ABSCHLUSSPRÜFUNG WINTER 2024/2025

Fachinformatiker für Anwendungsentwicklung Dokumentation zur betrieblichen Projektarbeit

Code-Generators

Abgabetermin: Dillingen, den 06.12.2024

Prüfungsbewerber:

Abdulaa Mousa
Schretzheimer Str. 15
89407 Dillingen

Ausbildungsbetrieb

CPU Consulting & Software GmbH
August-Wessels-Str. 23
86156 Augsburg

28.11.2024

Inhaltsverzeichnis

1. Einleitung

Diese Dokumentation beschreibt die Entwicklung eines Code-Generators für Dialoge in einer Angular- und TypeScript-Anwendung. Das Projekt wurde im Rahmen der Abschlussprüfung für Fachinformatiker Anwendungsentwicklung durchgeführt und umfasst die Implementierung eines Tools, das verschiedene UI-Komponenten in Dialogfenster einfügt und konfiguriert.

Ziel des Projekts ist es, eine benutzerfreundliche und flexible Anwendung zu entwickeln, die es ermöglicht, Dialogfenster dynamisch zu erstellen und anzupassen. Durch die Verwendung moderner Webtechnologien wie Angular und TypeScript soll eine leistungsfähige und wartbare Lösung bereitgestellt werden. Der Code-Generator soll dabei helfen, die Effizienz bei der Entwicklung von Benutzeroberflächen zu steigern und die Fehleranfälligkeit zu reduzieren.

Im Rahmen des Projekts wurden mehrere Schlüsselkomponenten entwickelt, darunter eine Direktive zur dynamischen Platzierung von Komponenten, Services zur Verwaltung und Generierung von Code sowie eine benutzerfreundliche Oberfläche zur Konfiguration der Dialoge. Die Entwicklung erfolgte in mehreren Phasen: von der Analyse über die Planung und Implementierung bis hin zur abschließenden Testung und Dokumentation.

Diese Dokumentation gibt einen umfassenden Überblick über die Projektumsetzung, erläutert die verwendeten Technologien und Architekturen und beschreibt die einzelnen Komponenten und ihre Funktionalitäten im Detail. Darüber hinaus werden die Herausforderungen und Lösungen, die während der Entwicklung auftraten, dargestellt und eine Zeitplanung sowie ein Fazit gegeben.

2. Projektumfeld

Das Projektumfeld umfasst die bestehenden technischen und organisatorischen Rahmenbedingungen, unter denen das Projekt realisiert wurde. Das Projekt wurde als Teil der bestehenden CPU-Banking entwickelt, die bereits Angular und TypeScript als wesentliche Technologien für die Frontend-Entwicklung verwendet.

Technologischer Rahmen:

- **Angular**: Ein weit verbreitetes Framework für die Entwicklung von Single-Page-Anwendungen (SPA). Angular bietet eine modulare Architektur, die es ermöglicht, einzelne Komponenten unabhängig voneinander zu entwickeln und zu testen. Es unterstützt die Zwei-Wege-Datenbindung, wodurch Änderungen in der Benutzeroberfläche und im Datenmodell synchronisiert werden.

- **TypeScript**: Eine strikte Supersprache von JavaScript, die statische Typisierung und moderne Features wie Klassen und Module bietet. TypeScript hilft dabei, Fehler frühzeitig zu erkennen und verbessert die Wartbarkeit des Codes durch eine klare Struktur und Typensicherheit.

- **Entwicklungsumgebung**: Visual Studio Code (VS-Code) als integrierte Entwicklungsumgebung (IDE). VS-Code bietet zahlreiche Erweiterungen für

Angular und TypeScript wie Linter, Debugger und Formatierungstools, die die Entwicklung effizienter machen.

- **Testing-Frameworks**: Jasmine, Karma und Angular's TestBed

Projektumgebung:

Das Projekt wird als Modul in eine bestehende Angular/TypeScript-Anwendung integriert, die zur Unterstützung von Geschäftsprozessen dient. Die bestehende Anwendung ist eine zentrale Plattform, die verschiedene Geschäftsprozesse abdeckt und bereits eine modulare Architektur verwendet. Die Integration des neuen Moduls soll nahtlos erfolgen, ohne die bestehende Struktur zu beeinträchtigen.

Motivation und Hintergrund:

- **Bestehende Anwendung**: Die aktuelle Methode zur Erstellung von Dialogfenstern ist ineffizient und fehleranfällig. Entwickler müssen viel Zeit aufwenden, um Dialoge manuell zu konfigurieren und zu erstellen.

- **Notwendigkeit**: Ein Tool, das diesen Prozess automatisiert, reduziert die Fehleranfälligkeit und spart wertvolle Entwicklungszeit. Die Integration eines solchen Tools in die bestehende Anwendung verbessert die Effizienz und Qualität der Entwicklung.

3. Projektumsetzung

Die Umsetzung des Projekts erfolgte innerhalb von 80 Stunden. Diese Zeit umfasst die gesamte Entwicklungsarbeit einschließlich der Analyse, Planung, Implementierung, Testung und Dokumentation. Das Projekt wurde in einem iterativen Ansatz entwickelt, bei dem die Funktionalitäten Schritt für Schritt hinzugefügt und getestet wurden.

Phasen:

1. **Analysephase**: In dieser Phase wurden die Anforderungen an das Tool ermittelt.

2. **Planungsphase / Entwurfsphase**: Basierend auf den Ergebnissen der Analysephase wurde die Architektur des neuen Tools entworfen. Es wurden die Hauptkomponenten und deren Interaktionen definiert sowie Architekturdiagramme erstellt. Diese Phase beinhaltete auch die Erstellung eines detaillierten Projektplans, der die einzelnen Entwicklungsschritte und Meilensteine festlegte.

3. **Entwicklungsphase**: In dieser Phase wurden die Hauptkomponenten des Tools implementiert. Die Entwicklung erfolgte iterativ, wobei in jeder Iteration eine funktionale Einheit implementiert und getestet wurde. Moderne Entwicklungsmethoden wie Test-Driven Development (TDD) und Continuous Integration/Continuous Deployment (CI/CD) wurden eingesetzt, um die Qualität des Codes sicherzustellen.

