Übungen

Business Case

Unser Kunde ist ein kleiner Steinmetzbetrieb. Er stellt sich eine Applikation vor, mit der er seine Produktpalette verwalten kann. In einem weiteren Schritt könnte man aus dieser Applikation auch den Verkauf abwickeln.

Einrichtung

Installation

- Installieren Sie (wenn nicht bereits vorhanden) node.js in der Version >=7. Testen Sie die installierten Versionen von node.js und npm.
- Installieren Sie das Angular CLI.
- Generieren Sie eine neue App. Verwenden Sie dafür einen passenden Namen für den Business Case.
- Testen Sie einige der CLI Befehle. (Siehe auch CLI Referenz)

IDE (Visual Studio Code)

- Installieren Sie folgende Plugins:
  - Path Intellisense
  - TSLint
  - Auto Import
  - Debugger for Chrome
  - npm Intellisense
  - Angular Language Service
- Öffnen Sie die generierte Applikation in VS Code. (via Konsole: code .)

Module

Aufgabe 1

- Generieren Sie eine neue Applikation via angular-cli mit
  - einem sinnvollen Präfix (bspw. *stn*)
  - Angular 4 als Basis
  - und vollziehen Sie das Bootstrapping nach.
- Generieren Sie ein Feature-Modul für die Produktverwaltung.
- Importieren Sie dieses Modul im Root-Modul.
- Starten Sie die Applikation.

Komponenten

Aufgabe 2.1

Schritt 1: Generieren

- Generieren Sie eine neue Komponente in dem Produkt-Modul. Sie sollen damit ein Produkt anzeigen können.
- Exportieren Sie die neue Komponente im produkt-Modul für die globale Verwendung. (Sie können dazu auch das --export flag im CLI verwenden)

Schritt2: Testen

- Führen sie ng testin einer separaten Shell aus und lassen Sie die Tests ab jetzt immer mitlaufen.
- Der Test für die gerade generierte Komponente sollte grün sein. Ergänzen Sie ihre Test möglichst immer sofort. Nach einer Zeit können Sie auch dazu übergehen die Tests vor der eigentlichen Implementierung zu schreiben.

Schritt 3: Implementieren

- Sie finden ein Beispiel-Produkt im src Folder. (product.json)
- Bauen Sie eine Domain-Klasse, die ein Produkt repräsentiert.

- Entwerfen sie ein einfaches Template in dem die Werte aus der Produkt-Klasse angezeigt werden.
- Zeigen Sie ein Produkt auf der Startseite der App an. Verwenden Sie dafür ein hartkodiertes Objekt im ViewModel der Produkt-Komponente.
- Integrieren Sie einen Button der bei Klick den aktuellen Preis um 5€ erhöht und via alert() ausgibt.
- Testen Sie die Klick-Funktion mit einem sinnvollen Testfall.

Aufgabe 2.2

- Definieren Sie nun das Produkt-Objekt in Ihrer Komponente als Input, sodass es via Databinding von außen gesetzt werden kann.
- Vergeben Sie (falls nötig) einen sprechenden Alias für die Property.
- Erstellen Sie ein anderes Produkt-Objekt in der App-Component und weisen Sie dieses Objekt der Produkt-Komponente zu.
- Schreiben Sie eine Methode in der App-Component, die aufgerufen werden soll, wenn der Preis der in der Komponente verändert wurde. Darin wird das Produkt-Objekt aktualisiert und in einem alert() der neue Preis ausgegeben.
- Definieren Sie in der Produkt-Komponente einen Output (eventEmitter), der jede Veränderung des Preises meldet.
- Binden Sie die Methode aus dem Parent an dieses Event.
- Testen Sie die Funktionalität. Stellen Sie auch mittels DOM-Komponententest sicher, dass das Binding in der Produktkomponente funktioniert.
- Versuchen Sie diesen UseCase alternativ via Two-Way Databinding (banana-in-a-box) abzubilden.

Aufgabe 2.3

- Ergänzen Sie die Produkt-Komponente um ein Formular mit einem Input für den Preis und einem Submit-Button.
- Greifen Sie via Template Variable auf den Wert des Inputs zu und aktualisieren Sie bei Klick auf den Button den Preis des Produkt-Objekts.

