInit : function(component, event) {
        var action = component.get("c.FindAll");

        action.setCallback(this, function(a) {
            component.set("v.renewals", a.getReturnValue());
        });

        $A.enqueueAction(action);
    },

    KeyChange : function(component, event) {
        var sKey = event.getParam("sKey");
        var action = component.get("c.FindByName");

        action.setParams({ "sKey": sKey });
        action.setCallback(this, function(a) {
            component.set("v.renewals", a.getReturnValue());
        });

        $A.enqueueAction(action);
    }
})
```

As always, don't forget to click **File** > **Save** in order to save any changes. The final step is to add the **`Search`** component to the Lightning application.

Adding the Search component to our app

We've reached the last part of this exercise, and we are ready to tie the components together.

In the **Developer Console**, go back to the QuickRenewals application and modify the **`div`** container as described in the following code listing in order to add the **`Search`** component that we've just created.

Code Listing 6-s: The Updated QuickRenewals Container Markup Code

```
<div class="container">
    <div class="row">
        <div class="col-sm-12">
            <edfreitas:Search/>
            <edfreitas:RenewalsList/>
        </div>
    </div>
</div>
```

The fully updated markup code looks Code Listing 6-t.

Code Listing 6-t: The Updated QuickRenewals Markup Code

```
<aura:application >
    <link href='/resource/Bootstrap/' rel="stylesheet"/>

    <div class="navbar navbar-default navbar-static-top"
        role="navigation">
        <div class="container">
            <div class="navbar-header">
                <a href="#" class="navbar-brand">Quick Renewals</a>
            </div>
        </div>
    </div>

    <div class="container">
      <div class="row">
        <div class="col-sm-12">
            <edfreitas:Search/>
            <edfreitas:RenewalsList />
        </div>
      </div>
    </div>
</aura:application>
```

Once again, don’t forget to save all changes. On the left-hand side of the screen, click **Preview** or **Update Preview** to view the changes and see the app running.

The applications will look like Figure 6-s.

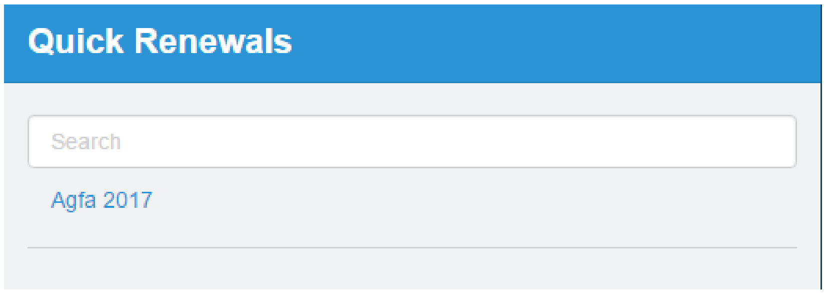


Figure 6-s: Preview of the QuickRenewals App

Cool! Now, let’s quickly test what we have just built to see if it actually works as we would expect.

If I type *2018*, nothing is shown—as expected. So that worked.

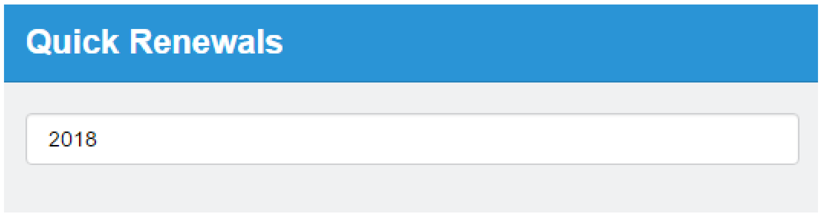


Figure 6-t: Testing the App (No Results Shown)

If I now type in the letter *A*, note that as expected—we get a result.

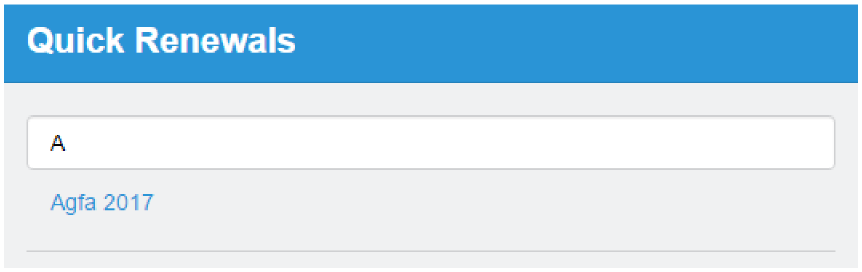


Figure 6-u: Testing the App (Results Shown)

So, our app is working!

Summary

Throughout this chapter, we've created a Lightning app step by step, and we explored how to write markup code, style the app, create components, use events, and create server-side code using Apex and Salesforce Object Query Language (SOQL) and client-side controllers in JavaScript.

It was a simple example, but we covered a lot of ground, and it was quite fun to do. We also looked at how namespaces in Force.com are the glue that tie all these app parts together.

Working with Force.com has allowed me to look at software development with a different mindset than what I am normally used to—especially having come into the Force.com world as a .NET developer.

However, this e-book has barely scratched the surface of what is still possible to do with this platform. There's an incredibly broad range of subjects that we weren't able to cover throughout this e-book, such as Apex Triggers, Visualforce, Workflow Automation, Schema Builder, and further exploring Lightning App development.

Force.com is an incredibly rich platform, and there are multiple learning tracks and certification paths available. There's also an incredibly rich ecosystem of third-party libraries and frameworks that expand even further what is possible to accomplish—for example, JSForce, which is an open-source library that allows anyone to create Force.com apps with JavaScript outside of the Force.com ecosystem—the code is hosted somewhere else.

If you have found this e-book inspiring and would like to keep exploring Force.com and learn even more, I highly encourage you to sign up for the Salesforce Trailhead training modules, which contain a ton of useful and easy-to-follow resources.

Thanks for following along, and if you feel up for it, you are welcome to keep improving the examples and concepts that we've covered here—and if you do so, I'd love to hear from you. All the best.