How to Set Up a Web Front End with Express, Axios, Ejs and Nodemon

A Step-by-Step Guide for Creating a Web Application that Calls Two API Endpoints and Displays Reports

# Introduction

In this document, you will learn how to set up a web front end that calls two API endpoints and displays reports, one as a tabular report and the other as a bar chart or histogram, using express, axios, ejs and nodemon. Express is a web framework for Node.js that allows you to create web applications and APIs. Axios is a library that helps you make HTTP requests from Node.js or the browser. Ejs is a templating engine that lets you embed JavaScript code in HTML pages. Nodemon is a tool that automatically restarts your Node.js application when you make changes to your code.

You will use the following API endpoints for this project:

* localhost:5023/Reports/getSales: This endpoint returns a JSON data of sales report that contains the fields: product, quantity, price, and total.
* localhost:5023/Reports/getInvoices: This endpoint returns a JSON data of invoices report that contains the fields: country, count, and amount.

You will display the sales report as a table and the invoices report as a bar chart or histogram in your web front end.

# Prerequisites

Before you start, you need to have the following installed on your system:

* Node.js: You can download and install it from https://nodejs.org/en/.
* NPM: This is a package manager for Node.js that comes with Node.js installation.
* Express: You can install it using the command npm install express.
* Axios: You can install it using the command npm install axios.
* Ejs: You can install it using the command npm install ejs.
* Nodemon: You can install it using the command npm install nodemon.

# Steps

Follow these steps to set up your web front end:

1. Create a new folder for your project and navigate to it in your terminal.
2. Initialize a new Node.js project using the command npm init -y. This will create a package.json file that contains the metadata of your project.
3. To set up the project, you'll need to install several npm packages. Here are the commands:

# Initialize a new Node.js project  
npm init -y  
# Install Express for the server  
npm **install** express  
# Install Axios for making HTTP requests  
npm **install** axios  
# Install EJS for the view engine  
npm **install** ejs  
# Install Nodemon for automatic server restarts during development  
npm **install** --save-dev nodemon

To create the report server, you can add a script to your package.json file to start the server using Nodemon. Here's how you can do it

"scripts": {  
  "start": "nodemon app.js"  
}

1. Then, you can start the server by running:

npm start

johndohoneyjr: Create an Express JS app that makes API Calls for a Sales Report. Display the API response as a table for the sales report. Call localhost:5023/Reports/getSales to get the JSON data. Make another API Call to display a report of Invoices by Country. Display this as a Table. Call localhost:5023/Reports/getInvoices to get the JSON response from the API. Show all necessary files to change with the code to implement the reports. Be sure to generate CSS for both reports.

GitHub Copilot: To create an Express JS app that makes API calls for a Sales Report and Invoices by Country, we'll need to do the following:

1. Set up an Express JS server.

2. Create routes to fetch data from the provided API endpoints.

3. Create views to display the fetched data in a table format.

4. Generate CSS for both reports.

Here's the step-by-step pseudocode:

1. Initialize a new Express app.

2. Set up middleware for parsing JSON and URL-encoded data.

3. Create a route for the sales report.

4. In the sales report route, make a GET request to the sales report API.

5. Parse the response and pass the data to the sales report view.

6. Create a route for the invoices report.

7. In the invoices report route, make a GET request to the invoices report API.

8. Parse the response and pass the data to the invoices report view.

9. Create views for the sales report and invoices report.

10. In each view, loop through the data and display each item in a table row.

11. Create CSS files for the sales report and invoices report views.

12. In each CSS file, style the table and its elements.

// app.js  
**const** express = require('express');  
**const** axios = require('axios');  
**const** app = express();  
  
app.**set**('view engine', 'ejs');  
  
app.**get**('/sales-report', **async** (req, res) => {  
 **const** response = **await** axios.**get**('http://localhost:5023/Reports/getSales');  
 res.render('sales-report', { data: response.data });  
});  
  
app.**get**('/invoices-report', **async** (req, res) => {  
 **const** response = **await** axios.**get**('http://localhost:5023/Reports/getInvoices');  
 res.render('invoices-report', { data: response.data });  
});  
  
app.listen(3000, () => console.log('Server running on port 3000'));

<!-- views/sales-report.ejs -->  
<**table**>  
 <**%** data.forEach(item => { %>  
 <**tr**>  
 <**td**><**%=** item.sales %></**td**>  
 <!-- Add more columns as needed -->  
 </**tr**>  
 <**%** }); %>  
</**table**>

```

```ejs

<!-- views/invoices-report.ejs -->  
<**table**>  
 <**%** data.forEach(item => { %>  
 <**tr**>  
 <**td**><**%=** item.invoice %></**td**>  
 <!-- Add more columns as needed -->  
 </**tr**>  
 <**%** }); %>  
</**table**>

/\* public/stylesheets/sales-report.css \*/  
**table** {  
 **width**: 100%;  
 **border-collapse**: collapse;  
}  
**table**, **th**, **td** {  
 **border**: 1px solid black;  
}

/\* public/stylesheets/invoices-report.css \*/  
**table** {  
 **width**: 100%;  
 **border-collapse**: collapse;  
}  
**table**, **th**, **td** {  
 **border**: 1px solid black;  
}

Remember to install necessary dependencies (`express`, `axios`, `ejs`) using npm and to serve static files (CSS) using `express.static` middleware. Also, adjust the table columns in the EJS files according to the structure of your API response.

Then, you can start the server by running:

npm start

Start Debugging in VSC

A black and white logo

Description automatically generated

Be sure to set some breakpoints in your code

A screenshot of a computer

Description automatically generated