```
how to call SearchContactFilter class in javascript
function LoadData() {
    var url = '/Dashboard/Contact?PageNumber=' + pageNumber +
'&PageSize=8&isInitialLoad=0';
    window.location.href = url;
}
public class DashboardController : Controller
{
    public async Task < | ActionResult >
        GetPaginationData(SearchContactFilter model)
{
}
```

- To call the SearchContactFilter class in your JavaScript function, you first need to construct the appropriate request to send data from JavaScript to your controller. Here's how you can approach it:
 - Modify the LoadData function to send an HTTP request to the GetPaginationData action in your controller. You can do this using fetch or ajax.
 - 2. Pass parameters to the controller action is a query string or request body (for POST or GET).

Example with fetch and GET method

Let's assume SearchContactFilter is a model that contains some properties like SearchTerm, PageNumber, and PageSize, which you want to pass from JavaScript.

JavaScript - LoadData function

Here's how you can send a GET request with query parameters:

```
javascript
                                                                              ி Copy code
function LoadData(pageNumber) {
   // Define the parameters you need
    var searchTerm = "exampleSearch"; // You can dynamically change this based on user in
   var pageSize = 8; // You can change this based on your needs
    // Construct the URL with query parameters
    var url = `/Dashboard/GetPaginationData?SearchTerm=${searchTerm}&PageNumber=${pageNumber}
    // Send a GET request to your controller
    fetch(url)
       .then(response => response.json()) // Assuming the response is JSON
       .then(data => {
           // Handle the data returned m the controller
           console.log(data);
        })
        .catch(error => {
           console.error('Error:', error);
        });
}
```

Controller Action - C# (Backend)

Your GetPaginationData action would look like this:

Here, SearchContactFilter could be something like this:

```
csharp

public class SearchContactFilter
{

public string SearchTerm { get; set; }

public int PageNumber { get; set; }
```

```
public int PageSize { get; set; }
}
```

The controller's GetPaginationData action will automatically bind the query parameters to the properties of SearchContactFilter when you make the GET request.

Example with fetch and POST method

If you want to send the data as a PoST reque ψ or example, if you are sending large data or sensitive information), you can modify the JavaScript like this:

```
Copy code
function LoadData(pageNumber) {
   var searchTerm = "exampleSearch";
   var pageSize = 8;
   // Prepare the data to send in the body
   var requestData = {
       SearchTerm: searchTerm,
       PageNumber: pageNumber,
       PageSize: pageSize
   };
    // Send a POST request with the data
    fetch('/Dashboard/GetPaginationData', {
       method: 'POST',
           'Content-Type': 'application/json'
       body: JSON.stringify(requestData)
   })
    .then(response => response.json())
    .then(data => {
       // Handle the data returned from the controller
       console.log(data);
   })
    .catch(error => {
       console.error('Error:', error);
   });
}
```

And in the controller:

```
csharp

[HttpPost]
public async Task<IActionResult> GetPaginationData([FromBody] SearchContactFilter model)

{
    // Handle the search/filter logic here using the model
    var data = await _yourDataService.GetContactsAsync(model);

    // Return the data as JSON
    return Json(data);
}
```

Explanation:

- JavaScript (GET and POST): You can choose either GET (use query parameters) or POST (send data in the body). The example above shows how to send the search filter as a query string or in the body of the request.
- C# Controller: The controller method receives the filter model (SearchContactFilter) either from the query string (for GET) or the request body (for POST), and processes it.
- SearchContactFilter: This model maps the filter criteria from the frontend (JavaScript) to the controller's action.

This approach should work for calling your <code>GetPaginationData</code> action and passing the filter parameters (<code>SearchTerm</code> , <code>PageNumber</code> , <code>PageSize</code> , etc.).

\$

If you want to call the SearchContactFilter class from JavaScript within the LoadData function and pass data to the GetPaginationData action in the DashboardController, there are a few key things to consider.

1. JavaScript Function:

- You need to send parameters to the backend (either through a query string or POST request).
- You are currently using window.location.href, which changes the browser's location to
 navigate to a URL. This is typically used for simple navigation or when you want to load a
 new page.

However, to send parameters (such as Sear-chContactFilter) to the GetPaginationData action, it's better to make an AJAX request (using fetch or XMLHttpRequest), so you can send the filter data dynamically without changing the page.

