3D print Club Web Application

Code Behind The Web Application

Nicholas Slaughter

2019

Table of Contents

1. Introduction
2. Razor Pages and Data Models
   1. Intro to Razor Pages
   2. Why We Chose Razor Pages
   3. Data
      1. Migrations Folder
      2. Model Folder
         1. Application User Model
         2. Printer Model
         3. Request Model
         4. Status Model
         5. Color Model
      3. Application Database Context
3. Pages
   1. Introduction to the Code Behind the Web Pages:
   2. Shared Tool Bar
   3. Home Page
   4. Login/Sign Up
   5. Messages
   6. Requests
   7. Admin Only Pages
      1. Pending Requests
      2. Approved Requests
      3. Denied Requests
      4. User Management
4. Extra Files and Utilities
   1. Start Up
   2. Utility: Authorization Html Helper

I. Introduction

This documentation exists to give information about the code that went into the 3D print queuing system for Oregon Institute of Technologies (OIT) Wilsonville 3D print lab. My team and I made this project as a course requirement for our Junior Project as well as knowing that the 3D print club at OIT Wilsonville needed a way to handle student’s print requests. The objective of this project was to create both an Android and web application that would let students make a request to the 3D print lab for their project to be printed which would then put them in a que with a time and date that their project would be printed. We also included a messaging system between both the students and the club admins because the admins wanted a way to somehow talk to the students about what their project was and why they needed it printed. For this set of documentation, I will be covering the code that went into the 3D print web application as I was one of the main people working on the web side of the project. This documentation will include information on simple tasks that can be done to add or edit something, information about razor pages and the data models that was created for it, the code for the web pages, as well as how the database is linked.

II. Razor Pages and Data Models

Introduction to Razor Pages

Razor pages was made by Microsoft as an extension of ASP.NET core MVC (Model-View-Controller) that was designed to help make web pages easier to code.

ASP.NET core MVC is a web application framework that makes it easy to set up web pages by separating the code into 3 components; the models, the views, and the controllers. The models are what the pages are based on, the views are what handle what is displayed on the screen, and the controllers are what handles the interaction the user has with the web application.

Razor pages uses the files extension of .cshtml which allows the blend of C Sharp and HTML into a file. This extension allows the coder to do things they otherwise wouldn’t if they just used HTML or C Sharp. This is due to them being able to use code they have written up in C Sharp for the web page (known as a code behind file) which can then be used in the .cshtml file. With the access of using both C Sharp and HTML the website can be made much faster and will look much cleaner. Another useful part of razor pages is that because it is a part of ASP.NET core MVC it is easy to make models that will let us hold information in our database and have the razor page base its page off of the model we want it to have the data for (models will be explained in the next section).

Why We Chose Razor Pages

My team and I decided on using razor pages because of its feasibility in setting up pages and the models we could use for our database. I was also part of the contribution to using razor pages as I had worked on a smaller yet similar project in the past of which I enjoyed using razor pages and I could base some of my code from that project on to this project.

Data

The data folder in the program holds the migrations, the models, and the context for the database. Without this folder there would be no way to contact the database and add information to it.

Migrations Folder

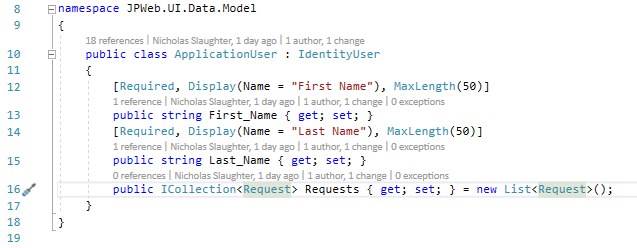
The migrations folder is what sets the tables for the database. If a table is ever created, destroyed, or edited then you must add a migration and update the database with the new information that was put into the new migration. If you do not add a migration and update the database after you change one of the tables, your database will not recognize what you have done and will work as if you haven’t changed the database until you update it.

Model Folder

The model folder is what allows you to create, destroy, or edit the tables in the database. Each model is considered a new table in the database and inside these models you can hold many sets of data. For our project we created four models. These models set the tables for the users (called Application User), the printers, the requests, and the statuses that can be set to each request.

Application User Model

The application user model holds the information we need from each user.

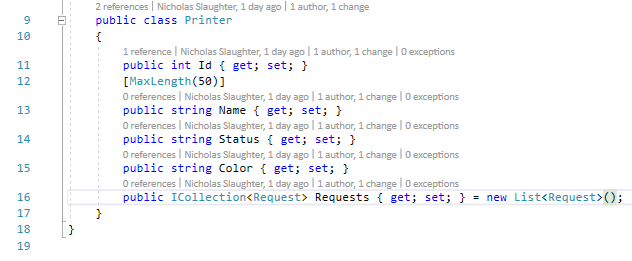
The fields we needed:

* Users first name
* Users last name

We don’t have users email and password as a field in the Application User model because that is already stored for us by the Areas folder. The Areas folder holds preloaded code for the project that is designed to help us (if you want more information on the Areas folder please refer to this link: <https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/areas?view=aspnetcore-2.2>). We don’t have an id set for the user due to the identity user extension which will give the user a more secure Id than 1 to N. This model also stores a collection of requests due to the logic that a student will have multiple requests.

