

CUSTOM BLAZOR OQTANE MODULES

SUCCINCTLY

BY **MICHAEL WASHINGTON**

Custom Blazor Oqtane Modules Succinctly

By

Michael Washington

Foreword by Daniel Jebaraj



Copyright © 2021 by Syncfusion, Inc.

2501 Aerial Center Parkway

Suite 200

Morrisville, NC 27560

USA

All rights reserved.

ISBN: 978-1-64200-218-8

Important licensing information. Please read.

This book is available for free download from www.syncfusion.com on completion of a registration form.

If you obtained this book from any other source, please register and download a free copy from www.syncfusion.com.

This book is licensed for reading only if obtained from www.syncfusion.com.

This book is licensed strictly for personal or educational use.

Redistribution in any form is prohibited.

The authors and copyright holders provide absolutely no warranty for any information provided.

The authors and copyright holders shall not be liable for any claim, damages, or any other liability arising from, out of, or in connection with the information in this book.

Please do not use this book if the listed terms are unacceptable.

Use shall constitute acceptance of the terms listed.

SYNCFUSION, SUCCINCTLY, DELIVER INNOVATION WITH EASE, ESSENTIAL, and .NET ESSENTIALS are the registered trademarks of Syncfusion, Inc.

Technical Reviewer: James McCaffrey

Copy Editor: Courtney Wright

Acquisitions Coordinator: Tres Watkins, VP of content, Syncfusion, Inc.

Proofreader: Jacqueline Bieringer, content producer, Syncfusion, Inc.

Table of Contents

About the Author	9
Introduction.....	10
Chapter 1 What Is Blazor?	12
Server-side Blazor	12
Client-side (WebAssembly).....	13
Chapter 2 What Is Oqtane, and Why Should You Use It?.....	14
Oqtane features.....	14
Custom modules	14
Chapter 3 The Help Desk Module.....	15
Help Desk users	15
Help Desk administrators	17
Chapter 4 Create the Help Desk Module.....	19
Install .NET Core (aka .NET 5) and Visual Studio.....	19
Install Oqtane	20
Oqtane module creator	23
Build the project.....	26
Complete the module installation.....	27
Explore the module	30
Dynamic routing	32
Chapter 5 Add Syncfusion	34
Install NuGet packages	34
Additional configuration	35
Add using statements to Imports.razor	36
Implement Syncfusion in IServerStartup.....	37

Support Oqtane WebAssembly mode.....	39
Including the JavaScript and .css files	40
Using IHostResources to include references in _Host.cshtml	42
Consume the Syncfusion controls	44
Chapter 6 Creating the Data Layer.....	49
Upgrade the module	49
Create the database tables	50
Shared classes.....	54
SyncfusionHelpDeskStatus.cs	54
SyncfusionHelpDeskTicketDetails.cs	55
SyncfusionHelpDeskTickets.cs	55
Repository code	57
HelpdeskContext.cs	57
IHelpdeskRepository.cs	58
HelpdeskRepository.cs	59
Controller code	63
Import and export support	67
Chapter 7 Creating Help Desk Tickets.....	71
Services layer.....	71
Forms and validation	74
Forms	74
Validation.....	74
Syncfusion Blazor controls	75
EditTicket control.....	75
Index control.....	78
New Help Desk Ticket tab.....	80

Existing Tickets tab.....	83
Remove all the code from Edit.Razor (for now).....	87
Test creating new tickets	87
Chapter 8 Help Desk Ticket Administration	90
Server project.....	90
Client project	95
Services.....	95
Edit.razor control.....	98
Data grid.....	100
Edit ticket.....	101
Delete ticket.....	103
Add the Administration button	105
Test the final module	106

The Story Behind the *Succinctly* Series of Books

Daniel Jebaraj, CEO
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.

Free? What is the catch?

There is no catch here. Syncfusion has a vested interest in this effort.

As a component vendor, our unique claim has always been that we offer deeper and broader frameworks than anyone else on the market. Developer education greatly helps us market and sell against competing vendors who promise to “enable AJAX support with one click,” or “turn the moon to cheese!”

Let us know what you think

If you have any topics of interest, thoughts, or feedback, please feel free to send them to us at succinctly-series@syncfusion.com.

We sincerely hope you enjoy reading this book and that it helps you better understand the topic of study. Thank you for reading.

Please follow us on Twitter and “Like” us on Facebook to help us spread the word about the *Succinctly* series!



About the Author

Michael Washington is a Microsoft MVP, an ASP.NET C# programmer, and the founder of BlazorHelpWebsite.com. He is the author of *An Introduction to Building Applications with Blazor*. He has extensive knowledge in process improvement, billing systems, and student information systems. He has a son, Zachary, and resides in Los Angeles with his wife, Valerie.

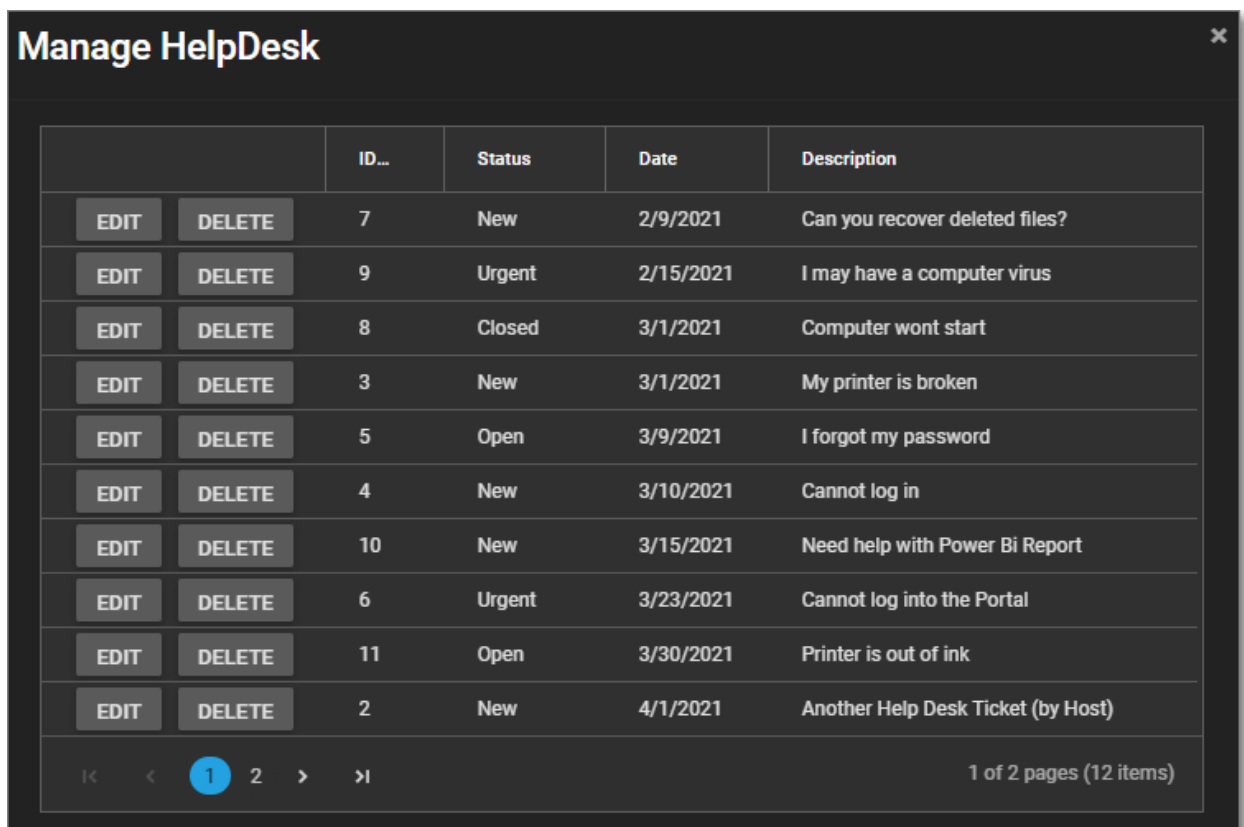
You can follow him on Twitter at [@ADefWebserver](https://twitter.com/ADefWebserver).

Introduction

Blazor is an exciting technology that allows you to create web-based applications using C# instead of JavaScript. However, you still can implement custom JavaScript when you desire.

Blazor is an alternative to other single-page application frameworks such as Angular, React, and Vue.js.

Oqtane is an application framework built using Microsoft's Blazor technology. It allows you to create and deploy modules written in Blazor.



	ID...	Status	Date	Description
EDIT DELETE	7	New	2/9/2021	Can you recover deleted files?
EDIT DELETE	9	Urgent	2/15/2021	I may have a computer virus
EDIT DELETE	8	Closed	3/1/2021	Computer wont start
EDIT DELETE	3	New	3/1/2021	My printer is broken
EDIT DELETE	5	Open	3/9/2021	I forgot my password
EDIT DELETE	4	New	3/10/2021	Cannot log in
EDIT DELETE	10	New	3/15/2021	Need help with Power Bi Report
EDIT DELETE	6	Urgent	3/23/2021	Cannot log into the Portal
EDIT DELETE	11	Open	3/30/2021	Printer is out of ink
EDIT DELETE	2	New	4/1/2021	Another Help Desk Ticket (by Host)

1 of 2 pages (12 items)

Figure 1: Help Desk Administration

In this book, we will build a sample Oqtane module called Syncfusion Help Desk.

This application will demonstrate the following:

- Creating a custom module in Oqtane.
- Adding Syncfusion controls to the custom module.
- Inserting, updating, and deleting data from the database.
- Using forms and validation.

The code for this ebook is available on [GitHub](#). The step-by-step instructions use Visual Studio 2019 Community edition, which is available for free at [this link](#).

You may want to bookmark the official Microsoft Blazor documentation, available at <https://blazor.net/>.

Chapter 1 What Is Blazor?

Blazor applications are composed of components that are constructed using C#, HTML-based Razor syntax, and CSS.

Blazor has two different runtime modes: server-side Blazor and client-side Blazor. Client-side Blazor is also known as Blazor WebAssembly. Both modes run in all modern web browsers, including web browsers on mobile phones.

Server-side Blazor

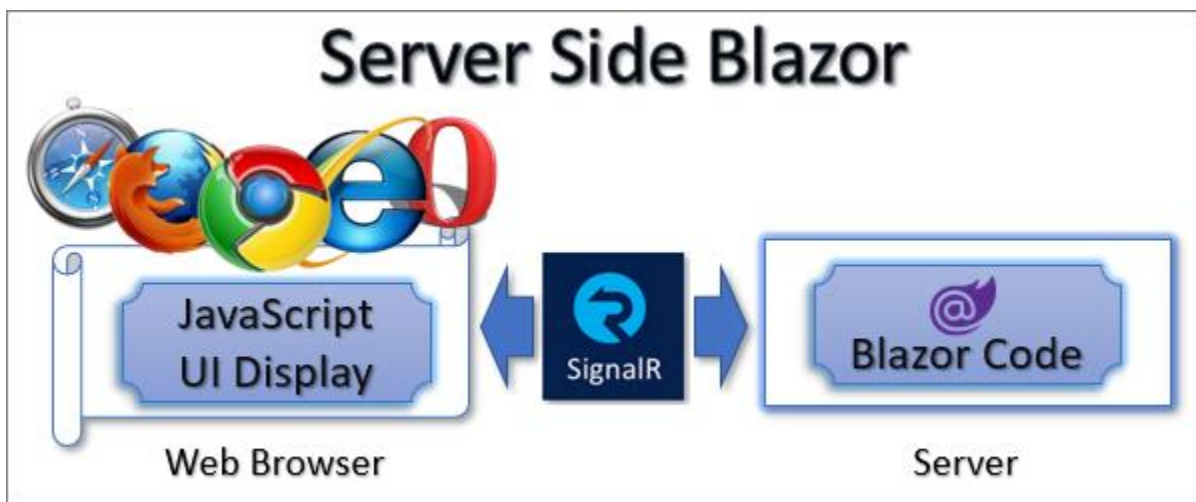


Figure 2: Server-Side Blazor
© BlazorHelpWebsite.com—used with permission

Server-side Blazor renders the Razor components on the server and updates the webpage using a SignalR asynchronous connection. The Blazor framework sends events from the web browser, such as button clicks and mouse movements, to the server. The Blazor runtime computes changes to the components on the server and sends a *diff-based* webpage back to the browser.

Client-side (WebAssembly)



Figure 3: Client-Side Blazor (WebAssembly)
© BlazorHelpWebsite.com - used by permission

Client-side Blazor is composed of the same code as server-side Blazor; however, it runs entirely in the web browser using a technology known as WebAssembly.

The primary difference in Blazor applications created in server-side Blazor versus client-side Blazor is that the client-side Blazor applications need to make web calls to access server data, whereas the server-side Blazor applications can omit this step, as all their code is executed on the server.

One way to think of Blazor is that it is a framework for creating SPA webpages using one of two architectures (client-side, server-side), using Razor technology written with the C# language.

Because client-side Blazor with WebAssembly executes entirely on a user's browser, for many applications, client-side Blazor is very fast.

When Oqtane is deployed, it can be run as client-side Blazor or as server-side Blazor.

The Syncfusion Help Desk sample application covered in this book can be run as server-side or client-side Blazor.

Chapter 2 What Is Oqtane, and Why Should You Use It?

Oqtane, built using Microsoft's Blazor technology, is a web application framework that automates the overhead in creating web applications. It allows you to deploy and run modules written in Blazor. It provides a dynamic web experience that can be run as client-side Blazor or as server-side Blazor.

It is a platform for hosting multiple Blazor applications, not just developing a single application. When you have a website that requires multiple types of functionality, such as forums, blogs, and content management, Oqtane can provide this.

Oqtane is open source (using the MIT license).

Oqtane features

Oqtane has several features, including:

- **Multitenant:** The ability to run multiple sites, each having its own database, on a single Oqtane installation.
- **Custom themes:** Customizable and easy-to-install styling for the Oqtane installation.
- **Modular framework:** Provides a clean separation between your custom module functionality and the underlying framework. This allows you to upgrade the underlying framework and other custom modules without breaking the other parts.
- **Granular permissions:** An Oqtane installation is composed of pages that contain modules contained in module instances. Granular permissions allow for view and edit permissions to be applied at the page level and the module-instance level.

Custom modules

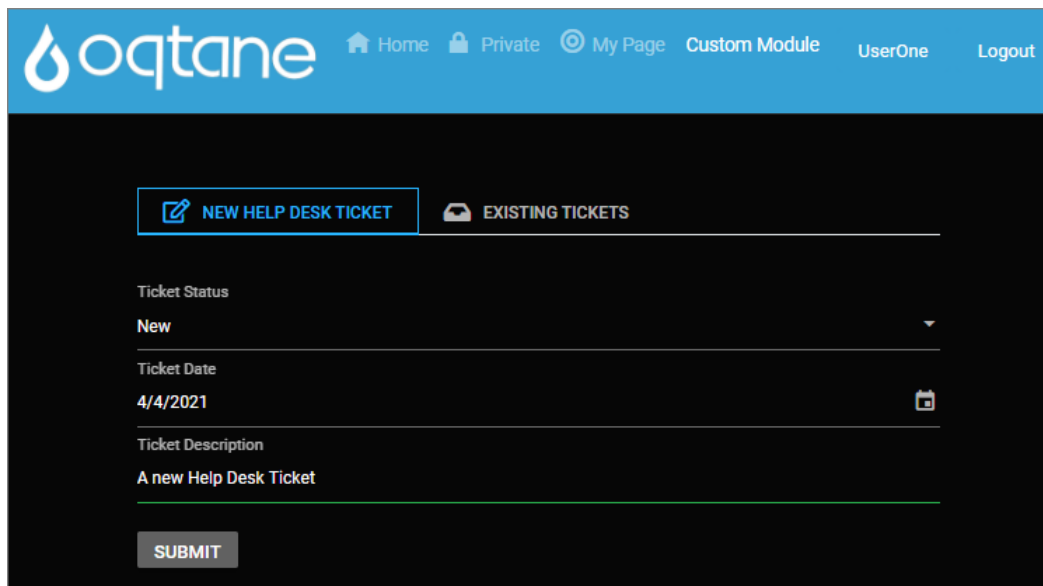
The Oqtane modular framework allows you to create or download and install custom modules. With custom modules, you only write "functionality-specific" code, not the entire website.

In this book, we will create a Help Desk custom module.

Chapter 3 The Help Desk Module

To demonstrate the features of Blazor and Oqtane, and how controls such as the suite available from Syncfusion can make development faster and easier, we will create a Oqtane Help Desk custom module.

Help Desk users



The screenshot shows the Oqtane web application interface. The top navigation bar is blue and contains the Oqtane logo, a home icon, and links for Home, Private, My Page, Custom Module, UserOne, and Logout. Below the navigation bar, there are two tabs: 'NEW HELP DESK TICKET' (active) and 'EXISTING TICKETS'. The form for creating a new ticket includes the following fields:

- Ticket Status:** A dropdown menu with 'New' selected.
- Ticket Date:** A date field showing '4/4/2021' with a calendar icon.
- Ticket Description:** A text area containing 'A new Help Desk Ticket'.
- SUBMIT:** A button at the bottom of the form.

Figure 4: New Help Desk Ticket

Logged-in users who only have view permission of the Oqtane module instance will see a form that allows them to create a new help desk ticket by entering the help desk ticket details and clicking the Submit button.

NEW HELP DESK TICKET		EXISTING TICKETS		
	ID #	Status	Date	Description
EDIT	13	New	4/7/2021	A new help desk ticket
EDIT	12	Open	4/4/2021	My printer doesn't work
<< < 1 > >>				1 of 1 pages (2 items)

Figure 5: Existing Tickets

By clicking the Existing Tickets tab, users can view and edit their tickets.

EDIT TICKET # 12

Ticket Status

Open

Ticket Date

4/4/2021

Ticket Description

My printer doesn't work

My printer is located in cubicle C1234

ADD

SAVE

Figure 6: User Adding Details

When the user clicks the Edit button, the ticket is opened in the Edit Ticket form. However, some of the fields of the help desk ticket are grayed-out and disabled. The user can only change the ticket status and add new details.

Help Desk administrators

Administration

NEW HELP DESK TICKET EXISTING TICKETS

Ticket Status
New

Ticket Date
4/7/2021

Ticket Description

SUBMIT

Figure 7: Administration Button

When a user who has edit permission to the Oqtane module instance views the module, they will see an Administration button. The button will take them to the section of the application that will allow them to administer *all* help desk tickets.

Manage HelpDesk Editing Sorting

	ID # ↑	Status	Date	Description
EDIT DELETE	11	Open	3/30/2021	Printer is out of ink
EDIT DELETE	12	Open	4/4/2021	My printer doesn't work
EDIT DELETE	13	New	4/7/2021	A new help desk ticket

Paging Resizing 2 of 2 pages (13 items)

Figure 8: Syncfusion DataGrid

This will display the Syncfusion DataGrid, which will allow the administrator to edit and delete records, as well as sort, page, and resize the data grid.

EDIT TICKET # 5

Ticket Status
Open

Ticket Date
3/9/2021

Ticket Description
I forgot my password

What is your username?

ADD

SAVE

Figure 9: Syncfusion Dialog

If a user clicks the Edit button next to a record in the DataGrid, it will open in the Syncfusion Dialog control. This dialog will allow the administrator to edit all fields of the help desk ticket, as well as add help desk ticket detail records at the bottom of the form.

	ID # ↑	Status	Date	Descrip
EDIT DELETE	11	Open	3/30/2021	Printer
EDIT DELETE	12	Open	4/4/2021	My pri
EDIT DELETE	13	New	4/7/2021	A new

DELETE RECORD ?

YES NO

Figure 10: Delete Confirmation

If the user clicks the Delete button next to a record in the data grid, the delete confirmation pop-up window will open in the Syncfusion Dialog.

The user can click Yes to delete the record or No to cancel the action.

Chapter 4 Create the Help Desk Module

In this chapter, we will cover the steps to create the Help Desk module.



Note: The source code for the completed application is available on GitHub [here](#).

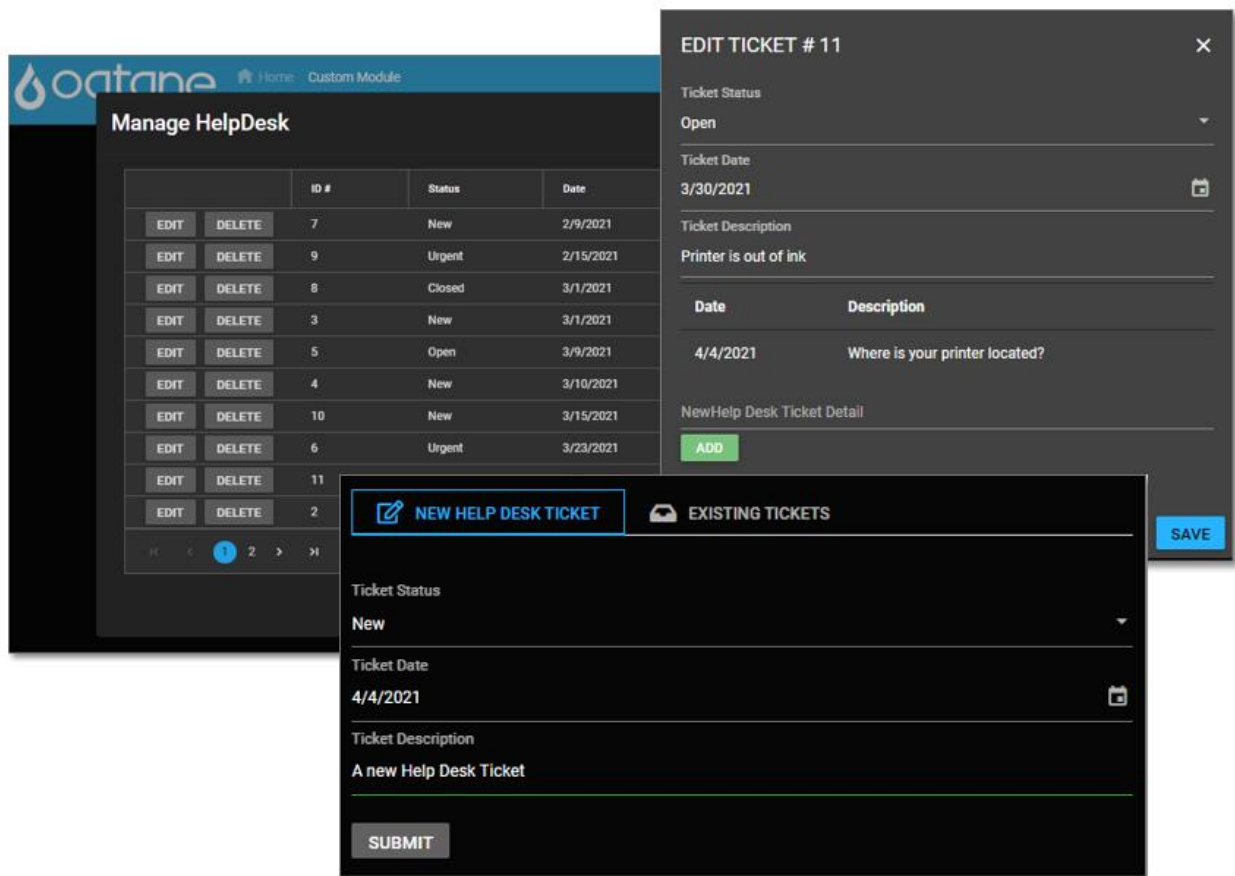


Figure 11: The Help Desk Application in Action

Install .NET Core (aka .NET 5) and Visual Studio

If you don't already have the following software installed, these steps are required:

- Install the .NET 5 SDK from [here](#).
- Install Visual Studio 2019 version 16.8 (or later), with the ASP.NET, web development, and .NET Core workload [here](#).