2. Backend (C# Controller):

 The GetPaginationData action in your DashboardController should accept parameters for SearchContactFilter and return the necessary data.

Solution Using fetch (AJAX Request):

Here's how you can modify your code to send SearchContactFilter data without navigating away from the page.

JavaScript (LoadData Function)

```
javascript
                                                                        Copy code
function LoadData(pageNumber) {
   // Define the filter data (you can adjust this based on user input)
   var pageSize = 8; // You can adjust this value
   // Create the filter object to send to the server
    var searchContactFilter = {
       SearchTerm: searchTerm,
       PageNumber: pageNumber,
       PageSize: pageSize
   };
    // Send the data via a POST request using the Fetch API
    fetch('/Dashboard/GetPaginationData', {
       method: 'POST', // Use POST to send the data to the server
       headers: {
           'Content-Type': 'application/json' // Indicate that we're sending JSON
       body: JSON.stringify(searchContactFilter) // Convert the filter object to JSON
   })
    .then(response => response.json()) // Parse the JSON response from the server
    .then(data => {
       // Handle the data returned from the controller (e.g., update the UI) \,
       console.log(data);
   })
    .catch(error => {
       console.error('Error:', error);
    });
```

Controller (Backend - C#)

In your DashboardController, you need to define the GetPaginationData action to accept the SearchContactFilter model. The model will be populated with the data sent from the frontend (JavaScript).

```
csharp

public class DashboardController: Controller
{

// This action handles the POST request and expects the SearchContactFilter model in to [HttpPost]

public async Task<IActionResult> GetPaginationData([FromBody] SearchContactFilter mode)

{

// You can now use the model (Set hoontactFilter) to fetch filtered data
```

```
var data = await _yourDataService.GetContactsAsync(model);

// Return the filtered data as JSON
    return Json(data);
}
```

SearchContactFilter Model

Make sure your SearchContactFilter class has the appropriate properties:

```
csharp

public class SearchContactFilter
{
    public string SearchTerm { get; set; }
    public int PageNumber { get; set; }
    public int PageSize { get; set; }
}
```

Explanation:

1. JavaScript:

- In the LoadData function, we construct the SearchContactFilter object with parameters like SearchTerm, PageNumber, and PageSize.
- We then send this object as a JSON payload to the /Dashboard/GetPaginationData endpoint using the fetch API (with a POST method).

2. C# Controller:

- The [FromBody] attribute tells ASP.NET to bind the incoming JSON request body to the SearchContactFilter model.
- The action will then process the filter data (e.g., by querying a database or service) and return the results as JSON.

3. Result Handling:

Once the request completes, the response is handled in the .then(data => {...}) block, where you can process the filtered data and update the UI.

Why Not Use window.location.href

- window.location.href is used to navigate to a new page. If you want to send data to the
 backend without reloading the page, an AJAX request (like fetch) is more appropriate.
- With window.location.href, you'd be navigating to a new URL, which would reload the page and wouldn't allow you to handle the data dynamically without reloading.

Alternative (GET Request via Query String)

If you absolutely prefer using window.location.href (i.e., navigating the browser with a query string), you could pass the parameters in the URL, but this would typically require modifications to your controller to accept query parameters. Here's how you could do it:

1. Modify JavaScript to send a GET request:

```
javascript

function LoadData(pageNumber) {

var searchTerm = "exampleSearch"; // Example search term

var pageSize = 8;

// Construct the URL with query string parameters

var url = '/Dashboard/GetPaginationData?SearchTerm=' + encodeURIComponent(searchTerm)

'%PageNumber=' + pageNumbé ↓

'%PageSize=' + pageSize;

// Navigate to the URL (this will reload the page)

window.location.href = url;
}
```

2. Controller (Backend): You would modify your controller to accept query parameters:

```
csharp

public class DashboardController : Controller

{

public async Task<IActionResult> GetPaginationData(string searchTerm, int pageNumber, :

{

// Create a SearchContactFilter object from guery parameters
```

```
var model = new SearchContactFilter
    SearchTerm = searchTerm,
    PageNumber = pageNumber,
    PageSize = pageSize
// Process the model, fetch the data
var data = await _yourDataService.GetContactsAsync(model);
// Return the filtered data as JSON
return Json(data);
```

