**4. Phân trang dữ liệu**

Khi làm việc với danh sách dữ liệu lớn, việc hiển thị tất cả dữ liệu trên một trang có thể làm giảm trải nghiệm người dùng và làm chậm ứng dụng. **Phân trang** là kỹ thuật chia nhỏ dữ liệu thành các trang để hiển thị và cho phép người dùng di chuyển giữa các trang này. Đây là cách quản lý hiệu quả khi hiển thị danh sách dữ liệu lớn.

**Cơ chế phân trang**

Phân trang được thực hiện bằng cách chia dữ liệu thành nhiều phần, mỗi phần gọi là một "trang", và chỉ hiển thị một phần dữ liệu tại một thời điểm. Để phân trang dữ liệu, ta cần xác định hai yếu tố:

1. **page**: Biến xác định trang hiện tại.
2. **PAGE\_SIZE**: Số lượng dữ liệu hiển thị trên mỗi trang.

**1. Phân trang dữ liệu với AJAX**

Khi gửi yêu cầu AJAX tới server để lấy dữ liệu, ta sẽ cần truyền thêm tham số page và PAGE\_SIZE. Tham số page giúp xác định trang nào đang được yêu cầu, còn PAGE\_SIZE xác định số lượng mục dữ liệu cần hiển thị trên mỗi trang.

**Ví dụ từ file claim-manager.js**:

javascript

const *PAGE\_SIZE* = 3; // Số lượng mục hiển thị trên mỗi trang  
  
function fetchClaims(page = 0) {  
 const claimCode = $('#search-ma-yeu-cau').val();  
 const fromDate = $('#from-date').val();  
 const toDate = $('#to-date').val();  
 const statusCode = $('#trang-thai-yeu-cau').val();  
  
 const requestData = {  
 claimCode,  
 fromDate,  
 toDate,  
 statusCode  
 };  
  
 $.ajax({  
 url: `http://localhost:8080/api/claims?page=${page}&size=` + *PAGE\_SIZE*, // Truyền tham số page và PAGE\_SIZE  
 type: 'POST',  
 contentType: 'application/json',  
 data: *JSON*.stringify(requestData),  
 success: function(response) {  
 renderTable(response); // Hiển thị dữ liệu trên bảng  
 renderPagination(response.pageNumber, response.totalPages); // Hiển thị phân trang  
 },  
 error: function(error) {  
 *console*.error('Lỗi khi lấy dữ liệu:', error);  
 }  
 });  
}

**Giải thích**:

* page: Tham số này được truyền trong URL để xác định trang dữ liệu cần lấy.
* PAGE\_SIZE: Đặt số lượng mục cần hiển thị trên mỗi trang (ví dụ: 3 mục).
* Server trả về dữ liệu tương ứng với trang đã yêu cầu, sau đó dữ liệu sẽ được hiển thị trong bảng HTML.

**2. Tạo và cập nhật giao diện phân trang**

Để người dùng có thể di chuyển giữa các trang, cần tạo giao diện phân trang với các nút chuyển trang (nút "Trang trước", "Trang sau", và các số trang). Các nút này sẽ cho phép người dùng chọn trang mà họ muốn xem.

**Ví dụ về hàm renderPagination trong file claim-manager.js**:

javascript

function renderPagination(pageNumber, totalPages) {  
 const pagination = $('#pagination');  
 pagination.empty(); // Xóa các nút phân trang cũ  
  
 // Thêm nút "Trang trước" nếu không phải là trang đầu tiên  
 if (pageNumber > 0) {  
 pagination.append(`<li class="page-item"><a class="page-link" href="#" data-page="${pageNumber - 1}">&laquo;</a></li>`);  
 }  
  
 // Tạo các nút số trang  
 for (let i = 0; i < totalPages; i++) {  
 pagination.append(`<li class="page-item ${i === pageNumber ? 'active' : ''}"><a class="page-link" href="#" data-page="${i}">${i + 1}</a></li>`);  
 }  
  
