import { Directive, ViewContainerRef } from '@angular/core';

@Directive({

  selector: '[dynamicComponent]'

})

export class DynamicComponentDirective {

  constructor(public viewContainerRef: ViewContainerRef) { }

}

import { ButtonComponent } from '@/visuals/components/button/button.component';

import { CheckboxComponent } from '@/visuals/components/checkbox/checkbox.component';

import { CurrencyComponent } from '@/visuals/components/currency/currency.component';

import { DateComponent } from '@/visuals/components/date/date.component';

import { DividerComponent } from '@/visuals/components/divider/divider.component';

import { InputTextComponent } from '@/visuals/components/input-text/input-text.component';

import { LiteralComponent } from '@/visuals/components/literal/literal.component';

import { MarginComponent } from '@/visuals/components/margin/margin.component';

import { NewlineComponent } from '@/visuals/components/newline/newline.component';

import { NumberComponent } from '@/visuals/components/number/number.component';

import { PercentComponent } from '@/visuals/components/percent/percent.component';

import { SelectComponent } from '@/visuals/components/select/select.component';

import { TextareaComponent } from '@/visuals/components/textarea/textarea.component';

import { Injectable, ComponentFactoryResolver, Type } from '@angular/core';

// Weitere Komponenten hier importieren

@Injectable({

  providedIn: 'root'

})

export class ComponentFactoryService {

  private components = new Map<string, Type<any>>([

    ['inputText', InputTextComponent],

    ['select', SelectComponent],

    ['checkbox', CheckboxComponent],

    ['currency', CurrencyComponent],

    ['percent', PercentComponent],

    ['button', ButtonComponent],

    ['number', NumberComponent],

    ['date', DateComponent],

    ['textarea', TextareaComponent],

    ['literal', LiteralComponent],

    ['newline', NewlineComponent],

    ['margin', MarginComponent],

    ['divider', DividerComponent],

    // Weitere Komponenten hier registrieren

  ]);

  constructor(private resolver: ComponentFactoryResolver) {}

  getComponentFactory(type: string) {

    const component = this.components.get(type);

    if (!component) {

      throw new Error(`Component of type ${type} not found`);

    }

    return this.resolver.resolveComponentFactory(component);

  }

}

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

import {EnvironmentService} from '@/\_services/environment.service';

import {DataService} from '@/\_services/data.service';

import {ComponentService} from "@/\_services/component.service";

import {AuthenticationService} from "@/\_services/authentication.service";

import {VorgangService} from "@/\_services/vorgang.service";

import {PersonService} from "@/\_services/person.service";

import {SessionService} from "@/\_services/session.service";

import {BautraegerService} from "@/\_services/bautraeger-service";

import { ControlObject, IFormControl } from '@/core/classes/ibase-component';

import { Router } from '@angular/router';

import { DialogResultButton } from '@/\_models/dialog-data';

export interface ExtendedControlObject extends IFormControl {

  cpType?: string;

  cpWidth?: string;

  // save as string

  // cpItems?: string;

  cpItemProvider?: string;

  cpItemProviderArg?: string;

  cpValidators?: string;

  cpOnValueChange?: string;

  cpAfterValueChanged?: string;

  cpDisabled?: string;

  cpSize?: string;

  cpLevel?: string;

}

@Injectable({

  providedIn: 'root',

})

export class codeGeneratorService {

  elementList : ExtendedControlObject = {};

  private dragSrcEl: any;

  /\*\*

   \* Constructor for generic data service

   \*/

  constructor(

    private ss: SessionService,

    private cs: ComponentService,

    private as: AuthenticationService,

    private ds: DataService,

    private env: EnvironmentService,

    public router: Router,

  ) {

    this.loadElementList();

  }

   // Lade elementList aus dem Local Storage

   loadElementList(): void {

    const savedElementList = localStorage.getItem('elementList');

    if (savedElementList) {

      this.elementList = JSON.parse(savedElementList);

    }

  }

  // Speichere elementList im Local Storage

  saveElementList(): void {

    localStorage.setItem('elementList', JSON.stringify(this.elementList));

  }

  // Füge ein neues Element hinzu

  addElementToView(type: string,key? : string , width? : string, size?: string): void {

    console.log(key);

    console.log('type',type);

    if (this.elementList == null) {

      this.elementList = {};

    }

    if(type === "literal"){

      this.elementList[key] = {

        cpWidth : width,

        cpType : type,

      };

    }else{

      this.elementList[key] = {

        label: key,

        disabled: true,

        cpWidth : width,

        cpType : type,

        cpSize : size,

        cpLevel : 2

      };

    }

    this.saveElementList();

  }

  reloadComponent() {

    this.router.navigateByUrl('/dummy-route', { skipLocationChange: true }).then(() => {

      this.router.navigate(['playground/tools/mainView']);

    });

  }

  removeItem(key: string): void {

    delete this.elementList[key];

    this.saveElementList();

    this.reloadComponent();

  }

  removeAllItem(): void {

    this.ss.confirm($localize`Hiermit wird Alle Felder entfernen. Soll dies ausgeführt werden?`)

      .subscribe(result => {

        if (+result.btn === DialogResultButton.yes) {

          this.elementList = {};

          this.saveElementList();

          this.reloadComponent();

        }

      });

  }

  moveItem(sourceIndex: number, targetIndex: number): void {

    const keys = Object.keys(this.elementList);

    const [movedItem] = keys.splice(sourceIndex, 1);

    keys.splice(targetIndex, 0, movedItem);

    const newElementList = {};

    keys.forEach((key, index) => {

      newElementList[key] = {

        ...this.elementList[key],

        Position: index

      };