4. **Testphase**: Nach der Implementierung wurden umfassende Tests durchgeführt, um sicherzustellen, dass das Tool den Anforderungen entspricht und fehlerfrei funktioniert. Dazu gehörten Unit-Tests, Integrationstests. Automatisierte Tests wurden eingerichtet, um zukünftige Änderungen und Erweiterungen zu erleichtern.

5. **Fehleranalyse**: Während der Testphase aufgetretene Fehler wurden analysiert und behoben. Diese Phase umfasste die Identifikation von Fehlerursachen, die Implementierung von Korrekturen und die erneute Testung der betroffenen Bereiche.

6. **Projektdokumentation**: Abschließend wurde eine umfassende Dokumentation erstellt, die alle Phasen und Komponenten des Projekts abdeckt. Dies umfasst technische Spezifikationen, Benutzerhandbücher und Wartungsdokumente.

4. Projektziel

Ziel des Projekts ist die Erstellung einer Benutzeroberfläche in Angular und TypeScript, die als Code-Generator für Dialoge dient. Die Anwendung ermöglicht es, verschiedene UI-Komponenten aus einem Menü in ein Dialogfenster einzufügen und zu konfigurieren. Die generierten Dialogfenster werden schließlich in HTML- und TypeScript-Code umgewandelt. Dieses Tool soll Entwicklern helfen, schneller Dialoge zu designen und zu erstellen.

Detaillierte Ziele:

- **Benutzerfreundlichkeit**: Die Anwendung soll eine intuitive und benutzerfreundliche Oberfläche bieten, die es den Entwicklern ermöglicht, UI-Komponenten einfach zu konfigurieren und zu verwalten. Eine klare und verständliche Benutzerführung soll die Bedienung erleichtern.

- **Flexibilität**: Unterstützung verschiedener UI-Komponenten, die aus einem Menü ausgewählt und in ein Dialogfenster eingefügt werden können. Die Komponenten sollen leicht konfigurierbar und anpassbar sein, um den unterschiedlichen Anforderungen der Entwickler gerecht zu werden.

- **Effizienz**: Steigerung der Effizienz bei der Entwicklung von Benutzeroberflächen durch die Drag-and-Drop-Funktionalität und die automatische Generierung von HTML- und TypeScript-Code. Dies soll die Entwicklungszeit reduzieren und die Fehleranfälligkeit minimieren.

- **Wartbarkeit**: Nutzung von Angular und TypeScript zur Sicherstellung, dass die Anwendung leicht wartbar und erweiterbar ist. Eine klare Trennung von Logik, Darstellung und Daten soll eine einfache Erweiterung und Anpassung des Systems ermöglichen.

Erwartete Ergebnisse:

- **Reduzierung der Fehleranfälligkeit**: Durch die Automatisierung und Standardisierung der Erstellung von Dialogfenstern sollen Fehler reduziert und die Zuverlässigkeit und Stabilität der Anwendung erhöht werden.

- **Erhöhung der Produktivität**: Bereitstellung von wiederverwendbaren Komponenten, die die Entwicklungszeit verkürzen und es den Entwicklern ermöglichen, sich auf die Implementierung neuer Funktionalitäten zu konzentrieren.

- **Verbesserung der Konsistenz und Qualität**: Konsistente und hochwertige Benutzeroberflächen durch den Einsatz von standardisierten UI-Komponenten, die für ein einheitliches Erscheinungsbild und eine verbesserte Benutzererfahrung sorgen.

5. Projektanalyse

5.1 Ist-Analyse

Die bestehende manuelle Erstellung von Dialogfenstern war ineffizient und zeitaufwendig. Manuelle Konfigurationen und fehlende Automatisierungen führten zu hoher Fehleranfälligkeit und unübersichtlichem Code.

Probleme der bestehenden Lösung:

- **Zeitaufwendig**: Die manuelle Erstellung und Konfiguration von Dialogfenstern erforderte viel Zeit und Aufwand von den Entwicklern.

- **Fehleranfälligkeit**: Manuelle Prozesse führten häufig zu Fehlern, die die Stabilität und Zuverlässigkeit der Anwendung beeinträchtigten.

- **Unübersichtlichkeit**: Der Mangel an Standardisierung und Automatisierung führte zu unübersichtlichem Code, der schwer zu warten und zu erweitern war.

5.2 Soll-Analyse

Die neue Lösung soll eine automatisierte, benutzerfreundliche Oberfläche bieten, die als Plattform um neue Dialoge zu designen und zu erstellen, und bietet die Möglichkeit, die Felder vorher zu konfigurieren. Die Anwendung soll die dynamisch geladenen und konfigurierten Felder für den Code erstellen und an die gewünschten Komponenten später kopieren können.

Anforderungen an die neue Lösung:

- **Automatisierung**: Die Erstellung und Konfiguration von Dialogen soll automatisiert werden, um Zeit zu sparen und Fehler zu reduzieren.

- **Benutzerfreundlichkeit**: Die Oberfläche soll intuitiv und einfach zu bedienen sein, sodass Entwickler schnell und effizient arbeiten können.

- **Flexibilität**: Unterstützung einer Vielzahl von UI-Komponenten, die leicht konfiguriert und angepasst werden können.

- **Standardisierung**: Die generierten Dialogfenster sollen einem einheitlichen Standard folgen, um die Wartung und Erweiterung zu erleichtern.

5.3 Projektbegründung

Durch die Implementierung der neuen Lösung werden Zeit und Ressourcen gespart, die Benutzerfreundlichkeit erhöht und die Fehleranfälligkeit reduziert. Eine automatisierte Lösung verbessert die Effizienz und Konsistenz bei der Erstellung von Dialogfenstern.

Vorteile der neuen Lösung:

- **Zeit- und Ressourceneinsparung**: Automatisierte Prozesse reduzieren den Zeit- und Arbeitsaufwand für die Erstellung von Dialogen.

- **Erhöhte Benutzerfreundlichkeit**: Eine benutzerfreundliche Oberfläche erleichtert die Bedienung und erhöht die Produktivität der Entwickler.