Note: The requirements to create applications using Blazor are constantly evolving. For the latest requirements, see [this page](#).



Note: When you install Visual Studio 2019 and select the .NET Core workload during installation, the .NET Core SDK and runtime, and SQL Server Express 2016 LocalDB will be installed for you. Learn more [here](#).

Install Oqtane









 Oqtane.Client.2.0.2.nupkg	328 KB
 Oqtane.Framework.2.0.2.Install.zip	22.9 MB
 Oqtane.Framework.2.0.2.nupkg	16.1 MB
 Oqtane.Framework.2.0.2.Upgrade.zip	22.9 MB
 Oqtane.Server.2.0.2.nupkg	138 KB
 Oqtane.Shared.2.0.2.nupkg	52.8 KB
 Source code (zip)	
 Source code (tar.gz)	

Figure 12: Download Oqtane

Go to [this webpage](#) to download and unzip the latest release version of the Oqtane source code to your local system.



Note: This book uses Oqtane version 2.0.2.

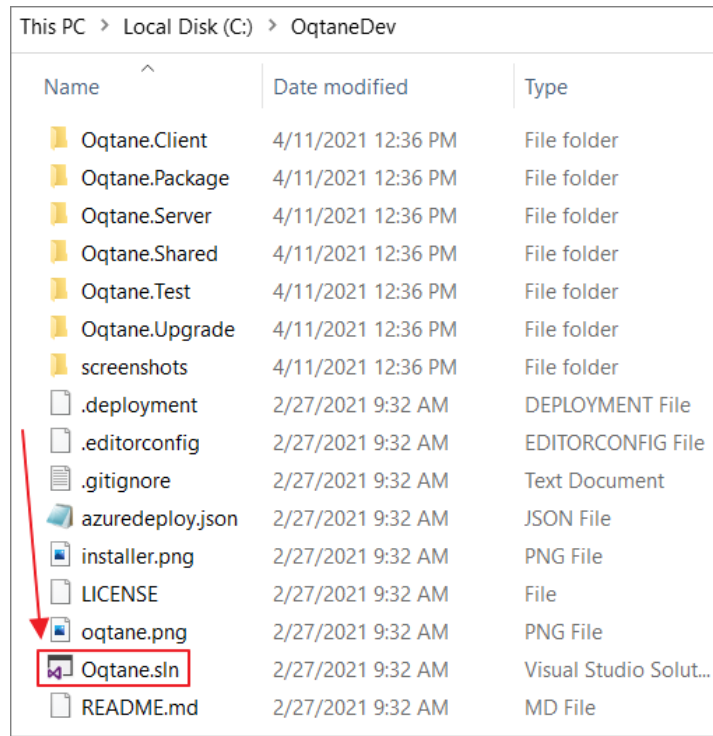


Figure 13: Open Oqtane.sln

Open the **Oqtane.sln** solution file in **Visual Studio** and select **Build > Build Solution**.

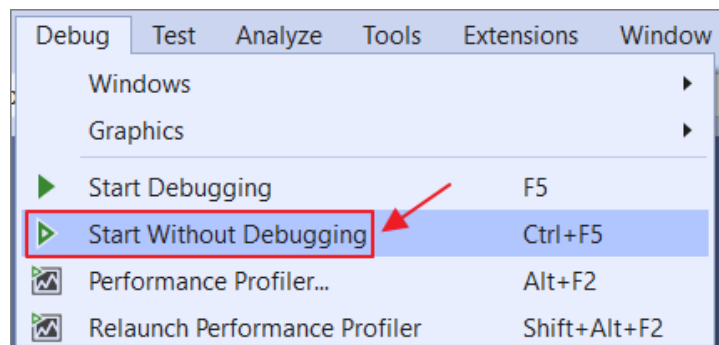
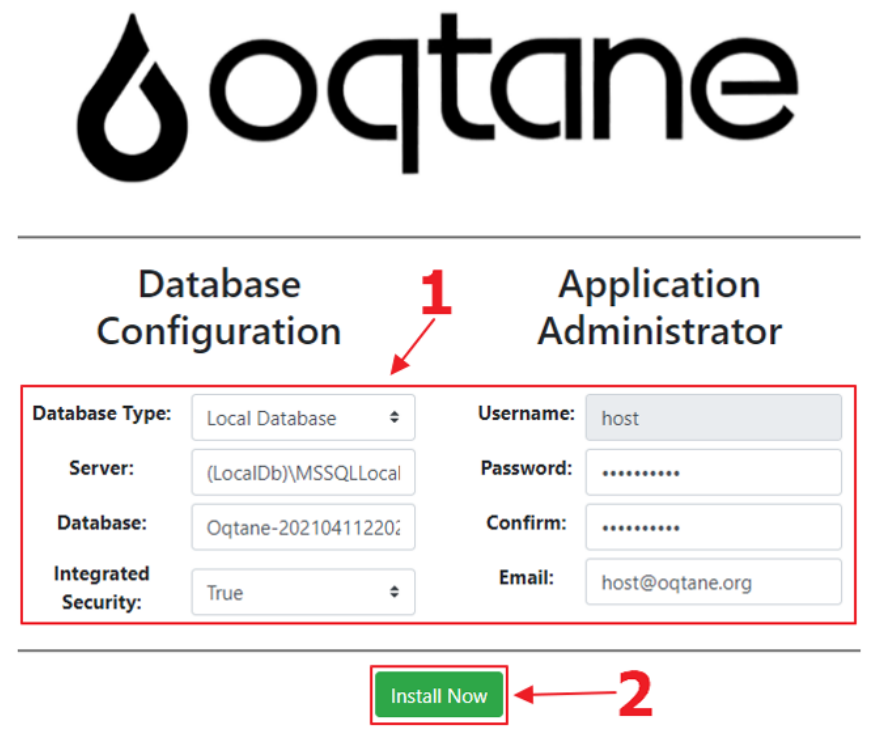


Figure 14: Start Without Debugging

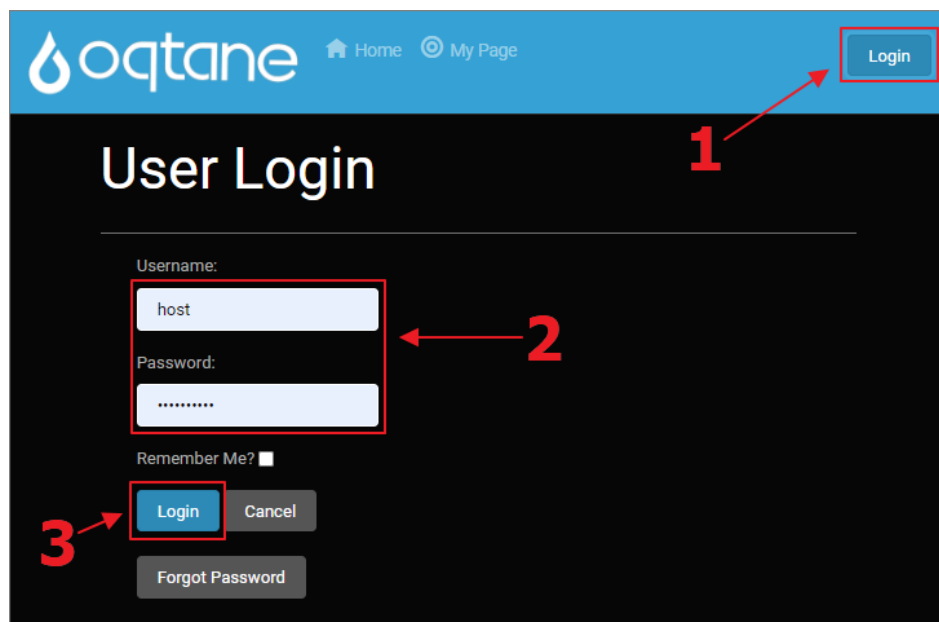
Make sure you specify **Oqtane.Server** as the **Startup Project**, and then select **Debug > Start Without Debugging** to run the application.



The screenshot shows the Oqtane Install Wizard interface. At the top is the Oqtane logo. Below it, the interface is divided into two main sections: "Database Configuration" and "Application Administrator". A red arrow labeled "1" points to the "Database Configuration" section. This section contains several input fields: "Database Type" (set to "Local Database"), "Server" (set to "(LocalDb)\MSSQLLocal"), "Database" (set to "Oqtane-20210411220"), "Integrated Security" (set to "True"), "Username" (set to "host"), "Password" (masked with dots), "Confirm" (masked with dots), and "Email" (set to "host@oqtane.org"). A red arrow labeled "2" points to a green "Install Now" button located at the bottom center of the form.

Figure 15: Oqtane Install Wizard

The Oqtane application will open in your web browser. Fill in the details required by the installation wizard and click **Install Now**.



The screenshot shows the Oqtane User Login page. At the top is a blue header with the Oqtane logo, navigation links for "Home" and "My Page", and a "Login" button. A red arrow labeled "1" points to this "Login" button. Below the header, the page has a dark background with the title "User Login". There are two input fields: "Username" (containing "host") and "Password" (masked with dots). A red arrow labeled "2" points to these input fields. Below the password field is a "Remember Me?" checkbox. At the bottom, there are three buttons: "Login" (highlighted with a red box and a red arrow labeled "3"), "Cancel", and "Forgot Password".

Figure 16: Log into Oqtane

When the installation is complete, click **Login**. Enter the username and password you created earlier and click **Login**.

Oqtane is now fully installed.

Oqtane module creator

There are two methods to create custom modules in Oqtane: *inline* and *external*.

Creating an external Oqtane module allows us to create a module that is more easily packaged to be distributed to other Oqtane instances. This is the method we will use.

An external Oqtane module is created outside of the Oqtane solution, as a separate Visual Studio project, in a folder at the same level as the Oqtane solution. It contains all the project and solution files already configured so that it can build and deploy the artifacts to the Oqtane instance. It can also package a module into a NuGet package.

For assistance in creating the custom module, we will use the built-in Oqtane module creator.

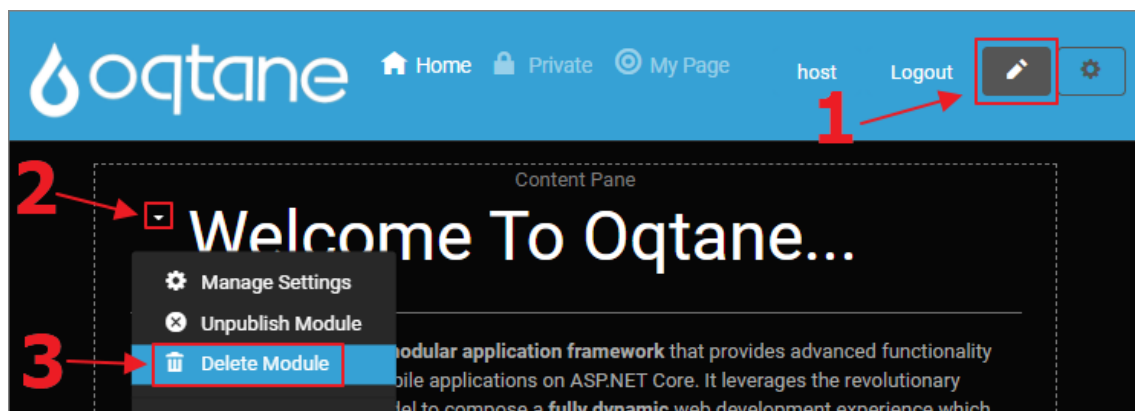


Figure 17: Delete Welcome Message Module

While logged in as the host account, click the pencil icon to open the **Edit** menu. Next, select the context menu for the HTML module on the page and click **Delete Module** to delete that module instance.

Repeat the process to remove the remaining HTML modules on the page.

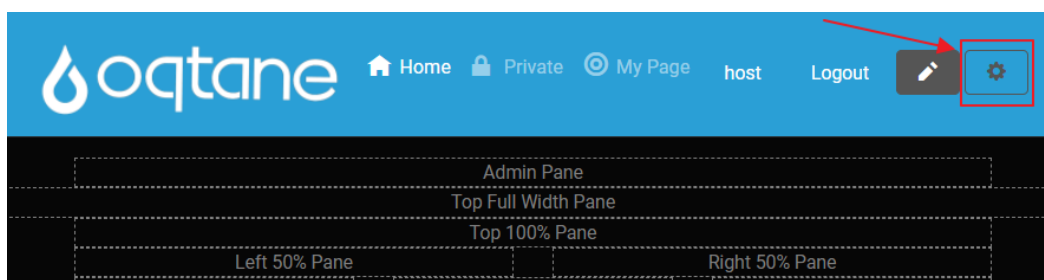


Figure 18: Open Control Panel

Click the gear icon to open the **Control Panel**.

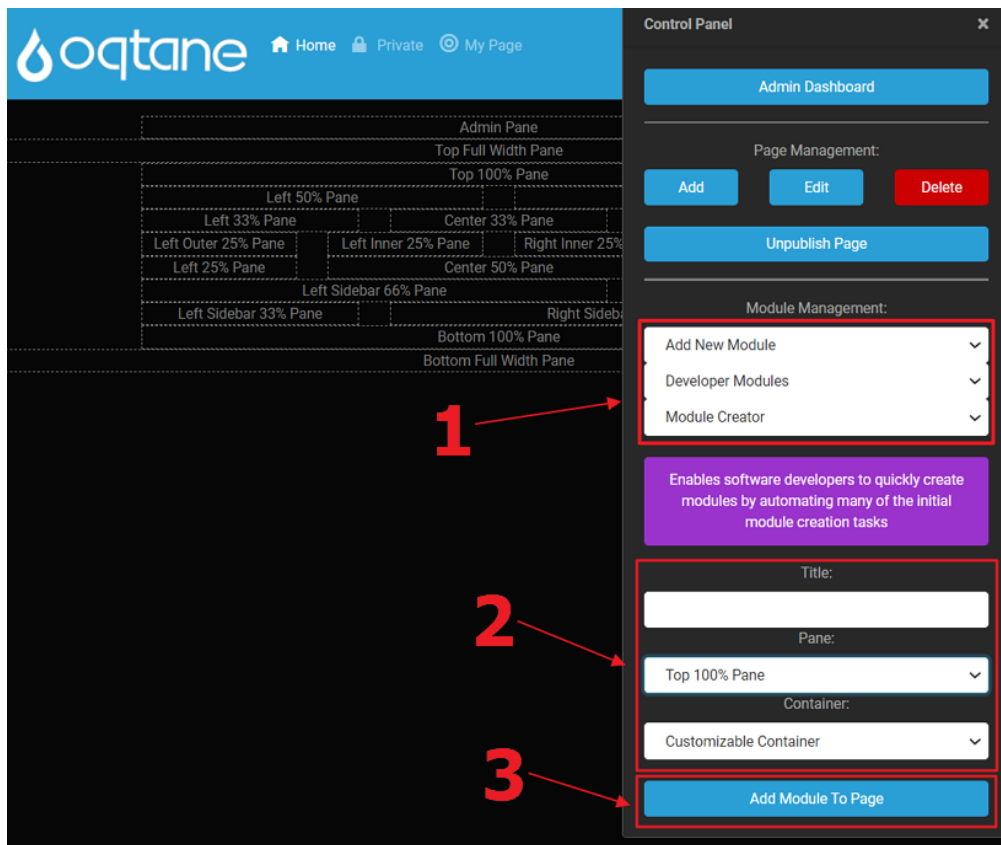


Figure 19: Add Module Creator

In the Module Management section, select **Add New Module > Developer Modules > Module Creator**. Select the **Top 100% Pane** and click the **Add Module To Page** button.

The module creator will be added to the page. Click the **X** in the upper-right corner of the Control Panel to close it.

oqtane Home Private My Page host Logout

Admin Pane
Top Full Width Pane
Top 100% Pane

Module Creator

Please Note That The Module Creator Is Only Intended To Be Used In A Development Environment

Owner Name: ?

Module Name: ? **1**

Description: ?

Template: ?

Framework Reference: ?

Location: ?

Create Module **2**

Left 50% Pane Right 50% Pane

Figure 20: Create Module

Fill in the details for the module according to the preceding image and click **Create Module**.

The Source Code For Your Module Has Been Created At The Location Specified Below And Must Be Compiled In Order To Make It Functional. Once It Has Been Compiled You Must Restart Your Application To Apply These Changes.

Figure 21: Restart Needed

Oqtane will need to restart to create the module.

Click the **Restart** link in the message that displays. Next, click the red **Restart Application** button. Finally, click the red **Restart Application** button.

Close your web browser (but leave Visual Studio open).

Build the project

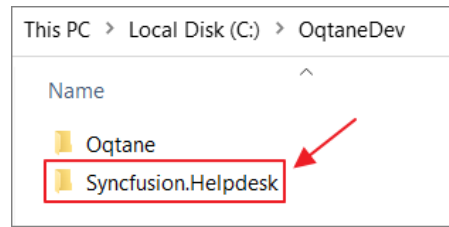


Figure 22: Module Folder Created

If we look at the file system at this point, we see that a new folder was created.

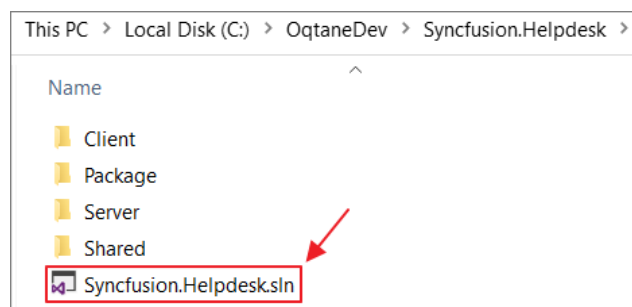


Figure 23: Help Desk Solution

This folder contains all the code created by the wizard. We can see that the wizard created a lot of code that gives us a good starting point for our custom module, including a Visual Studio solution file.

Open the **Visual Studio** solution file in *another instance* of **Visual Studio**.

We now want to build the project, create the **NuGet** package for the module, and put that **NuGet** package in the **Module** folder of the Oqtane solution.

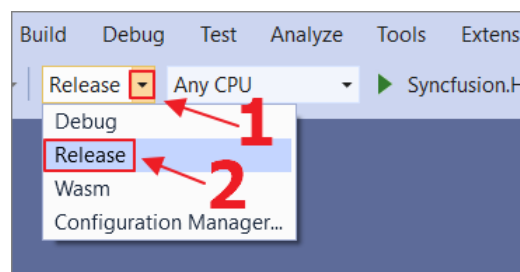


Figure 24: Switch to Release Mode

Switch to **Release** mode (by default, the NuGet package is only set to be created when building in Release mode).

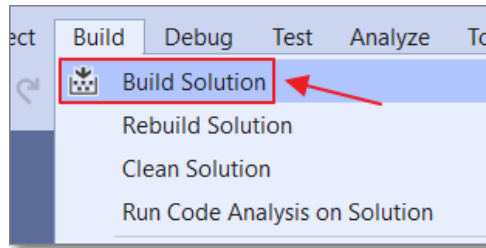


Figure 25: Build Help Desk Solution

Choose **Build Solution**. It should build without errors.

Switch the build mode back to **Debug**.

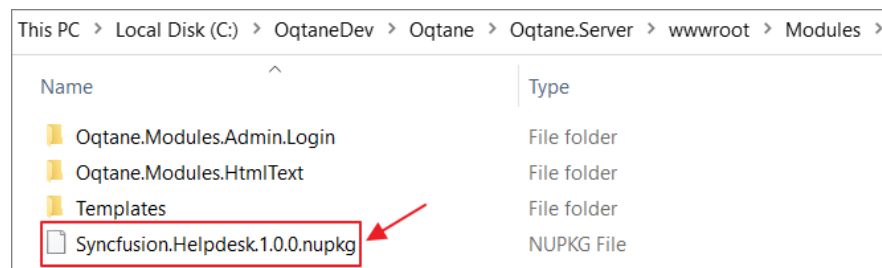



Figure 26: Syncfusion.Helpdesk.1.0.0.nupkg

When we view the **Oqtane.Server/wwwroot/Modules** directory of the Oqtane solution, we see the NuGet package for the module has been copied to that location. When we run the Oqtane application, it will scan this directory and see that the module is to be installed or upgraded.

 **Note:** You can install the **.nupkg** module in other Oqtane installations using **Module Management (in Administration)**, either by uploading it or downloading it from **NuGet in the Available Modules section**. The **Available Modules section in Oqtane** scans **nuget.org** for **NuGet packages that have the Oqtane tag**. The module creator creates a ***.nuspec** for your module, which contains the required **NuGet tags** automatically.

Complete the module installation

Return to the Visual Studio instance that contains the Oqtane solution and select **Start Without Debugging** to restart.

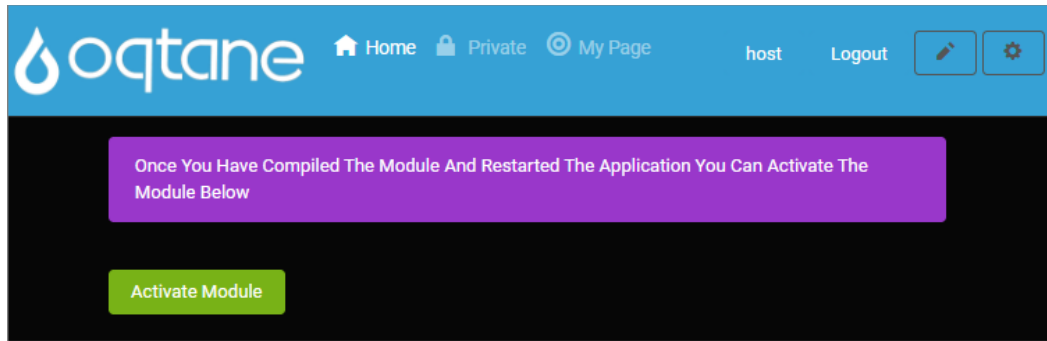


Figure 27: Activate Module Screen

Oqtane will detect that a new module has been added.

Log in as the host account and click **Activate Module** to complete the installation. Another **Activate Module** button will appear. Click it.