    });

    this.elementList = newElementList;

    this.saveElementList();

  }

  changeElementPosition(key: string, newPosition: number): void {

    newPosition  = newPosition -1;

    const keys = Object.keys(this.elementList);

    const oldPosition = keys.indexOf(key);

    if (oldPosition === -1) {

      console.error(`Key ${key} not found in element list.`);

      this.ss.info(`Key ${key} not found in element list.`);

      return;

    }

    if (newPosition < 0 || newPosition >= keys.length) {

      console.error(`New position ${newPosition} is out of bounds.`);

      this.ss.info(`New position ${newPosition + 1} is out of bounds.`);

      return;

    }

    // Entfernen des Elements an der alten Position

    const [element] = keys.splice(oldPosition, 1);

    // Einfügen des Elements an der neuen Position

    keys.splice(newPosition, 0, element);

    // Aktualisieren der elementList mit den neuen Positionen

    const newElementList = {};

    keys.forEach((key, index) => {

      newElementList[key] = {

        ...this.elementList[key],

        Position: index

      };

    });

    this.elementList = newElementList;

  }

  generateFormHtml(): string {

    let formHtml = `<form #formDirective="ngForm" [formGroup]="form">\n`;

    Object.keys(this.elementList).forEach(key => {

      const element = this.elementList[key];

      formHtml += this.generateElementHtml(element, key)+ '\n';

    });

    formHtml += `</form>`;

    return formHtml;

  }

  generateElementHtml(element: any, key: string): string {

    let type = element.cpType

    switch (type) {

      case 'inputText':

        return `<app-input-text [formGroup]="form" formName="${key}" width="${element.cpWidth}"></app-input-text>`;

      case 'select':

        return `<app-select [formGroup]="form" formName="${key}" width="${element.cpWidth}"></app-select>`;

      case 'checkbox':

        return `<app-checkbox [formGroup]="form" formName="${key}" width="${element.cpWidth}"></app-checkbox>`;

      case 'currency':

        return `<app-currency [formGroup]="form" formName="${key}" width="${element.cpWidth}"></app-currency>`;

      case 'percent':

        return `<app-percent [formGroup]="form" formName="${key}" width="${element.cpWidth}"></app-percent>`;

      case 'button':

        return `<app-button [formGroup]="form" formName="${key}" width="${element.cpWidth}"></app-button>`;

      case 'number':

        return `<app-number [formGroup]="form" formName="${key}" width="${element.cpWidth}"></app-number>`;

      case 'date':

        return `<app-date [formGroup]="form" formName="${key}" width="${element.cpWidth}"></app-date>`;

      case 'textarea':

        return `<app-textarea [formGroup]="form" formName="${key}" width="${element.cpWidth}"></app-textarea>`;

      case 'literal':

        return `<app-literal width="${element.cpWidth}"></app-literal>`;

      case 'divider':

        return `<app-divider i18n-label label="${element.label}" level="${element.cpLevel}" width="${element.cpWidth}"></app-divider>`;

      // Weitere Typen hier hinzufügen

      case 'newline':

        return `<app-newline></app-newline> `;

      case 'margin':

        return `<app-margin size="${element.cpSize}"></app-margin>`;

      default:

        return '';

    }

  }

  generateTsCode(): string {

    let code = `extends AppBaseComponent {\n`;

    code += `  controls: ControlObject = {\n`;

    Object.keys(this.elementList).forEach(key => {

      const element = this.elementList[key];

      if (['divider', 'margin', 'newline', 'literal'].includes(element.cpType)) return;

      code += `    ${key}: {\n`;

      Object.keys(element).forEach(prop => {

        if (['cpType', 'cpWidth', 'cpSize', 'cpLevel','Position','key','disabled'].includes(prop)) return;

        let newProp = prop;

        let value = element[prop];

        if (value !== null) {

          if (prop.startsWith('cp')) {

            newProp = prop.charAt(2).toLowerCase() + prop.slice(3);

          }

          if (['label', 'LongLabel'].includes(newProp)) {

            value = `$localize\`${value}\``;

          } else if (typeof value === 'string') {

            value = value;

          }

          code += `      ${newProp}: ${value},\n`;

        }

      });

      code += `    },\n`;

    });

    code += `  };\n\n`;

    code += `  constructor(public sessionService: SessionService,\n`;

    code += `    public cs: ComponentService,\n`;

    code += `    public ds: DataService,\n`;

    code += `    public ips: ItemProviderService,) {\n`;

    code += `    super(sessionService, cs);\n`;

    code += `  }\n\n`;

    code += `  readFromSession() {\n`;

    code += `    return 'null';\n`;

    code += `  }\n\n`;

    code += `  writeToSession(data: any): boolean {\n`;

    code += `    return false;\n`;

    code += `  }\n\n`;

    code += `}\n`;

    return code;

  }

}

import { ComponentService } from '@/\_services/component.service';

import { DataService } from '@/\_services/data.service';

import { ItemProviderService } from '@/\_services/item-provider.service';

import { ServiceBarControl, SessionService } from '@/\_services/session.service';

import { AppBaseComponent } from '@/core/classes/app-base-component';

import { ControlObject } from '@/core/classes/ibase-component';

import {Component, ComponentRef, Injector, QueryList, Renderer2, ViewChild, ViewChildren, ViewContainerRef} from '@angular/core';

import { codeGeneratorService } from '../codeGenerator.service';

import { DynamicComponentDirective } from '../dynamic-component.directive';

import { ComponentFactoryService } from '../component-factory.service';

import { KonfigurationPuppeComponent } from '../Konfiguration-puppe/konfigurationPuppe.component';