 // Thêm nút "Trang sau" nếu không phải là trang cuối cùng  
 if (pageNumber < totalPages - 1) {  
 pagination.append(`<li class="page-item"><a class="page-link" href="#" data-page="${pageNumber + 1}">&raquo;</a></li>`);  
 }  
}

**Giải thích**:

* Hàm renderPagination tạo ra các nút phân trang dựa trên tổng số trang (totalPages) và trang hiện tại (pageNumber).
* **Nút "Trang trước"**: Nếu không phải là trang đầu tiên, thêm một nút giúp người dùng quay lại trang trước đó.
* **Nút số trang**: Số trang được tạo dựa trên totalPages. Trang hiện tại được đánh dấu là "active" để người dùng biết họ đang ở trang nào.
* **Nút "Trang sau"**: Nếu không phải là trang cuối cùng, thêm một nút giúp người dùng chuyển sang trang tiếp theo.

**3. Xử lý sự kiện cho các nút phân trang**

Khi người dùng nhấn vào các nút phân trang (số trang, trang trước, trang sau), cần gửi yêu cầu AJAX để lấy dữ liệu của trang tương ứng.

**Ví dụ xử lý sự kiện phân trang**:

javascript

$(*document*).on('click', '#pagination .page-link', function(e) {  
 e.preventDefault(); // Ngăn chặn hành động mặc định của liên kết  
 const page = $(this).data('page'); // Lấy số trang từ thuộc tính data-page  
 fetchClaims(page); // Gọi lại hàm fetchClaims với trang mới  
});

**Giải thích**:

* Sử dụng sự kiện click để lắng nghe khi người dùng nhấn vào một trong các nút phân trang.
* Lấy số trang từ thuộc tính data-page của nút nhấn ($(this).data('page')).
* Gọi lại hàm fetchClaims để gửi yêu cầu AJAX với trang mới, từ đó cập nhật dữ liệu và giao diện.

**Tóm tắt**

* **Cơ chế phân trang**: Phân trang dữ liệu giúp chia nhỏ danh sách lớn thành nhiều trang và chỉ hiển thị một số lượng mục cố định trên mỗi trang. Điều này giúp cải thiện trải nghiệm người dùng và tối ưu hiệu suất.
  + Sử dụng các biến page và PAGE\_SIZE trong yêu cầu AJAX để xác định số trang và số lượng dữ liệu trên mỗi trang.
* **Tạo và cập nhật giao diện phân trang**:
  + Hàm renderPagination tạo các nút chuyển trang dựa trên số trang hiện tại và tổng số trang.
  + Nút "Trang trước", "Trang sau", và các số trang cho phép người dùng di chuyển qua lại giữa các trang dữ liệu.
* **Xử lý sự kiện phân trang**:
  + Khi người dùng nhấn vào các nút phân trang, hàm JavaScript sẽ được gọi để cập nhật dữ liệu của trang tương ứng.

Thông qua phân trang, trang web sẽ hiển thị dữ liệu một cách mượt mà, đồng thời cung cấp cho người dùng trải nghiệm tốt hơn khi làm việc với danh sách dữ liệu lớn.

**5. Giao tiếp với API Backend**

Giao tiếp với API backend là một kỹ thuật quan trọng giúp trang web tương tác với server để lấy dữ liệu hoặc gửi dữ liệu mới. Trong các ứng dụng web hiện đại, **AJAX** (Asynchronous JavaScript and XML) là phương thức phổ biến để thực hiện việc này mà không cần tải lại toàn bộ trang.

**Cách gửi yêu cầu tới API**

**AJAX** cho phép gửi và nhận dữ liệu từ server mà không cần tải lại trang. Bạn có thể sử dụng AJAX để gửi yêu cầu tới API backend, lấy dữ liệu từ server và hiển thị chúng trên trang web.

**1. Sử dụng AJAX để giao tiếp với API backend**

Trong ví dụ sau, chúng ta sẽ sử dụng AJAX để gửi yêu cầu tới API và nhận về danh sách yêu cầu bồi thường từ server. API backend có URL là http://localhost:8080/api/claims.