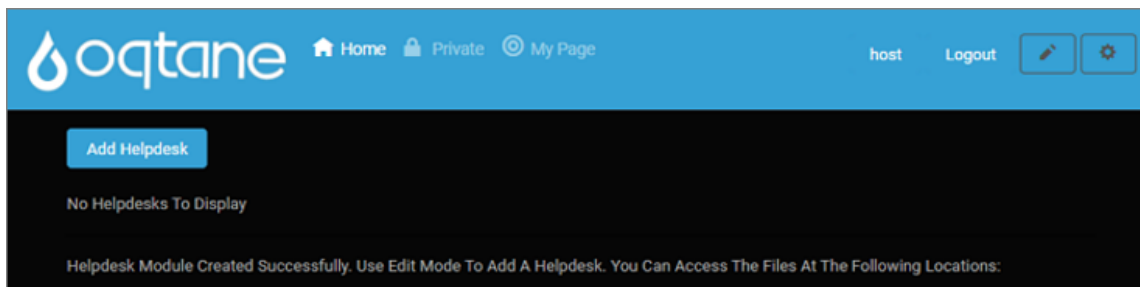


Figure 28: Module Installed

The module will appear.

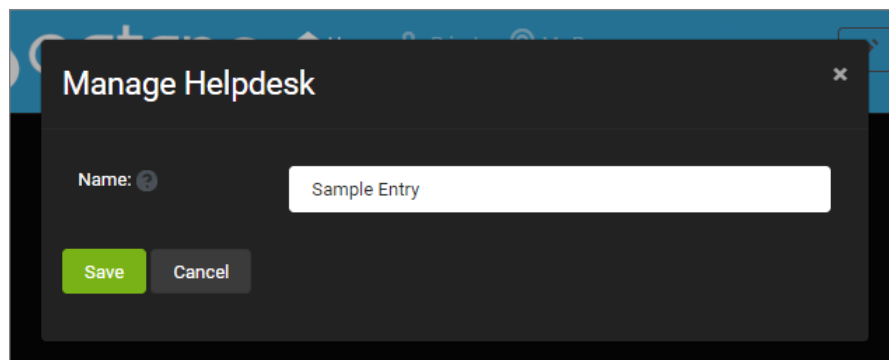


Figure 29: Insert a Record

Click the **Add Helpdesk** button to bring up the form that allows you to insert a sample entry. Enter a value such as **Sample Entry** in the Name field and click **Save**.

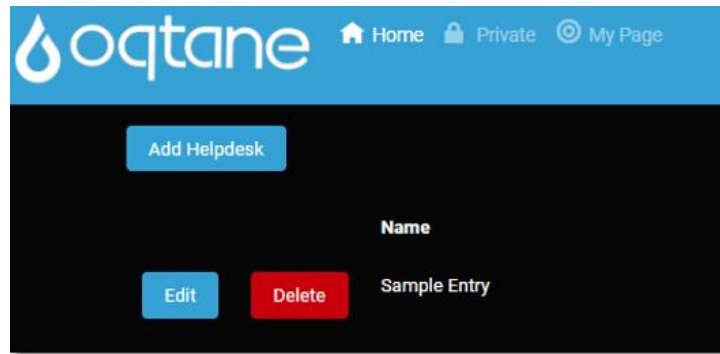


Figure 30: View Sample Entry

We see the record has been added, and we have the option to edit it or delete it.

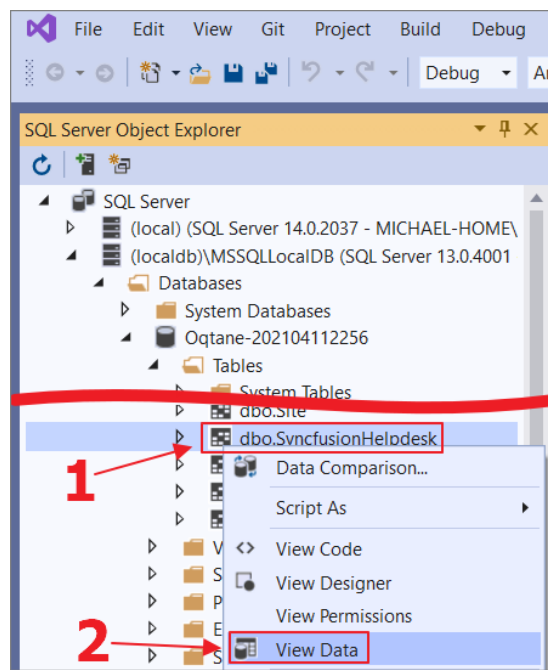


Figure 31: Select View Data

We can return to Visual Studio and select **View > SQL Server Object Explorer**.

We can locate the Oqtane database in the **MSSQLLocalDB** node. After expanding it, we see that a **SyncfusionHelpDesk** table has been created.

Right-click this table and select **View Data** to see the data.

	HelpdeskId	ModuleId	Name	CreatedBy	CreatedOn	ModifiedBy	ModifiedOn
1	1	28	Sample Entry	host	4/17/2021 2:55:...	host	4/17/2021 2:55:...
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 32: View Data

Note that the data records the **ModuleId**. This allows the data to be segmented by module instance.

Explore the module

When we return to the Visual Studio instance that contains the **Syncfusion.Helpdesk** module, we can examine the Visual Studio solution that has been created, which provides a useful starting point for creating our module.

The description of the files, created by the Oqtane module creator, are as follows.

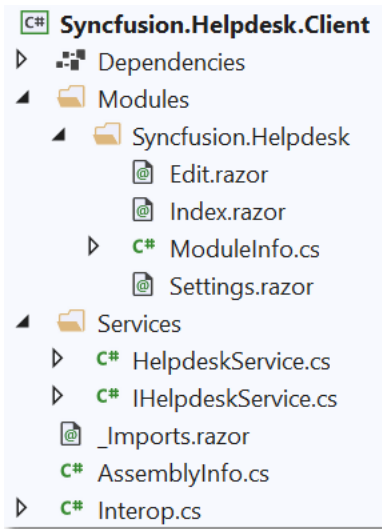


Figure 33: Syncfusion.Helpdesk.Client

Syncfusion.Helpdesk\Client\

- **Modules\Edit.razor**: Component for adding or editing content.
- **Modules\Index.razor**: Main component for your module.
- **ModuleInfo.cs**: Implements the **IModule** interface to provide configuration settings for your module.
- **Modules\Settings.razor**: Component for managing module settings.
- **Services\IHelpdeskService.cs**: Interface for defining service API methods.
- **Services\HelpdeskService.cs**: Implements service API interface methods.
- **_Imports.razor**: Global imports for module components.

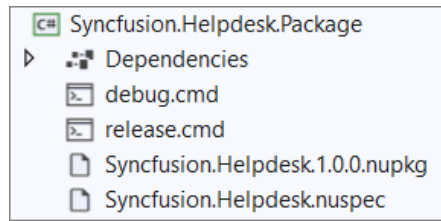


Figure 34: Syncfusion.Helpdesk.Package

Syncfusion.Helpdesk\Package\

- **debug.cmd**: Copies assemblies to Oqtane bin folder when in debug mode.
- **release.cmd**: Creates NuGet package and deploys to Oqtane **wwwroot/modules** folder when in release mode.
- **Syncfusion.Helpdesk.1.0.0.nupkg**: NuGet package that can be deployed to an Oqtane instance.
- **Syncfusion.Helpdesk.nuspec**: NuGet manifest for packaging module.

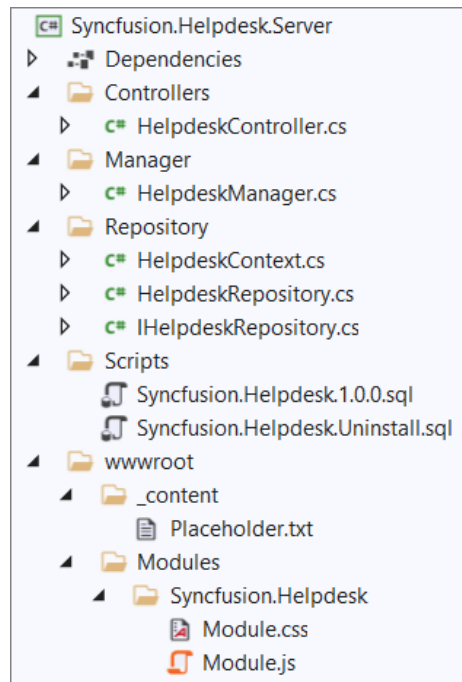


Figure 35: Syncfusion.Helpdesk.Server

Syncfusion.Helpdesk\Server\

- **Controllers\HelpdeskController.cs**: API methods implemented using a REST pattern.
- **Manager\HelpdeskManager.cs**: Implements optional module interfaces for features such as the import or export of content.
- **Repository\HelpdeskContext.cs**: Provides a **DbContext** for data access.
- **Repository\HelpdeskRepository.cs**: Implements repository interface methods for data access using EF Core.
- **Repository\IHelpdeskRepository.cs**: Interface for defining repository methods.
- **Scripts\Syncfusion.Helpdesk.1.0.0.sql**: Database schema definition script.

- Scripts**Syncfusion.Helpdesk.Uninstall.sql**: Database uninstall script.
- wwwroot**Module.css**: Module style sheet.
- wwwroot**Module.js**: Module JavaScript.

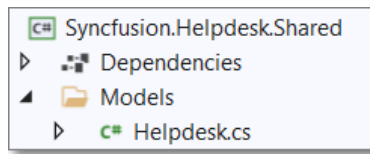


Figure 36: Syncfusion.Helpdesk.Shared

Syncfusion.Helpdesk\Shared\

- Syncfusion.Helpdesk.csproj: Shared project
- Models\Helpdesk.cs: Shared module class

Dynamic routing

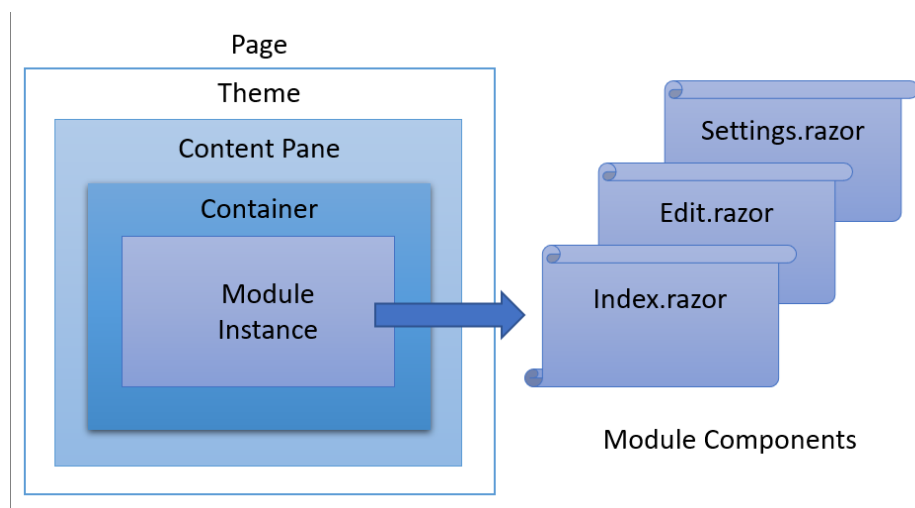


Figure 37: Dynamic Routing and Loading

Oqtane uses *dynamic routing* to load the Oqtane page and any modules on the page.

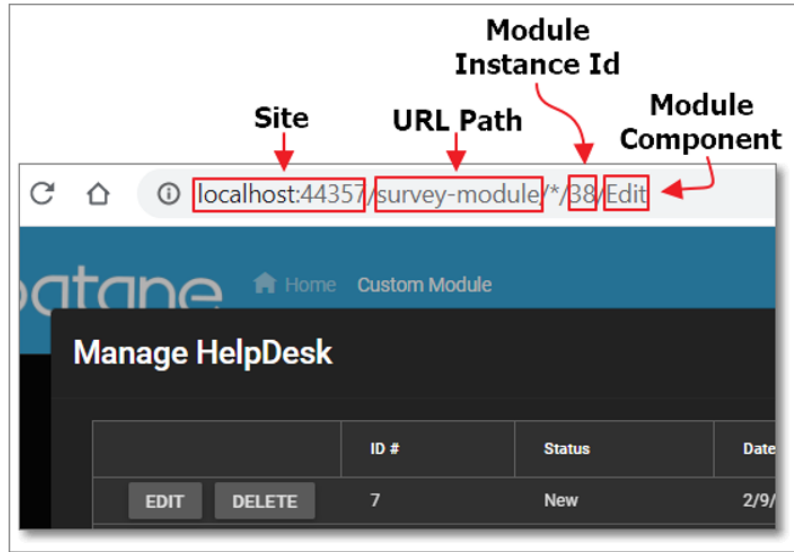


Figure 38: Oqtane URL

When looking at the routing for loading the the Edit module, in the completed module code, we see the the Oqtane URL in the web browser is composed of the following parts:

- **Site:** An Oqtane site can have several tenants, each with its own site URL.
- **URL Path:** An Oqtane page can have a custom name for the URL path to load the page; otherwise, the page name is used for the URL path.
- **Module Instance ID:** An Oqtane module can have multiple instances, each with a unique module ID, on a page and throughout the site.
- **Module Component:** The Index control is the default. Other typical controls are Edit and Settings. Additional controls can be created and loaded.

Chapter 5 Add Syncfusion

The Syncfusion controls allow us to easily implement advanced functionality with a minimum amount of code. The Syncfusion controls are contained in a NuGet package.

In this chapter, we will cover the steps to obtain that package and configure it for our Oqtane custom module.



Note: See the latest requirements to use Syncfusion [here](#).

Install NuGet packages

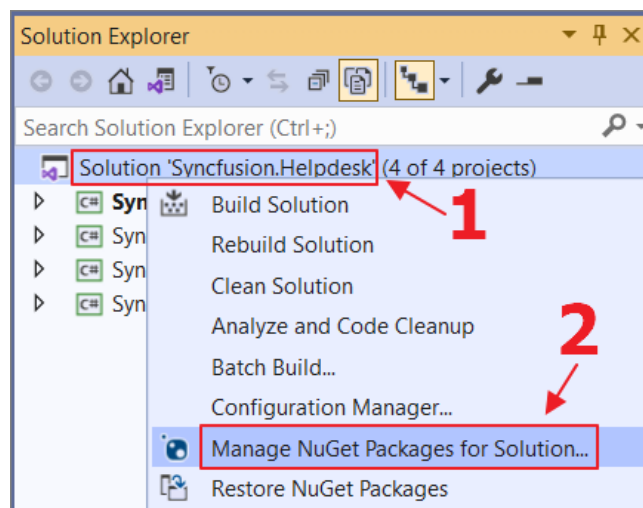


Figure 39: NuGet Packages

Right-click the Solution node and select **Manage NuGet Packages for Solution**.

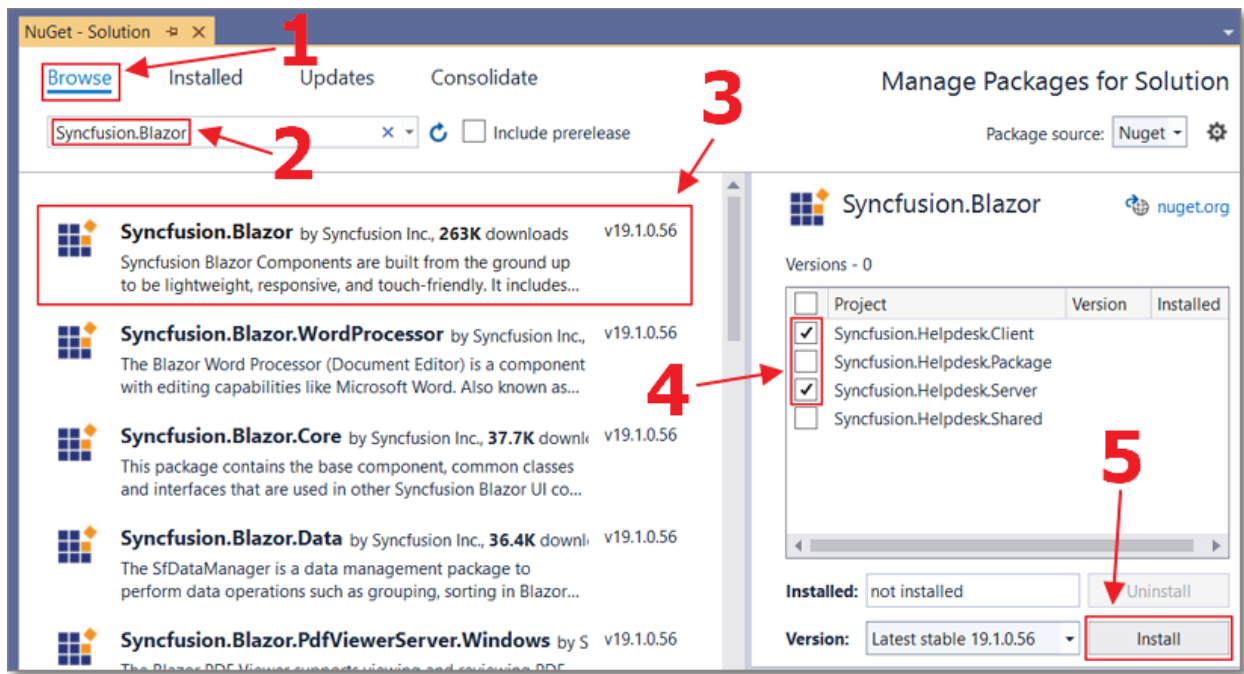


Figure 40: Install NuGet Packages

Go to the **Browse** tab and install the following NuGet packages:

- Syncfusion.Blazor (in the **Client** and **Server** project)
- Microsoft.AspNetCore.Mvc.NewtonsoftJson (in the **Client** project)

Additional configuration

The **Syncfusion.Helpdesk** module will build into the Oqtane framework (in the other Visual Studio instance).

We need to instruct the **Syncfusion.Helpdesk** module to include the Syncfusion assemblies. To do that, open the **debug.cmd** file in the **Syncfusion.Helpdesk.Package Dependencies** entry and remove the **Oqtane.Server\wwwroot** line.

Code Listing 1: debug.cmd

```
XCOPY "..\Server\wwwroot\*" "..\..\Oqtane\Oqtane.Server\wwwroot\" /Y /S
/I
```

Replace it with lines like the following (adjusting to match your directory structure).

Code Listing 2: Server Project Startup.cs Using Statement

```
XCOPY "..\Server\wwwroot\*" "..\..\Oqtane\Oqtane.Server\wwwroot\" /Y /S
/I
```

```
XCOPY "..\Server\bin\Debug\net5.0\Syncfusion.Blazor.dll"
"..\\Oqtane\Oqtane.Server\bin\Debug\net5.0\" /Y
XCOPY "..\Server\bin\Debug\net5.0\Syncfusion.ExcelExport.Net.dll"
"..\\Oqtane\Oqtane.Server\bin\Debug\net5.0\" /Y
XCOPY "..\Server\bin\Debug\net5.0\Syncfusion.Licensing.dll"
"..\\Oqtane\Oqtane.Server\bin\Debug\net5.0\" /Y
XCOPY "..\Server\bin\Debug\net5.0\Syncfusion.PdfExport.Net.dll"
"..\\Oqtane\Oqtane.Server\bin\Debug\net5.0\" /Y
XCOPY "..\Server\bin\Debug\net5.0\Newtonsoft.Json.dll"
"..\\Oqtane\Oqtane.Server\bin\Debug\net5.0\" /Y
```

Where this example has **Oqtane**, you will need to replace that with the name of the folder of your Oqtane installation. In this example, the Oqtane solution is installed at:
C:\OqtaneDev\Oqtane.

Open the **Syncfusion.Helpdesk.nuspec** file in the **Syncfusion.Helpdesk.Package** project and add the following lines to the **files** section.

Code Listing 3: Syncfusion.Helpdesk.nuspec

```
<file src="..\Server\bin\Release\net5.0\Syncfusion.Blazor.dll"
target="lib\net5.0" />
<file src="..\Server\bin\Release\net5.0\Syncfusion.ExcelExport.Net.dll"
target="lib\net5.0" />
<file src="..\Server\bin\Release\net5.0\Syncfusion.Licensing.dll"
target="lib\net5.0" />
<file src="..\Server\bin\Release\net5.0\Syncfusion.PdfExport.Net.dll"
target="lib\net5.0" />
<file src="..\Server\bin\Release\net5.0\Newtonsoft.Json.dll"
target="lib\net5.0" />
```

Add using statements to Imports.razor



Figure 41: _Imports.razor

To allow the use of the Syncfusion components without the need to include their **using** declaration on each component file, open the **_Imports.razor** file in the **Client** project, and add the following **using** statements.

Code Listing 4: _Imports.razor

```
@using Microsoft.AspNetCore.Components.Forms
@using Syncfusion.Blazor
```

```
@using Syncfusion.Blazor.Inputs
@using Syncfusion.Blazor.Popups
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.DropDowns
@using Syncfusion.Blazor.Layouts
@using Syncfusion.Blazor.Calendars
@using Syncfusion.Blazor.Navigations
@using Syncfusion.Blazor.Lists
@using Syncfusion.Blazor.Grids
@using Syncfusion.Blazor.Buttons
@using Syncfusion.Blazor.Notifications
```

Implement Syncfusion in IServerStartup

Syncfusion requires its service to be registered in the **Startup** class of the Oqtane framework application.

To do this, in the **Syncfusion.Helpdesk.Server** module project, we need to create a class that implements the Oqtane **IServerStartup** interface.

[According to Shaun Walker](#) (the creator of Oqtane):

“The **IServerStartup** interface matches the characteristics of the standard .NET Core Startup methods. You can create a class in your external module's Server project and add whatever service registration logic you would normally add in the Startup class. During startup, Oqtane will call the methods automatically.”

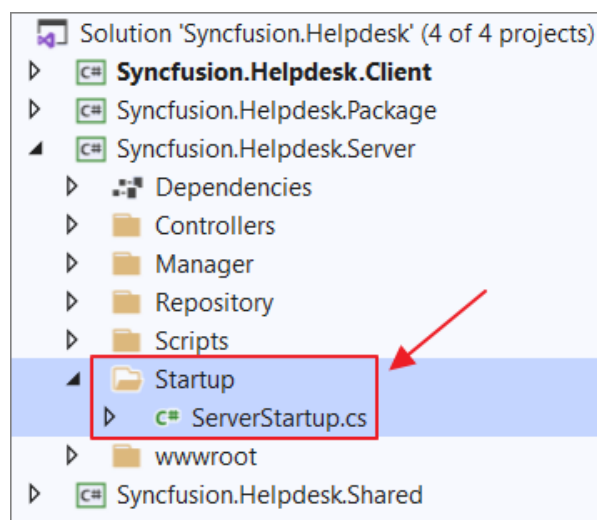


Figure 42: ServerStartup.cs

In the **Server** project, create a folder called **Startup**, and create a class file in it called **ServerStartup.cs** using the following code.

Code Listing 5: ServerStartup.cs

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.DependencyInjection;
using Oqtane.Infrastructure;
using Syncfusion.Blazor;
namespace Syncfusion.Helpdesk.Server.Startup
{
    public class ServerStartup : IServerStartup
    {
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddSyncfusionBlazor();
        }
        public void Configure(IApplicationBuilder app,
IWebHostEnvironment env)
        {
            //Register Syncfusion license.