@Component({

  selector: 'app-mainView',

  templateUrl: './mainView.component.html',

  styleUrls: ['./mainView.component.scss']

})

export class MainViewComponent extends AppBaseComponent {

  @ViewChildren(DynamicComponentDirective) dynamicComponents: QueryList<DynamicComponentDirective>;

  @ViewChild('dynamicComponentContainer', { read: ViewContainerRef, static: true }) dynamicComponentContainer: ViewContainerRef;

  private dragSrcEl: any;

  controls: ControlObject ={ ...this.cgs.elementList }

  constructor(public sessionService: SessionService,

    public cs: ComponentService,

    public ds: DataService,

    public ips: ItemProviderService,

    public cgs : codeGeneratorService,

    private componentFactoryService: ComponentFactoryService,

    private renderer: Renderer2) {

    super(sessionService, cs);

  }

  ngAfterViewInit(): void {

    this.renderElements();

  }

  readFromSession() {

    return 'null';

  }

  writeToSession(data: any): boolean {

    return false;

  }

  private renderElements(): void {

    if (this.dynamicComponentContainer) {

      this.dynamicComponentContainer.clear();

      Object.keys(this.cgs.elementList).forEach((key , index) => {

        // console.log('this.cgs.elementList[key].cpType', this.cgs.elementList[key].cpType);

        const type = this.cgs.elementList[key]?.cpType;

        const componentFactory = this.componentFactoryService.getComponentFactory(type);

        // Erstellen eines Wrappers

        const wrapper = this.renderer.createElement('ng-template');

        this.renderer.addClass(wrapper, 'wrapper');

        this.renderer.setAttribute(wrapper, 'draggable', 'true');

        // Event-Listener für Drag-and-Drop

        this.renderer.listen(wrapper, 'dragstart', (event) => this.handleDragStart(event, key, index));

        this.renderer.listen(wrapper, 'dragover', this.handleDragOver);

        this.renderer.listen(wrapper, 'drop', (event) => this.handleDrop(event, key, index));

        this.renderer.listen(wrapper, 'dragend',(event) => this.handleDragEnd(event));

        // Dynamische Komponente hinzufügen

        const componentRef = this.dynamicComponentContainer.createComponent(componentFactory);

        this.renderer.addClass(componentRef.location.nativeElement, 'wrapper');

        let isTypeNewline = type === 'newline' ? true : false;

        let isTypeMargin  = type === 'margin' ? true : false;

        let isTypeDivider = type === 'divider' ? true : false;

        let isTypeTextarea =  type === 'textarea' ? true : false;

        if(isTypeNewline){

          const textNode = this.renderer.createText('Newline');

          this.renderer.appendChild(wrapper, textNode);

        } else if(isTypeMargin){

          const textNode = this.renderer.createText('Margin');

          this.renderer.appendChild(wrapper, textNode);

          componentRef.instance.size = this.cgs.elementList[key].cpSize;

        }else{

          componentRef.instance.formGroup = this.form;

          componentRef.instance.formName = key;

          componentRef.instance.width = this.cgs.elementList[key].cpWidth;

          if(isTypeTextarea){

            componentRef.instance.rows = this.cgs.elementList[key].rows;

          }

          if(isTypeDivider){

            componentRef.instance.level = this.cgs.elementList[key].cpLevel;

          }

        }

        this.renderer.setStyle(componentRef.location.nativeElement, 'pointer-events', 'none');

        // Fügen Sie die Komponente in den Wrapper ein

        this.renderer.appendChild(wrapper, componentRef.location.nativeElement);

        // Fügen Sie den Wrapper in den ViewContainerRef ein

        this.renderer.appendChild(this.dynamicComponentContainer.element.nativeElement, wrapper);

        // Klick-Ereignis für den Wrapper hinzufügen

        const popupData: any ={

          ...this.cgs.elementList[key],

          key : key,

          Position : index +1

        };

        this.renderer.listen(wrapper, 'click', () => {

          this.showPopup($localize`Einstellungen für das Feld : ${key}`, KonfigurationPuppeComponent, popupData).subscribe(\_result => {

            // Logik nach dem Schließen des Popups (falls nötig)

          });

        });

        componentRef.changeDetectorRef.detectChanges(); // Sicherstellen, dass die Änderungen erkannt werden

      });

      // console.log(this.controls);

    }

  }

  handleDragStart(event: DragEvent, key: string, index: number): void {

    this.dragSrcEl = event.target;

    event.dataTransfer!.effectAllowed = 'move';

    event.dataTransfer!.setData('text/plain', index.toString());

    setTimeout(() => {

      this.renderer.setStyle(this.dragSrcEl, 'display', 'none');

    }, 0);

  }

  handleDragOver(event: DragEvent): void {

    event.preventDefault();

    event.dataTransfer!.dropEffect = 'move';

  }

  handleDrop(event: DragEvent, key: string, index: number): void {

    event.stopPropagation();

    const sourceIndex = Number(event.dataTransfer!.getData('text/plain'));

    if (sourceIndex !== index) {

      this.cgs.moveItem(sourceIndex, index);

    }

  }

  handleDragEnd(event: DragEvent): void {

    if (this.dragSrcEl) {

      this.cgs.reloadComponent();

      this.dragSrcEl = null;

    }

  }

}

.main-view {

  display: flex;

  justify-content: space-between; /\* Platz zwischen Hauptinhalt und Menü \*/

}

.main-content {

  flex: 1;

  overflow: auto;