**Ví dụ sử dụng AJAX để gửi yêu cầu tới API:**

javascript

function fetchClaims(page = 0) {  
 const claimCode = $('#search-ma-yeu-cau').val(); // Lấy giá trị mã yêu cầu từ input  
 const fromDate = $('#from-date').val(); // Lấy giá trị từ ngày  
 const toDate = $('#to-date').val(); // Lấy giá trị đến ngày  
 const statusCode = $('#trang-thai-yeu-cau').val(); // Lấy trạng thái yêu cầu  
  
 const requestData = {  
 claimCode,  
 fromDate,  
 toDate,  
 statusCode  
 };  
  
 $.ajax({  
 url: `http://localhost:8080/api/claims?page=${page}&size=3`, // Địa chỉ API với tham số page và PAGE\_SIZE  
 type: 'POST',  
 contentType: 'application/json', // Định dạng dữ liệu gửi lên server  
 data: *JSON*.stringify(requestData), // Chuyển dữ liệu request thành định dạng JSON  
 success: function(response) {  
 renderTable(response); // Hiển thị dữ liệu trên bảng  
 renderPagination(response.pageNumber, response.totalPages); // Hiển thị phân trang  
 },  
 error: function(error) {  
 *console*.error('Lỗi khi gửi yêu cầu:', error); // Xử lý lỗi nếu có  
 }  
 });  
}

**Giải thích:**

* **URL API**: Đây là địa chỉ API mà ứng dụng web gửi yêu cầu tới, ví dụ http://localhost:8080/api/claims.
* **AJAX request**: Sử dụng phương thức POST để gửi yêu cầu tới server.
* **contentType**: Dữ liệu được gửi lên server dưới dạng application/json (JSON format).
* **data**: Dữ liệu cần gửi (request payload), được chuyển thành định dạng JSON bằng hàm JSON.stringify().
* **success**: Nếu yêu cầu thành công, hàm này sẽ được gọi để xử lý dữ liệu trả về từ server và hiển thị lên giao diện.
* **error**: Nếu có lỗi xảy ra (ví dụ: không thể kết nối tới server), hàm này sẽ được gọi để hiển thị thông báo lỗi.

**Thiết kế dữ liệu gửi lên server (Request Payload)**

**Request payload** là dữ liệu được gửi từ frontend (trình duyệt) lên server (API) để thực hiện một yêu cầu nào đó. Trong trường hợp này, khi tìm kiếm danh sách yêu cầu bồi thường, ta cần gửi các thông tin như mã yêu cầu, ngày tháng, trạng thái.

**1. Dữ liệu gửi lên server (Request Payload)**

Dữ liệu gửi lên server được thiết kế dựa trên các tiêu chí tìm kiếm. Trong ví dụ về việc tìm kiếm yêu cầu bồi thường, request payload bao gồm các thông tin:

* **claimCode**: Mã yêu cầu bồi thường.
* **fromDate**: Ngày bắt đầu tìm kiếm.
* **toDate**: Ngày kết thúc tìm kiếm.
* **statusCode**: Trạng thái của yêu cầu bồi thường (ví dụ: đang xử lý, hoàn thành, bị từ chối).

**Ví dụ thiết kế request payload:**

javascript

const *requestData* = {  
 claimCode: $('#search-ma-yeu-cau').val(), // Lấy mã yêu cầu từ input  
 fromDate: $('#from-date').val(), // Lấy ngày bắt đầu từ input  
 toDate: $('#to-date').val(), // Lấy ngày kết thúc từ input  
 statusCode: $('#trang-thai-yeu-cau').val() // Lấy trạng thái yêu cầu từ dropdown  
};

**2. Truy vấn dữ liệu từ API bằng request payload**

Khi gửi dữ liệu từ frontend, request payload sẽ được chuyển thành JSON và gửi qua AJAX tới API backend. Server nhận request payload và trả về kết quả tương ứng với yêu cầu tìm kiếm.