//Syncfusion.Licensing.SyncfusionLicenseProvider.RegisterLicense("YOUR
LICENSE KEY");
        }
        public void ConfigureMvc(IMvcBuilder mvcBuilder)
        {
        }
    }
}
```

Support Oqtane WebAssembly mode

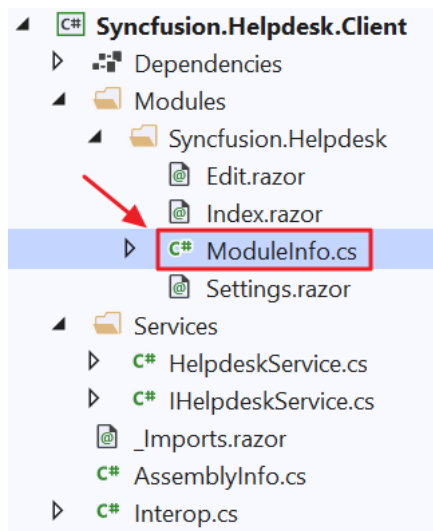


Figure 43: ModuleInfo.cs

Oqtane can run in two modes: **Server** and **Client (WebAssembly)**.



Note: You can switch modes by updating the *Runtime* property to either *Server* or *WebAssembly* in the *appsettings.json* file in the *Oqtane* website.

According to Shaun Walker:

“The **ModuleInfo.cs** file in the **Client** project contains the implementation for **IModule**, which contains a **Dependencies** property. This should contain a comma delimited list of all assemblies that need to be downloaded to the client when running on **WebAssembly**.”



Note: You can find more information on why this is needed at [this link](#).

Open the **ModuleInfo.cs** file, add a comma, and add the Syncfusion assemblies to the **Dependencies** property, so that it reads like the following.

Code Listing 6: Dependencies

```
Dependencies =  
"Syncfusion.Helpdesk.Shared.Oqtane,Syncfusion.Blazor,Syncfusion.ExcelExport  
.Net,Syncfusion.Licensing,Syncfusion.PdfExport.Net,Newtonsoft.Json"
```

Syncfusion is properly registered when running **Server** mode by creating a class that implements the **IServerStartup** interface. However, when Syncfusion is running in **WebAssembly** mode, it normally needs to be registered in the **Program.cs** file in the Blazor **Client** project.

To do this in Oqtane, in the **Client** project, create a folder named **Startup**, and in that folder, create a **ClientStartup.cs** file.

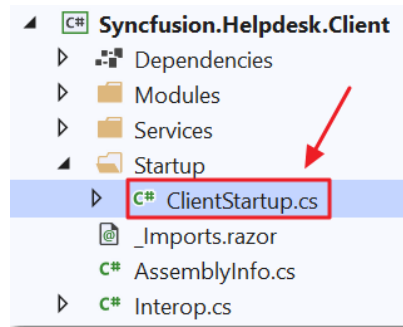


Figure 44: ClientStartup.cs

Complete this class to implement the **IClientStartup** interface by using the following code.

Code Listing 7: ClientStartup.cs

```
using Microsoft.Extensions.DependencyInjection;
using Oqtane.Services;
using Syncfusion.Blazor;
namespace Syncfusion.Helpdesk.Client.Startup
{
    public class ClientStartup : IClientStartup
    {
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddSyncfusionBlazor();
            //Syncfusion.Licensing.SyncfusionLicenseProvider
            //.RegisterLicense("Enter Your Syncfusion License Here");
        }
    }
}
```

Including the JavaScript and .css files

Syncfusion components require JavaScript and .css files to be loaded using the Blazor **_content** feature. Normally, these elements would be automatically extracted from the Syncfusion NuGet package and deployed to the Blazor **_content** folder when the project is built in **Visual Studio**.

However, due to the unique method Oqtane uses to dynamically load custom modules, we have to use the following method of manually extracting the required resources from the Syncfusion NuGet package and placing them in a location where Oqtane and the Syncfusion controls can consume them.

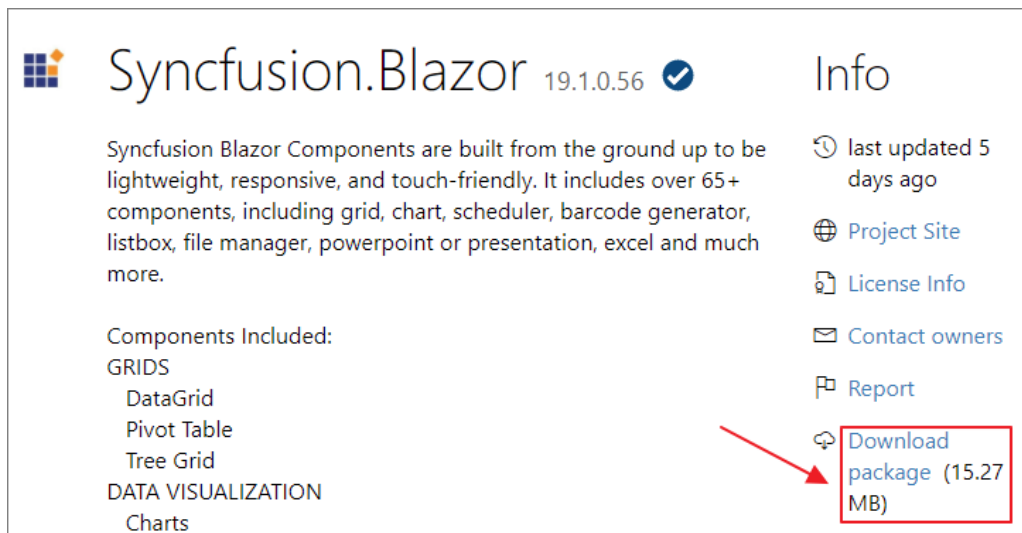


Figure 45: Download NuGet Package

First, we download the NuGet package [here](#).

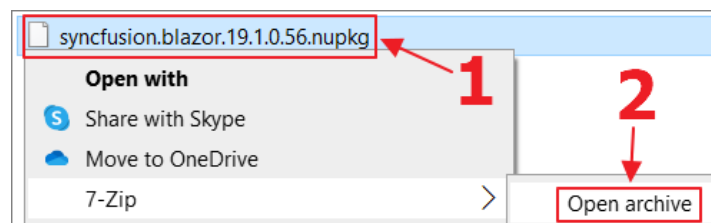


Figure 46: Extract Using 7-Zip

To unzip it, we use a tool such as 7-Zip, which you can [download here](#).

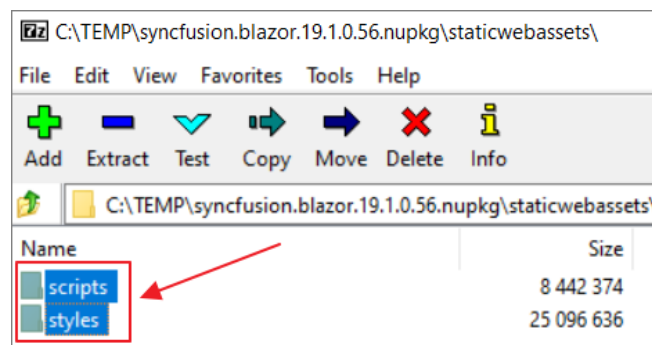


Figure 47: Extract Files

We then extract the required resources from the **staticwebassets** folder.

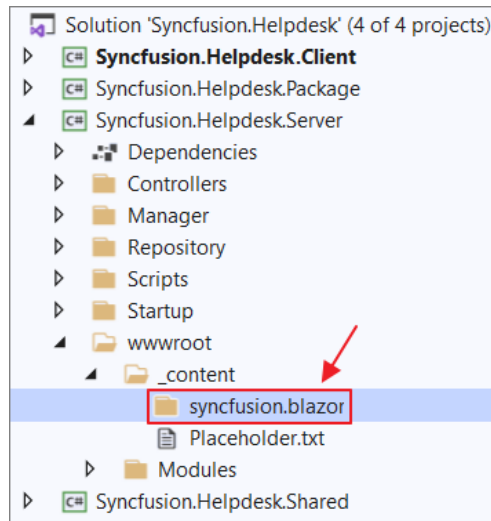


Figure 48: Create syncfusion.blazor

Next, we create the **syncfusion.blazor** directory under **Server\wwwroot_content**.

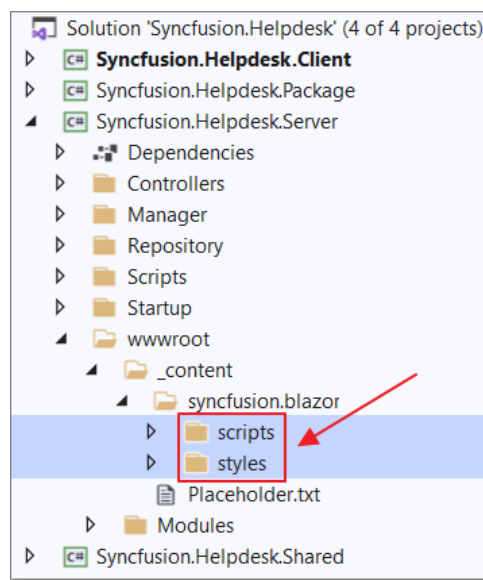


Figure 49: Copy and Paste Files

Finally, we copy and paste the files in the **Server\wwwroot_content\syncfusion.blazor** directory.

Using IHostResources to include references in _Host.cshtml

UI component suites, such as Syncfusion's, require .css, JavaScript, and fonts to be referenced in the **_Hosts.cshtml** file of the main Blazor application (in this case, the Oqtane framework).

To facilitate this, Oqtane requires your module to include a class that implements the **IHostResources** interface.

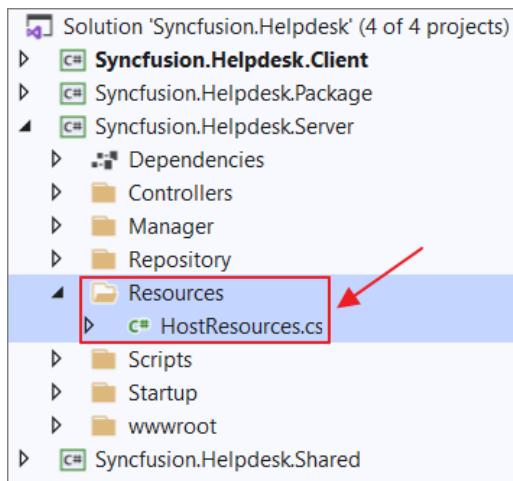


Figure 50: HostResources.cs

Create a **Resources** folder in the Server project and add a **HostResources.cs** file using the following code.

Code Listing 8: HostResources.cs

```
using System.Collections.Generic;
using Oqtane.Infrastructure;
using Oqtane.Models;
using Oqtane.Shared;
namespace Syncfusion.Helpdesk
{
    public class HostResources : IHostResources
    {
        public List<Resource> Resources => new List<Resource>()
        {
            // Only register .css files.
            // The JavaScript files will automatically be pulled
            // from the _content/syncfusion.blazor directory
            // so they do not need to be registered here.
            new Resource {
                ResourceType = ResourceType.Stylesheet,
                Url = "_content/Syncfusion.Blazor/" +
                    "styles/bootstrap4.css" },
            new Resource {
                ResourceType = ResourceType.Stylesheet,
                Url = "_content/Syncfusion.Blazor/" +
                    "styles/material-dark.css" },
            new Resource {
                ResourceType = ResourceType.Script,
```

```

        };
    }
}
    Url = "https://kit.fontawesome.com/a076d05399.js" }

```

Oqtane > Oqtane.Server > wwwroot > _content > syncfusion.blazor	
Name	Type
scripts	File folder
styles	File folder

Figure 51: JavaScript and CSS Deployed

If we now build the module in Visual Studio and look in the Oqtane `/Oqtane.Server/wwwroot/_content/syncfusion.blazor` directory, we see that the JavaScript and .css are properly deployed to the `_content` directory.



Note: If you get build errors, such as `exited with code 4`, close and reopen the instance of Visual Studio that contains the Oqtane solution. It can sometimes prevent the instance of Visual Studio that contains the custom module from building fully.

Consume the Syncfusion controls

We are now ready to consume the Syncfusion UI controls in our Oqtane module.

Open the `Index.razor` file in the **Client** project and replace the HTML markup with the following code (keep the C# code as is in the `@code` section).

Code Listing 9: Consume Syncfusion

```

@using Syncfusion.Helpdesk.Services
@using Syncfusion.Helpdesk.Models

@namespace Syncfusion.Helpdesk
@inherits ModuleBase
@inject IHelpdeskService HelpdeskService
@inject NavigationManager NavigationManager

@using Syncfusion.Blazor.Grids

@if (_Helpdesks == null)
{
    <p><em>Loading...</em></p> }
else
{

```

```

<br />
<ActionLink Action="Add"
            Security="SecurityAccessLevel.Edit"
            Text="Add" />

<br />
<br />
@if (_Helpdesks.Count != 0)
{
    <SfGrid DataSource="@_Helpdesks"
            AllowPaging="true">
        <GridPageSettings PageSize="5"></GridPageSettings>
    </SfGrid>
}
}

```

Rebuild the module in Visual Studio (ensure you are in debug mode in Visual Studio and not release mode). Run the application in the instance of Visual Studio that contains the Oqtane solution.

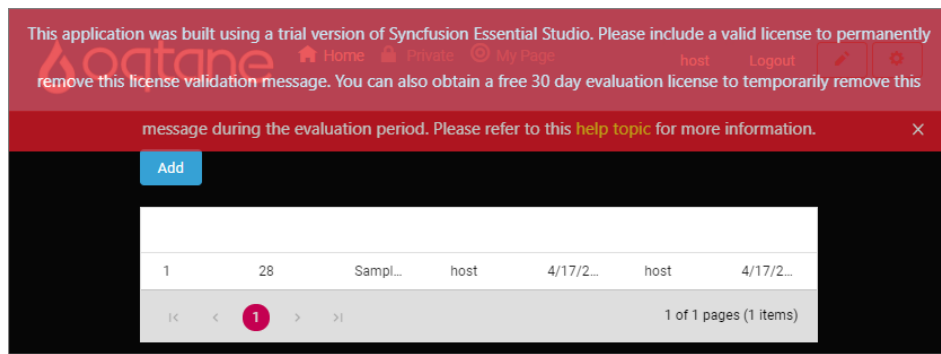


Figure 52: Consume Syncfusion Controls

The module will be displayed. You can add records using the Add button, and they will be displayed in the Syncfusion grid control.

You will see a Syncfusion license notice; to remove it, you will need to obtain a license. You can obtain a free one [here](#).



Note: The following only applies to Oqtane running in Server mode. If running in WebAssembly mode, you will have to hard-code the Syncfusion license in the *ClientStartup.cs* file.

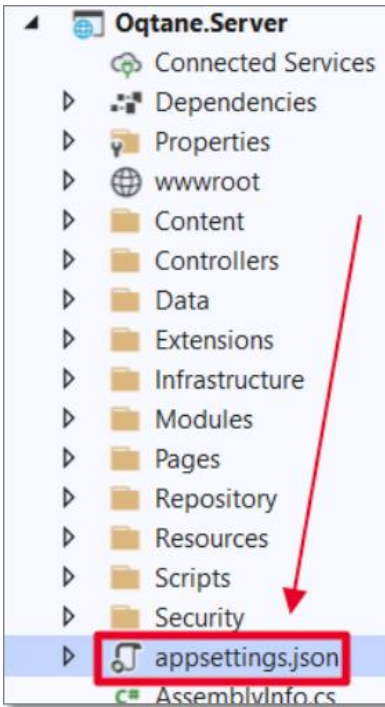


Figure 53: Open appsettings.json

In the Oqtane project, open the **appsettings.json** file.

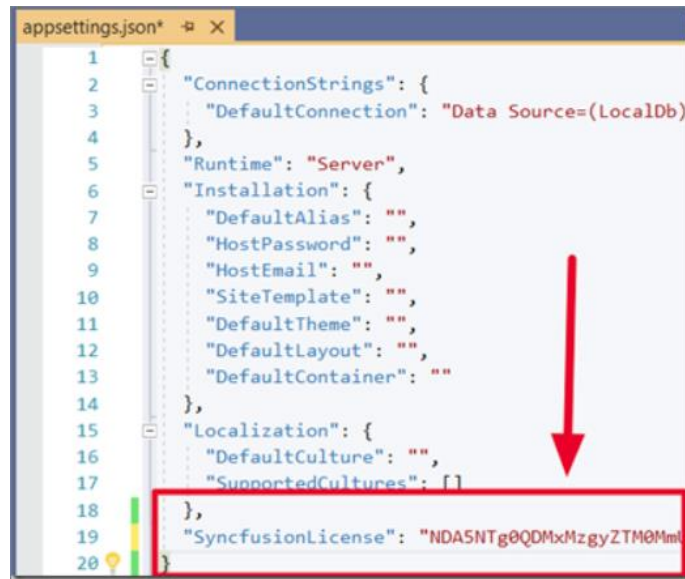


Figure 54: Insert Syncfusion License

Add a **SyncfusionLicense** node and the Syncfusion license as its value.

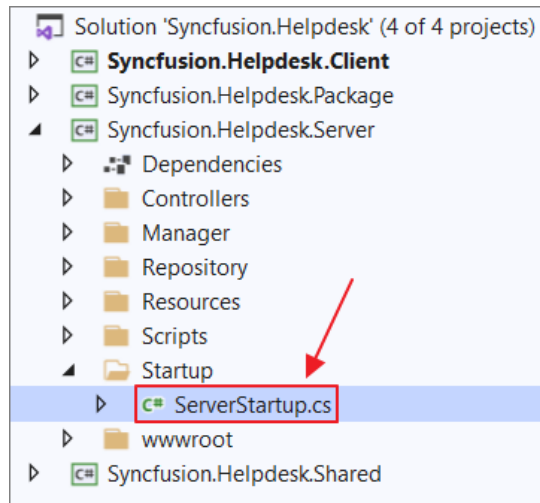


Figure 55: Add License Code

In the **Syncfusion.Helpdesk** solution in Visual Studio, open the **ServerStartup.cs** file.

Add the following **using** statement.

Code Listing 10: Add Microsoft.Extensions.Configuration

```
using Microsoft.Extensions.Configuration;
```

Change the **Configure** method to the following.

Code Listing 11: Configure Method

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    // Register Syncfusion license.
    // Get a free license here:
    // https://www.syncfusion.com/products/communitylicense
    var builder = new ConfigurationBuilder()
        .SetBasePath(env.ContentRootPath)
        .AddJsonFile("appsettings.json", optional: false,
            reloadOnChange: true);
    var Configuration = builder.Build();
    var SyncfusionLicense =
        Configuration.GetSection("SyncfusionLicense");
    if (SyncfusionLicense != null)
    {
        Syncfusion.Licensing.SyncfusionLicenseProvider
            .RegisterLicense(SyncfusionLicense.Value);
    }
}
```

Rebuild the **Syncfusion.Helpdesk** solution and restart the Oqtane site.

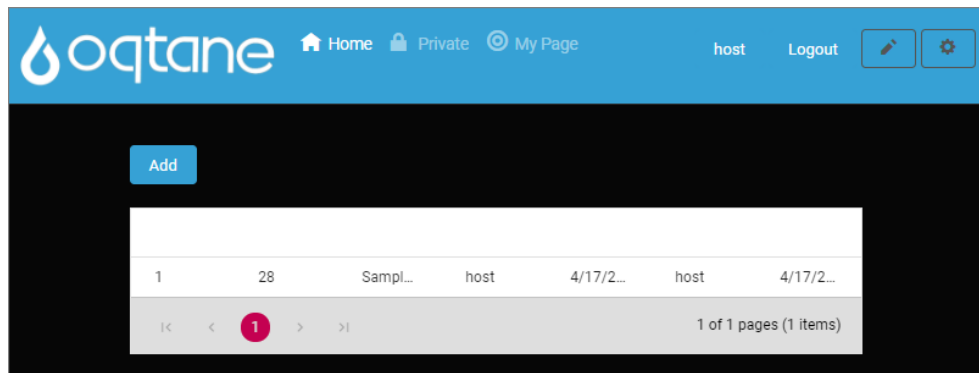


Figure 56: Oqtane Site with No License Warning

Now when you run the application, the warning no longer appears.

Chapter 6 Creating the Data Layer

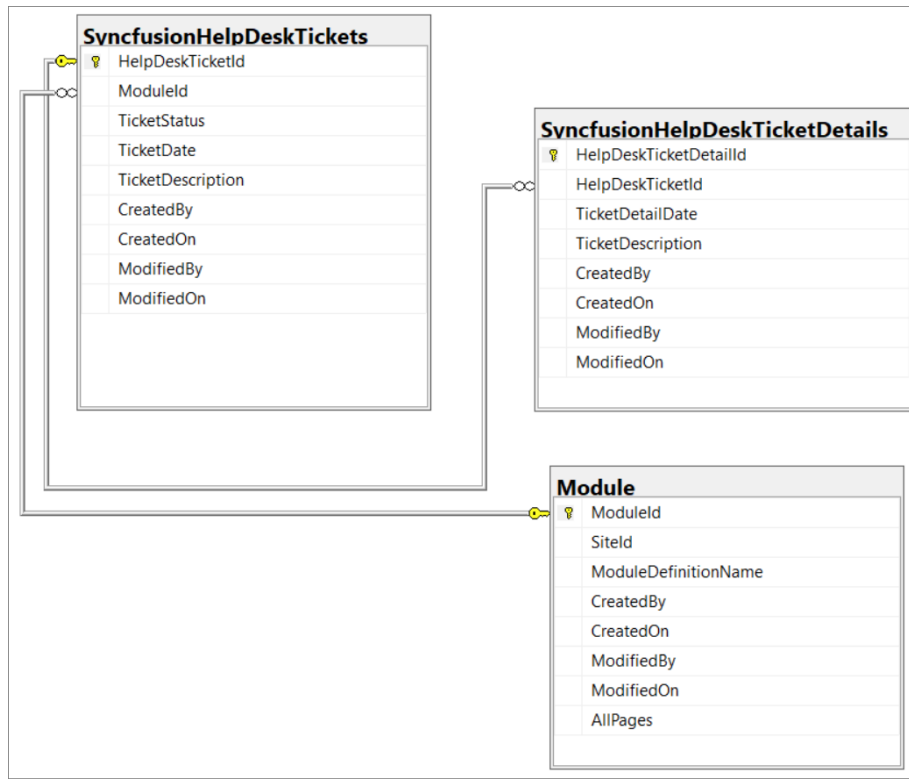


Figure 57: Database Diagram

We will now add tables to the database to support the custom code we plan to write. To allow our code to communicate with these tables, we require a data layer. This data layer will also allow us to organize and reuse code efficiently.

The diagram in Figure 57 shows the relationship between the tables. A **SyncfusionHelpDeskTickets** record is created first. As the issue is processed, multiple associated **SyncfusionHelpDeskTicketDetails** records are added.

The **SyncfusionHelpDeskTickets** table also has a relationship to the Oqtane framework **Module** table.

Upgrade the module

Oqtane has a technique to upgrade modules that allows us to create additional SQL scripts to add new database tables.

Create the database tables

The first step is to add the database tables we will need. To do this, we will add the SQL scripts required to create the database tables. Oqtane will execute the scripts when the module is reinstalled.