Fortgeschritten

- Erweitern Sie die Produkt-Komponente so, dass alle Werte in einem Formular erfasst und geändert werden können.
- Testen Sie den Zugriff auf das aktualisierte Produkt via Template Variable anstatt einen Output zu verwenden.
- Experimentieren Sie mit Template Projection

Direktiven

Aufgabe 3

- Definieren Sie einen Rahmen um die Darstellung der Produkt-Werte in der Produkt-Komponente.
- Verwenden Sie die ngStyle Direktive um die Farbe oder den Stil des Rahmens zu verändern.
- Verwenden Sie die ngClass Direktive, um die Farbe des Rahmens auf grün zu setzen, sobald der Preis kleiner 50 ist.
- Implementieren Sie eine Schaltfläche, die es erlaubt, das Formular in der Komponente aus und einzublenden (ngIf).
- Definieren Sie in der App-Component ein Array von Produkten (siehe practice/src/products.json).
- Iterieren Sie via ngFor über diese Liste und zeigen Sie jedes Produkt auf der Startseite an.
    - Verwenden Sie zunächst nur **One-Way Binding** für die Zuweisung an die Produkt-Komponente

Fortgeschritten

- Weisen Sie jedem zweiten Produkt einen hellgrauen Hintergrund zu.

Pipes

Aufgabe 4

- Formatieren Sie den Preis in der Komponente als Währung.
- Zeigen Sie über der Liste von Produkten das heutige Datum in einem sinnvollen Format an.
- Zeigen Sie (mithilfe einer Pipe) nur die ersten drei Produkte an.
- Generieren Sie eine Pipe in einem neuen Utils-Modul. Diese Pipe soll den Bruttopreis errechnen können.
- Dabei soll der Steuersatz konfiguriert werden können.
- Schreiben Sie einen Test, der die Funktionalität der Brutto/Netto Umrechnung absichert.

Fortgeschritten

- Flexibilisieren Sie die Pipe so, dass sie in beide Richtungen brutto->netto und netto->brutto funktioniert.

3rd Party Module / UI Frameworks

Aufgabe 5

- Installieren Sie angular-Material als UI-Framework in ihr Projekt.
- Vergessen Sie nicht das Angular-Material Modul in ihre Module zu importieren.
- Integrieren Sie eine MD-Toolbar in die Applikation.
- Passen Sie ihre Tests mit den neuen Abhängigkeiten an.

Fortgeschritten

- Migirieren sie alle Elemente auf Angular Material (z.B. das Produkt als Card o.ä.)
- Passen Sie das Theme von Angular-Material an (https://material.angular.io/guide/theming)

Hinweis: Sie können ab hier auch mit dem Code aus referenz\A5 weitermachen. Dazu müssen Sie den Code in einen neuen Ordner kopieren und npm install darin ausführen.

Formulare - TDF

Aufgabe 6.1

- Generieren Sie eine Komponente, in der ein neues Produkt angelegt werden kann. Verwenden Sie dafür eine Template-Driven Form.
- Sie können dafür die Vorlage *edit-product.VORLAGE.html* verwenden.
- Vergessen Sie nicht die Form-Direktiven (ngform und ngmodel) anzuwenden.
- Reichen Sie das erstellte Produkt via Output aus der Komponente heraus.
- Integrieren Sie das Produkt-Formular auf der Startseite. Hier soll das neue Produkt der Liste hinzugefügt werden.
- Erstellen Sie eine addProject() Methode in der App-Component und pushen sie das neue Produkt in das Produktarray.
- Das Produkt sollte in der Liste erscheinen.

Aufgabe 6.2

- Validieren Sie das Formular sinnvoll. Der Button soll erst klickbar werden, wenn das ganze Formular valide ist.
- Experimentieren Sie ach mit Validierungsmeldungen.

Fortgeschritten

- Implementieren Sie für jedes Feld eine sinnvolle Validierungsfehlermeldung.

Formulare - MDF

Aufgabe 6.3

- Erstellen Sie eine MDF-Entsprechung zu dem zuvor erstellten TDF-Formular.

Aufgabe 6.4

- Validieren Sie die Felder des Formulars sinnvoll mit core-Validatoren.

