ANGULAR AR-TOOL

PRODUCTION HOURS SCREEN

Components

\*Production-table

\*Production-header-section

\*Preferencemodal

Services

\*Produtionheader.service

\*column -names.service

Production-header-section

\*These componet contains header of the page its contains two input tag one is a WorkfromDate other is TimeZone. The workfromDate input is a datetime picker, its for user select what time they need the date time picker(npm install flatpickr) we use a flatpickr install in the project and some css class globally and add a script angular.json then you add the both script and style the only work the Flatpickr.

\*The TimeZone contains IST and EST

Logic for the Components

\* we use a two way binding in the angular [(ngModel)]=”workDateForm” if we use [(ngModel)] we should give a name=”workDateForm” its all give in the hlml page .

\*The workDateFrom:any=’’: line is decalring and initializing the workdateFrom property in your component.It sets the initial value of workDateFrom to an empty string

\*workDateFrom:this.workDateFrom the value bind the data ,What the value user give in the input,

\*TimeZone value also we need [(ngModel)]=”TimeZone” and pass the initial value of timezone:string=’’ empty string

We get time Zone value in this .timeZone.

ngOnInit()

\* These method is first load in the screen, in the ngOnIt method we use for call the jquery in the component

var s = document.createElement("script");

s.type = "text/javascript";

s.src = "../assets/js/quintessenceScript.js";

this.elementRef.nativeElement.appendChild(s);

The code you provided is using the JavaScript document.createElement() method to create a new <script> element dynamically.

\*var s = document.createElement("script") these line create the <script>element and assign it to the variable s,javascript file dynamically include in the web page.

\* s.type = "text/javascript"; the line set attribute of the <script> element to "text/javascript"

\*this.elementRef.nativeElement.appendChild(s); it used access the native html element in the componnets template,I allows to interact With DOM directly.

\*appendChild(s),your effecitenly added the<script>element to the component html element in the DOM.one added the,browser will interpt the <script> tag and load the javascript file specified by the src attribute.

(Document object Model changing the content of an HTML element)

\* Then we store a both value in the

this.data = {

workDateFrom: this.workDateFrom,

timeZone: this.timeZone

};

Service Page

After we create a service page for interacting the data with web API.The service name is production header service ,first we import in component

import { Injectable } from '@angular/core';

import { HttpClient, HttpEvent, HttpHeaders } from '@angular/common/http';

import { BehaviorSubject, Observable } from 'rxjs';

\*Then in the constractor we create a class for httpclient (private http:HttpClient)

\*The HttpClient class allows you to send HTTP requests (such as GET, POST, PUT, DELETE) and handle the responses in an asynchronous manner. It provides methods like get(), post(), put(), delete(), etc..

import { BehaviorSubject, Observable } from 'rxjs'; part of the RxJS library, which is a reactive programming library for JavaScript

\*BehaviourSubject observable that stores the current value and emits it to any new subscribers(Store a this.workDatrFrom ,this.TimeZone value Stored and emitts a new subsciper like component)

BehaviorSubject is commonly used to manage and share state across different components or services in an application.

\*BehaviorSubject and Observable are classes that provide different functionalities for managing and observing streams of data. They are not directly related to getter and setter methods.

\*BehaviorSubject(store data and emitt to new subscribe)

\*Observables can emit multiple values over time and provide powerful operators for transforming, combining, and filtering data streams.

\*private \_data = new BehaviorSubject<object>({}); create class

new BehaviorSubject<object>({}): This part initializes the \_data variable with a new instance of BehaviorSubject<object>. The BehaviorSubject is a type of Subject in RxJS.

<object>: It specifies the type of data that the BehaviorSubject will emit. In this case, it is set to object, indicating that the BehaviorSubject will emit objects.

({}): It provides an initial value for the BehaviorSubject. In this case, an empty object {} is passed as the initial value. This means that when you subscribe to \_data, it will emit an empty object as the initial value.

\* data$: Observable<Object> = this.\_data.asObservable();

\* $ is a convention often used in Angular to indicate that the property is an observable.

getprocessProdution(workDateFrom: any, timeZone: any): Observable<any> {}

\* Observable is a class provided by the RxJS library that represents a stream of values over time. It is commonly used for handling asynchronous operations, such as making HTTP requests or working with event-driven data.

\* Asynchronous operations are typically used for tasks that may take some time to complete, such as making HTTP requests, reading/writing to a database, or waiting for user input.(Non-blocking,Reactive programming,Error handling).

\* When I mentioned "working with event-driven data" in the context of observables in Angular, I was referring to scenarios where you have data that is updated or emitted in response to events or changes in the application.

\* Get API data in the WebApi and show in the table ,but the table in other componet the component name is productiontable component how we go to the other componet in angular ,we import (import { Router } from '@angular/router'; ) and create class in the constructor private router: Router

\*this.produtionTableComponent.getProcessProductionResponse();

this.router.navigate(['/prodution-table']);

ProductionTableComponent

\*this.produtionTableComponent.getProcessProductionResponse();

getProcessProductionResponse() {

this.produtionService.getprocessProdution(this.work, this.fromdate).subscribe(response => {

console.log('final'+ JSON.stringify(response));

this.table = response;

});

}

The purpose of this method is to make an HTTP request to the server to fetch process production data.