Printer Model

The printer model is used to set the fields that will hold the information for the printers.

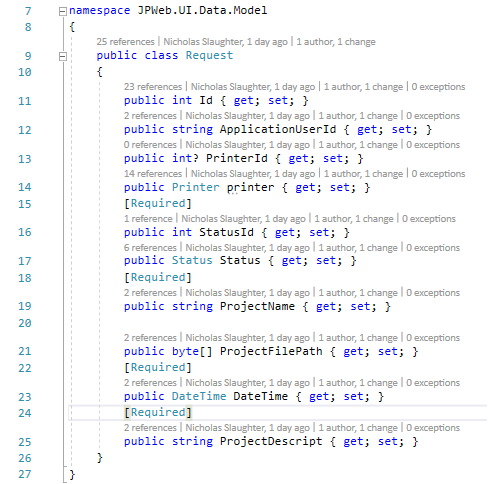
The fields we set for the printer model:

* Printer Id
* Printer Name
* Printer status
* Printer Color

We chose these as the fields based on the logic that there were multiple printers in the 3D print lab, each printer would have its own name, each printer would have a status, and the printer would need a color that the persons project would be printed as. The name field is rather useless because we can identify printers by their Id, but the 3D print lab had special names for each printer, so we decided to put it in as something fun for print club admins to see. The statuses for the printer are simple as the printer is either busy or free to print something. Finally, the color field exists due to the ability of having a project printed out in different colors that are available in the print lab. The printer model also holds a collection of requests due to the logic of a printer having multiple requests to print.

Request Model

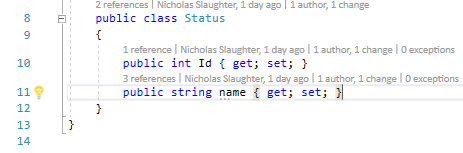
The request model is one of the most important models as it sets the table and information for all the requests that the students will make. With this being a model for requests we needed more information for this model compared to the others.

The request model fields:

* Request Id
* User Id
* Printer Id
* Status Id
* Project Name
* Project File Path
* Date and Time
* Project Description

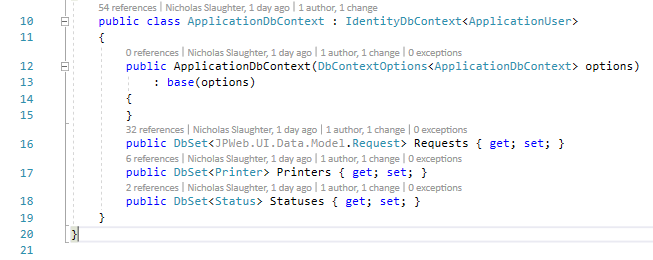
We had these fields set due to the logic that we needed to be able to identify a specific request, identify the user that made the request, identify the printer the request will be printed from, the requests will have a varying status, there needs to be a name for the project, we need a copy of the project, there needs to be a date and time to request to print, and there needs to be a description of the project. The project name field isn’t too important due to us being able to distinguish a request by its id, but the admin should be able to know the name of the project that is being requested to print. The project file path is an extremely important field because it will store the file that the student wants printed so that the admin can print the file. Date and time are also important and are required so the admin knows when the project should be printed. The project description is an important field that the 3D club admin requested because they need to know what it is the student wants to print before they approve the print job.

Status Model

It may seem weird that we are storing the status of a request in a separate table considering there would be hardly any information stored in this model (id and status name). The reason why we made status its own model is to make the code in the pages a bit cleaner. If we didn’t have a status model, we would have to add a bunch of ifelse cases in the code to find the status the request had, but with making the status its own model we only need one line of code which would get the status of the request. This also allows the admin to add more statuses, but it seems unlikely that more statuses would be added other than pending, approved, denied, and completed which are hard coded into the status table.

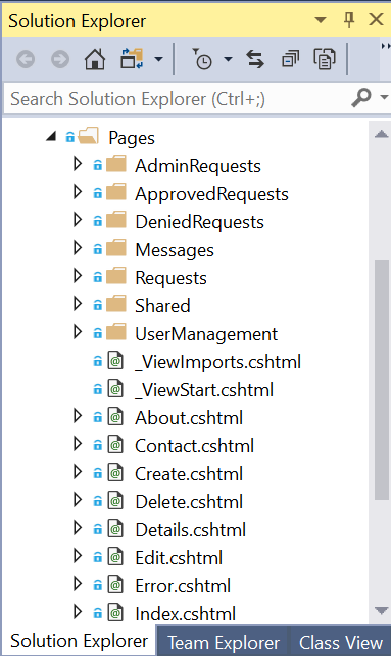
With these models we can store all the information we need for the 3D printer web app to work and have a successful queuing system.