}

app-elementMenu {

  flex: 0 0 250px; /\* Stellt sicher, dass das Menü eine feste Breite hat \*/

  order: 2; /\* Stellt sicher, dass das Menü an zweiter Stelle steht, also rechts \*/

}

.wrapper {

  padding-bottom: 22px;

  cursor: pointer !important; /\* Cursor anzeigen bei Hover \*/

  &:hover {

    background-color: #d2cfcf; /\* Etwas dunkleres Grau bei Hover \*/

    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1); /\* Leichter Schatten bei Hover \*/

  }

}

<div class="main-view">

  <!-- Der restliche Hauptinhalt -->

  <form class="main-content" #formDirective="ngForm" [formGroup]="form">

      <!-- Hauptinhalt hier -->

      <app-margin size ="small"></app-margin>

      <app-divider i18n-label label="Hauptansicht" level="2" width="4"></app-divider>

      <app-margin size ="big"></app-margin>

      <div #dynamicComponentContainer></div>

  </form>

  <app-elementMenu></app-elementMenu>

</div>

import { ComponentService } from '@/\_services/component.service';

import { DataService } from '@/\_services/data.service';

import { ItemProviderService } from '@/\_services/item-provider.service';

import { ServiceBarControl, SessionService } from '@/\_services/session.service';

import { AppBaseComponent } from '@/core/classes/app-base-component';

import { ControlObject } from '@/core/classes/ibase-component';

import {Component, OnInit} from '@angular/core';

import { Validators } from '@angular/forms';

import { codeGeneratorService } from '../codeGenerator.service';

import { MatDialogRef } from '@angular/material/dialog';

import { PopupComponent } from '@/core/components/popup/popup.component';

@Component({

  selector: 'app-konfigurationPuppe',

  templateUrl: './konfigurationPuppe.component.html',

  styleUrls: ['./konfigurationPuppe.component.scss']

})

export class KonfigurationPuppeComponent extends AppBaseComponent {

  isPopup = true;

  servicebarDef: ServiceBarControl[] = [

    {

      label: 'Einstellungen speichern',

      icon: 'add',

      position: 'right',

      method: () => {

        this.UptadeElement();

      },

    },

    {

      label: $localize`Feld  entfernen`,

      icon: 'delete',

      method: () => {

       this.deleteElement()

      }

    },

    {

      label: $localize`Schließen`,

      icon: 'cancel',

      name: 'btnCancelAll',

      method: () => {

        this.dialogRef.close();

      }

    },

  ];

  get isTypeSelect(): boolean {

    return this.readData.cpType === 'select' ? true : false;

  }

  get isTypeNewline(): boolean {

    return this.readData?.cpType === 'newline' ? true : false;

  }

  get isTypeMargin(): boolean {

    return this.readData?.cpType === 'margin' ? true : false;

  }

  get isTypeLiteral(): boolean {

    return this.readData?.cpType === 'literal' ? true : false;

  }

  get isTypeDivider(): boolean {

    return this.readData?.cpType === 'divider' ? true : false;

  }

  get isTypeTextarea(): boolean {

    return this.readData?.cpType === 'textarea' ? true : false;

  }

  controls: ControlObject =

  {

    ctrKey : {

      label: $localize` control Key`,

    },

    ctrPosition : {

      label: $localize` control Position`,

      validators: [ Validators.max(Object.keys(this.cgs.elementList).length), Validators.min(0)],

    },

    ctrWidth: {

      label: $localize` control Width`,

      validators: [ Validators.max(4), Validators.min(1)]

    },

    ctrSize: {

      label: $localize` Margin Size`,

      longLabel: $localize`zur Auswahl: gibt 'small' | 'default' | 'big'`

    },

    ctrlabel : {

      label: $localize` control Label`,

    },

    ctrlongLabel: {

      label: $localize` control LongLabel`,

    },

    ctrmin : {

      label: $localize` control Min`,

    },

    ctrmax: {

      label: $localize` control Max`,

    },

    ctrrows: {

      label: $localize` control Rows`,

    },

    ctrlevel: {

      label: $localize` control level`,

    },

    ctritemProvider : {

      label: $localize` control ItemProvider`,

    },

    ctritemProviderArg: {

      label: $localize` control ItemProviderArg`,

    },

    ctrnoEmptyItem : {

      label: $localize` control NoEmptyItem`,

    },

    ctrvalidators: {

      label: $localize` control Validators`,

    },

    ctrcssClass : {

      label: $localize` control CssClass`,

    },

    ctrdisabled: {

      label: $localize` control Disabled`,

    },

    ctronValueChange : {

      label: $localize` control OnValueChange`,

    },

    ctrafterValueChanged: {

      label: $localize` control AfterValueChanged`,

    },

  };

  constructor(public sessionService: SessionService,

    public cs: ComponentService,

    public ds: DataService,

    public ips: ItemProviderService,

    public cgs : codeGeneratorService,

    public dialogRef: MatDialogRef<PopupComponent<KonfigurationPuppeComponent>>) {

    super(sessionService, cs);

