Web Addon

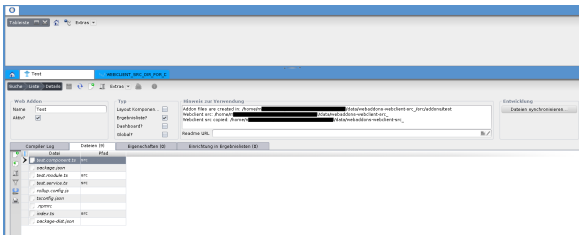
Definition

Menüaufruf: (Konfiguration) - (Web Addon)

Mit Web Addons kann man die Funktion des Web Clients flexibel erweitern.

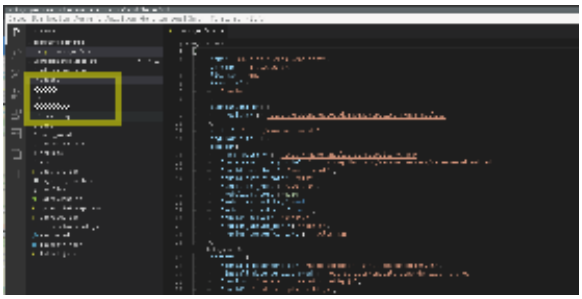
Voraussetzungen

- Entwicklung auf einer lokalen Dev Instanz: NodeJS v14.17.6 (vor 4.2022.x NodeJS v12) (über z. B. nvm installieren), npm 6.14.15, Visual Studio Code (oder ein anderer Editor), Nuclos Server im DEV Modus installiert
- Beispiel 1: NodeJS auf dem Nuclos-Server



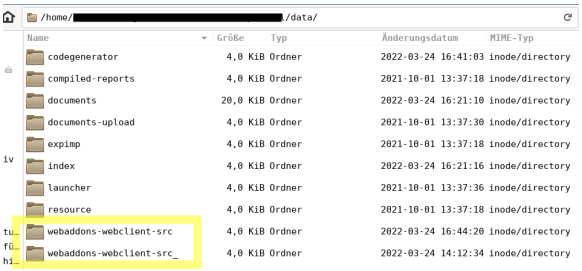
Konfiguration auf eine Dev Instanz

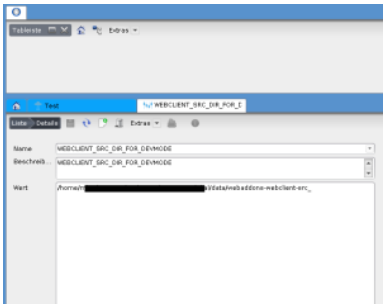
Das Addon muss wie o. g. angelegt werden, einem Nuclet zugewiesen sein (hier auch test). Danach ist es im Nuclos Installationspfad "[nuclos Instanzname]/data/webaddons-webclient-src/addons" zu finden.

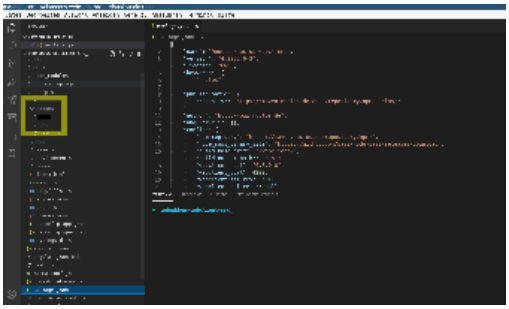


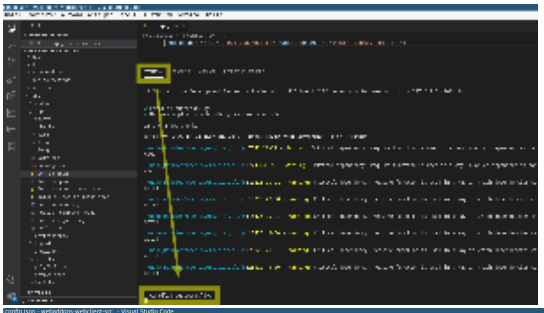
Dieser Ordner kann jetzt z. B. mit Visual Studio Code (über Ordner öffnen) angezeigt werden:

Da aber eine lokale Entwicklung mit optischen Feedback im Browser möglich sein soll. Muss zusätzlich zur Development Installation noch ein Parameter (WEBCLIENT_SRC_DIR_FOR_DEVMODE) gesetzt werden. Dieser Parameter muss auf eine Kopie des Verzeichnisses "webaddons-webclient-src" verweisen (in diesem Beispiel "webaddons-webclient-src_") die vorher angelegt werden muss.



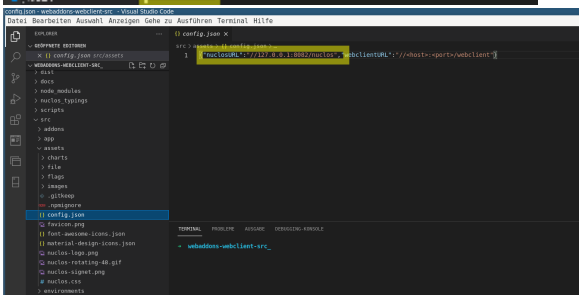






Nach einem Serverneustart stellt der Server das WebAddon jetzt aus dem neuen Verzeichnis zur Verfügung.
Zu erkennen ist das - nachdem das Vereichnis "webaddons-webclient-src_" mit Visual Studio Code erneut geöffnet wurde - am verschobenen Addons Verzeichnis. Dies wird jetzt aus dem src Verzeichnis zur Verfügung gestellt.