Application Database Context

The application database context is what tells the database what tables need to be added or removed from the database. Whenever a model is added, code will also need to be added to the application database context to let it know that a new table is going to be made (refer to lines 16, 17, and 18 in the picture for how this is done). With the model in ankle brackets of the DbSet function, this lets the application database context get all the fields needed for the tables they go to. This piece of code is very important as it is what communicates with the database so you can insert tables into it.

III. Pages

Introduction to the Code Behind the Web Pages:

The pages folder holds all of the pages that you can go to on the website. Inside the folder is a section of subfolders that refer to what specific page they deal with and the pages that focus on the home page. The page specific folders include the cshtml code for that page, the C# code behind the web page, and the crud operations attached to the page.

The cshtml code is how the website is made to look different. In the cshtml we can add new elements to a page, delete elements from a page, or edit the elements in the page. The cshtml code is important as it lets us manipulate what the users can see and how they see it to make the page easier to comprehend without them seeing stuff they shouldn’t.

Behind each cshtml file there is a C# file which lets us manipulate what happens in the website when an action is performed. The C# file is what lets us take the information put in by the user and insert it into the database. Without the C# file there would be no way for the information to be stored in the database and is a very important part of the code for the website because of that.

The crud operation are sub-pages that deal with the elements of the specific page you are in. These operations deal with creating an element that is specified in the page, editing the element(s), view the details of an element, and deleting an element.

The different pages you can find in the solution explorer are as follows: Shared, the general pages for home and login, Requests, Messages, Pending Requests, Approved Requests, Denied Requests, and User Management. In the next section I will go over each of these pages including what their index pages do as well as what their crud operations do if they have them.

Shared:

The shared folder is a folder that is a premade when the razor page program is first created. This folder holds the code for items that are present on every page on the website. This includes the tool bar, the users name at the top, the logout button, and the acknowledgement at the bottom. This folder is extremely important as it lets us manipulate what the user can see depending on their status as a user (user, admin, super admin) and provides a quick way to any of the main pages of the website.

Shared Layout Page Tab Inclusion:

As stated above the shared folder can provide a quick way to any of the main pages on the website. This is done in the layout.cshtml file. In the file you can include the link to any page you want to appear on the tool bar. Including a page, you want into the tool bar is very simple as you can see in the image below. All you have to include is the following:

<li><a asp-page=”/folder of page you want/Index”>How you want the tab to be named</a></li>

Once you do this you will be able to see the tab for the page in the tool bar and if you click on it you will be taken to that page.

The Shared Layout Page Restrictions:

To manipulation what tabs users can see is done in the layout.cshtml file inside the shared folder. Inside the file is many pages that are wrapped around if statements and only a couple that aren’t. These if statements check the status you have as a user and lets you view them if you have the required status or higher. For example, anybody can view the Make A Request and Messages tab because the condition asks for some of a user or higher status, but only admins and super admins can see the Pending, Approved, and Denied Requests tabs because they have the required status or higher.

Making A Condition for A Page To Be Seen:

To make a condition for a page to be seen mirror one of the if statements above and wrap it around the tab you don’t want some users to see. This is done by typing the following code:

@{

If(AuthorizeHtmlHelper.UserIs(AdminOrHigher or SuperAdmin)(user))

{

<li><a asp-page=”/folder of page you want/Index”>How you want the tab to be named</a></li>

}}

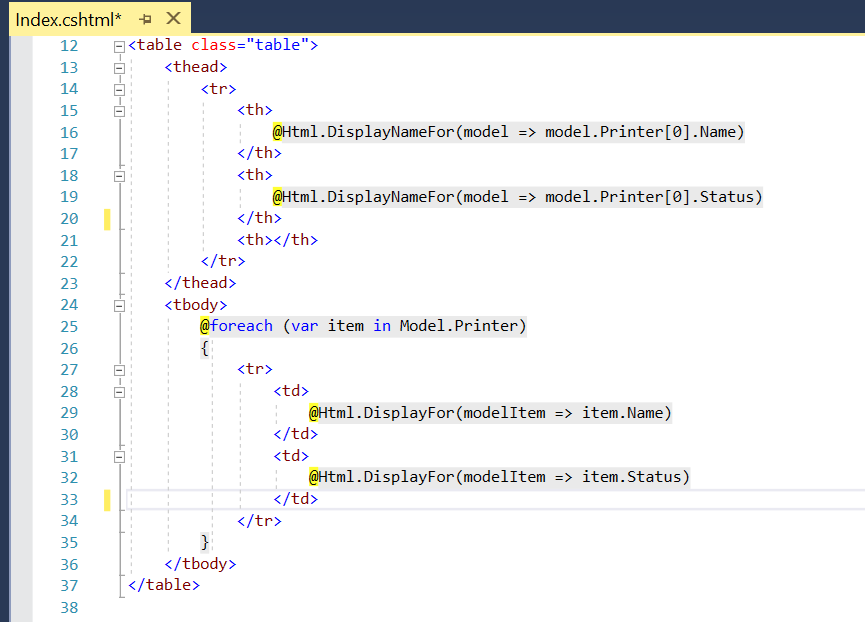
Once the condition is written, run the program and if your status is below the required status you will not be able to see the tab and if you have the required status you will be able to see the tab.