- **Reduzierte Fehleranfälligkeit**: Standardisierte und automatisierte Prozesse minimieren die Fehlerquote und erhöhen die Zuverlässigkeit der Anwendung.

- **Verbesserte Effizienz und Konsistenz**: Eine einheitliche und konsistente Gestaltung der Dialogfenster verbessert die Qualität und Wartbarkeit der Anwendung.

5.4 "Make-or-Buy"-Entscheidung

Nach Prüfung bestehender Lösungen wurde entschieden, eine individuelle Lösung zu entwickeln, um spezifische Anforderungen und Anpassungen optimal zu erfüllen. Bestehende Lösungen auf dem Markt boten nicht die erforderliche Flexibilität und Anpassungsfähigkeit, die für die speziellen Bedürfnisse dieses Projekts erforderlich sind.

Gründe für die Entscheidung zur Eigenentwicklung:

- **Anpassungsfähigkeit**: Die Möglichkeit, die Lösung vollständig an die spezifischen Anforderungen des Projekts anzupassen, ohne auf die Einschränkungen von Drittanbieterlösungen Rücksicht nehmen zu müssen.

- **Kostenersparnis**: Die Entwicklung einer eigenen Lösung kann langfristig kostengünstiger sein, da keine Lizenzgebühren anfallen und die Wartung und Weiterentwicklung intern erfolgen kann.

- **Kontrolle und Sicherheit**: Vollständige Kontrolle über den Quellcode und die Sicherheit der Anwendung, ohne Abhängigkeiten von Drittanbietern.

6. Projektplanung

6.1 Projektphasen

Die Projektplanung umfasst mehrere Phasen, die jeweils bestimmte Aktivitäten und Ziele abdecken. Jede Phase ist essenziell für den erfolgreichen Abschluss des Projekts.

Phasen:

1. **Analysephase**:
    - Ziele: Identifikation der Anforderungen und Spezifikation der notwendigen Funktionalitäten.
    - Aktivitäten: Analyse der bestehenden Prozesse und Systeme.
2. **Planungsphase / Entwurfsphase**:
    - Ziele: Entwurf der Systemarchitektur und Planung der Entwicklungsphasen.
    - Aktivitäten: Erstellung von Architekturdiagrammen, Definition der Hauptkomponenten und ihrer Interaktionen, Erstellung eines detaillierten Projektplans mit Meilensteinen.
3. **Entwicklungsphase**:
    - Ziele: Implementierung der Hauptkomponenten und Services.
    - Aktivitäten: Entwicklung der Komponenten und Services nach dem iterativen Ansatz, Integration und kontinuierliche Überprüfung des Fortschritts.
4. **Testphase**:
    - Ziele: Sicherstellung der Qualität und Funktionalität der Anwendung.
    - Aktivitäten: Durchführung von Unit-Tests, Integrationstests, Einrichtung von automatisierten Tests, Identifikation und Behebung von Fehlern.
5. **Fehleranalyse**:
    - Ziele: Identifikation und Behebung von Fehlern, die während der Testphase aufgetreten sind.
    - Aktivitäten: Analyse von Fehlerberichten, Implementierung von Korrekturen, erneute Testung der betroffenen Bereiche.

6. **Projektdokumentation**:
    - Ziele: Erstellung einer umfassenden Dokumentation, die alle Phasen und Komponenten des Projekts abdeckt.
    - Aktivitäten: Dokumentation der technischen Spezifikationen, Zusammenstellung aller relevanten Projektinformationen.

6.2 Ressourcenplanung

Die Ressourcenplanung ist entscheidend für den reibungslosen Ablauf des Projekts. Sie stellt sicher, dass alle notwendigen Ressourcen zur Verfügung stehen und effizient genutzt werden.

Ressourcen:

- **Hardware**: Leistungsfähiger Entwicklungsrechner, der die benötigten Entwicklungs- und Testumgebungen unterstützt.
- **Software**: Alle notwendigen Softwaretools und Frameworks, die für die Entwicklung und das Testen der Anwendung erforderlich sind.
- **Entwicklungsumgebung**: Visual Studio Code (VS-Code).
- **Frameworks**: Angular, TypeScript.
- **Kosten**: Es entstehen keine zusätzlichen Kosten, da alle verwendeten Tools und Frameworks bereits lizenziert waren oder kostenlos zur Verfügung standen.

6.3 Zielplattform

Die Zielplattform definiert die Umgebung, in der die Anwendung letztendlich ausgeführt wird.

Plattform:

- **Angular-Anwendung**: Die Anwendung wird als Single-Page-Anwendung (SPA) entwickelt, die in Angular und TypeScript geschrieben ist.
- **Bereitstellung**: Die Anwendung wird auf einem Webserver gehostet und ist über einen Webbrowser zugänglich.

6.4 Architekturdesign

Das Architekturdesign des Projekts folgt dem Model-View-Controller (MVC)-Prinzip, um eine klare Trennung von Datenmodell, Benutzeroberfläche und Anwendungslogik zu gewährleisten. Dies fördert die Wartbarkeit und Erweiterbarkeit der Anwendung.

Architekturprinzipien:

- **Model (Modell)**: Repräsentiert die Daten und Geschäftslogik der Anwendung. Es verwaltet den Zustand der Anwendung und kommuniziert mit der Datenbank oder anderen externen Quellen.
- **View (Ansicht)**: Verantwortlich für die Darstellung der Daten auf der Benutzeroberfläche. Sie empfängt Eingaben vom Benutzer und leitet diese an den Controller weiter.
- **Controller**: Vermittelt zwischen Modell und Ansicht. Es verarbeitet Benutzereingaben, aktualisiert das Modell und sorgt dafür, dass die Ansicht entsprechend aktualisiert wird.

Vorteile des MVC-Architekturprinzips:

- **Trennung der Verantwortlichkeiten**: Eine klare Trennung von Daten, Logik und Darstellung erleichtert die Wartung und Erweiterung der Anwendung.
- **Wiederverwendbarkeit**: Komponenten können unabhängig voneinander entwickelt und getestet werden, was die Wiederverwendbarkeit von Code fördert.
- **Skalierbarkeit**: Die Anwendung kann leicht erweitert und angepasst werden, um neuen Anforderungen gerecht zu werden.