  }

  ngAfterViewInit(): void {

    this.setFormValue('ctrKey', this.readData?.key);

    this.setFormValue('ctrPosition', this.readData?.Position);

    this.setFormValue('ctrWidth', this.readData?.cpWidth);

    this.setFormValue('ctrSize', this.readData?.cpSize);

    this.setFormValue('ctrlevel', this.readData?.cpLevel);

    this.setFormValue('ctrrows', this.readData?.rows);

    this.setFormValue('ctrlabel', this.readData?.label);

    this.setFormValue('ctrlongLabel', this.readData?.LongLabel);

    this.setFormValue('ctrmin', this.readData?.min);

    this.setFormValue('ctrmax', this.readData?.max);

    this.setFormValue('ctritemProvider', this.readData?.cpItemProvider);

    this.setFormValue('ctritemProviderArg', this.readData?.cpItemProviderArg);

    this.setFormValue('ctrnoEmptyItem', this.readData?.noEmptyItem);

    this.setFormValue('ctrvalidators', this.readData?.cpValidators);

    this.setFormValue('ctrcssClass', this.readData?.cssClass);

    this.setFormValue('ctrdisabled', this.readData?.cpDisabled);

    this.setFormValue('ctronValueChange', this.readData?.cpOnValueChange);

    this.setFormValue('ctrafterValueChanged', this.readData?.cpAfterValueChanged);

  }

  readFromSession() {

    return 'null';

  }

  writeToSession(data: any): boolean {

    return false;

  }

  UptadeElement(): void {

    // this.cgs.elementList[this.readData.key].Position = this.getFormValue('ctrPosition');

    this.cgs.elementList[this.readData.key].cpWidth = this.getFormValue('ctrKey');

    this.cgs.elementList[this.readData.key].cpWidth = this.getFormValue('ctrWidth');

    this.cgs.elementList[this.readData.key].cpSize = this.getFormValue('ctrSize');

    this.cgs.elementList[this.readData.key].cpLevel = this.getFormValue('ctrlevel');

    this.cgs.elementList[this.readData.key].rows = this.getFormValue('ctrrows');

    this.cgs.elementList[this.readData.key].label =  this.getFormValue('ctrlabel');

    this.cgs.elementList[this.readData.key].LongLabel =  this.getFormValue('ctrlongLabel');

    this.cgs.elementList[this.readData.key].min =  this.getFormValue('ctrmin');

    this.cgs.elementList[this.readData.key].max = this.getFormValue('ctrmax');

    this.cgs.elementList[this.readData.key].cpItemProvider =  this.getFormValue('ctritemProvider');

    this.cgs.elementList[this.readData.key].cpItemProviderArg = this.getFormValue('ctritemProviderArg');

    this.cgs.elementList[this.readData.key].noEmptyItem =  this.getFormValue('ctrnoEmptyItem');

    this.cgs.elementList[this.readData.key].cpValidators =  this.getFormValue('ctrvalidators');

    this.cgs.elementList[this.readData.key].cssClass =  this.getFormValue('ctrcssClass');

    this.cgs.elementList[this.readData.key].cpDisabled =  this.getFormValue('ctrdisabled');

    this.cgs.elementList[this.readData.key].cpOnValueChange =  this.getFormValue('ctronValueChange');

    this.cgs.elementList[this.readData.key].cpAfterValueChanged = this.getFormValue('ctrafterValueChanged');

    this.cgs.changeElementPosition(this.readData.key,this.getFormValue('ctrPosition'));

    this.renameKey(this.readData.key,this.getFormValue('ctrKey'));

    this.cgs.saveElementList();

    console.log(this.cgs.elementList);

    setTimeout(() => {

     this.markAsPristine();

     // this.dialogRef.close();

     this.cgs.reloadComponent();

    }, 50);

  }

  deleteElement(){

    if (this.readData.key in this.cgs.elementList) {

        this.cgs.removeItem(this.readData.key);

      this.dialogRef.close()

    }

  }

  renameKey(oldKey: string, newKey: string): void {

    if (oldKey !== newKey && this.cgs.elementList[oldKey]) {

      const elements = { ...this.cgs.elementList }; // Kopiere die gesamte Liste

      const keys = Object.keys(elements);

      const index = keys.indexOf(oldKey);

      // Erstelle das neue Element mit dem neuen Schlüssel

      this.cgs.elementList = {}; // Leere die Liste

      keys.forEach((key, i) => {

        if (i === index) {

          this.cgs.elementList[newKey] = { ...elements[oldKey]};

          this.readData.key = newKey;

        } else {

          this.cgs.elementList[key] = elements[key];

        }

      });

    }

  }

}

<form #formDirective="ngForm" [formGroup]="form">

    <app-divider i18n-label label="Standardkonfiguration" level="2" width="4"></app-divider>

    <app-margin size="big"></app-margin>

    <app-input-text [formGroup]="form" formName="ctrKey" width="2"></app-input-text>

    <app-margin size="big"></app-margin>

    <app-number [formGroup]="form" formName="ctrPosition" width="2"></app-number>

    <app-number \*ngIf="!isTypeMargin && !isTypeNewline" [formGroup]="form" formName="ctrWidth" width="2"></app-number>

    <app-input-text \*ngIf="isTypeMargin" [formGroup]="form" formName="ctrSize" width="2"></app-input-text>

    <app-margin size="big"></app-margin>

    <app-number \*ngIf="isTypeTextarea" [formGroup]="form" formName="ctrrows" width="2"></app-number>

    <app-margin size="big"></app-margin>

    <app-number \*ngIf="isTypeDivider" [formGroup]="form" formName="ctrlevel" width="2"></app-number>

    <app-input-text \*ngIf="!isTypeLiteral && !isTypeMargin && !isTypeNewline" [formGroup]="form" formName="ctrlabel" width="2"></app-input-text>

    <ng-container \*ngIf="!isTypeDivider && !isTypeLiteral && !isTypeMargin && !isTypeNewline">