Subscribe()Methods takes callback function as argument,which well be execute when the response is received from the serever

\*In the callback function, the Response parameter represents the data returned by the server in response to the HTTP request.

\*The this.table=response statement assigns the received response data to the table property of the component or service. It is likely that the table property is used to store and manage the process production data in the component or service.

Now we fill a data in the table

<th scope="col" \*ngFor="let columnName of getColumnNames()"

[hidden]="!isColumnVisible(columnName)" class="th-w180">

{{ columnName }}

</th>

We use a \*ngFor = \*ngFor="let columnName of getColumnNames()" : This directive iterates over an array of column names returned by the getcolumnNames**()** method in the component. It dynamically generates a **<th>** element for each column name in the array[hidden]="!isColumnVisible(columnName)": This attribute binding is used to conditionally hide or show the header cell based on the result of the iscolumnvisible**()** method in the component. The method likely returns a boolean value indicating whether the column should be visible or not.

\***{{** columnName **}}**: This interpolation syntax is used to display the value of the columnName variable within the header cell. It will show the actual column name for each iteration.

\*It starts by initializing an array called Visiblecolumns with initial column names that are always visible (**['Account', 'Username']** in this case).

Then, it iterates over each row in the **this.table** data.

getColumnNames(): string[] {

const visibleColumns: string[] = ['Account', 'Username'];

for (const row of this.table) {

const keys = Object.keys(row);

for (const key of keys) {

if (!visibleColumns.includes(key)) {

visibleColumns.push(key);

}

}

}

const tableColumns = visibleColumns.filter(column => !this.selectedColumns[column]);

this.columnNamesService.setColumnNames(tableColumns);

return tableColumns;

}

isColumnVisible(column: string): boolean {

return !this.selectedColumns.includes(column);

}

\* if (!visibleColumns.includes(key)) {

visibleColumns.push(key);

}

It checks if each key is already present in the visibleColumns array using the **includes()** method. If the key is not present, it adds it to the visibleColumns array using push**()**.

For each row, it retrieves all the keys (property names) using **Object.keys(row)**.

**<**ng-container \*ngIf="table.length > 0; else noRecords">

<tr \*ngFor="let row of table">

<td>

<ng-container \*ngIf="table.length > 0; else noRecords">: This condition checks if the **table** array has any records. If it has records, the content inside the **<ng-container>** will be rendered; otherwise, it will render the content defined in the **noRecords** template (not shown in the provided code).

\*<tr \*ngFor="let row of table">: This directive iterates over each row in the table array and creates a table row (<tr>) for each row.

\*<td \*ngFor="let columnName of getColumnNames()" [hidden]="!isColumnVisible(columnName)">: This directive creates a table cell for each columnName in the getColumnNames() array. The **[hidden]** attribute dynamically hides the cell if the column is not visible, determined by the isColumnVisible() method.

Prefference Component------The production-header-section component have a setting button If user click the button they only show what colums is present in the table that column shoe in model--IF user decide which column they want if user click column name and click a confrim that column not visible in table

We create separate service page for these component

\*Column-names service

<div class="ui-tools mt-2">

<ul id="jPagination" class="ui-tools-list"></ul>

<div class="ui-tools-preferences">

<button class="ui\_trigger-btn" aria-label="Preferences" data-bs-toggle="modal"

data-bs-target="#preferencesModal" (click)="getColumnNamess()" type="button">

<span><svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 16 16" focusable="false" aria-hidden="true">

</svg></span>

</button>

</div> when user click the setting button #preferencesModal globally we call preferencemodal id and we create a method in production-header-service component

getColumnNamess() {

this.preferencesmodalComponent.getProcessColumn();

}

First import { PreferencesmodalComponent } from '../preferencesmodal/preferencesmodal.component'; and create a class for the preference modal component

private preferencesmodalComponent: PreferencesmodalComponent

We go to the preferencemodalcomponent getProcessColumn()

getProcessColumn(): string[] {

console.log("column split-------" + this.columnNamesService.getColumnNames1().toString().split(','));

this.columnssplit = this.columnNamesService.getColumnNames1().toString().split(',');

return this.columnssplit;

}

This line logs the column names retrieved from the columnNamesService after splitting them by a comma (','). It helps to debug and verify the retrieved column names.

this.columnssplit = this.columnNamesService.getColumnNames1().toString().split(',');: This line assigns the split column names to the columnssplit variable. getColumnNames1() is a method from the columnNamesService that returns the column names as a string. The string is then split into an array using the comma (',') as the separator.

This is the method in preference modal component we stored a column value in the this.columnssplit and go to the columnNamesService we have a method in getColumnNames1().

return this.columnNames;

}

getColumnNames1(): any[] {

console.log('Value\_\_\_\_\_\_\_\_\_\_\_\_>'+ JSON.stringify(this.columnNames));

return this.columnNames;

}