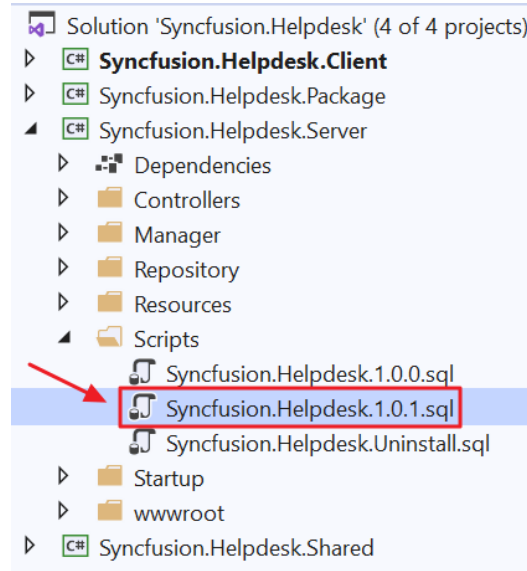


Figure 58: Syncfusion.Helpdesk.1.0.1.sql

Add a new file, **Syncfusion.Helpdesk.1.0.1.sql**, to the **Server** project (in the **Scripts** folder) using the following code.

Code Listing 12: Syncfusion.Helpdesk.1.0.0.sql

```
/*
Create SyncfusionHelpDesk tables.
*/

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[SyncfusionHelpDeskTicketDetails](
    [HelpDeskTicketDetailId] [int] IDENTITY(1,1) NOT NULL,
    [HelpDeskTicketId] [int] NOT NULL,
    [TicketDetailDate] [datetime] NOT NULL,
    [TicketDescription] [nvarchar](max) NOT NULL,
    [CreatedBy] [nvarchar](256) NOT NULL,
    [CreatedOn] [datetime] NOT NULL,
    [ModifiedBy] [nvarchar](256) NOT NULL,
    [ModifiedOn] [datetime] NOT NULL,
    CONSTRAINT [PK_SyncfusionHelpDeskTicketDetails] PRIMARY KEY CLUSTERED
```

```

(
    [HelpDeskTicketDetailId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[SyncfusionHelpDeskTickets](
    [HelpDeskTicketId] [int] IDENTITY(1,1) NOT NULL,
    [ModuleId] [int] NOT NULL,
    [TicketStatus] [nvarchar](50) NOT NULL,
    [TicketDate] [datetime] NOT NULL,
    [TicketDescription] [nvarchar](max) NOT NULL,
    [CreatedBy] [nvarchar](256) NOT NULL,
    [CreatedOn] [datetime] NOT NULL,
    [ModifiedBy] [nvarchar](256) NOT NULL,
    [ModifiedOn] [datetime] NOT NULL,
    CONSTRAINT [PK_HelpDeskTickets] PRIMARY KEY CLUSTERED
(
    [HelpDeskTicketId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
ALTER TABLE [dbo].[SyncfusionHelpDeskTicketDetails] WITH CHECK ADD
CONSTRAINT [FK_SyncfusionHelpDeskTicketDetails_SyncfusionHelpDeskTickets]
FOREIGN KEY([HelpDeskTicketId])
REFERENCES [dbo].[SyncfusionHelpDeskTickets] ([HelpDeskTicketId])
ON DELETE CASCADE
GO
ALTER TABLE [dbo].[SyncfusionHelpDeskTicketDetails] CHECK CONSTRAINT
[FK_SyncfusionHelpDeskTicketDetails_SyncfusionHelpDeskTickets]
GO

/*
Create foreign key relationships.
*/

ALTER TABLE [dbo].[SyncfusionHelpDeskTickets] WITH CHECK ADD CONSTRAINT
[FK_SyncfusionHelpDeskTickets_Module] FOREIGN KEY([ModuleId])
REFERENCES [dbo].[Module] ([ModuleId])
ON DELETE CASCADE
GO

```

```
ALTER TABLE [dbo].[SyncfusionHelpDeskTickets] CHECK CONSTRAINT  
[FK_SyncfusionHelpDeskTickets_Module]  
GO
```

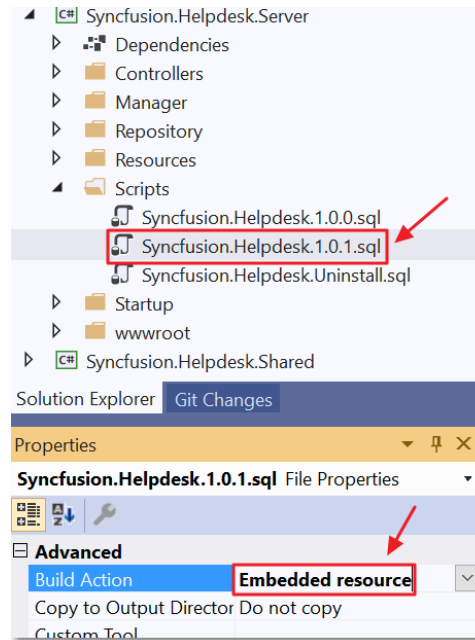


Figure 59: Set to Embedded resource

Click the file in the Solution Explorer, and in the **Properties**, set the **Build Action** to **Embedded resource**.

So that the database tables will be removed when the module is uninstalled, open the **Syncfusion.Helpdesk.Uninstall.sql** file and replace all the code with the following code.

Code Listing 13: Syncfusion.Helpdesk.Uninstall.sql

```
/*  
Remove SyncfusionHelpdesk table.  
*/  
  
DROP TABLE [dbo].[SyncfusionHelpdesk]  
GO  
DROP TABLE [dbo].[SyncfusionHelpDeskTicketDetails]  
GO  
DROP TABLE [dbo].[SyncfusionHelpDeskTickets]  
GO
```

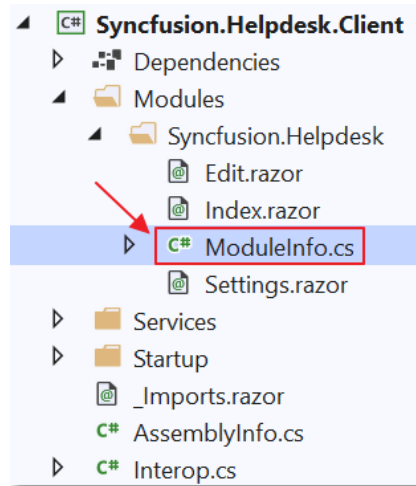


Figure 60: Update ModuleInfo.cs

In the Client project, open the **ModuleInfo.cs** file and update the **Version** property to the following.

Code Listing 14: Update Version

```
Version = "1.0.1",
```

Update the **ReleaseVersions** property to the following.

Code Listing 15: Update ReleaseVersions

```
ReleaseVersions = "1.0.0,1.0.1",
```

Switch the **Syncfusion.Helpdesk** solution to **Release** mode and then rebuild it. This will cause a new, updated NuGet package to be created.

Switch the **Syncfusion.Helpdesk** solution back to **Debug** mode and restart the Oqtane site. The Oqtane site will detect the updated NuGet package and run the updated **.sql** script.

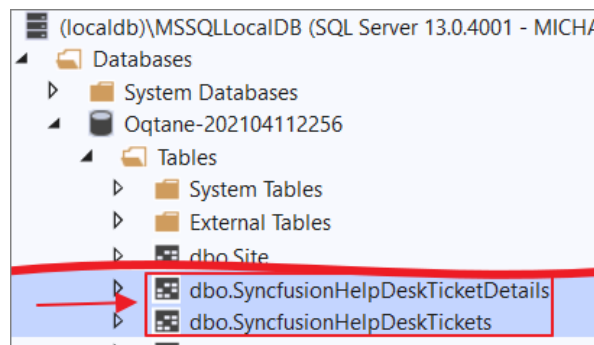


Figure 61: New Tables Created

We can look in the database and confirm the new database tables have been created.

Shared classes

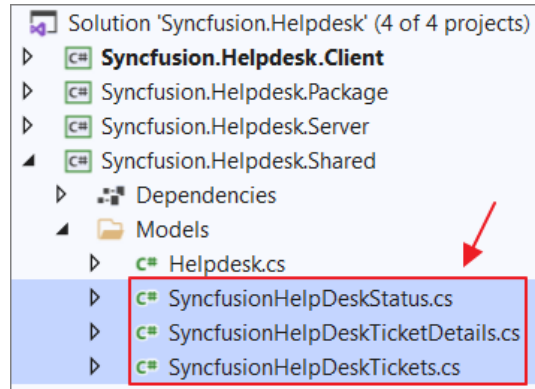


Figure 62: Shared Classes

We will now create classes that will be shared between the **Server** and **Client** projects.

In the **Shared** project, create the following files using the following code.

SyncfusionHelpDeskStatus.cs

Code Listing 16: SyncfusionHelpDeskStatus.cs

```
using System.Collections.Generic;

namespace Syncfusion.Helpdesk.Models
{
    public class SyncfusionHelpDeskStatus
    {
        public string ID { get; set; }
        public string Text { get; set; }

        public static List<SyncfusionHelpDeskStatus> Statuses =
            new List<SyncfusionHelpDeskStatus>() {
                new SyncfusionHelpDeskStatus(){ ID= "New", Text= "New" },
                new SyncfusionHelpDeskStatus(){ ID= "Open", Text= "Open" },
                new SyncfusionHelpDeskStatus(){ ID= "Urgent", Text= "Urgent" },
                new SyncfusionHelpDeskStatus(){ ID= "Closed", Text= "Closed" },
            };
    }
}
```

SyncfusionHelpDeskTicketDetails.cs

Code Listing 17: SyncfusionHelpDeskTicketDetails.cs

```
using System;
using System.ComponentModel.DataAnnotations;
using Oqtane.Models;

namespace Syncfusion.Helpdesk.Models
{
    public class SyncfusionHelpDeskTicketDetails : IAuditable
    {
        [Key]
        public int HelpDeskTicketDetailId { get; set; }
        public int HelpDeskTicketId { get; set; }
        public DateTime TicketDetailDate { get; set; }
        public string TicketDescription { get; set; }
        public string CreatedBy { get; set; }
        public DateTime CreatedOn { get; set; }
        public string ModifiedBy { get; set; }
        public DateTime ModifiedOn { get; set; }

        public virtual SyncfusionHelpDeskTickets HelpDeskTicket { get; set; }
    }
}
```



Note: The class in the previous code implements the Oqtane *IAuditable* interface. Therefore, the Oqtane framework will automatically populate the *CreatedBy*, *CreatedOn*, *ModifiedBy*, and *ModifiedOn* fields.

SyncfusionHelpDeskTickets.cs

Code Listing 18: SyncfusionHelpDeskTickets.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using Oqtane.Models;

namespace Syncfusion.Helpdesk.Models
{
    public class SyncfusionHelpDeskTickets : IAuditable
    {
        public SyncfusionHelpDeskTickets()
        {

```

```

        SyncfusionHelpDeskTicketDetails =
            new HashSet<SyncfusionHelpDeskTicketDetails>();
    }

    [Key]
    public int HelpDeskTicketId { get; set; }
    public int ModuleId { get; set; }

    [Required]
    public string TicketStatus { get; set; }

    [Required]
    public DateTime TicketDate { get; set; }

    [Required]
    [StringLength(50, MinimumLength = 2,
        ErrorMessage =
            "Description must be a minimum of 2 and " +
            "maximum of 50 characters.")]
    public string TicketDescription { get; set; }

    public string CreatedBy { get; set; }
    public DateTime CreatedOn { get; set; }
    public string ModifiedBy { get; set; }
    public DateTime ModifiedOn { get; set; }

    public virtual ICollection<SyncfusionHelpDeskTicketDetails>
        SyncfusionHelpDeskTicketDetails { get; set; }
    }
}

```



Note: *Blazor provides an `EditForm` control (that will be implemented later in client code) that allows us to validate a form using data annotations. These data annotations are contained in the `SyncfusionHelpDeskTickets` class.*

Repository code

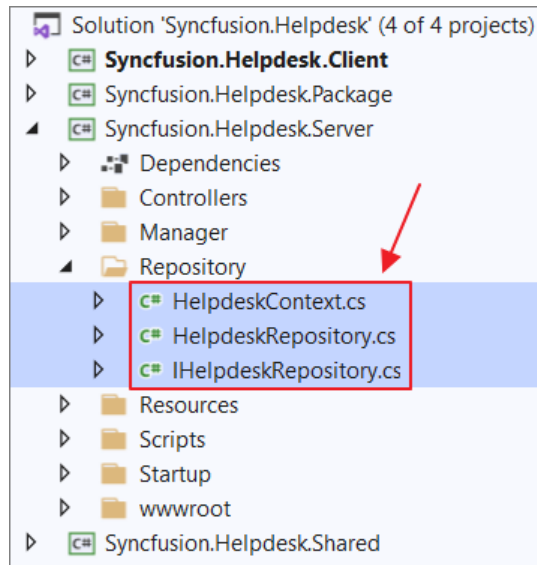


Figure 63: Repository Code

The repository code directly communicates with the database.

Update the files with the following code.

HelpdeskContext.cs

Oqtane uses Microsoft Entity Framework for database access. The **HelpdeskContext** class defines our custom tables and allows Entity Framework to connect to them in the database.

To add the new tables we have created, update this class to the following code.

Code Listing 19: HelpdeskContext.cs

```
using Microsoft.EntityFrameworkCore;
using Microsoft.AspNetCore.Http;
using Oqtane.Modules;
using Oqtane.Repository;
using Syncfusion.Helpdesk.Models;

namespace Syncfusion.Helpdesk.Repository
{
    public class HelpdeskContext : DbContextBase, IService
    {
        public virtual
            DbSet<Models.SyncfusionHelpDeskTickets>
            SyncfusionHelpDeskTickets
        { get; set; }
    }
}
```

```

        public virtual
            DbSet<SyncfusionHelpDeskTicketDetails>
            SyncfusionHelpDeskTicketDetails
        { get; set; }

        public HelpdeskContext(
            ITenantResolver tenantResolver, IHttpContextAccessor accessor)
        :
            base(tenantResolver, accessor)
        {
            // ContextBase handles multitenant database connections.
        }
    }
}

```

IHelpdeskRepository.cs

We use an interface called **IHelpdeskRepository** to define the repository methods that will perform specific operations on the tables in the database (**GetSyncfusionHelpDeskTickets**, **GetSyncfusionHelpDeskTicket**, **AddSyncfusionHelpDeskTickets**, **UpdateSyncfusionHelpDeskTickets**, and **DeleteSyncfusionHelpDeskTickets**).

Update this class to the following code.

Code Listing 20: IHelpdeskRepository.cs

```

using System.Collections.Generic;
using System.Linq;
using Syncfusion.Helpdesk.Models;

namespace Syncfusion.Helpdesk.Repository
{
    public interface IHelpdeskRepository
    {
        IQueryable<Models.SyncfusionHelpDeskTickets>
            GetSyncfusionHelpDeskTickets(int ModuleId);

        Models.SyncfusionHelpDeskTickets
            GetSyncfusionHelpDeskTicket(int Id);

        Models.SyncfusionHelpDeskTickets
            AddSyncfusionHelpDeskTickets(
                Models.SyncfusionHelpDeskTickets SyncfusionHelpDeskTicket);

        Models.SyncfusionHelpDeskTickets
            UpdateSyncfusionHelpDeskTickets(

```

```

        string UpdateMode,
        Models.SyncfusionHelpDeskTickets SyncfusionHelpDeskTicket);

    void DeleteSyncfusionHelpDeskTickets(int Id);
}

```

HelpdeskRepository.cs

Finally, we implement the repository methods specified in the **IHelpdeskRepository** interface. Replace all the current code with the following code.

Code Listing 21: HelpdeskRepository.cs

```

using Microsoft.EntityFrameworkCore;
using System.Linq;
using System.Collections.Generic;
using Oqtane.Modules;
using Syncfusion.Helpdesk.Models;
using System;

namespace Syncfusion.Helpdesk.Repository
{
    public class HelpdeskRepository : IHelpdeskRepository, IService
    {
        private readonly HelpdeskContext _db;

        public HelpdeskRepository(HelpdeskContext context)
        {
            _db = context;
        }
    }
}

```

Add the following method to return all the help desk tickets associated with a module instance.

Code Listing 22: GetSyncfusionHelpDeskTickets

```

public IQueryable<Models.SyncfusionHelpDeskTickets>
    GetSyncfusionHelpDeskTickets(int ModuleId)
{
    return _db.SyncfusionHelpDeskTickets.Where(
        item => item.ModuleId == ModuleId);
}

```

Add the following method to return a single help desk ticket and all its associated help desk ticket details.

Code Listing 23: GetSyncfusionHelpDeskTicket

```
public Models.SyncfusionHelpDeskTickets
GetSyncfusionHelpDeskTicket(int Id)
{
    var HelpDeskTicket = _db.SyncfusionHelpDeskTickets
        .Where(item => item.HelpDeskTicketId == Id)
        .Include(x =>
x.SyncfusionHelpDeskTicketDetails).FirstOrDefault();

    // Strip out HelpDeskTicket from
SyncfusionHelpDeskTicketDetails
    // to avoid trying to return self-referencing object.

    var objDeskTicket = new SyncfusionHelpDeskTickets();

    objDeskTicket.HelpDeskTicketId =
HelpDeskTicket.HelpDeskTicketId;
    objDeskTicket.ModuleId = HelpDeskTicket.ModuleId;
    objDeskTicket.TicketDate = HelpDeskTicket.TicketDate;
    objDeskTicket.TicketDescription =
HelpDeskTicket.TicketDescription;
    objDeskTicket.TicketStatus = HelpDeskTicket.TicketStatus;
    objDeskTicket.CreatedBy = HelpDeskTicket.CreatedBy;
    objDeskTicket.CreatedOn = HelpDeskTicket.CreatedOn;
    objDeskTicket.ModifiedBy = HelpDeskTicket.ModifiedBy;
    objDeskTicket.ModifiedOn = HelpDeskTicket.ModifiedOn;

    objDeskTicket.SyncfusionHelpDeskTicketDetails =
        new List<SyncfusionHelpDeskTicketDetails>();

    foreach (var item in
HelpDeskTicket.SyncfusionHelpDeskTicketDetails)
    {
        item.HelpDeskTicket = null;
        objDeskTicket.SyncfusionHelpDeskTicketDetails.Add(item);
    }

    return objDeskTicket;
}
```

Add the following method to add a new help desk ticket.

Code Listing 24: AddSyncfusionHelpDeskTickets

```
public Models.SyncfusionHelpDeskTickets
AddSyncfusionHelpDeskTickets
    (Models.SyncfusionHelpDeskTickets SyncfusionHelpDeskTicket)
{
    _db.SyncfusionHelpDeskTickets.Add(SyncfusionHelpDeskTicket);
    _db.SaveChanges();
    return SyncfusionHelpDeskTicket;
}
```

Add the following method to the module to update an existing help desk ticket (and all its associated help desk ticket details).

Code Listing 25: UpdateSyncfusionHelpDeskTickets

```
public Models.SyncfusionHelpDeskTickets
UpdateSyncfusionHelpDeskTickets(
    string UpdateMode,
    Models.SyncfusionHelpDeskTickets UpdatedSyncfusionHelpDeskTickets)
{
    // Get the existing record.
    var ExistingTicket =
        _db.SyncfusionHelpDeskTickets
        .Where(x => x.HelpDeskTicketId ==
        UpdatedSyncfusionHelpDeskTickets.HelpDeskTicketId)
        .FirstOrDefault();

    if (ExistingTicket != null)
    {
        if (UpdateMode == "Admin")
        {
            // Only Admin can update these fields.

            ExistingTicket.TicketDate =
                UpdatedSyncfusionHelpDeskTickets.TicketDate;

            ExistingTicket.TicketDescription =
                UpdatedSyncfusionHelpDeskTickets.TicketDescription;
        }

        ExistingTicket.TicketStatus =
            UpdatedSyncfusionHelpDeskTickets.TicketStatus;

        // Insert any new TicketDetails.

        if (UpdatedSyncfusionHelpDeskTickets
            .SyncfusionHelpDeskTicketDetails
            != null)
    }
```

```

        {
            foreach (var item in
                UpdatedSyncfusionHelpDeskTickets
                .SyncfusionHelpDeskTicketDetails)
            {
                if (item.HelpDeskTicketDetailId == 0)
                {
                    // Create New HelpDeskTicketDetails record.
                    SyncfusionHelpDeskTicketDetails
newHelpDeskTicketDetails =
                        new SyncfusionHelpDeskTicketDetails();

                    newHelpDeskTicketDetails.HelpDeskTicketId =
UpdatedSyncfusionHelpDeskTickets.HelpDeskTicketId;

                    newHelpDeskTicketDetails.TicketDetailDate =
                        DateTime.Now;

                    newHelpDeskTicketDetails.TicketDescription =
                        item.TicketDescription;

                    _db.SyncfusionHelpDeskTicketDetails
                        .Add(newHelpDeskTicketDetails);

                    _db.SaveChanges();
                }
            }

            _db.Entry(ExistingTicket).State = EntityState.Modified;
            _db.SaveChanges();
        }

        return ExistingTicket;
    }
}

```

Finally, add the following method to delete a help desk ticket.

Code Listing 26: DeleteSyncfusionHelpDeskTickets

```

public void DeleteSyncfusionHelpDeskTickets(int Id)
{
    Models.SyncfusionHelpDeskTickets SyncfusionHelpDeskTicket =
        _db.SyncfusionHelpDeskTickets.Find(Id);

    _db.SyncfusionHelpDeskTickets.Remove(SyncfusionHelpDeskTicket);
    _db.SaveChanges();
}

```

```
}
```

Controller code

We will now create the server-side controller code that will provide the data access methods. This controller will be called by code contained in the **Services** folder of the Client project.

Code Listing 27: HelpDeskController.cs

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Authorization;
using System.Collections.Generic;
using Microsoft.AspNetCore.Http;
using Oqtane.Shared;
using Oqtane.Enums;
using Oqtane.Infrastructure;
using Syncfusion.Helpdesk.Models;
using Syncfusion.Helpdesk.Repository;
using System.Linq;
using System.Threading.Tasks;
using Oqtane.Repository;

namespace Syncfusion.Helpdesk.Controllers
{
    [Route(ControllerRoutes.Default)]
    public class HelpdeskController : Controller
    {
        private readonly IHelpdeskRepository _HelpDeskRepository;
        private readonly IUserRepository _users;
        private readonly ILoggerManager _logger;
        protected int _entityId = -1;

        public HelpdeskController(
            IHelpdeskRepository HelpDeskRepository,
            IUserRepository users,
            ILoggerManager logger,
            IHttpContextAccessor accessor)
        {
            _HelpDeskRepository = HelpDeskRepository;
            _users = users;
            _logger = logger;

            if (accessor.HttpContext.Request.Query.ContainsKey("entityid"))
            {
                _entityId = int.Parse(
                    accessor.HttpContext.Request.Query["entityid"]);
            }
        }
    }
}
```

```

        }
    }
}

```

Next, we will add methods to the class. Each method is decorated with an **Authorize** tag that specifies an *access policy*. This will ensure the method can be called only by users who have the specified permission.