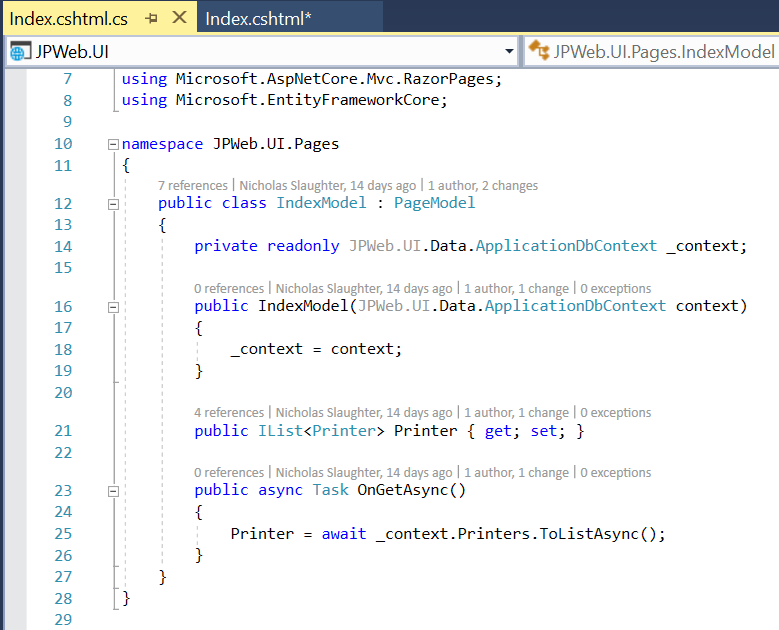
Home Page:

The home page is the very first page a user sees when the go to the website. This page shows the printers and their status. This is to show users what printers they can make a request for to get their project printed as soon as possible.

Index.cshtml For Home Page

The index.cshtml for the home page can be located as the index.cshtml that is in only in the pages folder. Inside the index.cshtml for the home page all that can be scene is the code that makes a table for the printers. The code for the table is split up into two parts the tables headings and the tables body.

The upper body of the table code represents the table headings. Anything between the <thead> (line 13) and </thead> (line 23) are considered headers in the table. The headers that are in the tables are set as the name of the printer (line 16), and the status of the printer (line 19).

The lower body of the table code represents the contents that go into the table otherwise known as the body of the table. Anything between <tbody> (line 24) and </tbody> (line 36) are considered elements that will appear below their specified heading. Inside the body the program will look for each printer in the database (line 25). When it finds a printer it will then display the name and status of the printer in the table (line 29 and 32).

Index Code Behind

The only thing happening inside the code behind for the home page is the instantiation of a list of printers that are in the database so that the Index.cshtml can parse through the list of printers to get their name and status.

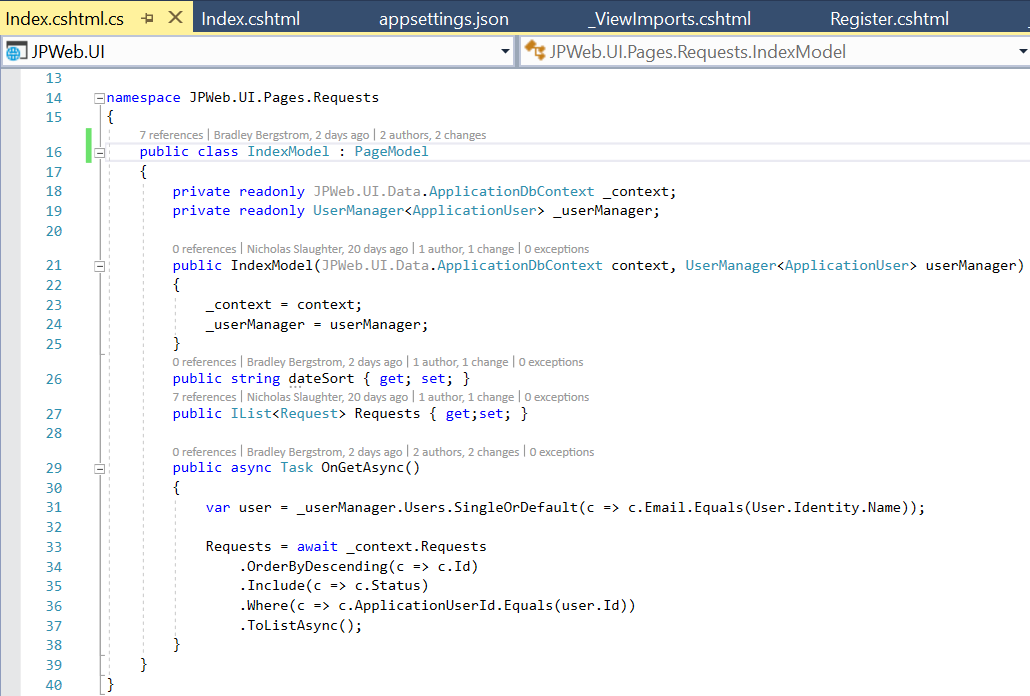
Request:

The requests folder holds the code that lets users make a request. When a user clicks on the “Make a Request” tab they will be directed to the request page which will show output a table of the requests they have made including the projects status, name, the file extension, the date requested, and the description. From there the user is also able to create a new request or delete their request. This functionality happens in the index.cshtml, create.cshtml, and the delete.cshtml files including their code behind files as well.

Index.cshtml for Requests:

Inside the index.csthml for requests there is code for the table that gets the information from each request that the current logged in user has made. the table first assigns the headings such as status, name, project file path, date requested, and project description. Next the table gets each request that the current user has made and displays their elements in their respected columns.

Aside from the table that shows the requests for the current logged in user, the index page also let’s go to the create a request page and lets you go to the delete page for each request that the current logged in user has made.

 Code Behind for Index:

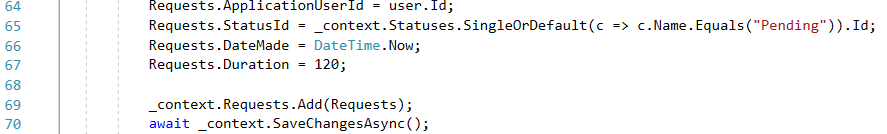
Inside the code behind for the index.cshtml we create a UserManager (line 19) which lets us get the email for the current user that is logged in. This then lets us get the requests only for the current user (line 36). The \_context (line 18) is what lets us get the requests that are currently in the database. We then create a list of requests out of the requests in the database so we can manipulate and parse through that list in the index.cshtml (line 33).

 Create.cshtml for Requests:

The create.cshtml for the requests page is fairly simple as it prints out empty text boxes with heading above them to signify what information you are supposed to insert into the empty text box. The page also includes a check box that indicates whether a project is for personal use or not by making the Boolean of “PersonalUse” in Requests equal true if it is checked. Then at the bottom there is code for a button that is called submit and when the button is clicked the information that you entered will be inserted into the Requests table in the database.

Code Behind for Create:

Inside the code behind for the create.cshtml we create a request (line 33) and a message hub (line 35) to insert the new information into the database. When the submit button is pressed the program goes to the OnPostASync() class to insert the new information into the database and create or add a message hub for a user.

From lines 64 to 70 we store the information that the user entered, and we store their id, we set the request status to pending, we get the current date time, and we set the duration to a default of 2 hours.

From lines 74 – 97 we check to see if this is the users first time making a request and if it is, we create a new message hub letting the user and admins know that a project has been submitted. If they have made a request before then a message will be sent to the user and the admins to let them know a new project has been submitted.

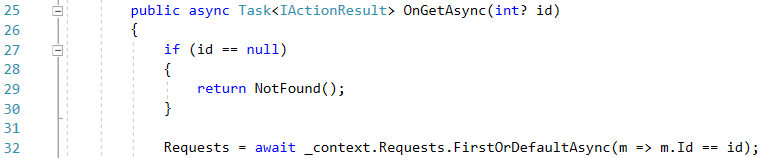
Once we have updated the database and the message hub we then go back to the index page (line 99).

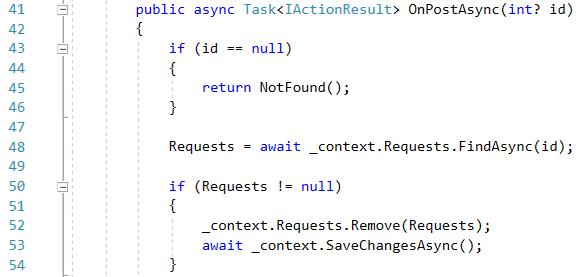
Delete.cshtml for Requests:

The delete.cshtml in the Requests folder is accessed when you click delete on any request in the request table. When the delete page is accessed the information about the requests is displayed so the user knows what exactly the request is. If the user does choose to delete the request, then they click a button that says delete and return to the index page.

Code Behind for Delete:

In the code behind for the delete.cshtml we get the information from the request to display it and we delete the request when the delete button is clicked.

When the delete page is accessed the OnGetASync() is called in the code behind for the and sets a request equal to the request the user is wanting to delete (line 32). We do this so the delete.cshtml can display the information about the request.

When the delete button is clicked the OnPostASync() is called in the code behind and sets a request equal to the request the user wants to delete. This is done for safety reason to ensure that the user isn’t deleting a request that is null. If the request isn’t null, the request is removed from the database (line 48-54).