**Ví dụ chi tiết về việc gửi request payload:**

javascript

$.ajax({  
 url: `http://localhost:8080/api/claims?page=0&size=3`, // URL API và tham số phân trang  
 type: 'POST',  
 contentType: 'application/json', // Định dạng dữ liệu gửi lên  
 data: *JSON*.stringify(requestData), // Chuyển request payload thành JSON  
 success: function(response) {  
 renderTable(response); // Hiển thị dữ liệu trên bảng  
 renderPagination(response.pageNumber, response.totalPages); // Hiển thị phân trang  
 },  
 error: function(error) {  
 *console*.error('Lỗi khi gửi yêu cầu:', error); // Xử lý lỗi  
 }  
});

**Giải thích:**

* **data**: Dữ liệu được gửi lên server, là một đối tượng JavaScript chứa các tiêu chí tìm kiếm. Trước khi gửi, nó được chuyển thành định dạng JSON bằng JSON.stringify().
* **success**: Khi yêu cầu thành công, server sẽ trả về một đối tượng JSON chứa dữ liệu yêu cầu bồi thường tương ứng với các tiêu chí đã gửi.
* **renderTable**: Hàm này sẽ hiển thị dữ liệu trả về trên bảng HTML.

**Ví dụ tổng thể: Gửi yêu cầu tìm kiếm yêu cầu bồi thường**

Giả sử chúng ta có một form tìm kiếm yêu cầu bồi thường với các tiêu chí: mã yêu cầu, ngày bắt đầu, ngày kết thúc, và trạng thái yêu cầu.

**HTML form**:

html

<div class="row mb-3">  
 <div class="col-md-3">  
 <label for="search-ma-yeu-cau">Mã yêu cầu bồi thường</label>  
 <input type="text" id="search-ma-yeu-cau" class="form-control" placeholder="Nhập mã yêu cầu">  
 </div>  
 <div class="col-md-3">  
 <label for="from-date">Từ ngày</label>  
 <input type="date" id="from-date" class="form-control">  
 </div>  
 <div class="col-md-3">  
 <label for="to-date">Đến ngày</label>  
 <input type="date" id="to-date" class="form-control">  
 </div>  
 <div class="col-md-3">  
 <label for="trang-thai-yeu-cau">Trạng thái yêu cầu</label>  
 <select id="trang-thai-yeu-cau" class="form-control">  
 <option value="">Tất cả trạng thái</option>  
 <option value="Đang xử lý">Đang xử lý</option>  
 <option value="Hoàn thành">Hoàn thành</option>  
 <option value="Đã lên lịch">Đã lên lịch</option>  
 <option value="Đang chờ">Đang chờ</option>  
 <option value="Bị từ chối">Bị từ chối</option>  
 </select>  
 </div>  
</div>  
<div class="row mb-3">  
 <div class="col-md-12 text-right">  
 <button id="search-button" class="btn btn-primary">Tìm kiếm</button>  
 </div>  
</div>

**JavaScript (AJAX request với dữ liệu từ form)**:

javascript

$('#search-button').click(function() {  
 const requestData = {  
 claimCode: $('#search-ma-yeu-cau').val(), // Lấy mã yêu cầu từ input  
 fromDate: $('#from-date').val(), // Lấy ngày bắt đầu  
 toDate: $('#to-date').val(), // Lấy ngày kết thúc  
 statusCode: $('#trang-thai-yeu-cau').val() // Lấy trạng thái yêu cầu  
 };  
  
 $.ajax({  
 url: `http://localhost:8080/api/claims?page=0&size=3`, // Gửi yêu cầu đến API  
 type: 'POST',  
 contentType: 'application/json', // Định dạng dữ liệu gửi lên server  
 data: *JSON*.stringify(requestData), // Chuyển đổi request thành JSON  
 success: function(response) {  
 renderTable(response); // Hiển thị kết quả trên bảng  
 },  
 error: function(error) {  
 *console*.error('Lỗi:', error); // Xử lý khi có lỗi  
 }  
 });  
});