At the time of publication, the available options are: **ViewPage**, **EditPage**, **ViewModule**, **EditModule**, **ViewFolder**, **EditFolder**, and **ListFolder**.

Add the following code to implement the method that will allow a user to access their help desk tickets.

Code Listing 28: Get List of SyncfusionHelpDeskTickets

```

// A non-Administrator can only query their Tickets.
// GET: api/<controller>?username=x&entityid=y
[HttpGet]
[Authorize(Policy = PolicyNames.ViewModule)]
public IEnumerable<SyncfusionHelpDeskTickets> Get(
    string username, string entityid)
{
    // Get User
    var User = _users.GetUser(this.User.Identity.Name);

    if (User.Username.ToLower() != username.ToLower())
    {
        return null;
    }

    var HelpDeskTickets =
        _HelpDeskRepository.GetSyncfusionHelpDeskTickets(
            int.Parse(entityid))
            .Where(x => x.CreatedBy == username)
            .OrderBy(x => x.HelpDeskTicketId)
            .ToList();

    return HelpDeskTickets;
}

```

Add the following method that will allow a user to retrieve the complete details of a single help desk ticket.

Code Listing 29: Get SyncfusionHelpDeskTickets

```
// A non-Administrator can only get a Ticket they created.
// GET: api/<controller>/1?username=y&entityid=z
[HttpGet("{HelpDeskTicketId}")]
[Authorize(Policy = PolicyNames.ViewModule)]
public SyncfusionHelpDeskTickets Get(
    string HelpDeskTicketId, string username, string entityid)
{
    // Get User
    var User = _users.GetUser(this.User.Identity.Name);

    if (User.Username.ToLower() != username.ToLower())
    {
        return null;
    }

    var HelpDeskTicket =
        _HelpDeskRepository.GetSyncfusionHelpDeskTicket(
            int.Parse(HelpDeskTicketId));

    if (HelpDeskTicket.CreatedBy != User.Username)
    {
        return null;
    }

    return HelpDeskTicket;
}
```

Add the following method that will allow a user to create a new help desk ticket.

Code Listing 30: Post SyncfusionHelpDeskTickets

```
// All users can Post.
// POST api/<controller>
[HttpPost]
[Authorize(Policy = PolicyNames.ViewModule)]
public Task Post(
    [FromBody] Models.SyncfusionHelpDeskTickets
SyncfusionHelpDeskTickets)
{
    if (ModelState.IsValid
        && SyncfusionHelpDeskTickets.ModuleId == _entityId)
    {
        // Add a new Help Desk Ticket.
        SyncfusionHelpDeskTickets =
            _HelpDeskRepository.AddSyncfusionHelpDeskTickets(
                SyncfusionHelpDeskTickets);
    }
}
```

```

        _logger.Log(LogLevel.Information, this, LogFunction.Create,
            "HelpDesk Added {newSyncfusionHelpDeskTickets}",
            SyncfusionHelpDeskTickets);
    }

    return Task.FromResult(SyncfusionHelpDeskTickets);
}

```

Add the following method that will allow a user to update an existing help desk ticket.

Code Listing 31: Update SyncfusionHelpDeskTickets

```

// Only users who created Ticket
// can call this method to update Ticket.
// POST api/<controller>/1
[HttpPost("{Id}")]
[Authorize(Policy = PolicyNames.ViewModule)]
public void Post(
    int Id,
    [FromBody] Models.SyncfusionHelpDeskTickets
UpdateSyncfusionHelpDeskTicket)
{
    // Get User
    var User = _users.GetUser(this.User.Identity.Name);

    if (User != null)
    {
        // Ensure logged in user is the creator.
        if (UpdateSyncfusionHelpDeskTicket.CreatedBy.ToLower() ==
User.Username.ToLower())
        {
            // Update Ticket
            _HelpDeskRepository.UpdateSyncfusionHelpDeskTickets(
                "User",
                UpdateSyncfusionHelpDeskTicket);
        }
    }
}

```

Import and export support

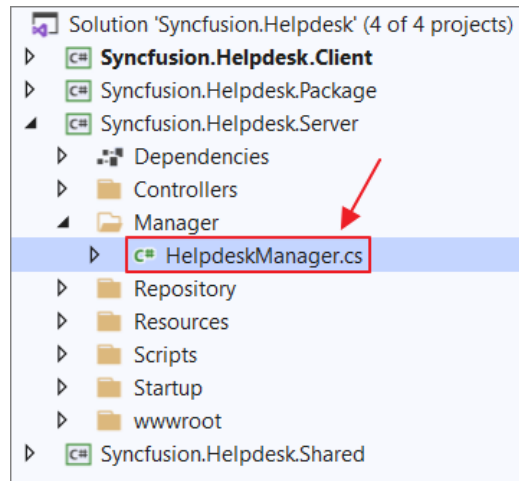


Figure 64: HelpdeskManager.cs

Finally, we will update the code in `HelpdeskManager.cs`, a class that allows a user to import and export module content.

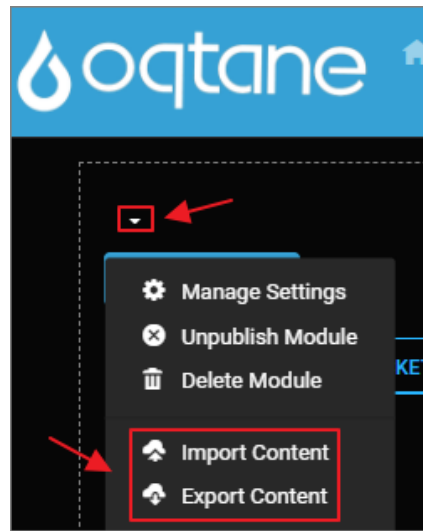


Figure 65: Import and Export Content

When the module is complete and working again, if we log into the Oqtane site as the host account and navigate to the page containing the module, we can click the **Edit** pencil icon to enter edit mode. This allows us to select the context menu for the module and access the **Import Content** and **Export Content** options.

To enable this functionality, we create `import` and `export` methods in a class that implements the `IPortable` interface.

Replace all the code of the existing `HelpdeskManager.cs` file with the following code.

```

using System.Collections.Generic;
using System.Linq;
using System.Text.Json;
using Oqtane.Modules;
using Oqtane.Models;
using Oqtane.Infrastructure;
using Oqtane.Repository;
using Syncfusion.Helpdesk.Models;
using Syncfusion.Helpdesk.Repository;

namespace Syncfusion.Helpdesk.Manager
{
    public class HelpdeskManager : IInstallable, IPortable
    {
        private IHelpdeskRepository _HelpDeskRepository;
        private ISqlRepository _sql;

        public HelpdeskManager(
            IHelpdeskRepository HelpDeskRepository,
            ISqlRepository sql)
        {
            _HelpDeskRepository = HelpDeskRepository;
            _sql = sql;
        }

        public bool Install(Tenant tenant, string version)
        {
            return _sql.ExecuteScript(
                tenant,
                GetType().Assembly,
                "Syncfusion.Helpdesk." + version + ".sql");
        }

        public bool Uninstall(Tenant tenant)
        {
            return _sql.ExecuteScript(
                tenant,
                GetType().Assembly,
                "Syncfusion.Helpdesk.Uninstall.sql");
        }
    }
}

```



Note: This code also implements the *Install* and *Uninstall* methods of the *IInstallable* interface that is called when the module is installed and uninstalled in *Oqtane*.

To implement the code that will be called when the export functionality is invoked, add the following method to the class.

Code Listing 33: Export Module

```
public string ExportModule(Module module)
{
    string content = "";

    List<Models.SyncfusionHelpDeskTickets> HelpDesks =
        new List<SyncfusionHelpDeskTickets>();

    var AllTickets = _HelpDeskRepository
        .GetSyncfusionHelpDeskTickets(module.ModuleId).ToList();

    foreach (var Ticket in AllTickets)
    {
        var HelpDeskTicket = _HelpDeskRepository
            .GetSyncfusionHelpDeskTicket(Ticket.HelpDeskTicketId);

        HelpDesks.Add(HelpDeskTicket);
    }

    if (HelpDesks != null)
    {
        content = JsonSerializer.Serialize(HelpDesks);
    }
    return content;
}
```

To implement the code that will be called when the import functionality is invoked, add the following method to the class.

Code Listing 34: Import Module

```
public void ImportModule(Module module, string content, string
version)
{
    List<Models.SyncfusionHelpDeskTickets> HelpDesks = null;
    if (!string.IsNullOrEmpty(content))
    {
        HelpDesks =
```

```

        JsonSerializer
    .Deserialize<List<Models.SyncfusionHelpDeskTickets>>(content);
    }
    if (HelpDesks != null)
    {
        foreach (var HelpDesk in HelpDesks)
        {
            Models.SyncfusionHelpDeskTickets NewHelpDesk =
                new SyncfusionHelpDeskTickets();

            NewHelpDesk.ModuleId = module.ModuleId;
            NewHelpDesk.TicketDate = HelpDesk.TicketDate;
            NewHelpDesk.TicketStatus = HelpDesk.TicketStatus;
            NewHelpDesk.TicketDescription =
HelpDesk.TicketDescription;

            NewHelpDesk.SyncfusionHelpDeskTicketDetails =
                new List<SyncfusionHelpDeskTicketDetails>();

            foreach (var TicketDetail in
                HelpDesk.SyncfusionHelpDeskTicketDetails)
            {
                SyncfusionHelpDeskTicketDetails NewDetail =
                    new SyncfusionHelpDeskTicketDetails();

                NewDetail.TicketDetailDate =
TicketDetail.TicketDetailDate;
                NewDetail.ModifiedBy = TicketDetail.ModifiedBy;
                NewDetail.ModifiedOn = TicketDetail.ModifiedOn;
                NewDetail.TicketDescription =
TicketDetail.TicketDescription;

                NewHelpDesk.SyncfusionHelpDeskTicketDetails.Add(NewDetail);
            }

            _HelpDeskRepository.AddSyncfusionHelpDeskTickets(NewHelpDesk);
        }
    }
}

```

Next, build the solution. The solution should build without errors.

Chapter 7 Creating Help Desk Tickets

In this chapter, we will create a page that will allow a user to create new help desk tickets. We will also allow a user to add additional details to an existing help desk ticket.

Services layer

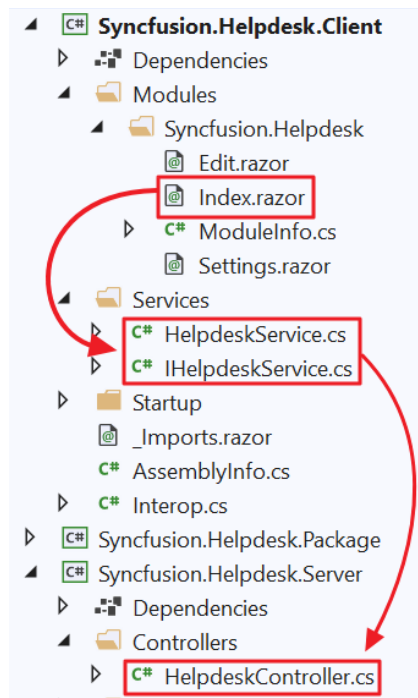


Figure 66: Services Layer

The code we will create will communicate with code in the Server project through the *services layer*. We will implement this layer by modifying the code in the **Services** folder in the **Client** project.

The first step is to define the interface for our service methods. Replace all the code in the **IHelpdeskService.cs** file with the following code.

Code Listing 35: IHelpdeskService.cs

```
using System.Collections.Generic;
using System.Threading.Tasks;
using Syncfusion.Helpdesk.Models;

namespace Syncfusion.Helpdesk.Services
{
```

```

public interface IHelpdeskService
{
    Task<List<SyncfusionHelpDeskTickets>>
        GetSyncfusionHelpDeskTicketsByUserAsync(
            int ModuleId, string username);

    Task<SyncfusionHelpDeskTickets>
        GetSyncfusionHelpDeskTicketByUserAsync(
            int HelpDeskTicketId, int ModuleId, string username);

    Task<SyncfusionHelpDeskTickets>
        AddSyncfusionHelpDeskTicketsAsync(
            SyncfusionHelpDeskTickets SyncfusionHelpDeskTickets);

    Task<SyncfusionHelpDeskTickets>
        UpdateSyncfusionHelpDeskTicketsAsync(
            SyncfusionHelpDeskTickets SyncfusionHelpDeskTickets);
}
}

```

We will now implement the service methods defined in the interface we just created. Replace all the code in the **HelpdeskService.cs** file with the following code.

Code Listing 36: HelpdeskService.cs

```

using System.Collections.Generic;
using System.Linq;
using System.Net.Http;
using System.Threading.Tasks;
using Oqtane.Modules;
using Oqtane.Services;
using Oqtane.Shared;
using Syncfusion.Helpdesk.Models;

namespace Syncfusion.Helpdesk.Services
{
    public class HelpdeskService :
        ServiceBase, IHelpdeskService, IService
    {
        private readonly SiteState _siteState;

        public HelpdeskService(
            HttpClient http, SiteState siteState) : base(http)
        {
            _siteState = siteState;
        }
    }
}

```



```

        private string Apiurl => CreateApiUrl(_siteState.Alias,
        "Helpdesk");
    }
}

```

Add the following method to allow help desk tickets related to the current **Module** instance to be retrieved. Notice that this method, and all other methods we will add, use the Oqtane framework helper method, **CreateAuthorizationPolicyUrl**, that properly formats the request to add the **ModuleId**.

Oqtane security will not allow the server-side method to be invoked if the user does not have proper security, as defined in the **Authorize** policy that decorates the method, for the current module instance.

Code Listing 37: GetSyncfusionHelpDeskTicketsByUserAsync

```

public async Task<List<SyncfusionHelpDeskTickets>>
    GetSyncfusionHelpDeskTicketsByUserAsync(
        int ModuleId, string username)
{
    return await GetJsonAsync<List<SyncfusionHelpDeskTickets>>(
        CreateAuthorizationPolicyUrl(
            $"{Apiurl}?username={username}", ModuleId));
}

```

Add the following method to allow a help desk ticket to be retrieved.

Code Listing 38: GetSyncfusionHelpDeskTicketByUserAsync

```

public async Task<SyncfusionHelpDeskTickets>
    GetSyncfusionHelpDeskTicketByUserAsync(
        int HelpDeskTicketId, int ModuleId, string username)
{
    return await GetJsonAsync<SyncfusionHelpDeskTickets>(
        CreateAuthorizationPolicyUrl(
            $"{Apiurl}/{HelpDeskTicketId}?&username={username}",
ModuleId));
}

```

Add the code for the following method to allow a help desk ticket to be created.

Code Listing 39: AddSyncfusionHelpDeskTicketsAsync

```

public async Task<SyncfusionHelpDeskTickets>
    AddSyncfusionHelpDeskTicketsAsync(
        SyncfusionHelpDeskTickets SyncfusionHelpDeskTickets)

```

```

{
    return await PostJsonAsync<SyncfusionHelpDeskTickets>(
        CreateAuthorizationPolicyUrl($"{Apiurl}",
        SyncfusionHelpDeskTickets.ModuleId),
        SyncfusionHelpDeskTickets);
}

```

The next method we implement will allow a help desk ticket to be updated.

Code Listing 40: UpdateSyncfusionHelpDeskTicketsAsync

```

public async Task<SyncfusionHelpDeskTickets>
    UpdateSyncfusionHelpDeskTicketsAsync(
        SyncfusionHelpDeskTickets objSyncfusionHelpDeskTicket)
{
    return await PostJsonAsync(
        CreateAuthorizationPolicyUrl(
            $"{Apiurl}/{objSyncfusionHelpDeskTicket.HelpDeskTicketId}",
            objSyncfusionHelpDeskTicket.ModuleId),
        objSyncfusionHelpDeskTicket);
}

```

Forms and validation

Blazor provides a method for you to create forms with validation to collect data.

Forms

Blazor provides an **EditForm** control that allows us to validate a form using data annotations. These data annotations are defined in the **SyncfusionHelpDeskTickets** class (in the **Shared** project) and will be specified in the **Model** property of the **EditForm** control (to be implemented in code covered later).

Validation

The **EditForm** control defines a method to handle **OnValidSubmit**. This method is triggered only when the data in the form satisfies all the validation rules defined by the data annotations. Any validation errors are displayed using the **DataAnnotationsValidator** and the **ValidationSummary** control.

Syncfusion Blazor controls

We will also employ the following Syncfusion controls in our form:

- **SfTextBox**: Allows us to gather data from the user and later display existing data.
- **SfDatePicker**: Allows the user to enter a date value and later display an existing date value.
- **SfComboBox**: Presents a list of predefined values (in our example, a list of ticket status values) that allows the user to choose a single value. Later, we will use this control to also display an existing selected value.
- **SfGrid**: Displays data in a tabular form. Allows sorting, reordering of columns, and the ability to trigger edits of data rows. It will be used to display existing help desk tickets.
- **SfTab**: Allows functionality to be divided into individual tabs. This control will contain the form to create tickets on one tab, and the **SfGrid** to display existing tickets on another tab.
- **SfDialog**: The Dialog control is used to display information and accept user input. It can be displayed as a modal control that requires the user to interact with it before continuing to use any other part of the application. It will be used to display the form to edit a help desk ticket and to display confirmation boxes.



Note: For full details on using the Syncfusion controls, see the [Syncfusion Blazor documentation](#).

EditTicket control

We will construct an **EditTicket** control that will be placed inside a dialog and displayed when a user wants to edit a help desk ticket (to add additional details). We do this to allow this control to be reused in the **Edit.razor** page (covered in the following chapter).

In the **Client** project, create a new control called **EditTicket.razor** using the following code.

Note that the control takes a **SelectedTicket** parameter. The values updated or added in the **EditTicket.razor** control will be passed back to the parent Razor control that invokes it. That control will save the values.

Code Listing 41: EditTicket.razor

```
@using System.Security.Claims;
@using Syncfusion.Blazor.DropDowns
@using Syncfusion.Helpdesk.Models

@code {
    [Parameter]
    public SyncfusionHelpDeskTickets SelectedTicket { get; set; }

    [Parameter]
    public bool isAdmin { get; set; }
```

```
string NewHelpDeskTicketDetailText = "";
```

```
}
```

Add the following code to the markup section of the **EditTicket** control.

The **EditTicket** control allows the user to select the **Ticket Status** from a dropdown menu and the **Ticket Date** from a calendar control, as well as to edit the **Ticket Description** using a text box control.

Also note that the **Ticket Date** and **Ticket Description** are only enabled if the **isAdmin** flag is set.

Code Listing 42: EditTicket Markup

```
<div>
    <SfComboBox TValue="string"
        TItem="SyncfusionHelpDeskStatus"
        PopupHeight="230px"
        Placeholder="Ticket Status"
        FloatLabelType="@FloatLabelType.Always"
        @bind-Value="@SelectedTicket.TicketStatus"
        DataSource="@SyncfusionHelpDeskStatus.Statuses">
        <ComboBoxEvents TValue="string"
            TItem="SyncfusionHelpDeskStatus">
        </ComboBoxEvents>
        <ComboBoxFieldSettings Text="Text" Value="ID">
        </ComboBoxFieldSettings>
    </SfComboBox>
</div>
<div>
    <SfDatePicker ID="TicketDate"
        Enabled="isAdmin"
        Placeholder="Ticket Date"
        FloatLabelType="@FloatLabelType.Always"
        @bind-Value="@SelectedTicket.TicketDate"
        Max="DateTime.Now"
        ShowClearButton="false">
    </SfDatePicker>
</div>
<div>
    <SfTextBox Enabled="isAdmin"
        Placeholder="Ticket Description"
        FloatLabelType="@FloatLabelType.Always"
        @bind-Value="@SelectedTicket.TicketDescription">
    </SfTextBox>
</div>
```

```
</div>
```

The **EditTicket** control allows the user to add ticket details.

Add the following code to the markup to support this functionality.

Code Listing 43: EditTicket Details

```
@if (SelectedTicket.SyncfusionHelpDeskTicketDetails != null)
{
    @if (SelectedTicket.SyncfusionHelpDeskTicketDetails.Count() > 0)
    {
        <table class="table">
            <thead>
                <tr>
                    <th>Date</th>
                    <th>Description</th>
                </tr>
            </thead>
            <tbody>
                @foreach (var TicketDetail in
                    SelectedTicket.SyncfusionHelpDeskTicketDetails)
                {
                    <tr>
                        <td>
                            @TicketDetail.TicketDetailDate.ToShortDateString()
                        </td>
                        <td>
                            @TicketDetail.TicketDescription
                        </td>
                    </tr>
                }
            </tbody>
        </table>
    }
    <SfTextBox Placeholder="NewHelp Desk Ticket Detail"
                @bind-Value="@NewHelpDeskTicketDetailText">
    </SfTextBox>
    <SfButton CssClass="e-small e-success"
                @onclick="AddHelpDeskTicketDetail">
        Add
    </SfButton>
}
}
```

Finally, add the following method to the **@code** section of the control to add any details entered to the **SyncfusionHelpDeskTicketDetails** collection.

Code Listing 44: AddHelpDeskTicketDetail

```
private void AddHelpDeskTicketDetail()
{
    // Create New HelpDeskTicketDetails record.
    SyncfusionHelpDeskTicketDetails NewHelpDeskTicketDetail =
    new SyncfusionHelpDeskTicketDetails();

    NewHelpDeskTicketDetail.HelpDeskTicketId =
    SelectedTicket.HelpDeskTicketId;

    NewHelpDeskTicketDetail.TicketDetailDate =
    DateTime.Now;

    NewHelpDeskTicketDetail.TicketDescription =
    NewHelpDeskTicketDetailText;

    // Add to collection.
    SelectedTicket.SyncfusionHelpDeskTicketDetails
        .Add(NewHelpDeskTicketDetail);

    // Clear the Text Box.
    NewHelpDeskTicketDetailText = "";
}
```

Index control

The **Index** control is the first control that will load when a user accesses the module.

Replace all the code in the **Index.razor** control with the following code that adds the required **import** and **using** statements, properties and variables, and the markup that will display the tab control, with **New Help Desk Ticket** and **Existing Tickets** tabs.