Once the request is deleted the user then returns to the index page (line 56).

Admin Only Pages

Pending Requests:

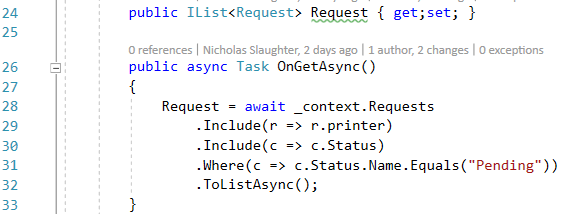
The pending requests shows all the requests that have a status of pending.

Index.cshtml For Pending Requests:

The Index.cshtml for pending requests displays a table of requests much like that of the Request index page but it only shows the requests that have pending requests. The page also lets you edit any request in the table.

Code Behind For Index

The code behind for the index.cshtml gets a list of all the requests in the database that have a status of “pending”. We do this so the only requests that show in the Index.cshtml table are pending requests.

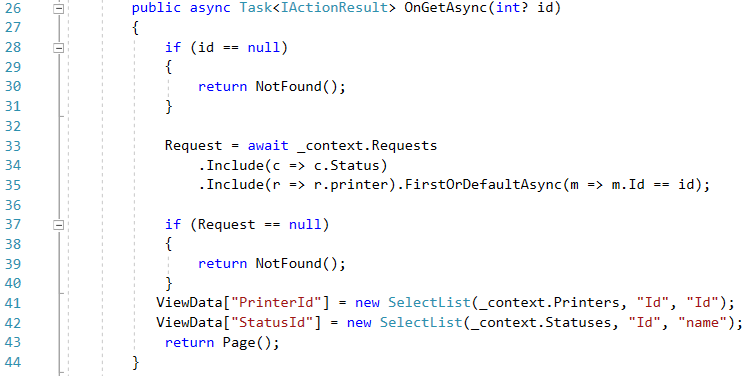


Edit.cshtml For Pending Requests

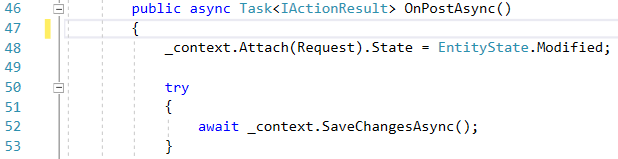
The edit.cshtml for pending requests displays a drop down menu for statuses and printers that the admin wants to assign to the request. Once the status and printer is selected the admin can click the submit button which will then update the status Id and printer Id of the request in the database.

Code Behind For Edit

The code behind for the edit.cshtml stores the request in a variable so it can be modified, gets the statuses and printers that can be used for their drop down menu, and updates the request into the database.

When the edit page is accessed the OnGetASync() function is called and stores the request into a variable so it can be edited (line 33-35). Then the statuses and printers that are in the database are accessed so the drop-down menu can get filled with their contents (line 41-42).

When the submit button is clicked the OnPostASync() function is called and this then inserts the modified data into the request which then gets synced to the database (line 48-52). Once the database is updated the admin returns to the index page.



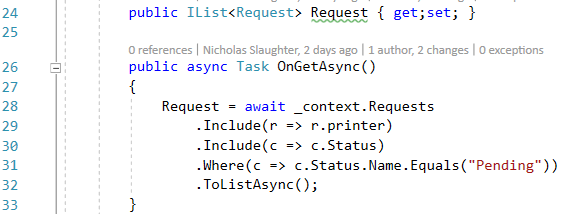
Approved Requests:

Index.cshtml For Approved Requests:

The Index.cshtml for approved requests displays a table of requests much like that of the Request index page but it only shows the requests that have approved requests. The page also lets you edit any request in the table.

Code Behind For Index

The code behind for the index.cshtml gets a list of all the requests in the database that have a status of “approved”. We do this so the only requests that show in the Index.cshtml table are approved requests.

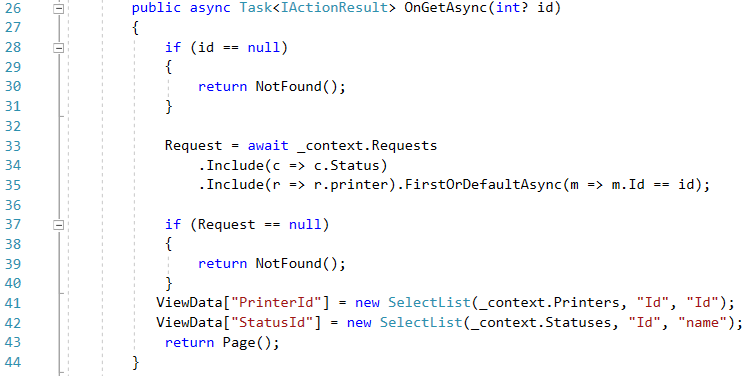


Edit.cshtml For Approved Requests

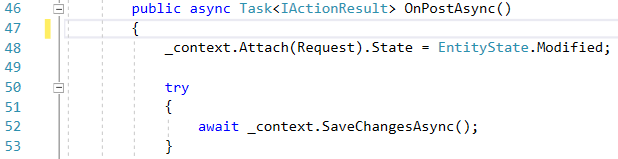
The edit.cshtml for approved requests displays a drop down menu for statuses and printers that the admin wants to assign to the request. Once the status and printer is selected the admin can click the submit button which will then update the status Id and printer Id of the request in the database.