**Tóm tắt**

* **Gửi yêu cầu tới API**: Sử dụng AJAX để gửi dữ liệu từ frontend tới backend API. Yêu cầu này có thể là POST hoặc GET tùy theo mục đích, và dữ liệu có thể là JSON hoặc định dạng khác.
* **Request Payload**: Là dữ liệu gửi lên server, thường là các tiêu chí tìm kiếm, trạng thái, và các thông tin liên quan khác. Trước khi gửi, dữ liệu này cần được chuyển thành JSON.
* **AJAX**: AJAX giúp giao tiếp với API mà không cần tải lại trang, giúp trải nghiệm người dùng mượt mà hơn.

Với các kỹ thuật này, sinh viên có thể xây dựng các tính năng tìm kiếm, lọc dữ liệu trên giao diện web và giao tiếp với API backend một cách hiệu quả.

**6. Tìm kiếm và lọc dữ liệu**

Tìm kiếm và lọc dữ liệu là những tính năng quan trọng trong các ứng dụng web, giúp người dùng dễ dàng tìm thấy thông tin mà họ cần. Để thực hiện việc tìm kiếm và lọc dữ liệu, chúng ta cần thiết kế một form cho phép người dùng nhập hoặc chọn các tiêu chí tìm kiếm. Sau đó, sử dụng AJAX để gửi các tiêu chí này lên server và hiển thị kết quả.

**Thiết kế form tìm kiếm**

Một form tìm kiếm sẽ bao gồm nhiều phần tử điều khiển (form controls) như ô nhập liệu (input), danh sách thả xuống (select), và các nút để thực hiện tìm kiếm. Dưới đây là ví dụ về form tìm kiếm với các tiêu chí như mã yêu cầu, ngày tháng và trạng thái yêu cầu.

**1. Tạo giao diện form tìm kiếm trong HTML**

Form tìm kiếm thường bao gồm các input để nhập mã yêu cầu, chọn ngày bắt đầu, ngày kết thúc, và trạng thái yêu cầu. Dưới đây là cách tạo form tìm kiếm với nhiều tiêu chí.

**Ví dụ về form tìm kiếm:**

html

<div class="row mb-3">  
 <div class="col-md-3">  
 <label for="search-ma-yeu-cau">Mã yêu cầu bồi thường</label>  
 <input type="text" id="search-ma-yeu-cau" class="form-control" placeholder="Nhập mã yêu cầu">  
 </div>  
 <div class="col-md-3">  
 <label for="from-date">Từ ngày</label>  
 <input type="date" id="from-date" class="form-control">  
 </div>  
 <div class="col-md-3">  
 <label for="to-date">Đến ngày</label>  
 <input type="date" id="to-date" class="form-control">  
 </div>  
 <div class="col-md-3">  
 <label for="trang-thai-yeu-cau">Trạng thái yêu cầu</label>  
 <select id="trang-thai-yeu-cau" class="form-control">  
 <option value="">Tất cả trạng thái</option>  
 <option value="Đang xử lý">Đang xử lý</option>  
 <option value="Hoàn thành">Hoàn thành</option>  
 <option value="Đã lên lịch">Đã lên lịch</option>  
 <option value="Đang chờ">Đang chờ</option>  
 <option value="Bị từ chối">Bị từ chối</option>  
 </select>  
 </div>  
</div>  
<div class="row mb-3">  
 <div class="col-md-12 text-right">  
 <button id="search-button" class="btn btn-primary">Tìm kiếm</button>  
 </div>  
</div>

**Giải thích:**

* #search-ma-yeu-cau: Input để nhập mã yêu cầu bồi thường.
* #from-date và #to-date: Input kiểu ngày (date) để người dùng chọn khoảng thời gian tìm kiếm.
* #trang-thai-yeu-cau: Dropdown cho phép người dùng chọn trạng thái yêu cầu.
* Nút "Tìm kiếm" (#search-button) khi được nhấn sẽ thực hiện việc gửi yêu cầu tìm kiếm dựa trên các tiêu chí đã nhập.