Summary:

- AJAX (fetch) is the recommended approach for sending data to the server without navigating away from the page.
- window.location.href is only useful for simple page navigation and will cause the page to reload.

how to setup username and password in Ubuntu



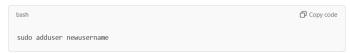
Setting up a username and password in Ubuntu can be done in several ways, depending on your goal. Here are the most common methods to create and manage user accounts on an Ubuntu system:

1. Create a New User (via Terminal)

If you're looking to add a new user with a specific username and password, follow these steps:

Step-by-Step:

- 1. Open Terminal: Press Ctrl + Alt + T to open the terminal.
- 2. Create a New User: To add a new user, use the adduser command followed by the desired username. For example:



Replace newusername with the actual username you want to create.

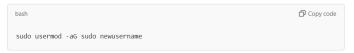
3. Set the Password: The adduser command will prompt you to set a password for the new user. You'll need to type the password twice for confirmation.

Example:

```
∩ Copy code
markdown
Enter new UNIX password: ******
Retype new UNIX password: ******
```

After that, it will ask for additional information (e.g., Full Name, Room Number, etc.), but you can press Enter to leave these blank.

4. Add User to Sudo Group (Optional): If you want the new user to have administrative (sudo) privileges, you need to add them to the sudo group:



This will give the user newusername sudo access, which allows them to run commands with administrative privileges.

5. Verify the User: You can verify that the user has been created by checking the /home directory or using the id command:



This will show the UID, GID, and groups the user is part of.

6. Switch to the New User: To switch to the new user account, use the su or sudo command:

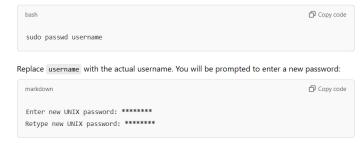
```
bash
                                                                            Copy code
```



2. Change the Password for an Exis & g User

If you need to change the password of an existing user, you can use the passwd command:

1. Change Password: Use the following command to change the password for a specific user:



3. Managing Users via GUI (Graphical Interface)

If you're using the **Ubuntu Desktop**, you can manage users from the graphical interface as well.

- Open "Settings": Go to the Activities overwew and search for Settings, or click on the system menu (top-right corner) and click Settings.
- Navigate to "Users": In the Settings window, go to the Users tab (under Details in newer versions).
- Unlock for Changes: You'll need to unlock the settings to make changes. Click the Unlock button at the top-right and enter your current password.
- 4. Add a New User:
 - Click the + button to add a new user.
 - Choose the account type (e.g., Standard or Administrator) and fill in the Full Name, Username, and Password.
 - Click Add to create the user.
- 5. Modify Existing Users:
 - Click on any existing user to change their settings, including password or user type.

4. Set Up Automatic Login (Option \downarrow

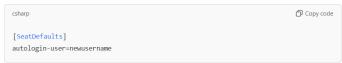
If you want your newly created user to automatically log in when you start your system, you can set that up from the **Settings** > **Users** interface in the GUI. However, this is **not recommended for security reasons** unless it's a single-user machine.

Alternatively, you can enable automatic login by modifying the configuration file manually:

1. Edit the LightDM Config (for Ubuntu Desktop with LightDM):



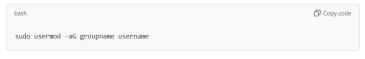
2. Add the Following Lines:



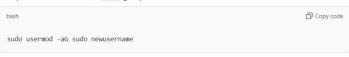
Replace newusername with the username you want to log in automatically.



To add the user to specific groups (like sudo for administrative access or other groups), use the usermod command:



Example to add a user to the sudo group:



Common Commands for User Management:

Here's a summary of the most common user n agement commands:

- Add User: sudo adduser username
- Change Password: sudo passwd username
- Add User to Group: sudo usermod -aG groupname username
- Delete User: sudo deluser username
- Check User Info: id username
- List All Users: cat /etc/passwd

Summary:

- Create a new user using the adduser command and set a password.
- Manage user passwords with the passwd command.
- Use the graphical interface to create or modify users on Ubuntu Desktop.
- Add users to groups for specific privileges (e.g., sudo for administrative access).

ð

Message ChatGPT

ChatGPT can make mistakes. Check important info.