7. Projektentwicklung

7.1 Hauptkomponenten und ihre Methoden

7.1.1 DynamicComponentDirective

- **Beschreibung**: Diese Direktive ermöglicht das dynamische Platzieren von Komponenten in der Ansicht.
- **Warum verwendet**: Erhöht die Flexibilität, UI-Komponenten zur Laufzeit zu laden und zu rendern.
- **Wichtige Methode**:
    - constructor(viewContainerRef: ViewContainerRef): Initialisiert die Direktive und speichert eine Referenz auf den Container, der die dynamischen Komponenten enthält.
- **Beziehung**: Diese Direktive wird von MainViewComponent verwendet, um dynamische Komponenten in der Ansicht zu platzieren. viewContainerRef ermöglicht das Hinzufügen und Entfernen von Komponenten zur Laufzeit.

7.1.2 ComponentFactoryService

- **Beschreibung**: Ein Service zur Erstellung und Verwaltung von UI-Komponenten.
- **Warum verwendet**: Zentrale Steuerung der Erstellung und Verwaltung dynamischer UI-Komponenten.
- **Wichtige Methode**:
    - getComponentFactory(type: string): Liefert die Factory für die angeforderte Komponente basierend auf dem übergebenen Typ.
- **Beziehung**: Dieser Service wird von MainViewComponent und AddElementComponent genutzt, um Komponenten dynamisch zu erstellen. Der ComponentFactoryService stellt die notwendigen Factories für die DynamicComponentDirective bereit.

7.1.3 CodeGeneratorService

- **Beschreibung**: Verwalten und Generieren von Code basierend auf konfigurierten UI-Komponenten.
- **Warum verwendet**: Dynamisches Erstellen, Speichern und Laden des generierten Codes.
- **Wichtige Methoden**:
    - generateFormHtml(): Generiert das HTML-Formular basierend auf den ausgewählten Komponenten.
        - **Verwendung**: Diese Methode wird von CodeViewPopupComponent aufgerufen, um den HTML-Code anzuzeigen.
    - generateTsCode(): Generiert den TypeScript-Code für die definierten Komponenten.
        - **Verwendung**: Diese Methode wird von CodeViewPopupComponent aufgerufen, um den TypeScript-Code anzuzeigen.
    - addElementToView(type: string, key?: string, width?: string, size?: string): Fügt ein neues Element zur Ansicht hinzu.
        - **Verwendung**: Diese Methode wird von AddElementComponent und MainViewComponent verwendet, um neue UI-Elemente hinzuzufügen.
    - removeItem(key: string): Entfernt ein Element aus der Ansicht.
        - **Verwendung**: Diese Methode wird von MainViewComponent verwendet, um ein Element basierend auf seinem Schlüssel zu entfernen.

- moveItem(sourceIndex: number, targetIndex: number): Verschiebt ein Element innerhalb der Ansicht.
    - **Verwendung**: Diese Methode wird von MainViewComponent verwendet, um die Position eines Elements zu ändern.
- changeElementPosition(key: string, newPosition: number): Ändert die Position eines Elements in der Ansicht.
    - **Verwendung**: Diese Methode wird von MainViewComponent verwendet, um die Position eines Elements basierend auf dem übergebenen Schlüssel zu ändern.

- **Beziehung**: Dieser Service wird von MainViewComponent und CodeViewPopupComponent verwendet, um den generierten Code zu verwalten und anzuzeigen. CodeGeneratorService speichert und lädt Konfigurationen, die in der Benutzeroberfläche dargestellt werden.

7.1.4 MainViewComponent

- **Beschreibung**: Hauptansichtsmodul der Anwendung, das die Benutzeroberfläche und Benutzerinteraktionen verwaltet.
- **Warum verwendet**: Verwaltung der zentralen Logik und Darstellung der Hauptbenutzeroberfläche.
- **Wichtige Methoden**:
    - renderElements(): Rendert die dynamischen Komponenten basierend auf der Konfiguration.
        - **Funktionsweise**: Diese Methode durchläuft die Liste der konfigurierten Elemente und erstellt für jedes Element die entsprechende Komponente. Innerhalb dieser Methode wird ein Wrapper-Element erstellt, das als Container für die dynamischen Komponenten dient. Event-Listener werden hinzugefügt, um Drag-and-Drop-Funktionalitäten zu ermöglichen. DynamicComponentDirective und ComponentFactoryService werden verwendet, um die Komponenten dynamisch zu erstellen und in die Ansicht einzufügen.
    - handleDragStart(event: DragEvent, key: string, index: number): Handhabt den Beginn einer Drag-and-Drop-Operation.
        - **Funktionsweise**: Diese Methode speichert Informationen über die gezogene Komponente, um den Drop-Vorgang zu ermöglichen.
    - handleDragOver(event: DragEvent): Erlaubt das Ablegen eines Elements.
        - **Funktionsweise**: Diese Methode verhindert das Standardverhalten und zeigt an, dass das Element ablegbar ist.

    - handleDrop(event: DragEvent, key: string, index: number): Handhabt das Ablegen eines Elements.
        - **Funktionsweise**: Diese Methode fügt die Komponente an der neuen Position ein und aktualisiert die Ansicht.
    - handleDragEnd(event: DragEvent): Handhabt das Ende einer Drag-and-Drop-Operation.
        - **Funktionsweise**: Diese Methode bereinigt die gespeicherten Informationen und stellt den normalen Zustand der Anwendung wieder her.
- **Beziehung**: Diese Komponente interagieren mit DynamicComponentDirective, ComponentFactoryService und CodeGeneratorService, um die dynamische Erstellung, Positionierung und Verwaltung von UI-Komponenten zu ermöglichen. MainViewComponent fungiert als zentrale Steuerungseinheit für Benutzerinteraktionen und die Darstellung der dynamischen Elemente.