**Lấy giá trị từ các input để gửi lên server**

Sau khi người dùng nhập thông tin vào các input hoặc chọn các tiêu chí tìm kiếm, bạn cần lấy các giá trị này từ DOM và gửi chúng lên server để thực hiện tìm kiếm.

**1. Lấy giá trị từ các input**

Để lấy giá trị từ các input trong form, bạn có thể sử dụng jQuery với cú pháp đơn giản như sau:

* **$('#search-ma-yeu-cau').val()**: Lấy giá trị từ ô nhập mã yêu cầu.
* **$('#from-date').val()**: Lấy giá trị ngày bắt đầu.
* **$('#to-date').val()**: Lấy giá trị ngày kết thúc.
* **$('#trang-thai-yeu-cau').val()**: Lấy giá trị từ dropdown trạng thái yêu cầu.

**Ví dụ: Lấy dữ liệu từ form tìm kiếm**

javascript

$('#search-button').click(function() {  
 const claimCode = $('#search-ma-yeu-cau').val(); // Lấy mã yêu cầu bồi thường từ input  
 const fromDate = $('#from-date').val(); // Lấy giá trị ngày bắt đầu  
 const toDate = $('#to-date').val(); // Lấy giá trị ngày kết thúc  
 const statusCode = $('#trang-thai-yeu-cau').val(); // Lấy giá trị trạng thái yêu cầu  
  
 const requestData = {  
 claimCode: claimCode, // Dữ liệu mã yêu cầu  
 fromDate: fromDate, // Dữ liệu từ ngày  
 toDate: toDate, // Dữ liệu đến ngày  
 statusCode: statusCode // Dữ liệu trạng thái  
 };  
  
 // Gửi dữ liệu lên server qua AJAX  
 $.ajax({  
 url: `http://localhost:8080/api/claims`, // URL API  
 type: 'POST',  
 contentType: 'application/json', // Định dạng dữ liệu gửi đi  
 data: *JSON*.stringify(requestData), // Chuyển đổi đối tượng requestData thành JSON  
 success: function(response) {  
 renderTable(response); // Hiển thị dữ liệu kết quả  
 },  
 error: function(error) {  
 *console*.error('Lỗi khi lấy dữ liệu:', error); // Xử lý lỗi  
 }  
 });  
});

**Giải thích**:

* **Lấy giá trị từ input**: Sử dụng phương thức .val() của jQuery để lấy giá trị mà người dùng nhập hoặc chọn.
* **requestData**: Đây là đối tượng chứa tất cả các tiêu chí tìm kiếm từ form (mã yêu cầu, ngày tháng, trạng thái).
* **AJAX**: Dữ liệu này sau đó được gửi lên server thông qua AJAX, và khi server trả về kết quả, chúng sẽ được hiển thị trên giao diện người dùng.

**2. Gửi dữ liệu tìm kiếm lên server**

Dữ liệu được gửi lên server có thể bao gồm nhiều thông tin khác nhau từ form tìm kiếm. Chúng ta thường gửi dữ liệu dưới dạng JSON để server có thể xử lý dễ dàng.

**Ví dụ chi tiết về request payload**:

javascript

Sao chép mã

const *requestData* = {  
 claimCode: $('#search-ma-yeu-cau').val(), // Lấy mã yêu cầu từ input  
 fromDate: $('#from-date').val(), // Lấy giá trị ngày bắt đầu  
 toDate: $('#to-date').val(), // Lấy giá trị ngày kết thúc  
 statusCode: $('#trang-thai-yeu-cau').val() // Lấy trạng thái từ dropdown  
};  
  
$.ajax({  
 url: `http://localhost:8080/api/claims`, // Gửi yêu cầu đến API server  
 type: 'POST', // Phương thức POST  
 contentType: 'application/json', // Dữ liệu được gửi lên dưới dạng JSON  
 data: *JSON*.stringify(*requestData*), // Chuyển đổi dữ liệu thành chuỗi JSON  
 success: function(response) {  
 renderTable(response); // Hiển thị dữ liệu kết quả trên bảng  
 },  
 error: function(error) {  
 *console*.error('Lỗi khi lấy dữ liệu:', error); // Xử lý lỗi nếu có  
 }  
});

**Giải thích**:

* **claimCode, fromDate, toDate, statusCode**: Được lấy từ các input trong form và tạo thành một đối tượng requestData.
* **JSON.stringify(requestData)**: Chuyển đối tượng requestData thành chuỗi JSON để gửi lên server.
* **success**: Khi dữ liệu được trả về từ server, ta sẽ sử dụng hàm renderTable để hiển thị dữ liệu trong bảng.