Code Behind For Edit:

The code behind for the edit.cshtml stores the request in a variable so it can be modified, gets the statuses and printers that can be used for their drop down menu, and updates the request into the database.

When the edit page is accessed the OnGetASync() function is called and stores the request into a variable so it can be edited (line 33-35). Then the statuses and printers that are in the database are accessed so the drop-down menu can get filled with their contents (line 41-42).

When the submit button is clicked the OnPostASync() function is called and this then inserts the modified data into the request which then gets synced to the database (line 48-52). Once the database is updated the admin returns to the index page.



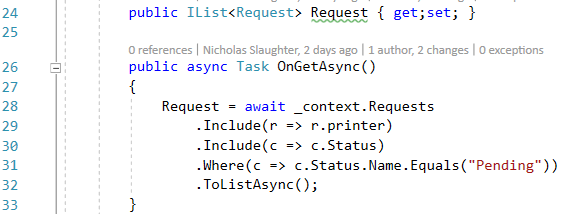
Denied Requests:

Index.cshtml For Denied Requests:

The Index.cshtml for denied requests displays a table of requests much like that of the Request index page but it only shows the requests that have denied requests. The page also lets you edit any request in the table.

Code Behind For Index

The code behind for the index.cshtml gets a list of all the requests in the database that have a status of “denied”. We do this so the only requests that show in the Index.cshtml table are denied requests.

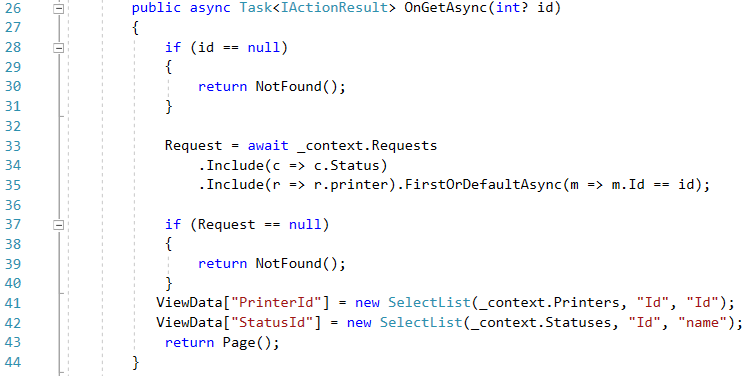


Edit.cshtml For Denied Requests

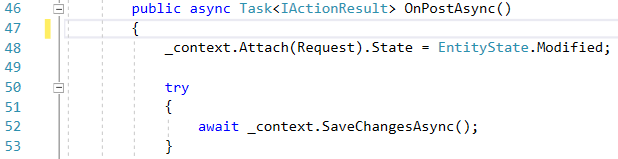
The edit.cshtml for denied requests displays a drop down menu for statuses and printers that the admin wants to assign to the request. Once the status and printer is selected the admin can click the submit button which will then update the status Id and printer Id of the request in the database.

Code Behind For Edit:

The code behind for the edit.cshtml stores the request in a variable so it can be modified, gets the statuses and printers that can be used for their drop down menu, and updates the request into the database.

When the edit page is accessed the OnGetASync() function is called and stores the request into a variable so it can be edited (line 33-35). Then the statuses and printers that are in the database are accessed so the drop-down menu can get filled with their contents (line 41-42).

When the submit button is clicked the OnPostASync() function is called and this then inserts the modified data into the request which then gets synced to the database (line 48-52). Once the database is updated the admin returns to the index page.



User Management:

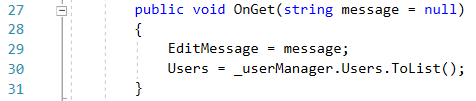
The user management page is a unique page as it is the only page that can only be accessed by a super admin. This high restriction is because this page allows the super admin to change the policies of other users. Meaning that this allows the super admin to set a user to the status of admin or super admin. The super admin can do this by first accessing the list of users in the index page and then editing their requests in the edit page.

Index.cshtml For User Management:

The index.cshtml for user management prints out a list of all of the users registered in the database. It prints out their information in the format of a table with the headings being Id, name, and email. The program then parses through the list of users outputting the id, name, and email of each user. The program will also show an edit button next to each user allowing you to go to the edit.cshtml page.

Code Behind For Index.cshtml:

In the code behind for index.cshtml all the users in the database are put into a list so that the list can be parsed in the index.cshtml. The list of users is got when the super admin clicks on the user management tab in the tool bar. This calls the OnGet() function in the code behind which then sets the list equal to the users in the database (line 30).