Code Listing 45: Index.razor

```
@using Syncfusion.Helpdesk.Services
@using Syncfusion.Helpdesk.Models
@using Syncfusion.Blazor.Buttons
@using Microsoft.AspNetCore.Components.Authorization
@using Syncfusion.Helpdesk.Client.Modules.Syncfusion_Helpdesk

@namespace Syncfusion.Helpdesk
@inherits ModuleBase
@inject AuthenticationStateProvider AuthenticationStateProvider
@inject IHelpdeskService HelpDeskService
@inject NavigationManager NavigationManager
```

```

<br />
<br />
@if (isAuthenticated)
{
    <div id="target" style="height: 500px;"
        class="col-lg-12 control-section">
        <div class="e-sample-resize-container">
            <SfTab CssClass="default-tab">
                <TabItems>
                    <TabItem>
                        <ChildContent>
                            <TabHeader Text="New Help Desk Ticket"
                                IconCss="far fa-edit">
                                </TabHeader>
                            </ChildContent>
                            <ContentTemplate>
                                <br /><br />
                            </ContentTemplate>
                        </TabItem>
                        <TabItem>
                            <ChildContent>
                                <TabHeader Text="Existing Tickets"
                                    IconCss="fa fa-inbox">
                                    </TabHeader>
                                </ChildContent>
                                <ContentTemplate>
                                    <br /><br />
                                </ContentTemplate>
                            </TabItem>
                        </TabItems>
                    </SfTab>
                </div>
            </div>
        }
    else
    {
        <b>You must be logged in to submit a Ticket</b>
    }
    @code {
        public override List<Resource> Resources =>
            new List<Resource>()
        {
            new Resource { ResourceType = ResourceType.Stylesheet,
                Url = ModulePath() + "Module.css" },
            new Resource { ResourceType = ResourceType.Script,
                Url = ModulePath() + "Module.js" }
        }
    }
}

```

```

};

public string Content = "Submit";

private Dictionary<string, object> submit =
    new Dictionary<string, object>() {
        { "type", "submit"}
    };

// Global property for the Help Desk Ticket.
SyncfusionHelpDeskTickets objHelpDeskTicket =
    new SyncfusionHelpDeskTickets()
{
    TicketDate = new DateTime(
        DateTime.Now.Year,
        DateTime.Now.Month,
        DateTime.Now.Day),
    TicketStatus = "New"
};

// Global property for the selected Help Desk Ticket.
private SyncfusionHelpDeskTickets SelectedTicket =
    new SyncfusionHelpDeskTickets();

SfGrid<SyncfusionHelpDeskTickets> gridObj;

public List<SyncfusionHelpDeskTickets>
    colHelpDeskTickets { get; set; }

AuthenticationState authState;
bool isAuthenticated = false;
string Status = "";

// EXISTING HELP DESK TICKETS

public bool EditDialogVisibility { get; set; } = false;
}

```

New Help Desk Ticket tab

To allow a user to create a new help desk ticket, add the following markup to the **ContentTemplate** section of the **New Help Desk Ticket** tab.

Code Listing 46: New Help Desk Ticket

```
<EditForm ID="new-form"
```



```

        Model="@objHelpDeskTicket"
        OnValidSubmit="@HandleValidSubmit">
<DataAnnotationsValidator></DataAnnotationsValidator>
<div>
    <SfComboBox TValue="string" TItem="SyncfusionHelpDeskStatus"
        PopupHeight="230px" Placeholder="Ticket Status"
        FloatLabelType="@FloatLabelType.Always"
        @bind-Value="@objHelpDeskTicket.TicketStatus"
        DataSource="@SyncfusionHelpDeskStatus.Statuses">
        <ComboBoxEvents TValue="string"
            TItem="SyncfusionHelpDeskStatus">
        </ComboBoxEvents>
        <ComboBoxFieldSettings Text="Text" Value="ID">
        </ComboBoxFieldSettings>
    </SfComboBox>
</div>
<div>
    <SfDatePicker ID="TicketDateInput" Placeholder="Ticket Date"
        FloatLabelType="@FloatLabelType.Always"
        @bind-Value="@objHelpDeskTicket.TicketDate"
        Max="DateTime.Now"
        ShowClearButton="false"></SfDatePicker>
    <ValidationMessage For="@(() => objHelpDeskTicket.TicketDate)" />
</div>
<div>
    <SfTextBox Placeholder="Ticket Description"
        FloatLabelType="@FloatLabelType.Always"
        @bind-Value="@objHelpDeskTicket.TicketDescription">
    </SfTextBox>
    <ValidationMessage For="@(() =>
objHelpDeskTicket.TicketDescription)" />
</div>
<br />
    <SfButton Content="@Content" HtmlAttributes="@submit"></SfButton>
    <br /><br /><b>@Status</b>
</EditForm>

```

When the user submits data in the form that passes validation, the **HandleValidSubmit** method is called.

Add the following to the code section to implement that method.

Code Listing 47: HandleValidSubmit

```

public async Task HandleValidSubmit(EditContext context)
{

```

```

try
{
    Status = "";

    // Save the new Help Desk Ticket.

    SyncfusionHelpDeskTickets HelpDesk =
        new SyncfusionHelpDeskTickets();

    HelpDesk.ModuleId = ModuleState.ModuleId;
    HelpDesk.TicketStatus = objHelpDeskTicket.TicketStatus;
    HelpDesk.TicketDate =
        Convert.ToDateTime(objHelpDeskTicket.TicketDate);
    HelpDesk.TicketDescription = objHelpDeskTicket.TicketDescription;
    HelpDesk.CreatedBy = authState.User.Identity.Name;
    HelpDesk.CreatedOn = DateTime.Now;
    HelpDesk.ModifiedBy = authState.User.Identity.Name;
    HelpDesk.ModifiedOn = DateTime.Now;
    HelpDesk.SyncfusionHelpDeskTicketDetails =
        new List<SyncfusionHelpDeskTicketDetails>();

    HelpDesk =
        await
        HelpDeskService.AddSyncfusionHelpDeskTicketsAsync(HelpDesk);

    await logger.LogInformation("HelpDesk Added {HelpDesk}", HelpDesk);

    Status = "Saved!";

    // Clear the form.
    ResetForm();

    // Refresh Tickets.
    await RefreshTickets();
}
catch (Exception ex)
{
    Status = ex.Message;
}
}

```

HandleValidSubmit requires a **ResetForm** method to restore the form values to their defaults and a **RefreshTickets** method that will update the **Existing Tickets** (to be implemented later).

Add the following to implement those methods.

Code Listing 48: ResetForm and RefreshTickets

```
public void ResetForm()
{
    objHelpDeskTicket = new SyncfusionHelpDeskTickets()
    {
        TicketDate = new DateTime(
            DateTime.Now.Year,
            DateTime.Now.Month,
            DateTime.Now.Day),
        TicketStatus = "New"
    };
}

public async Task RefreshTickets()
{
    var Tickets =
        await HelpDeskService.GetSyncfusionHelpDeskTicketsByUserAsync(
            ModuleState.ModuleId, authState.User.Identity.Name);

    colHelpDeskTickets =
        Tickets.OrderByDescending(x => x.HelpDeskTicketId).ToList();
}
```

Existing Tickets tab

To allow a user to view and update their existing help desk tickets, add the following markup to the **ContentTemplate** section of the **Existing Tickets** tab.

This adds the Syncfusion DataGrid control, which will display the tickets.

Code Listing 49: Existing Tickets Tab

```
<div>
    <div>
        <SfGrid ID="Grid"
            @ref="gridObj"
            DataSource="@colHelpDeskTickets"
            AllowPaging="true"
            AllowSorting="true"
            AllowResizing="true"
            AllowReordering="true">
            <GridPageSettings PageSize="5"></GridPageSettings>
            <GridEvents CommandClicked="OnCommandClicked"
                TValue="SyncfusionHelpDeskTickets">
            </GridEvents>
            <GridColumns>
```

```

        <GridColumn HeaderText="" TextAlign="TextAlign.Left"
Width="50">
            <GridCommandColumns>
                <GridColumn Type=CommandButtonType.Edit
                    ButtonOption=
                        @(new CommandButtonOptions()
                            { Content = "Edit" })">
                    </GridColumn>
                </GridCommandColumns>
            </GridColumn>
            <GridColumn IsPrimaryKey="true"

Field=@nameof(SyncfusionHelpDeskTickets.HelpDeskTicketId)
                HeaderText="ID #" TextAlign="@TextAlign.Left"
                Width="70">
            </GridColumn>
            <GridColumn
Field=@nameof(SyncfusionHelpDeskTickets.TicketStatus)
                HeaderText="Status" TextAlign="@TextAlign.Left"
                Width="80">
            </GridColumn>
            <GridColumn
Field=@nameof(SyncfusionHelpDeskTickets.TicketDate)
                HeaderText="Date" TextAlign="@TextAlign.Left"
                Format="d" Type="ColumnType.Date"
                Width="80">
            </GridColumn>
            <GridColumn
Field=@nameof(SyncfusionHelpDeskTickets.TicketDescription)
                HeaderText="Description"
                TextAlign="@TextAlign.Left"
                Width="150">
            </GridColumn>
        </GridColumns>
    </SfGrid>
</div>
</div>

```

The **Existing Tickets** data grid contains an **Edit** button that opens the selected ticket in a Syncfusion Dialog control. This control will display the selected ticket in the **EditTicket.razor** control (added earlier).

Add the following markup code to support this functionality.

Code Listing 50: Edit Ticket Dialog

```

<SfDialog Target="#target"
        Width="500px"

```

```

        Height="500px"
        IsModal="true"
        ShowCloseIcon="true"
        @bind-Visible="EditDialogVisibility">
<DialogTemplates>
    <Header> EDIT TICKET # @SelectedTicket.HelpDeskTicketId</Header>
    <Content>
        <EditTicket SelectedTicket="@SelectedTicket" isAdmin="false" />
    </Content>
    <FooterTemplate>
        <div class="button-container">
            <button type="submit"
                class="e-btn e-normal e-primary"
                @onclick="SaveTicket">
                Save
            </button>
        </div>
    </FooterTemplate>
</DialogTemplates>
</SfDialog>

```

Add the following method to the code section that opens the dialog when a help desk ticket is selected in the **Existing Tickets** data grid.

Code Listing 51: OnCommandClicked

```

public async void OnCommandClicked(
    CommandClickEventArgs<SyncfusionHelpDeskTickets> args)
{
    if (args.CommandColumn.ButtonOption.Content == "Edit")
    {
        // Get the selected Help Desk Ticket.
        var HelpDeskTicket = (SyncfusionHelpDeskTickets)args.RowData;

        SelectedTicket =
            await
            HelpDeskService.GetSyncfusionHelpDeskTicketByUserAsync(
                HelpDeskTicket.HelpDeskTicketId,
                ModuleState.ModuleId,
                authState.User.Identity.Name);

        // Open the Edit dialog.
        this.EditDialogVisibility = true;
        StateHasChanged();
    }
}

```

Add the **SaveTicket** method that will be called when a user clicks the **Submit** button on the Dialog control.

Code Listing 52: SaveTicket

```
public async Task SaveTicket()
{
    // Update the selected Help Desk Ticket.
    await
    HelpDeskService.UpdateSynCFusionHelpDeskTicketsAsync(SelectedTicket);

    // Update the Status of the Ticket in the collection.
    var TicketToUpdate =
        colHelpDeskTickets.Where(
            x => x.HelpDeskTicketId ==
                SelectedTicket.HelpDeskTicketId).FirstOrDefault();

    TicketToUpdate.TicketStatus = SelectedTicket.TicketStatus;

    // Close the Edit dialog.
    this.EditDialogVisibility = false;

    // Refresh the SfGrid
    // so the changes to the selected
    // Help Desk Ticket are reflected.
    gridObj.Refresh();
}
```

Finally, add the following method that will be invoked when the **Index.razor** control loads.

Code Listing 53: OnInitializedAsync

```
protected override async Task OnInitializedAsync()
{
    try
    {
        // Get user.
        authState =
            await
            AuthenticationStateProvider.GetAuthenticationStateAsync();

        if (authState.User.Identity.IsAuthenticated)
        {
            isAuthenticated = true;

            await RefreshTickets();
        }
    }
    catch (Exception ex)
```

```

{
    await logger.LogError(
        ex, "Error Loading HelpDesk {Error}", ex.Message);

    AddModuleMessage("Error Loading HelpDesk", MessageType.Error);
}
}

```

Remove all the code from Edit.Razor (for now)

The methods in the service layer have changes, so the code in the **Edit.Razor** control will no longer build. So we can test out the code in the **Index.razor** control, remove all the code from **Edit.razor**.

Test creating new tickets

Rebuild the **Syncfusion.Helpdesk** solution. It should build without errors.

In the instance of Visual Studio that contains the Oqtane solution, restart using the **Start Without Debugging** option.

Log in as the host account.

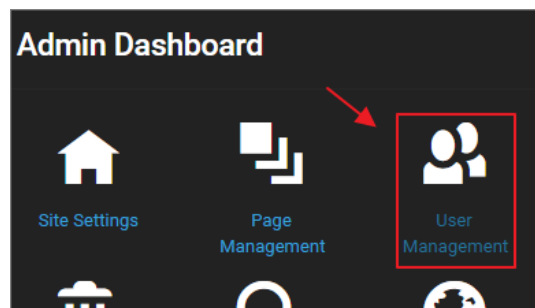
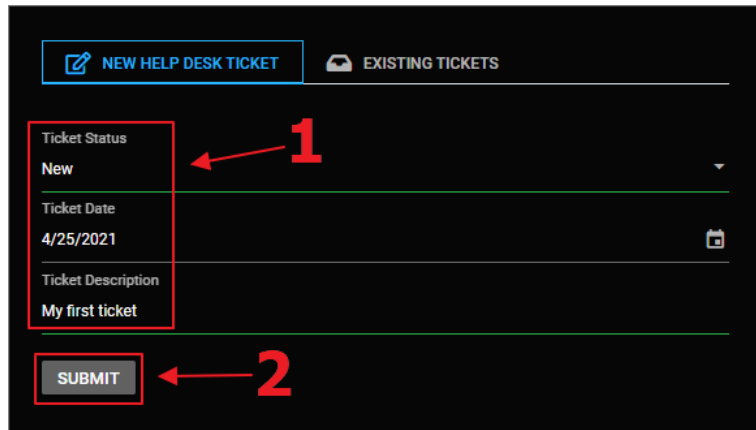


Figure 67: Create New User

Select the gear icon to open the **Control Panel** and click the button to navigate to the **Admin Dashboard**. Select **User Management > Add User** and create a new user who is not in the administrator role.

Log out and log back in as the new user.



NEW HELP DESK TICKET EXISTING TICKETS

Ticket Status
New

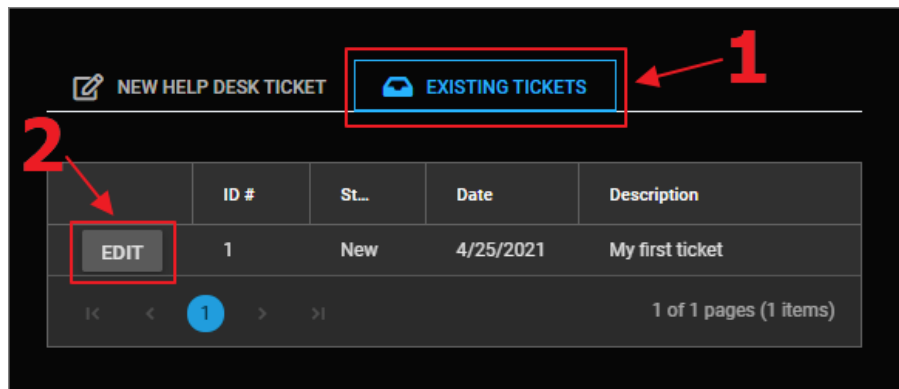
Ticket Date
4/25/2021

Ticket Description
My first ticket

SUBMIT

Figure 68: New Ticket

Enter details for a new help desk ticket and click **Submit**.



NEW HELP DESK TICKET EXISTING TICKETS

	ID #	St...	Date	Description
EDIT	1	New	4/25/2021	My first ticket

1 of 1 pages (1 items)

Figure 69: Existing Tickets

You can click the **Existing Tickets** tab to see the ticket and open it by clicking the **Edit** button next to the record in the data grid.

The screenshot shows a dark-themed modal window titled "EDIT TICKET # 1" with a close button (X) in the top right corner. The form contains the following fields and controls:

- Ticket Status:** A dropdown menu currently showing "New".
- Ticket Date:** A date field showing "4/25/2021" with a calendar icon to its right.
- Ticket Description:** A text input field containing "My first ticket".
- New item:** A text input field located below the description field.
- ADD:** A green button located below the "New item" field.
- SAVE:** A blue button located at the bottom right of the modal.

Three red annotations with arrows point to specific elements:

- 1:** Points to the "New item" text input field.
- 2:** Points to the green "ADD" button.
- 3:** Points to the blue "SAVE" button.

Figure 70: Add New Details

The ticket will open in the **Edit Ticket** control.

You can add additional details. Click **Save** to save the changes.

Chapter 8 Help Desk Ticket Administration

In this final chapter, we will create an administration page that implements an important function of the Syncfusion DataGrid: the ability to provide server-side paging and sorting.

The existing data grid (implemented on the **Existing Tickets** tab) provides paging and sorting, but all records for the module instance, for the user, are retrieved in a single request.

This would be problematic for the administration page because there could potentially be thousands of records, as the administrator can see all help desk tickets, for all users, for the module instance.

In addition, we will demonstrate how to create controls in Oqtane that are only available to users in specified (and configurable) roles.

Server project

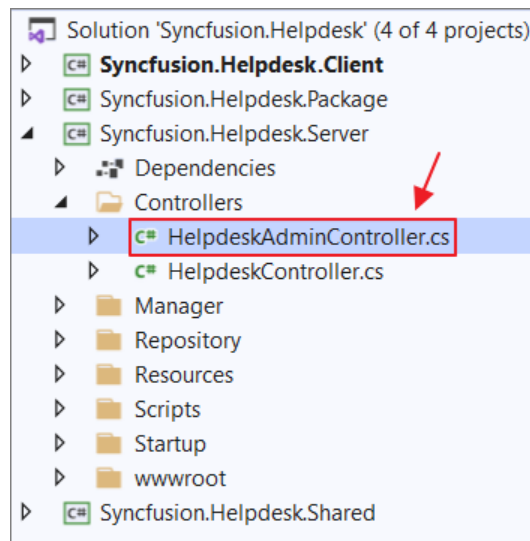


Figure 71: HelpdeskAdminController.cs

We will first create a new controller that will be paired with new services we will create in the **Client** project.

Create a new file in the **Controllers** folder of the **Server** project, called **HelpdeskAdminController.cs**, using the following code.

Code Listing 54: HelpDeskController.cs

```
using Microsoft.AspNetCore.Mvc;  
using Microsoft.AspNetCore.Authorization;
```

```

using Microsoft.Extensions.Primitives;
using System.Collections.Generic;
using Microsoft.AspNetCore.Http;
using Oqtane.Shared;
using Oqtane.Enums;
using Oqtane.Infrastructure;
using Syncfusion.Helpdesk.Models;
using Syncfusion.Helpdesk.Repository;
using System.Linq;
using System.Threading.Tasks;
using Oqtane.Repository;
using System.Linq.Expressions;
using System;

namespace Syncfusion.Helpdesk.Controllers
{
    [Route(ControllerRoutes.Default)]
    public class HelpdeskAdminController : Controller
    {
        private readonly IHelpdeskRepository _HelpDeskRepository;
        private readonly IUserRepository _users;
        private readonly ILoggerManager _logger;
        protected int _entityId = -1;

        public HelpdeskAdminController(
            IHelpdeskRepository HelpDeskRepository,
            IUserRepository users,
            ILoggerManager logger,
            IHttpContextAccessor accessor)
        {
            try
            {
                _HelpDeskRepository = HelpDeskRepository;
                _users = users;
                _logger = logger;

                if (accessor.HttpContext
                    .Request.Query.ContainsKey("entityid"))
                {
                    _entityId = int.Parse(
                        accessor.HttpContext
                            .Request.Query["entityid"]);
                }
            }
            catch (System.Exception ex)
            {
                string error = ex.Message;
            }
        }
    }
}

```

```

    }
}

```

Add the following method to the namespace, in the file (but outside of the **HelpdeskAdminController** class), that will support the sorting functionality in the method that will be implemented next.

Code Listing 55: IQueryableExtensions

```

// From: https://bit.ly/30ypMCp
public static class IQueryableExtensions
{
    public static IOOrderedQueryable<T> OrderBy<T>(
        this IQueryable<T> source, string propertyName)
    {
        return source.OrderBy(ToLambda<T>(propertyName));
    }

    public static IOOrderedQueryable<T> OrderByDescending<T>(
        this IQueryable<T> source, string propertyName)
    {
        return source.OrderByDescending(ToLambda<T>(propertyName));
    }

    private static Expression<Func<T, object>> ToLambda<T>(
        string propertyName)
    {
        var parameter = Expression.Parameter(typeof(T));
        var property = Expression.Property(parameter, propertyName);
        var propAsObject = Expression.Convert(property,
typeof(object));
        return Expression.Lambda<Func<T, object>>(propAsObject,
parameter);
    }
}

```

Add the following method that will be called by the Syncfusion DataGrid (we will add it later) and will allow paging and sorting.

Also note that because this method will be called directly by the Syncfusion DataGrid, it will not have a method to call it in the service methods we will add later to the **Client** project.

Code Listing 56: Get Method for Data Grid

```

// Only an Administrator can query all Tickets.
// GET: api/<controller>?entityid=x

```

```

[HttpGet]
[Authorize(Policy = PolicyNames.EditModule)]
public object Get(string entityid)
{
    StringValues Skip;
    StringValues Take;
    StringValues OrderBy;

    // Filter the data.
    var TotalRecordCount =
        _HelpDeskRepository.GetSyncfusionHelpDeskTickets(
            int.Parse(entityid)).Count();

    int skip = (Request.Query.TryGetValue("$skip", out Skip))
        ? Convert.ToInt32(Skip[0]) : 0;

    int top = (Request.Query.TryGetValue("$top", out Take))
        ? Convert.ToInt32(Take[0]) : TotalRecordCount;

    string orderby =
        (Request.Query.TryGetValue("$orderby", out OrderBy))
        ? OrderBy.ToString() : "TicketDate";

    // Handle OrderBy direction.
    if (orderby.EndsWith(" desc"))
    {
        orderby = orderby.Replace(" desc", "");

        return new
        {
            Items =
                _HelpDeskRepository.GetSyncfusionHelpDeskTickets(
                    int.Parse(entityid))
                    .OrderByDescending(orderby)
                    .Skip(skip)
                    .Take(top),
            Count = TotalRecordCount
        };
    }
    else
    {
        System.Reflection.PropertyInfo prop =
            typeof(SyncfusionHelpDeskTickets).GetProperty(orderby);

        return new
        {
            Items =
                _HelpDeskRepository.GetSyncfusionHelpDeskTickets(
                    int.Parse(entityid))

```

```

        .OrderBy(orderby)
        .Skip(skip)
        .Take(top),
        Count = TotalRecordCount
    };
}
}

```

Add the following code for the method that will allow the details for a single help desk ticket to be retrieved.

Code Listing 57: Get Method

```

// Only an Administrator can call this method.
// GET: api/<controller>/1?entityid=z
[HttpGet("{HelpDeskTicketId}")]
[Authorize(Policy = PolicyNames.EditModule)]
public SyncfusionHelpDeskTickets Get(
    string HelpDeskTicketId,
    string entityid)
{
    return _HelpDeskRepository.GetSyncfusionHelpDeskTicket
        (int.Parse(HelpDeskTicketId));
}

```

Add the following code for the method that will allow a help desk ticket to be updated.