        <app-input-text [formGroup]="form" formName="ctrlongLabel" width="2"></app-input-text>

        <app-margin size="big"></app-margin>

        <app-input-text [formGroup]="form" formName="ctrmin" width="2"></app-input-text>

        <app-input-text [formGroup]="form" formName="ctrmax" width="2"></app-input-text>

        <app-margin size="big"></app-margin>

        <app-input-text [formGroup]="form" formName="ctrcssClass" width="2"></app-input-text>

        <app-input-text [formGroup]="form" formName="ctrdisabled" width="1"></app-input-text>

        <div \*ngIf="isTypeSelect">

            <app-divider i18n-label label="Einstellungen für App-Select" level="2" width="4"></app-divider>

            <app-input-text [formGroup]="form" formName="ctritemProvider" width="2"></app-input-text>

            <app-input-text [formGroup]="form" formName="ctritemProviderArg" width="1"></app-input-text>

            <app-checkbox [formGroup]="form" formName="ctrnoEmptyItem" width="1"></app-checkbox>

            <app-margin size="big"></app-margin>

        </div>

        <app-divider i18n-label label="Funktionen" level="2" width="4"></app-divider>

        <app-input-text [formGroup]="form" formName="ctrvalidators" width="4"></app-input-text>

        <app-margin size="big"></app-margin>

        <app-textarea [formGroup]="form" formName="ctronValueChange" width="4"></app-textarea>

        <app-margin size="big"></app-margin>

        <app-textarea [formGroup]="form" formName="ctrafterValueChanged" width="4"></app-textarea>

        <app-margin size="big"></app-margin>

    </ng-container>

</form>

import { ComponentService } from '@/\_services/component.service';

import { DataService } from '@/\_services/data.service';

import { ItemProviderService } from '@/\_services/item-provider.service';

import { ServiceBarControl, SessionService } from '@/\_services/session.service';

import { AppBaseComponent } from '@/core/classes/app-base-component';

import { ControlObject } from '@/core/classes/ibase-component';

import {Component, OnInit} from '@angular/core';

import { codeGeneratorService } from '../codeGenerator.service';

import { AddElementComponent } from '../add-element/addElement.component';

import { CodeViewPopupComponent } from '../codeViewPopup/codeViewPopup.component';

@Component({

  selector: 'app-elementMenu',

  templateUrl: './elementMenu.component.html',

  styleUrls: ['./elementMenu.component.scss']

})

export class ElementMenuComponent  extends AppBaseComponent {

  servicebarDef: ServiceBarControl[] = [

    {

      label: 'Quellcode erzeugen',

      icon: 'add',

      position: 'right',

      method: (self: ElementMenuComponent) => {

        this.showPopup($localize`Ansicht von HTML- und TypeScript-Code`, CodeViewPopupComponent).subscribe(\_result => {

        });

      },

    },

    {

      label: $localize`Alle Felder entfernen`,

      icon: 'delete',

      method: () => {

       this.cgs.removeAllItem();

      }

    },

  ]

  controls: ControlObject =

  {

    inputText: {

      label: $localize`input`,

      disabled : true,

    },

    select: {

      label: $localize`select`,

      disabled : true,

    },

    checkbox: {

      label: $localize`checkbox`,

      disabled : true,

    },

    currency: {

      label: $localize`currency`,

      disabled : true,

    },

    percent: {

      label: $localize`percent`,

      disabled : true,

    },

    button: {

      label: $localize`button`,

      disabled : true,

    },

    number: {

      label: $localize`number`,

      disabled : true,

    },

    date: {

      label: $localize`date`,

      disabled : true,

    },

    textarea: {

      label: $localize`textarea`,

      disabled : true,

    },

    literal: {

      label: $localize`literal`,

      disabled : true,

    },

    // newline: {

    //   label: $localize`newline`,

    //   disabled : true,

    // },

    divider: {

      label: $localize`divider`,

      disabled : true,

    },

    // margin: {

    //   label: $localize`margin`,

    //   disabled : true,

    // },

  };

  constructor(public sessionService: SessionService,

    public cs: ComponentService,

    public ds: DataService,

    public ips: ItemProviderService,

    public cgs : codeGeneratorService) {

    super(sessionService, cs);

  }

  readFromSession() {

    return 'null';

  }

  writeToSession(data: any): boolean {

    return false;

  }

  openPopup(\_type: string): void {

    this.showPopup($localize`${\_type}-Komponente einfügen`, AddElementComponent,

      \_type).subscribe(\_result => {

    });

  }

}

.element-menu-container {

    height: 100%; /\* Füllt die gesamte Höhe des übergeordneten Containers \*/

    width: 250px; /\* Setzen Sie die gewünschte Breite \*/

    overflow-y: auto; /\* Ermöglicht das vertikale Scrollen \*/

  }

  .element-menu {

    padding: 10px;

    box-sizing: border-box;

  }

  .menu-item {

    background-color: #eceaea; /\* Leichtes Grau als Hintergrundfarbe \*/

    padding: 10px;

    margin-bottom: 10px;

    border-radius: 5px;

    transition: background-color 0.3s, box-shadow 0.3s; /\* Für weiche Übergänge \*/

    cursor: pointer !important; /\* Zeigt den Klickbereich an \*/

    &:hover {

      background-color: #d2cfcf; /\* Etwas dunkleres Grau bei Hover \*/

      box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1); /\* Leichter Schatten bei Hover \*/