7.1.5 CodeViewPopupComponent

- **Beschreibung**: Zeigt generierten HTML- und TypeScript-Code im Popup-Fenster an.
- **Warum verwendet**: Ermöglicht die Anzeige und Bearbeitung des generierten Codes.
- **Wichtige Methoden**:
    - open (): Öffnet das Popup-Fenster, in dem der generierte Code angezeigt wird.
        - **Funktionsweise**: Diese Methode öffnet das Popup-Fenster und stellt den generierten Code dar.
    - close(): Schließt das Popup-Fenster.
        - **Funktionsweise**: Diese Methode schließt das Popup-Fenster.
    - copyHTMLCode(): Kopiert den generierten HTML-Code in die Zwischenablage.
        - **Funktionsweise**: Diese Methode kopiert den angezeigten HTML-Code in die Zwischenablage, sodass er in andere Anwendungen eingefügt werden kann.
    - copyTsCode(): Kopiert den generierten TypeScript-Code in die Zwischenablage.
        - **Funktionsweise**: Diese Methode kopiert den angezeigten TypeScript-Code in die Zwischenablage.

- **Beziehung**: Diese Komponente interagiert mit dem CodeGeneratorService, um den generierten Code anzuzeigen und zu verwalten. CodeViewPopupComponent bietet eine Schnittstelle zur Bearbeitung und Verwaltung des generierten Codes.

7.1.6 AddElementComponent

- **Beschreibung**: Ermöglicht das Hinzufügen neuer UI-Elemente.
- **Warum verwendet**: Benutzern die Möglichkeit geben, neue UI-Komponenten hinzuzufügen und zu konfigurieren.
- **Wichtige Methoden**:
    - addElement(type: string): Fügt ein neues UI-Element hinzu.
        - **Funktionsweise**: Diese Methode nimmt die Benutzereingaben entgegen und erstellt basierend darauf ein neues Element, das der Liste der Elemente hinzugefügt wird. Sie stellt sicher, dass das neue Element korrekt konfiguriert und angezeigt wird.
- **Beziehung**: Diese Komponente verwendet den ComponentFactoryService, um neue Komponenten zu erstellen und hinzuzufügen. Sie interagiert auch mit dem CodeGeneratorService. AddElementComponent bietet die Funktionalität zur Erweiterung der Benutzeroberfläche durch Hinzufügen neuer Elemente.

7.1.7 ElementMenuComponent

- **Beschreibung**: Zeigt das Menü der verfügbaren UI-Komponenten an.
- **Warum verwendet**: Einfache Auswahl und Einfügung von UI-Komponenten.
- **Wichtige Methoden**:
    - selectComponent(type: string): Wählt eine Komponente aus dem Menü aus.
        - **Funktionsweise**: Diese Methode wird verwendet, um die gewünschte Komponente auszuwählen und in die Ansicht einzufügen. Sie interagiert mit MainViewComponent und AddElementComponent, um die ausgewählten Komponenten in die Ansicht einzufügen und zu konfigurieren.
- **Beziehung**: Diese Komponente interagiert mit MainViewComponent und AddElementComponent, um die ausgewählten Komponenten in die Ansicht einzufügen und zu konfigurieren. ElementMenuComponent bietet eine benutzerfreundliche Oberfläche zur Auswahl von UI-Komponenten.

7.1.8 KonfigurationPuppeComponent

- **Beschreibung**: Ermöglicht die Konfiguration der UI-Komponenten.
- **Warum verwendet**: Benutzern die Möglichkeit geben, die Eigenschaften der UI-Komponenten zu ändern und anzupassen.
- **Wichtige Methoden**:
    - configureComponent(key: string, config: any): Konfiguriert die ausgewählte Komponente.
        - **Funktionsweise**: Diese Methode ermöglicht es dem Benutzer, die Eigenschaften der ausgewählten Komponente zu ändern und die Änderungen zu speichern. Sie interagiert direkt mit dem CodeGeneratorService, um die Konfiguration der Komponenten zu ändern und die aktualisierte Konfiguration zu speichern.
- **Beziehung**: Diese Methode interagiert direkt mit dem CodeGeneratorService, um die Konfiguration der Komponenten zu ändern und die aktualisierte Konfiguration zu speichern.

Zusammenfassung der Beziehungen der Komponenten und Services

- **MainViewComponent und DynamicComponentDirective**: MainViewComponent verwendet DynamicComponentDirective, um dynamische Komponenten in der Ansicht zu platzieren.
- **MainViewComponent und ComponentFactoryService**: MainViewComponent verwendet ComponentFactoryService, um die entsprechenden ComponentFactories für die dynamischen Komponenten zu erhalten.
- **MainViewComponent und CodeGeneratorService**: MainViewComponent verwendet CodeGeneratorService, um die Konfiguration der UI-Komponenten zu verwalten und den generierten Code anzuzeigen.
- **AddElementComponent und ComponentFactoryService**: AddElementComponent verwendet ComponentFactoryService, um neue Komponenten zu erstellen und hinzuzufügen.
- **ElementMenuComponent und AddElementComponent**: ElementMenuComponent interagiert mit AddElementComponent, um die ausgewählten Komponenten in die Ansicht einzufügen und zu konfigurieren.
- **CodeViewPopupComponent und CodeGeneratorService**: CodeViewPopupComponent interagiert mit dem CodeGeneratorService, um den generierten Code anzuzeigen und zu verwalten.

- **KonfigurationPuppeComponent und CodeGeneratorService**: KonfigurationPuppeComponent interagiert direkt mit dem CodeGeneratorService, um die Konfiguration der Komponenten zu ändern und die aktualisierte Konfiguration zu speichern.

8. Herausforderungen und Lösungen im Detail

Dynamisches Laden und Konfigurieren von UI-Komponenten zur Laufzeit

- **Herausforderung**: Es war erforderlich, UI-Komponenten dynamisch zu laden und zu konfigurieren, basierend auf Benutzerinteraktionen.

- **Lösung**: Die DynamicComponentDirective und der ComponentFactoryService wurden verwendet, um diese Flexibilität zu erreichen. Die DynamicComponentDirective ermöglicht es, Komponenten dynamisch in der Ansicht zu platzieren, während der ComponentFactoryService die Erstellung und Verwaltung dieser Komponenten zentral steuert.

Implementierung von Drag-and-Drop-Funktionalitäten für UI-Komponenten

- **Herausforderung**: Die Benutzerfreundlichkeit sollte durch Drag-and-Drop-Funktionalitäten verbessert werden, um UI-Komponenten einfach zu verschieben und neu anzuordnen.