Um die Anpassungen über den Browser anzeigen zu können (npm sorgt bei Änderungen dann für eine Aktualisierung der Daten) muss die Datei "src/assets/config.json" noch angepasst werden. Diese muss auf den lokalen Nuclos Server verweisen (im Beispiel wird der Parameter nuclosURL von "//:<host>:<port>/nuclos" auf den u. g. Wert geändert).



Jetzt müssen noch die npm Installationsskript im Order "webaddons-webclient-src_" ausgeführt werden (hier über das Terminal aus Visual Studio Code).

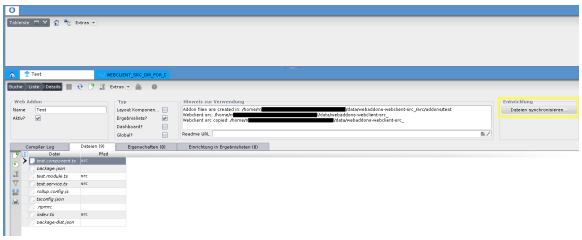
```
#npm install
#npm run install-addon-deps
```

Danach kann der lokale Server gestartet werden.

```
npm run start --webclient-host="127.0.0.1" --
webclient-port=4200
```

Nachdem der Start abgeschlossen ist, kann die Seite über den Browser geöffnet werden.

Anpassungen die vorgenommen werden, werden direkt im genutzten Browser angezeigt. Sollen die Änderungen fertiggestellt sein können Sie in der Web Addon Verwaltung direkt in das Nuclet übernommen und dann exportiert werden.



Beispiel 1: Berechnete Felder

In diesem Fall legt man ein Web Addon z.B. mit dem Namen "BerechneteFelder" und der Option "global" an. Der Beispiel-Code erzeugt clientseitige Berechnung für Unterformulare.

berechnete-felder.service.ts

```
import { Injectable } from '@angular/core';
import {GlobalContext, ISubEntityObject, Logger}
from "@nuclos/nuclos-addon-api";
import {
        IEntityObject,
        IEntityObjectDependents,
        IEntityObjectEventListener
} from "@nuclos/nuclos-addon-api/api/entity-
object";

@Injectable({
        providedIn: 'root'
})

class CalculationRule {

        constructor(public attributeNames : string
[], public targetAttributeName : string, public
funcCalculate : (eo : IEntityObject, attributeName
: string) => number) {
        }
}

@Injectable()
export class BerechneteFelderService {
        private attributeChangeListener :
IEntityObjectEventListener;
        private selectedEo : IEntityObject;
        private calculationRules : CalculationRule
[];

        callForDependents(eo : IEntityObject,
func: {(entity:ISubEntityObject):void}) :any {
                eo.getDependents
("test_SubBo_refMainBo").asObservable().subscribe
(subEntities => {
                        if( subEntities ===
undefined)
                                return;
                        for(let i in subEntities) {
                                let entity :
ISubEntityObject = subEntities[i];
                                func(entity);
                                this.logger.log
('BerechneteFelder: onEoSelection() ' + func.
prototype.name);
                        }
                })
        }

        constructor(private addonCtx:
GlobalContext, private logger: Logger) {
                this.logger.log('BerechneteFelder:
constructor');
                this.calculationRules = [];
                this.calculationRules.push(new
CalculationRule(["subbo_attribute1",
"subbo_attribute2"], "subbo_attribute3",
```

```
calculate_Attribute));
                this.attributeChangeListener = {
                        afterAttributeChange:
(entityObject: IEntityObject, attributeName:
string, oldValue: any, newValue: any) => {
                                        this.
logger.log("BerechneteFelder: entityObject: " +
entityObject.getEntityClassId() + ",
attributeName: " + attributeName +
                                                ",
oldValue: " + oldValue + ", newValue: " +
newValue);
                                        this.
calculationRules.forEach(calculationRule => {
                                                if
(calculationRule.attributeNames.includes
(attributeName)) {

let result = calculationRule.funcCalculate
(entityObject, attributeName);

entityObject.setAttribute(calculationRule.
targetAttributeName, result);
                                                }
                                        })
                        }
                }

                addonCtx.getEntityObjectApi().
onEoSelection().subscribe(eo => {
                        this.logger.log
('BerechneteFelder: onEoSelection()');
                        if (this.selectedEo !==
undefined) {
                                this.selectedEo.
getDependents("test_TestBo_ref1").asObservable().
subscribe((entities => {
                                        entities?.
forEach((subEo) => subEo.removeListener(this.
attributeChangeListener));
                                }))
                        }

                        if (eo !== undefined) {
                                eo.getDependents
("test_TestBo_ref1").asObservable().subscribe
((entities => {
                                        entities?.
forEach((subEo) => {

subEo.addListener(this.attributeChangeListener)
                                        });
                                }))
                                this.selectedEo =
eo;
                        }
                })
                addonCtx.getEntityObjectApi().
onEoModification().subscribe(eo => {
                });
        }
}

function calculate_Attribute(eo : IEntityObject,
attributeName : string) : number {

        let dAttribute1 = eo.getAttribute
("subbo_attribute1");
        let dAttribute2 = eo.getAttribute
("subbo_attribute2");
```

```
        return dAttribute1 * dAttribute12;
}
```

`test_SubBo_refMainBo` ist der BO-Name des Unterformulars.

Die fachliche Implementierung findet in `calculate_Attribute()` statt. eo ist in diesem Fall eine Instanz des SubBos. Die Funktion wird bei jeder Änderung der Attribute `subbo_attribute1` und `subbo_attribute2` aufgerufen. Das Ergebnis wird in `subbo_attribute3` geschrieben.