    }

    /\* Stellt sicher, dass Kinderkomponenten das click-Event nicht verhindern \*/

    app-input-text,

    app-select,

    app-checkbox,

    app-currency,

    app-percent,

    app-button,

    app-number,

    app-date,

    app-textarea {

      pointer-events: none !important; /\* Ignoriere Pointer-Events auf Kinderkomponenten \*/

    }

  }

<form #formDirective="ngForm" [formGroup]="form" class="app--bautraeger">

    <div class="element-menu-container">

        <div class="element-menu">

            <app-divider i18n-label label="Eingaben-Komponente" level="2" width="1"></app-divider>

            <div class="menu-item" (click)="openPopup('inputText')">

                <app-input-text [formGroup]="form" formName="inputText"></app-input-text>

            </div>

            <app-margin size="big"></app-margin>

            <div class="menu-item" (click)="openPopup('select')">

                <app-select [formGroup]="form" formName="select"></app-select>

            </div>

            <app-margin size="big"></app-margin>

            <div class="menu-item" class="menu-item" (click)="openPopup('checkbox')">

                <app-checkbox [formGroup]="form" formName="checkbox"></app-checkbox>

            </div>

            <app-margin size="big"></app-margin>

            <div class="menu-item" class="menu-item" (click)="openPopup('currency')">

                <app-currency [formGroup]="form" formName="currency"></app-currency>

            </div>

            <app-margin size="big"></app-margin>

            <div class="menu-item" class="menu-item" (click)="openPopup('percent')">

                <app-percent [formGroup]="form" formName="percent"></app-percent>

            </div>

            <app-margin size="big"></app-margin>

            <div class="menu-item" class="menu-item" (click)="openPopup('button')">

                <app-button [formGroup]="form" formName="button"></app-button>

            </div>

            <app-margin size="big"></app-margin>

            <div class="menu-item" class="menu-item" (click)="openPopup('number')">

                <app-number [formGroup]="form" formName="number"></app-number>

            </div>

            <app-margin size="big"></app-margin>

            <div class="menu-item" class="menu-item" (click)="openPopup('date')">

                <app-date [formGroup]="form" formName="date"></app-date>

            </div>

            <app-margin size="big"></app-margin>

            <div class="menu-item" class="menu-item" (click)="openPopup('textarea')">

                <app-textarea [formGroup]="form" formName="textarea" width="1"></app-textarea>

            </div>

            <app-margin size="big"></app-margin>

            <app-divider i18n-label label="Layout-Komponente" level="2" width="1"></app-divider>

            <app-margin size="big"></app-margin>

            <div class="menu-item" class="menu-item" (click)="openPopup('divider')">

                <!-- <app-divider [formGroup]="form" formName="divider"></app-divider> -->

                Titelbereich

            </div>

            <app-margin size="big"></app-margin>

            <div class="menu-item" class="menu-item" (click)="openPopup('literal')">

                <!-- <app-literal [formGroup]="form" formName="literal"></app-literal> -->

                Platzhalter

            </div>

            <app-margin size="big"></app-margin>

            <div class="menu-item" class="menu-item" (click)="openPopup('newline')">

                <!-- <app-newline></app-newline> -->

                Zeilenwechsel

            </div>

            <app-margin size="big"></app-margin>

            <div class="menu-item" class="menu-item" (click)="openPopup('margin')">

                Abstandshalter

                <!-- <app-margin [formGroup]="form" formName="margin"></app-margin> -->

            </div>

            <app-margin size="big"></app-margin>

    <!-- <app-table #tableKostendeckung [formGroup]="form" formName="tableKostendeckung" width="2"></app-table> -->

    <!-- <app-table #aufgabenlisteTable

    [columnDescriptions]="columnDescriptions"

    [formGroup]="form"

    [highlightHovered]="true"

    [highlightSelection]="true"

    [tableDescription]="tableDescription"

    formName="aufgabenlisteTable"

    innerWidth="4"

    outerWidth="4"

    ></app-table> -->

        </div>

    </div>

</form>

import { ComponentService } from '@/\_services/component.service';

import { DataService } from '@/\_services/data.service';

import { ItemProviderService } from '@/\_services/item-provider.service';

import { ServiceBarControl, SessionService } from '@/\_services/session.service';

import { AppBaseComponent } from '@/core/classes/app-base-component';

import { ControlObject } from '@/core/classes/ibase-component';

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

import { codeGeneratorService } from '../codeGenerator.service';

import { MatDialogRef } from '@angular/material/dialog';

import { PopupComponent, PopupData } from '@/core/components/popup/popup.component';

import { Validators } from '@angular/forms';

import { TestComponetComponent } from '../Test-componet/testComponet.component';

@Component({

  selector: 'app-codeViewPopup',

  templateUrl: './codeViewPopup.component.html',

  styleUrls: ['./codeViewPopup.component.scss']

})

export class CodeViewPopupComponent extends AppBaseComponent {

  isPopup = true;

  servicebarDef: ServiceBarControl[] = [

    {

      label: $localize`TestKomponente anzeigen`,

      method: () => {

        this.showPopup($localize`Ansicht der TestKomponente`, TestComponetComponent).subscribe(\_result => {

          console.log(\_result);

          return this.dialogRef.afterClosed();

        });

      }

    },

    {

      label: $localize`HTML-Code kopieren`,

      method: () => {

        this.copyHTMLCode();

      }

    },

    {

      label: $localize`TS-Code kopieren`,

      method: () => {

        this.copyTsCode();

      }

    },

    {

      label: $localize`Schließen`,

      icon: 'cancel',

      name: 'btnCancelAll',

      position: 'right',

      method: () => {

        this.dialogRef.close();