- **Lösung**: Methoden wie handleDragStart(), handleDragOver(), handleDrop(), handleDragEnd() wurden in der MainViewComponent implementiert, um Drag-and-Drop-Operationen zu unterstützen. Diese Methoden handhaben die verschiedenen Phasen des Drag-and-Drop-Prozesses und aktualisieren die UI entsprechend.

Speichern und Laden von Konfigurationen der UI-Komponenten

- **Herausforderung**: Die Konfigurationen der UI-Komponenten mussten gespeichert und bei Bedarf wieder geladen werden, um die Benutzerarbeit zu erleichtern.

- **Lösung**: Im CodeGeneratorService wurden Methoden wie saveElementList() und loadElementList() implementiert, um die Konfigurationen im LocalStorage zu speichern und zu laden. Dadurch können Benutzer ihre Arbeit speichern und später fortsetzen.

9. Zeitplanung

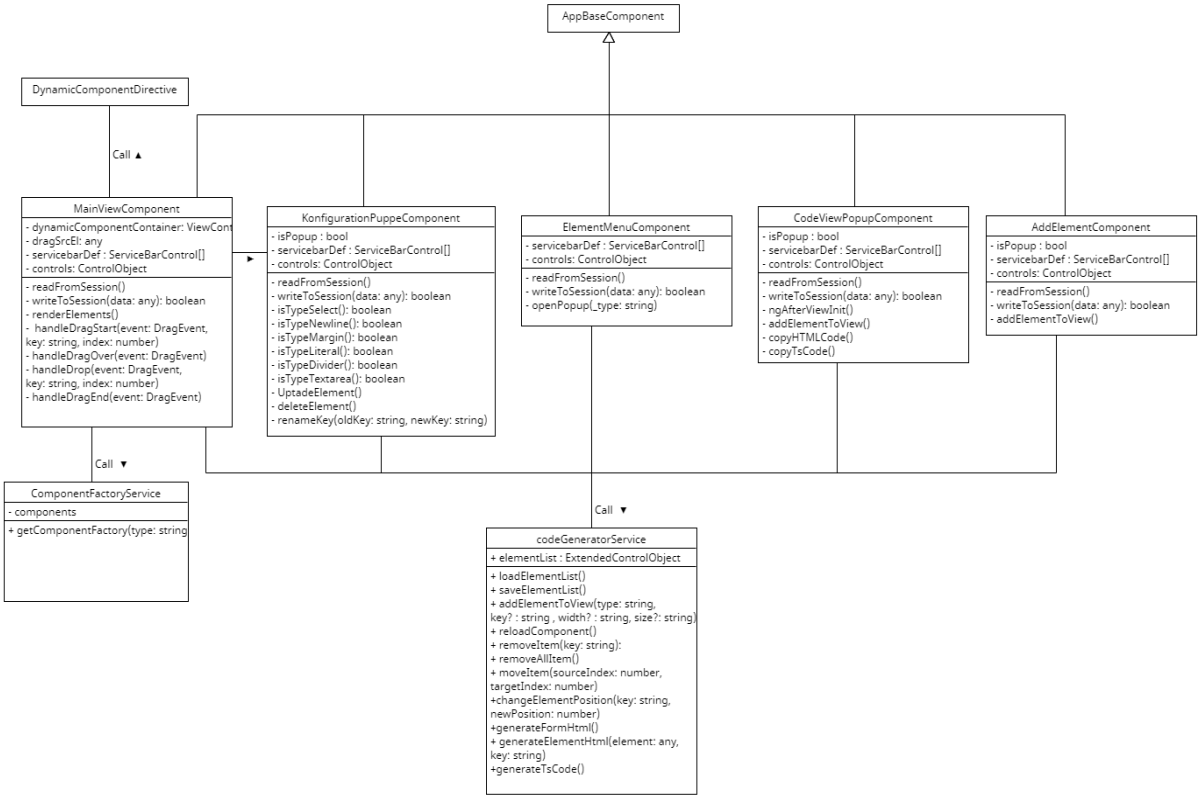
Aufgabe	Dauer (Stunden)
Analysephase	5
Planungsphase	6
Entwicklungsphase	40
Testphase	7
Fehleranalyse und Behebung	7
Projektdokumentation	10

10. Fazit

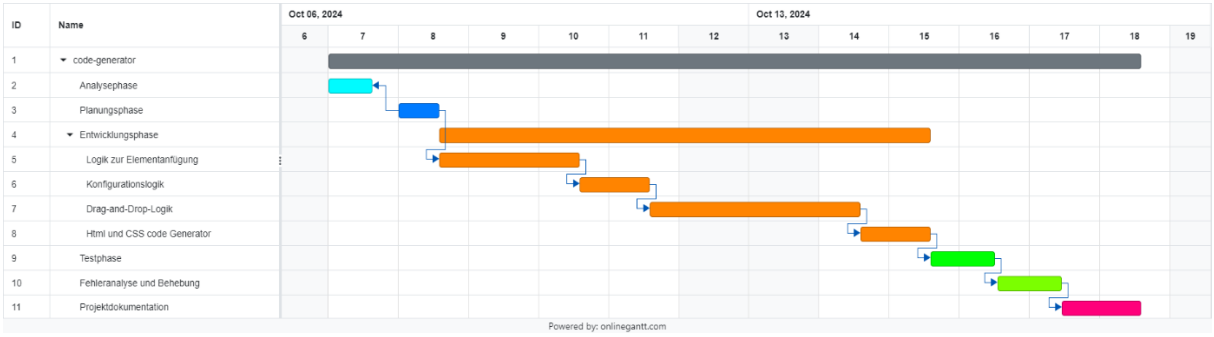
In dieser Projektarbeit wurde ein Tool zur Erstellung und Konfiguration von UI-Komponenten in Angular und TypeScript entwickelt. Die Implementierung umfasste die Entwicklung von mehreren Komponenten und Services sowie die Erstellung einer umfassenden Dokumentation. Das Projekt wurde innerhalb des vorgegebenen Zeitrahmens erfolgreich abgeschlossen.