Code Listing 58: Put Method

```

// Only an Administrator can update using this method.
// PUT api/<controller>/5
[HttpPut("{id}")]
[Authorize(Policy = PolicyNames.EditModule)]
public Models.SyncfusionHelpDeskTickets Put(
    int id,
    [FromBody] Models.SyncfusionHelpDeskTickets
    updatedSyncfusionHelpDeskTickets)
{
    if (ModelState.IsValid &&
        updatedSyncfusionHelpDeskTickets.ModuleId == _entityId)
    {
        updatedSyncfusionHelpDeskTickets =
            _HelpDeskRepository.UpdateSyncfusionHelpDeskTickets(
                "Admin",
                updatedSyncfusionHelpDeskTickets);

        _logger.Log(LogLevel.Information, this, LogFunction.Update,
            "HelpDesk Updated {updatedSyncfusionHelpDeskTickets}",

```

```

        updatedSyncfusionHelpDeskTickets);
    }

    return updatedSyncfusionHelpDeskTickets;
}

```

Add the following code for the method that will allow a help desk ticket to be deleted.

Code Listing 59: Delete Method

```

// DELETE api/<controller>/5
[HttpDelete("{id}")]
[Authorize(Policy = PolicyNames.EditModule)]
public void Delete(int id)
{
    Models.SyncfusionHelpDeskTickets
deletedSyncfusionHelpDeskTickets =
    _HelpDeskRepository.GetSyncfusionHelpDeskTicket(id);
    if (deletedSyncfusionHelpDeskTickets != null &&
        deletedSyncfusionHelpDeskTickets.ModuleId == _entityId)
    {
        _HelpDeskRepository.DeleteSyncfusionHelpDeskTickets(id);
        _logger.Log(LogLevel.Information, this, LogFunction.Delete,
            "HelpDesk Deleted {HelpDeskId}", id);
    }
}

```

Client project

We will now create the code in the **Client** project. We will first add additional **Services** layer code and then complete the module by updating the code in the **Edit.razor** control.

Services

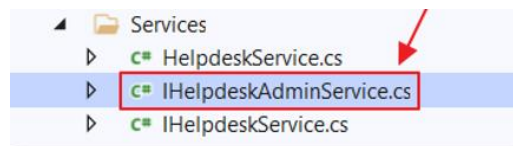


Figure 72: IHelpdeskAdminService.cs

The code we will now create will communicate with code in the **HelpdeskAdminController.cs** file in the **Server** project.

The first step is to define the interface for our new service methods. Add a new file called **IHelpdeskAdminService.cs** using the following code.

Code Listing 60: *IHelpdeskAdminService.cs*

```
using Syncfusion.Helpdesk.Models;
using System.Threading.Tasks;

namespace Syncfusion.Helpdesk.Services
{
    public interface IHelpdeskAdminService
    {
        // Admin Methods.

        Task<SyncfusionHelpDeskTickets>
            GetSyncfusionHelpDeskTicketAdminAsync(
                int HelpDeskTicketId, int ModuleId);

        Task<SyncfusionHelpDeskTickets>
            UpdateSyncfusionHelpDeskTicketsAdminAsync(
                SyncfusionHelpDeskTickets objSyncfusionHelpDeskTicket);

        Task DeleteSyncfusionHelpDeskTicketsAsync(
            int Id, int ModuleId);
    }
}
```

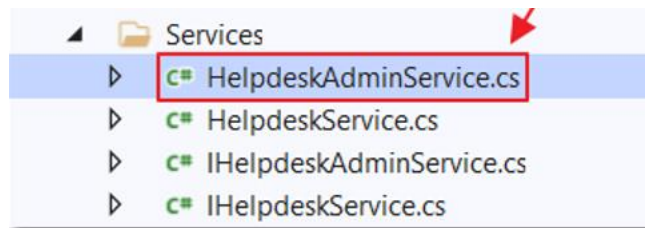


Figure 73: *HelpdeskAdminService.cs*

The next step is to implement the methods defined in the interface. Add a new file called **HelpdeskAdminService.cs** using the following code.

Code Listing 61: *HelpDeskAdminService.cs*

```
using Oqtane.Modules;
using Oqtane.Services;
using Oqtane.Shared;
using System.Net.Http;
using System.Threading.Tasks;
using Syncfusion.Helpdesk.Models;

namespace Syncfusion.Helpdesk.Services
{
    public class HelpdeskAdminService :
```



```

        ServiceBase, IHelpdeskAdminService, IService
    {
        private readonly SiteState _siteState;

        public HelpdeskAdminService(
            HttpClient http,
            SiteState siteState) : base(http)
        {
            _siteState = siteState;
        }

        private string Apiurl => CreateApiUrl(
            _siteState.Alias, "HelpdeskAdmin");

        public async Task<SyncfusionHelpDeskTickets>
            GetSyncfusionHelpDeskTicketAdminAsync(
                int HelpDeskTicketId, int ModuleId)
        {
            return await GetJsonAsync<SyncfusionHelpDeskTickets>(
                CreateAuthorizationPolicyUrl(
                    $"{Apiurl}/{HelpDeskTicketId}",
                    ModuleId));
        }

        public async Task<SyncfusionHelpDeskTickets>
            UpdateSyncfusionHelpDeskTicketsAdminAsync(
                Models.SyncfusionHelpDeskTickets objSyncfusionHelpDeskTicket)
        {
            return await PutJsonAsync<SyncfusionHelpDeskTickets>(
                CreateAuthorizationPolicyUrl(
                    $"{Apiurl}/{objSyncfusionHelpDeskTicket.HelpDeskTicketId}",
                    objSyncfusionHelpDeskTicket.ModuleId),
                objSyncfusionHelpDeskTicket);
        }

        public async Task DeleteSyncfusionHelpDeskTicketsAsync(
            int HelpDeskId, int ModuleId)
        {
            await DeleteAsync(CreateAuthorizationPolicyUrl(
                $"{Apiurl}/{HelpDeskId}", ModuleId));
        }
    }
}

```

Edit.razor control

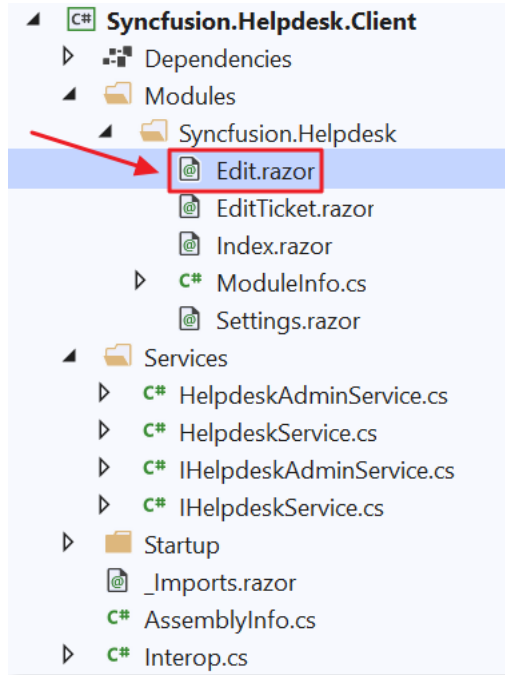


Figure 74: Edit.Razor

To complete the module, the next control to code is the **Edit.razor** control. Previously, we removed the code that was created for us automatically by the module creation wizard, because it would no longer compile when we altered the service methods.

Replace all the code with the following code.

Code Listing 62: Edit.razor

```
@using Oqtane.Modules.Controls
@using Syncfusion.Helpdesk.Services
@using Syncfusion.Helpdesk.Models
@using Syncfusion.Helpdesk.Client.Modules.Syncfusion_Helpdesk

@namespace Syncfusion.Helpdesk
@inherits ModuleBase
@inject IHelpdeskAdminService HelpdeskAdminService
@inject NavigationManager NavigationManager

@code {
    // This ensures only users with the security level Edit
    // can open this control.
    public override SecurityAccessLevel
    SecurityAccessLevel => SecurityAccessLevel.Edit;

    public override string Actions => "Add,Edit";
```

```

public override string Title => "Manage HelpDesk";

public override List<Resource> Resources => new List<Resource>()
{
    new Resource {
        ResourceType = ResourceType.Stylesheet,
        Url = ModulePath() + "Module.css" }
};

// Global property for the selected Help Desk Ticket.
private SyncfusionHelpDeskTickets SelectedTicket =
    new SyncfusionHelpDeskTickets();

SfGrid<SyncfusionHelpDeskTickets> gridObj;

public List<SyncfusionHelpDeskTickets>
    colHelpDeskTickets
{ get; set; }

string SfDataManagerURL = "";

protected override async Task OnInitializedAsync()
{
    try
    {
        SfDataManagerURL =
            $"{ModuleState.SiteId}/api/HelpdeskAdmin?" +
            $"entityid={ModuleState.ModuleId}";
    }
    catch (Exception ex)
    {
        await logger.LogError(ex, "Error Loading HelpDesk {Error}",
            ex.Message);

        AddModuleMessage("Error Loading HelpDesk", MessageType.Error);
    }
}

public bool EditDialogVisibility { get; set; } = false;

// Property to control the delete dialog.
public bool DeleteRecordConfirmVisibility { get; set; } = false;
}

```

Data grid

We will now add the Syncfusion DataGrid. This is similar to the Syncfusion DataGrid we added earlier, in the **Index.razor** control. However, this data grid has the **SfDataManager** property configured to point directly to the **HelpdeskAdminController** that we added previously. This will allow the data grid to perform server-side paging and sorting automatically.

Add the following markup for the DataGrid.

Code Listing 63: DataGrid

```
<div id="admintarget" style="height: 500px;" class="col-lg-12 control-
section">
    <SfGrid ID="Grid"
        @ref="gridObj"
        DataSource="@colHelpDeskTickets"
        AllowPaging="true"
        AllowSorting="true"
        AllowResizing="true"
        AllowReordering="true">
        <SfDataManager Url="@SfDataManagerURL "
            Adaptor="Adaptors.WebApiAdaptor">
        </SfDataManager>
        <GridPageSettings PageSize="10"></GridPageSettings>
        <GridEvents CommandClicked="OnCommandClicked"
            TValue="SyncfusionHelpDeskTickets">
        </GridEvents>
        <GridColumns>
            <GridColumn HeaderText="" TextAlign="TextAlign.Left"
Width="100">
                <GridCommandColumns>
                    <GridCommandColumn Type=CommandButtonType.Edit
                        ButtonOption="@(<new
CommandButtonOptions()
                                { Content = "Edit"
}))">
                    </GridCommandColumn>
                    <GridCommandColumn Type=CommandButtonType.Delete
                        ButtonOption="@(<new
CommandButtonOptions()
                                { Content = "Delete"
}))">
                    </GridCommandColumn>
                </GridCommandColumns>
            </GridColumn>
            <GridColumn IsPrimaryKey="true"
Field=@nameof(SyncfusionHelpDeskTickets.HelpDeskTicketId)
                HeaderText="ID #" TextAlign="@TextAlign.Left"
```

```

        Width="70">
    </GridColumn>
    <GridColumn
Field=@nameof(SyncfusionHelpDeskTickets.TicketStatus)
        HeaderText="Status" TextAlign="@TextAlign.Left"
        Width="80">
    </GridColumn>
    <GridColumn Field=@nameof(SyncfusionHelpDeskTickets.TicketDate)
        HeaderText="Date" TextAlign="@TextAlign.Left"
        Format="d" Type="ColumnType.Date"
        Width="80">
    </GridColumn>
    <GridColumn
Field=@nameof(SyncfusionHelpDeskTickets.TicketDescription)
        HeaderText="Description"
TextAlign="@TextAlign.Left"
        Width="150">
    </GridColumn>
</GridColumn>
</SfGrid>
</div>

```

Edit ticket

The data grid contains an **Edit** button that opens the selected ticket in a Syncfusion Dialog control. This control will display the selected ticket in the **EditTicket.razor** control, which was added earlier (and also used in the **Index.razor** control).

Add the following markup code to support this functionality. Notice that this version passes the value of **true** to the **isAdmin** property. This will cause the **EditTicket.razor** control to enable all fields to be editable.

Code Listing 64: Edit Ticket Dialog (Admin)

```

<SfDialog Target="#admintarget"
        Width="500px"
        Height="500px"
        IsModal="true"
        ShowCloseIcon="true"
        @bind-Visible="EditDialogVisibility">
    <DialogTemplates>
        <Header> EDIT TICKET # @SelectedTicket.HelpDeskTicketId</Header>
        <Content>
            <EditTicket SelectedTicket="@SelectedTicket" isAdmin="true" />
        </Content>
        <FooterTemplate>
            <div class="button-container">

```

```

        <button type="submit"
            class="e-btn e-normal e-primary"
            @onclick="SaveTicket">
            Save
        </button>
    </div>
</FooterTemplate>
</DialogTemplates>
</SfDialog>

```

Add the following method to the code section that opens the dialog when a help desk ticket is selected in the data grid.

Code Listing 65: OnCommandClicked

```

public async void OnCommandClicked(
    CommandClickEventArgs<SyncfusionHelpDeskTickets> args)
{
    if (args.CommandColumn.ButtonOption.Content == "Edit")
    {
        // Get the selected Help Desk Ticket.
        var HelpDeskTicket = (SyncfusionHelpDeskTickets)args.RowData;

        SelectedTicket =
            await
            HelpdeskAdminService.GetSyncfusionHelpDeskTicketAdminAsync
                (HelpDeskTicket.HelpDeskTicketId, ModuleState.ModuleId);

        // Open the Edit dialog.
        this.EditDialogVisibility = true;
        StateHasChanged();
    }
}

```

Add the **SaveTicket** method, which will be called when a user clicks the **Submit** button on the dialog control.

Code Listing 66: SaveTicket Method

```

public async Task SaveTicket()
{
    // Update the selected Help Desk Ticket.
    await HelpdeskAdminService.
        UpdateSyncfusionHelpDeskTicketsAdminAsync(SelectedTicket);

    // Close the Edit dialog.
    this.EditDialogVisibility = false;
}

```

```

        // Refresh the SfGrid
        // so the changes to the selected
        // Help Desk Ticket are reflected.
        gridObj.Refresh();
    }

```

Delete ticket

The data grid contains a **Delete** button that opens a Syncfusion Dialog control that asks the user to confirm they want to delete the selected record.

Add the following markup code to support this functionality.

Code Listing 67: Delete Confirmation Dialog

```

<SfDialog Target="#admintarget"
    Width="100px"
    Height="130px"
    IsModal="true"
    ShowCloseIcon="false"
    @bind-Visible="DeleteRecordConfirmVisibility">
    <DialogTemplates>
        <Header> DELETE RECORD ? </Header>
        <Content>
            <div class="button-container">
                <button type="submit"
                    class="e-btn e-normal e-primary"
                    @onclick="ConfirmDeleteYes">
                    Yes
                </button>
                <button type="submit"
                    class="e-btn e-normal"
                    @onclick="ConfirmDeleteNo">
                    No
                </button>
            </div>
        </Content>
    </DialogTemplates>
</SfDialog>

```

Add the following code to the **OnCommandClicked** method in the code section that opens the dialog when the **Delete** button next to a record in the data grid is clicked.

Code Listing 68: Delete Button Clicked

```

if (args.CommandColumn.ButtonOption.Content == "Delete")
{

```

```

        SelectedTicket = new SyncfusionHelpDeskTickets();
        SelectedTicket.HelpDeskTicketId =
args.RowData.HelpDeskTicketId;

        // Open Delete confirmation dialog.
        this.DeleteRecordConfirmVisibility = true;
        StateHasChanged();
    }

```

The **Delete** confirmation dialog contains two buttons, labeled **Yes** and **No**. If the user selects **No** (to cancel the delete), add the following code to simply close the dialog.

Code Listing 69: ConfirmDeleteNo

```

public void ConfirmDeleteNo()
{
    this.DeleteRecordConfirmVisibility = false;
}

```

If the user selects **Yes** (to delete the record), add the following code to delete the record and close the dialog.

Code Listing 70: ConfirmDeleteYes

```

public async void ConfirmDeleteYes()
{
    // The user selected Yes to delete the
    // selected Help Desk Ticket.
    // Delete the record.
    await HelpdeskAdminService.
        DeleteSyncfusionHelpDeskTicketsAsync(
            SelectedTicket.HelpDeskTicketId,
            ModuleState.ModuleId);

    // Close the dialog.
    this.DeleteRecordConfirmVisibility = false;
    StateHasChanged();

    // Refresh the SfGrid
    // so the deleted record will not show.
    gridObj.Refresh();
}

```


Add the Administration button

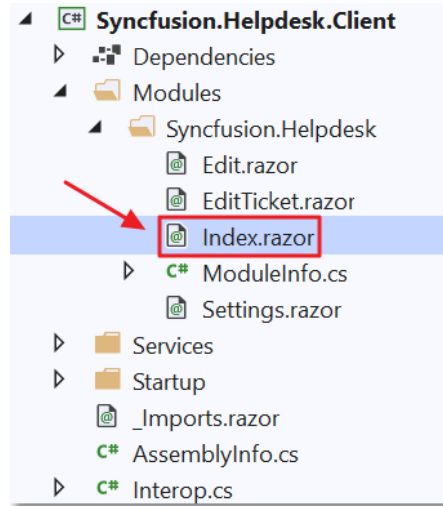


Figure 75: Open Index.razor

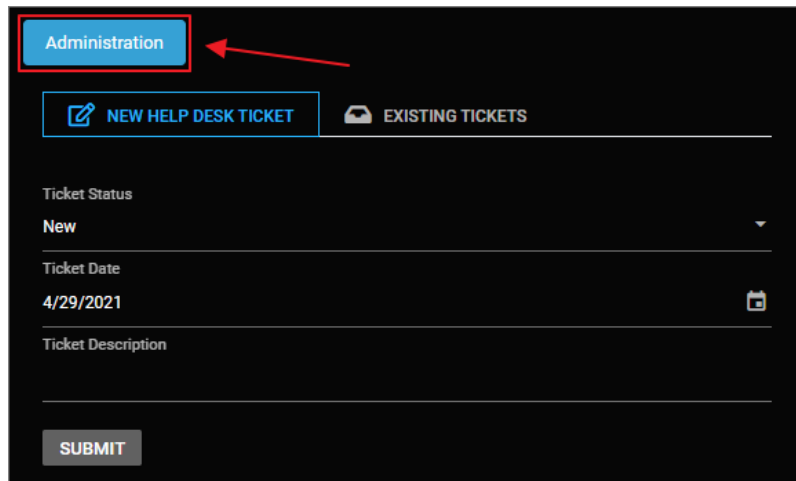
The final code needed to complete the module is to add an **Administration** button that will only be visible to users who have **Edit** access to the module. It will navigate to the **Edit.razor** control we have just completed.

Open the **Index.razor** control and add the following markup.

Code Listing 71: Administration Button

```
<ActionLink Action="Edit"  
            Security="SecurityAccessLevel.Edit"  
            Text="Administration" />
```

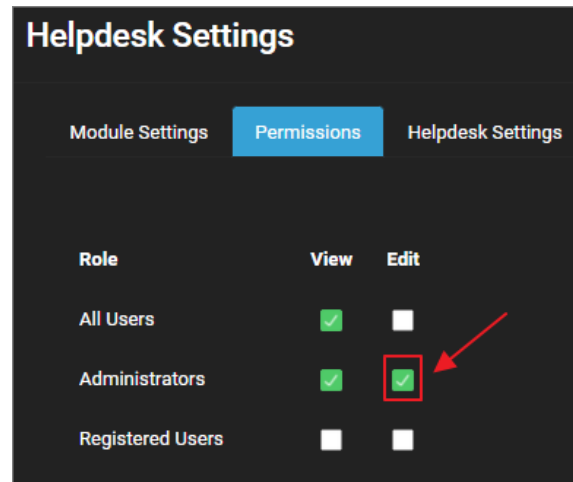
Test the final module



The screenshot shows a dark-themed user interface. At the top left, a blue button labeled "Administration" is highlighted with a red rectangle and a red arrow points to it from the right. Below this, there are two buttons: "NEW HELP DESK TICKET" (with a pencil icon) and "EXISTING TICKETS" (with a folder icon). Further down, there are input fields for "Ticket Status" (with a dropdown menu showing "New"), "Ticket Date" (with a calendar icon and the date "4/29/2021"), and "Ticket Description". At the bottom left, there is a grey "SUBMIT" button.

Figure 76: Administration Button

When we run the module in Oqtane and log in as the host account, we will now see the Administration button.



The screenshot shows the "Helpdesk Settings" page with the "Permissions" tab selected. It contains a table with columns "Role", "View", and "Edit".

Role	View	Edit
All Users	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Administrators	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Registered Users	<input type="checkbox"/>	<input type="checkbox"/>

A red rectangle highlights the "Edit" checkbox for the "Administrators" role, with a red arrow pointing to it from the right.

Figure 77: Edit Permissions

The Administration button shows because the host account is in the Administrators role. The module instance is configured to allow access to the Edit control to users in the Administration role.

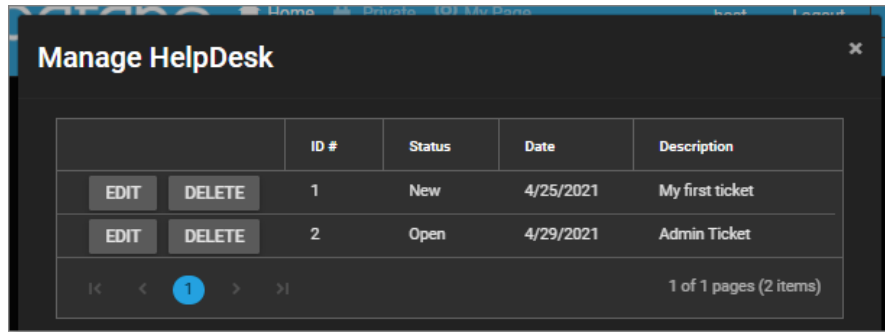


Figure 78: Administration

The Administration button will navigate to the **Edit.razor** control that will display the data grid allowing you to edit and delete records, sort, and page.

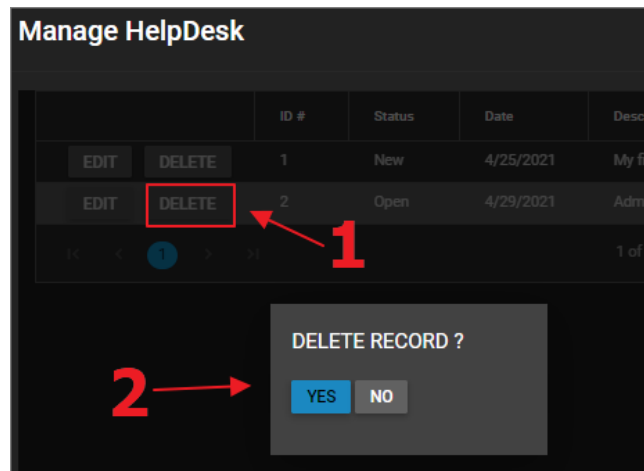


Figure 79: Confirm Delete

If the user clicks the **Delete** button next to a record in the data grid, the delete confirmation is opened. If the user clicks **Yes**, the record will be deleted, and if the user clicks **No**, the action is cancelled.

We've now seen how Blazor technology, together with Oqtane and Syncfusion controls, enables you to create sophisticated, manageable, and extensible single-page applications using C# and Razor syntax. Try it for yourself!