**Ví dụ tổng thể: Tìm kiếm và lọc dữ liệu yêu cầu bồi thường**

Giả sử bạn có một bảng để hiển thị kết quả tìm kiếm, và form tìm kiếm sẽ thực hiện yêu cầu tìm kiếm khi người dùng nhập dữ liệu và nhấn nút "Tìm kiếm".

**Form HTML**:

html

<div class="row mb-3">  
 <div class="col-md-3">  
 <label for="search-ma-yeu-cau">Mã yêu cầu bồi thường</label>  
 <input type="text" id="search-ma-yeu-cau" class="form-control" placeholder="Nhập mã yêu cầu">  
 </div>  
 <div class="col-md-3">  
 <label for="from-date">Từ ngày</label>  
 <input type="date" id="from-date" class="form-control">  
 </div>  
 <div class="col-md-3">  
 <label for="to-date">Đến ngày</label>  
 <input type="date" id="to-date" class="form-control">  
 </div>  
 <div class="col-md-3">  
 <label for="trang-thai-yeu-cau">Trạng thái yêu cầu</label>  
 <select id="trang-thai-yeu-cau" class="form-control">  
 <option value="">Tất cả trạng thái</option>  
 <option value="Đang xử lý">Đang xử lý</option>  
 <option value="Hoàn thành">Hoàn thành</option>  
 <option value="Đã lên lịch">Đã lên lịch</option>  
 <option value="Đang chờ">Đang chờ</option>  
 <option value="Bị từ chối">Bị từ chối</option>  
 </select>  
 </div>  
</div>  
<div class="row mb-3">  
 <div class="col-md-12 text-right">  
 <button id="search-button" class="btn btn-primary">Tìm kiếm</button>  
 </div>  
</div>

**AJAX request trong JavaScript**:

javascript

$('#search-button').click(function() {  
 const requestData = {  
 claimCode: $('#search-ma-yeu-cau').val(), // Lấy mã yêu cầu từ input  
 fromDate: $('#from-date').val(), // Lấy giá trị ngày bắt đầu  
 toDate: $('#to-date').val(), // Lấy giá trị ngày kết thúc  
 statusCode: $('#trang-thai-yeu-cau').val() // Lấy giá trị trạng thái  
 };  
  
 $.ajax({  
 url: `http://localhost:8080/api/claims`, // URL API  
 type: 'POST',  
 contentType: 'application/json', // Định dạng dữ liệu gửi đi  
 data: *JSON*.stringify(requestData), // Gửi dữ liệu tìm kiếm lên server  
 success: function(response) {  
 renderTable(response); // Hiển thị kết quả trên bảng  
 },  
 error: function(error) {  
 *console*.error('Lỗi khi lấy dữ liệu:', error); // Xử lý lỗi  
 }  
 });  
});

**Tóm tắt**

* **Thiết kế form tìm kiếm**: Bạn có thể tạo form tìm kiếm với nhiều tiêu chí (mã yêu cầu, ngày tháng, trạng thái) bằng cách sử dụng các phần tử HTML như input và select.
* **Lấy giá trị từ input**: Sử dụng jQuery để lấy giá trị từ các input của người dùng và tạo request payload chứa các tiêu chí tìm kiếm.
* **Gửi dữ liệu lên server**: Dữ liệu tìm kiếm được gửi lên server bằng AJAX và server sẽ trả về kết quả phù hợp với các tiêu chí tìm kiếm.