11. Anhang

- **UML-Klassendiagramme**

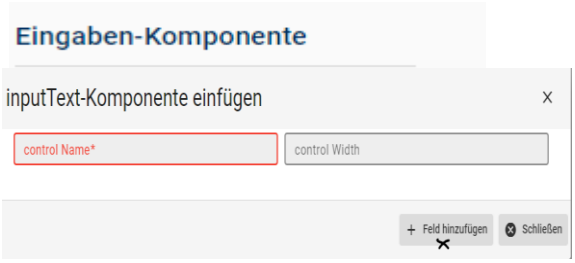


- **Gantt Chart**



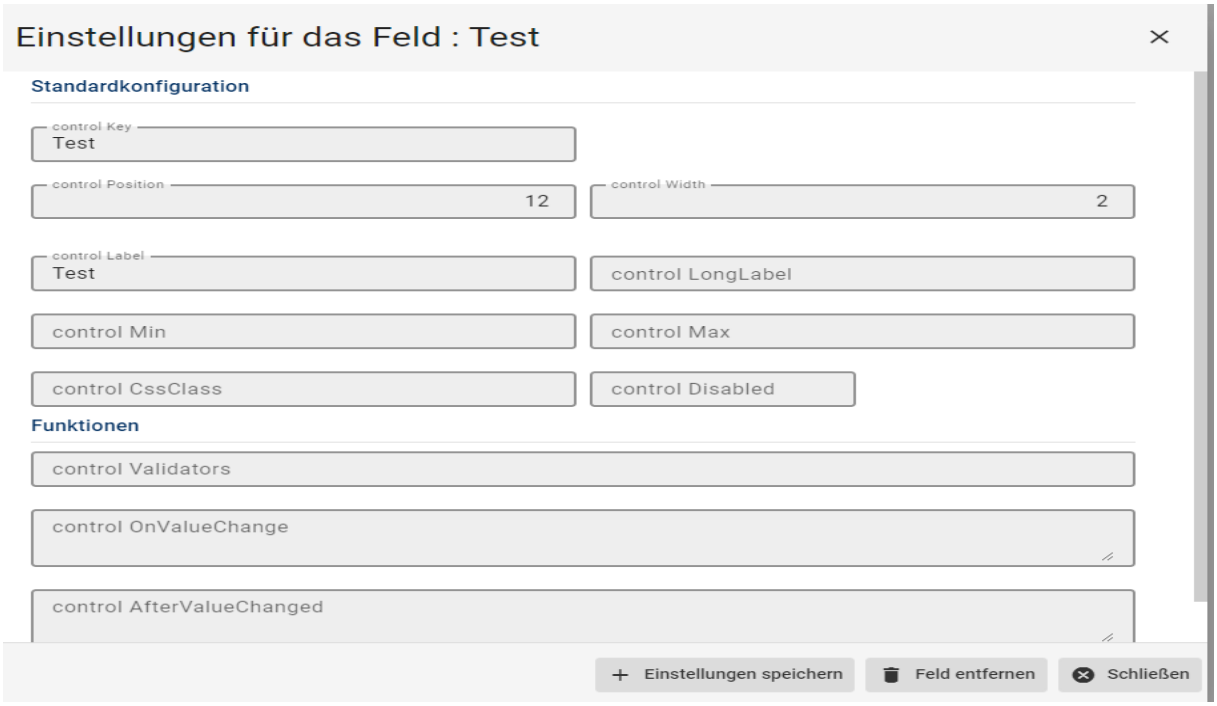
- **Benutzerdokumentation Code-Generator**
  In dieser Dokumentation finden Sie einige Hinweise und Hilfestellungen zur Benutzung des Code-Generator. Bei technischen Problemen, Fragen oder sonstigen Anliegen betreffend der Anwendung wenden Sie sich bitte an Abdulaa Mousa.

1. Klicken Sie auf das gewünschte Element, das Sie einfügen möchten. Es wird ein Pop-up geöffnet, in dem Sie den Namen des Elements angeben müssen (dieser Name sollte mit dem Namen der Datensätze übereinstimmen, die Sie anzeigen möchten).

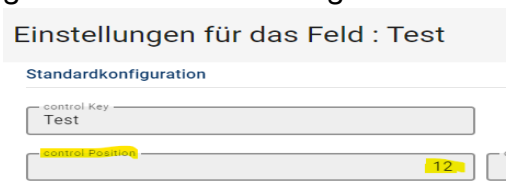


In diesem Pop-up können Sie auch die Breite des Feldes angeben. Die maximale Breite beträgt 4. Wenn Sie keine Breite angeben, wird standardmäßig die Breite 1 übernommen. Für den Elementabstand (Margin) gibt es keine Breitenangabe, sondern hier können Sie zwischen den Optionen "groß" und "klein" wählen. Nachdem Sie das gemacht haben, klicken Sie auf den Button "Feld hinzufügen". Ein neues Feld wird in der Hauptansicht angezeigt. Wenn Sie ein Feld desselben Typs einfügen möchten, können Sie dies tun, indem Sie den Namen ändern. Es darf keine zwei Felder mit gleichem Namen geben. Wenn Sie fertig sind, können Sie das Pop-up schließen.