      }

    },

  ];

  controls: ControlObject =

  {

    htmlCode : {

      label: $localize`HTML-Code`,

      rows: 15,

      disabled : true,

    },

    tsCode: {

      label: $localize`TypeScript-Code`,

      rows: 15,

      disabled : true,

    },

  };

  constructor(public sessionService: SessionService,

    public cs: ComponentService,

    public ds: DataService,

    public ips: ItemProviderService,

    public cgs : codeGeneratorService,

    public dialogRef: MatDialogRef<PopupComponent<CodeViewPopupComponent>>) {

    super(sessionService, cs);

  }

  ngAfterViewInit(): void {

    this.setFormValue('htmlCode', this.cgs.generateFormHtml());

    this.setFormValue('tsCode', this.cgs.generateTsCode());

  }

  readFromSession() {

    return 'null';

  }

  writeToSession(data: any): boolean {

    return false;

  }

  copyHTMLCode(): void {

    const textarea = document.createElement('textarea');

    textarea.value = this.cgs.generateFormHtml();

    document.body.appendChild(textarea);

    textarea.select();

    document.execCommand('copy');

    document.body.removeChild(textarea);

  }

  copyTsCode(): void {

    const textarea = document.createElement('textarea');

    textarea.value = this.cgs.generateTsCode();

    document.body.appendChild(textarea);

    textarea.select();

    document.execCommand('copy');

    document.body.removeChild(textarea);

  }

}

<form #formDirective="ngForm" [formGroup]="form">

    <app-divider i18n-label label="HTML-Code" level="2" width="4"></app-divider>

    <app-margin size ="big"></app-margin>

    <app-textarea rows="15" width="4" formName="htmlCode" [formGroup]="form"></app-textarea>

    <app-margin size ="big"></app-margin>

    <app-divider i18n-label label="TypeScript-Code" level="2" width="4"></app-divider>

    <app-textarea rows="15" width="4" formName="tsCode" [formGroup]="form"></app-textarea>

    <app-margin size ="big"></app-margin>

</form>

import { ComponentService } from '@/\_services/component.service';

import { DataService } from '@/\_services/data.service';

import { ItemProviderService } from '@/\_services/item-provider.service';

import { ServiceBarControl, SessionService } from '@/\_services/session.service';

import { AppBaseComponent } from '@/core/classes/app-base-component';

import { ControlObject } from '@/core/classes/ibase-component';

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

import { codeGeneratorService } from '../codeGenerator.service';

import { MatDialogRef } from '@angular/material/dialog';

import { PopupComponent, PopupData } from '@/core/components/popup/popup.component';

import { Validators } from '@angular/forms';

@Component({

  selector: 'app-addElement',

  templateUrl: './addElement.component.html',

  styleUrls: ['./addElement.component.scss']

})

export class AddElementComponent extends AppBaseComponent {

  isPopup = true;

  servicebarDef: ServiceBarControl[] = [

    {

      label: 'Feld hinzufügen',

      icon: 'add',

      position: 'right',

      hideIfWSReadonly: true,

      method: (self: AddElementComponent) => {

        self.addElementToView();

      },

      name: 'btnCreate'

    },

    {

      label: $localize`Schließen`,

      icon: 'cancel',

      name: 'btnCancelAll',

      method: () => {

        this.dialogRef.close();

      }

    },

  ];

  controls: ControlObject =

  {

    ctrName : {

      label: $localize` control Name`,

      validators : Validators.required,

    },

    ctrWidth: {

      label: $localize` control Width`,

    },

    ctrSize: {

      label: $localize` Margin Size`,

      longLabel: $localize`zur Auswahl: gibt 'small' | 'default' | 'big'`

    }

  };

  constructor(public sessionService: SessionService,

    public cs: ComponentService,

    public ds: DataService,

    public ips: ItemProviderService,

    public cgs : codeGeneratorService,

    public dialogRef: MatDialogRef<PopupComponent<AddElementComponent>>) {

    super(sessionService, cs);

  }

  get isTypeNewline(): boolean {

    return this.readData === 'newline' ? true : false;

  }

  get isTypeMargin(): boolean {

    return this.readData === 'margin' ? true : false;

  }

  readFromSession() {

    return 'null';

  }

  writeToSession(data: any): boolean {

    return false;

  }

  addElementToView(): void {

    let name = this.form.get('ctrName').value;

    let width = this.form.get('ctrWidth').value;

    let size = this.form.get('ctrSize').value;

    if (name === null || name === " " || name === undefined) {

      return;

    }

     this.cgs.addElementToView(this.readData,name,width,size);

     console.log(this.cgs.elementList);

     setTimeout(() => {

      this.markAsPristine();

      // this.dialogRef.close();

      this.cgs.reloadComponent();

     }, 50);

  }

}

<form #formDirective="ngForm" [formGroup]="form">

    <div \*ngIf="isTypeNewline">

        <app-input-text [formGroup]="form" formName="ctrName" width="2"></app-input-text>

    </div>

    <div \*ngIf="isTypeMargin">

        <app-input-text [formGroup]="form" formName="ctrName" width="2"></app-input-text>

        <app-input-text [formGroup]="form" formName="ctrSize" width="2"></app-input-text>

        <app-margin size="big"></app-margin>

    </div>

    <div \*ngIf="!isTypeMargin && !isTypeNewline">

        <app-input-text [formGroup]="form" formName="ctrName" width="2"></app-input-text>

        <app-input-text [formGroup]="form" formName="ctrWidth" width="2"></app-input-text>

        <app-margin size="big"></app-margin>

    </div>

</form>