Fortgeschritten

- Erstellen Sie einen Custom-Validator, der sicherstellt dass ein numerischer Wert positiv ist.
- Validieren Sie, damit das Preis-Feld.

Routing

Jetzt wird unsere Applikation komplexer. Wir arbeiten nicht mehr nur auf der App-Component. Daher sind jetzt einige Umbauarbeiten nötig, bevor der Router eingesetzt werden kann.

Aufgabe 7

- Transferieren Sie die Produktlisten-Funktionalität in eine neue Komponente im Produkt-Modul (z.B. product-list)
- Implementieren Sie ein <router-outlet> in der App-Component unter der Toolbar.
- Konfigurieren Sie den Router im Root-Modul und konfigurieren Sie eine Weiterleitung von der Startseite auf die Route "/products".
- Konfigurieren Sie den Router im Produkt-Featuremodul. Routen Sie "/products" auf die Produktlisten-Komponente.
- Die Produktliste sollte nun dort gerendert werden.
- Konfigurieren Sie eine Route für das Formular, mit dem ein neues Produkt angelegt wird. (z.B. unter "/products/new")
- Für die Kommunikation zwischen der Produkt-Liste und dem newProdukt-Formular benötigen wir einen Service, der in einer folgenden Übung implementiert wird.
- Implementieren Sie mit der <md-toolbar> eine globale Navigation in der App-Component.

Fortgeschritten

- Implementieren Sie eine parametrisierte Route, unter der das Produkt-Formular mit einer Produkt-ID aufgerufen werden kann.
- Füllen Sie das ID-Feld in dem Fall mit dem Parameter vor.
- Ergänzen Sie einen Button für jedes Produkt in der Produktliste, von dem aus die neue Route mit der aktuellen ID aufgerufen werden kann.
- Implementieren Sie den Guard "CanDeactivate" für die Formular-Komponente und verhindern Sie, dass der Benutzer wegnavigiert, bevor er gespeichert hat.

Services

Aufgabe 8

- Generieren und implementieren Sie einen Produkte-Service, der sowohl von der Liste als auch vom Formular genutzt wird.
- Realisieren sie die Kommunikation zwischen den beiden Komponenten mit dem Service.
- Der Service hält vorerst die globale Instanz der Produktliste als property und bietet mindestens eine getList() und eine newProduct() Methode.
- Produkte, die über das Formular hinzugefügt wurden, sollen auch in der Liste erscheinen (zunächst ohne Persistierung).
- Sichern Sie die Methoden des Service mit geeigneten Test ab.

Observables / HTTP & Rest

Aufgabe 9

- Implementieren Sie einen Live-Counter der Länge des Name-Feldes im Produkt-Formular und zeigen sie diesen Wert an.

Aufgabe 10

- Sie erhalten vom Kursleiter eine URL für einen REST-Service.
- Ersetzen Sie die statische Produktliste im Produkt-Service mit einem get-Aufruf auf den Produkt-Service.
- Geben Sie im Service nun ein Observable zurück, damit die Komponente die Subscription machen kann.
- Verbinden Sie ihren new-Produkt Mechanismus mit dem POST Endpoint des Produkt-Services (Achtung die id darf bei einem neuen Produkt nicht gesetzt sein, da dies serverseitig geschieht).
- Nun sollten Sie in ihrer Produktliste ebenfalls die Produkte der anderen Teilnehmer abrufen können.
- Mocken Sie die Aufrufe des ProduktService in ihren Komponententests

Fortgeschritten

- Experimentieren Sie mit verschiedenen Observable-Operatoren mit einem HTTP-GET (z.B. auf der Produktliste).

Testing

Aufgabe 11

- Führen Sie *ng test* aus. Vermutlich werden einige/viele der generierten Test fehlschlagen.
- Lassen Sie *ng test* in einem Konsolenfenster laufen, dann werden die Testfälle automatisch ausgeführt.
- Ergänzen Sie Unittests für ihre Applikation wo nötig.
- Lassen Sie sich einen Code-coverage Report dafür ausgeben.

Fortgeschritten

- Verwenden Sie die *async* oder *fakeAsync* Zone in einem ihrer Komponententests (z.B. product-list)
- Mocken Sie den Aufruf des HTTP-Services, sodass der Product-Service isoliert vom Backend getestet wird.