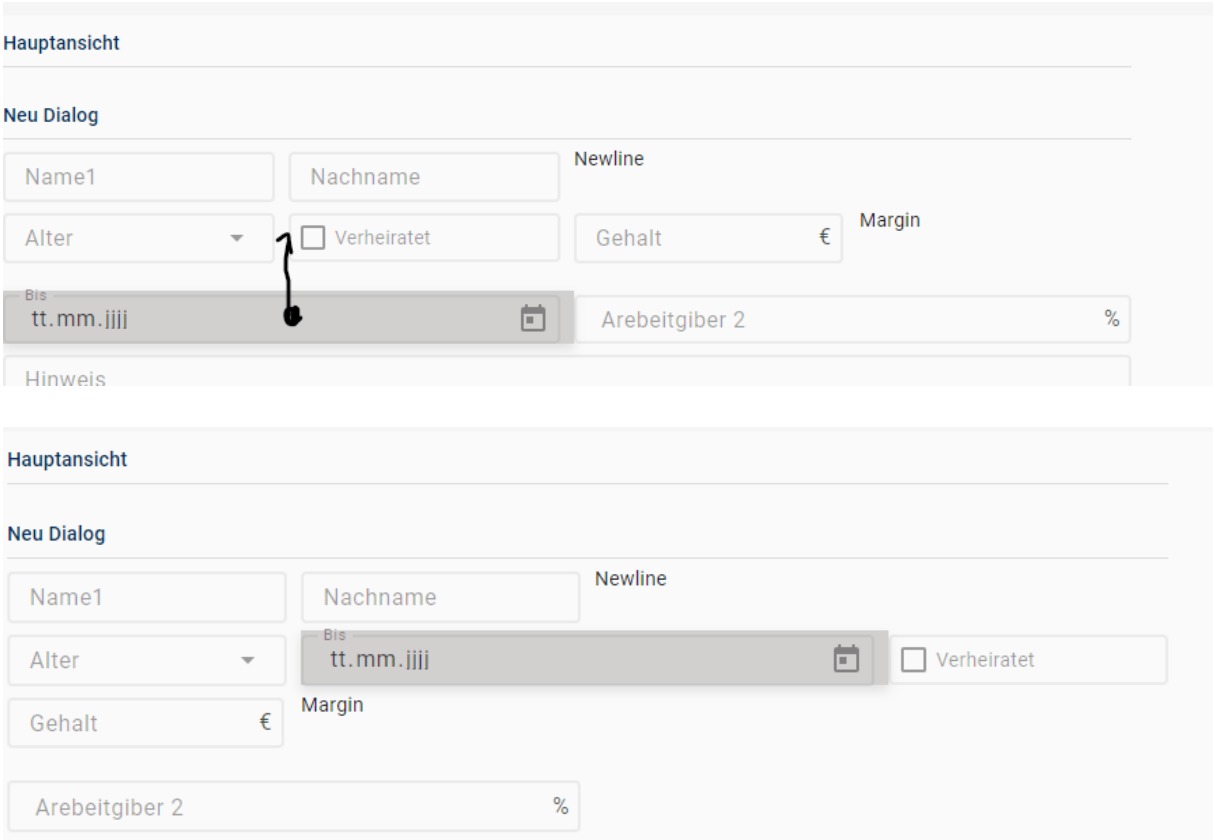
2. Um ein Feld zu konfigurieren, klicken Sie in der Hauptansicht auf das gewünschte Feld. Ein Pop-up-Fenster öffnet sich, in dem Sie verschiedene Einstellungen vornehmen können, wie z.B. das Label, das Langlabel und weitere Optionen. Sie haben auch die Möglichkeit, bestimmte Funktionen zu konfigurieren. Zum Beispiel können Sie bei "disabled" eine Bedingung als String in das Eingabefeld eintragen oder bei "onValueChange" Ihre Logik in Form eines Codeschnipsels hinzufügen, wie z.B. () => { logic }. Sobald Sie das Feld konfiguriert haben, können Sie Ihre Änderungen speichern. Im Konfigurations-Pop-up besteht auch die Möglichkeit, das Feld zu lösche



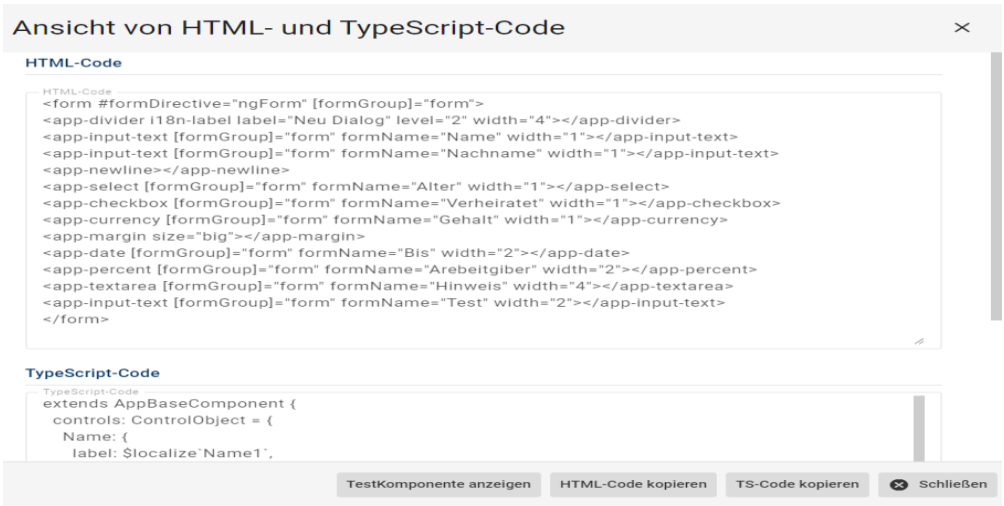
3. Wenn Sie die Position des Feldes ändern möchten, gibt es zwei Möglichkeiten. Erstens, im Konfigurations-Pop-up können Sie im Feld "Position" die neue gewünschte Position angeben.



Zweitens, in der Hauptansicht können Sie das Feld mit der linken Maustaste halten und an die gewünschte Position ziehen. Hinweis: Wenn die Position nicht erlaubt ist, wird das Feld an seinen ursprünglichen Platz zurückkehren.



4. Wenn Sie das Design und die Konfiguration Ihres Dialogs abgeschlossen haben, klicken Sie auf den Button "Quellcode erzeugen". Es wird ein Pop-up geöffnet, in dem Sie den HTML- und TypeScript-Code finden. Klicken Sie auf den Button "HTML kopieren" oder "TypeScript kopieren", um den entsprechenden Code zu kopieren. Fügen Sie den kopierten Code dann in die gewünschte Komponente ein (im Beispiel "TestComponentComponent"). Sie können den Code kopieren und dann auf den Button "Testkomponente anzeigen" klicken, um ihn zu testen.



- **Codeausschnitte der Kernmethoden**

```
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.nativ
eElement, 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);
    }
  }

  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>`;
 }
}
 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`;
  return code;

}
  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();
  }
```