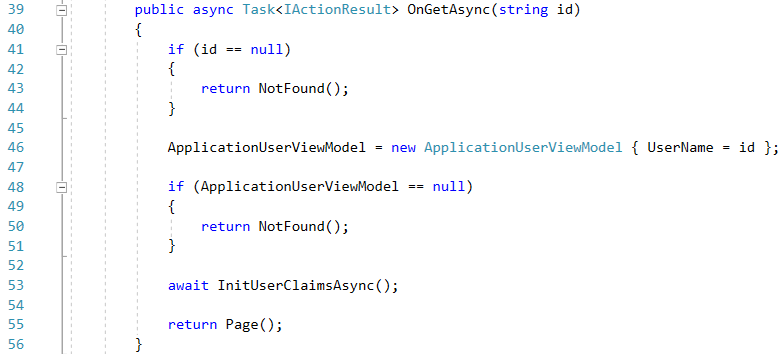
Edit.cshtml For User Management:

The edit.cshtml for the user management folder prints out the email of the user you are editing the claim of, and check boxes that represent each claim. If a check box is checked this sets the claim for the user to whatever check box is checked. This claim is not set in the database, however, until the submit button is pressed.

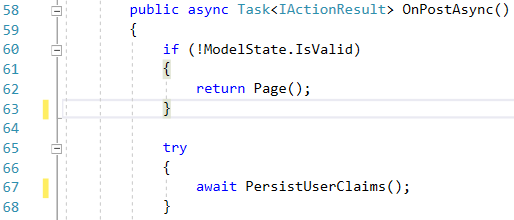
Code Behind For Edit.cshtml:

The code behind for the edit.cshtml gets the users email so that it can display on the edit page, and it sets the claim of the user to whatever check box was clicked for them.

When a super admin clicks on the edit button in the index page the OnGetASync() function is called which gets the users email and their current claim (line 46 – 53).



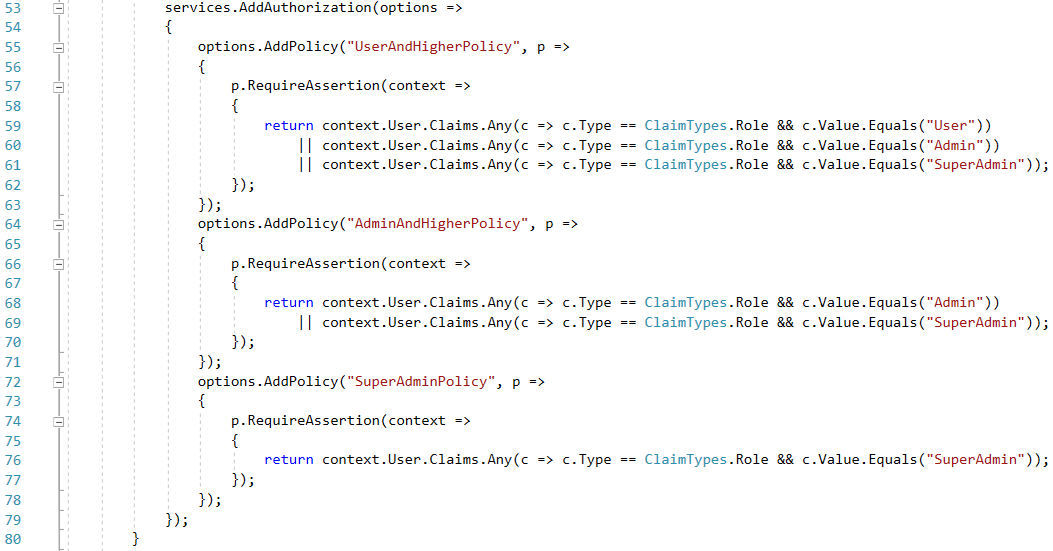
When a super admin clicks the submit button on the edit page the OnPostASync() function is called. This then sets the claim of the user to whatever check boxes were checked (line 69). Once the super admin edits the users claim they are then brought back to the index page with a message prompting that the user whose claim was edited needs to be logged out for their claim to change(line 81).



IV. Extra Files and Utilities

Start Up:

The start up page is a premade class when you create the razor page program. This class is very important because it sets up the connection of the program to the database and lets us set the user policies (line 53-80). If you want to know more about the start up class and how it works please refer to <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/startup?view=aspnetcore-2.2>.



Utility: Authorization Html Helper:

The authorization html helper is a utility for our program to make the code for restricting a user from seeing pages they shouldn’t a lot simpler. Inside the authorization html helper class, there are three function that check to see if a user has a high enough claim to view a page. These functions are as follows: UserOrHigher(ClaimsIdentity user), UserIsAdminOrHigher(ClaimsIdentity user), and UserIsSuperAdmin(ClaimsIdentity user). These functions are called in the layout index.cshtml, the admin pages, and the user management page. The functions are called there because each of these pages has something that should not be viewed by regular users of the website.

