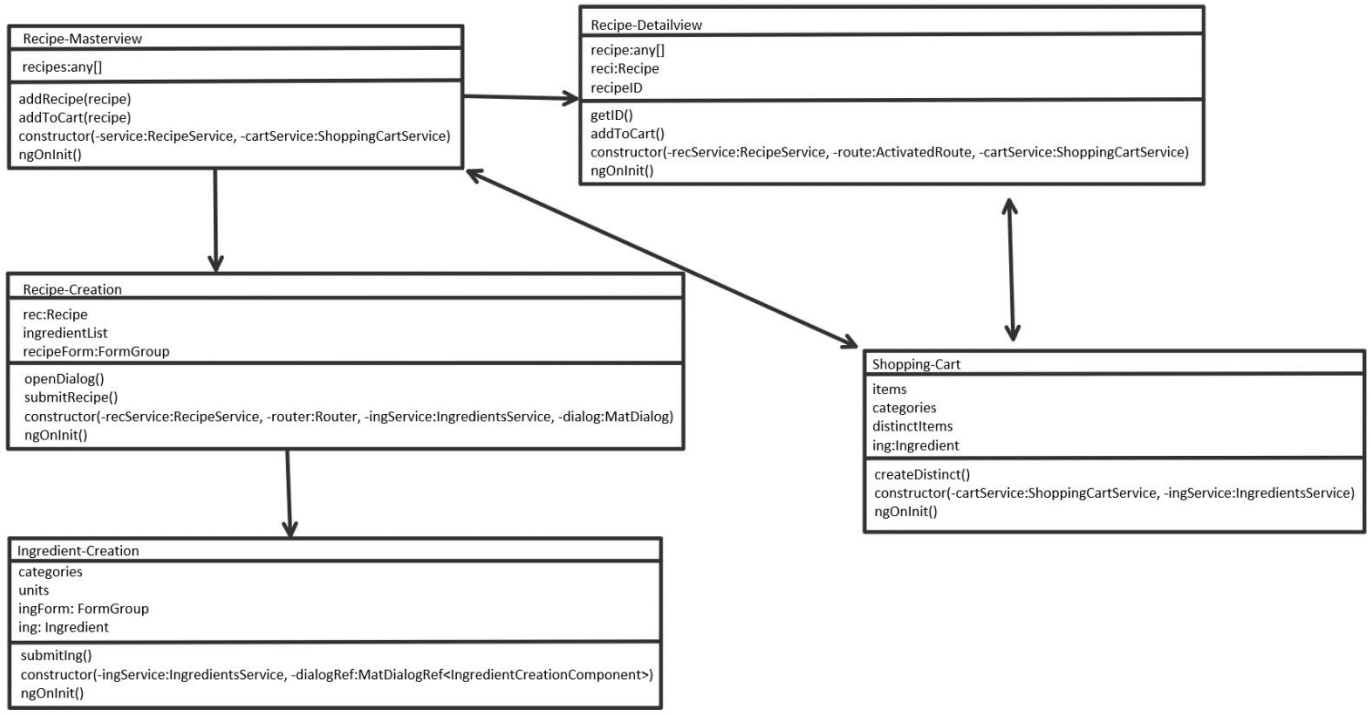
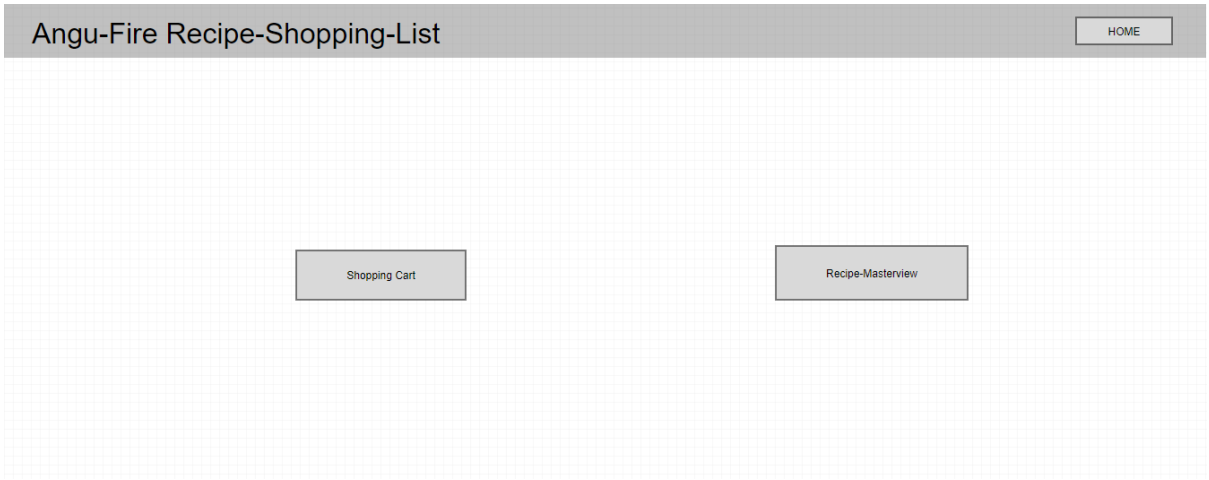
Abstraktes Klassendiagramm:



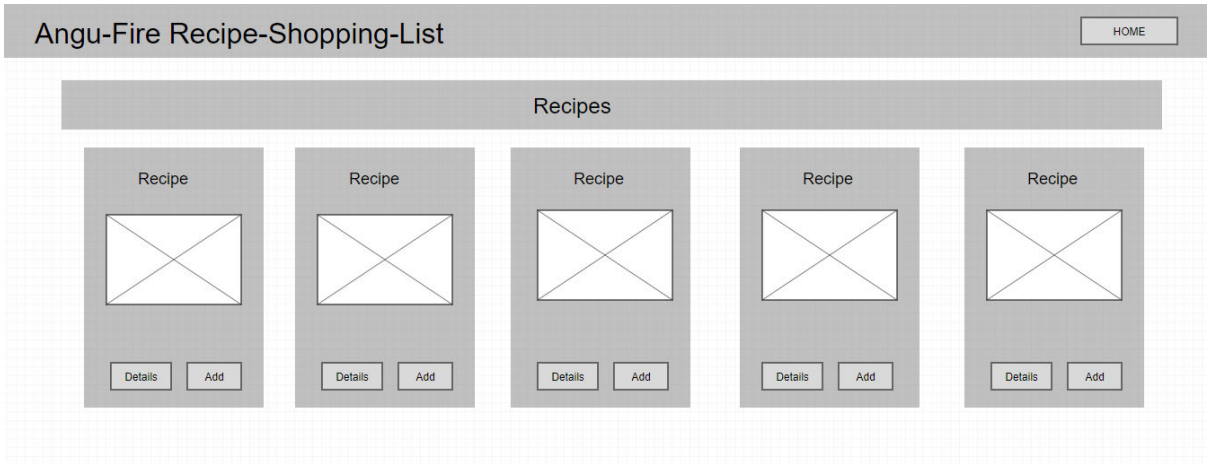
Mock-Up:

- Entry:

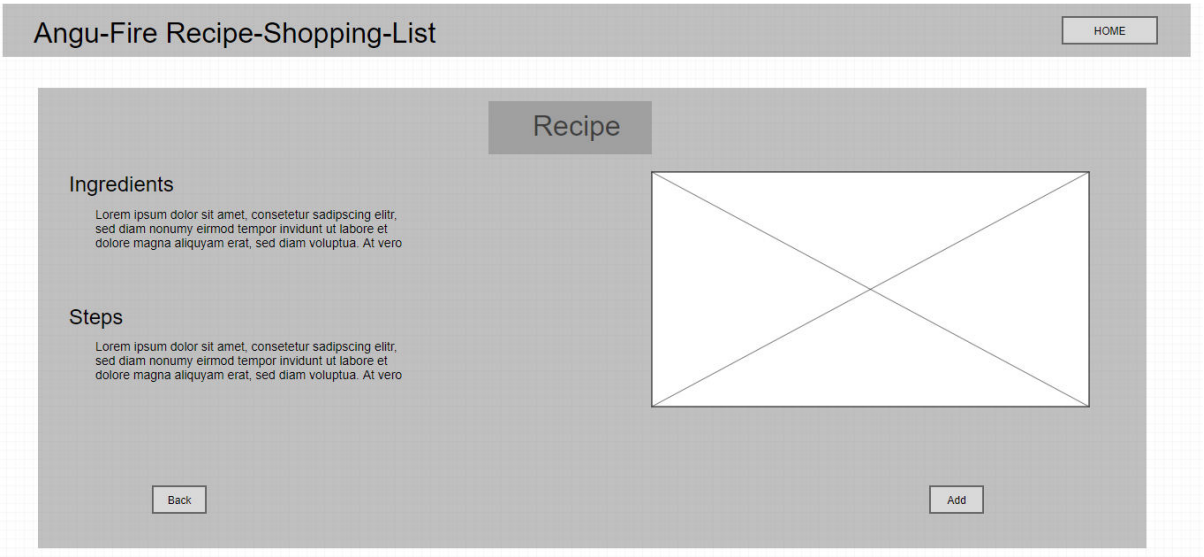


Matrikelnummer: 9679885

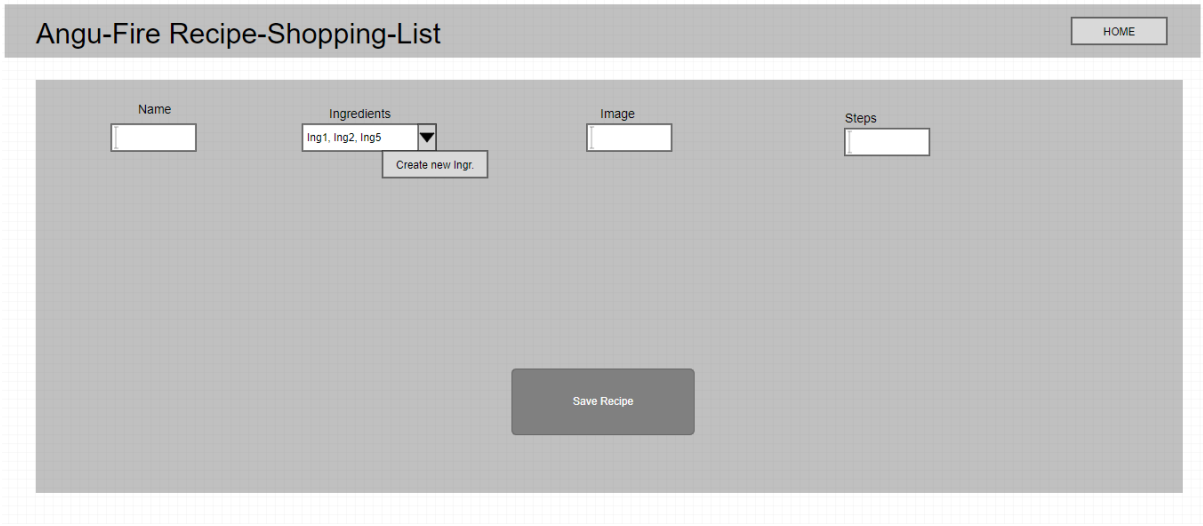
- Recipe-Masterview:



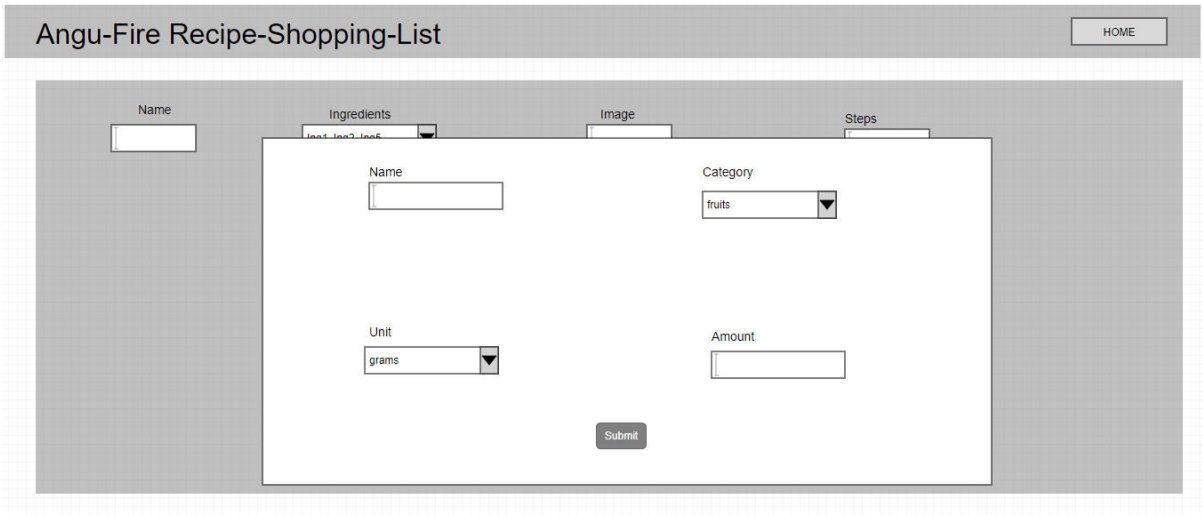
- Recipe-Detailview:



- Recipe-Creation:



- Ingredient-Creation:



- Shopping-Cart:

Requirements und Konzepte:

Grundprinzip:

> Rezepte bestehen aus einem Namen, einer Beschreibung („Steps“), einem Bild (falls kein Pfad angegeben ist, wird ein Placeholder angezeigt) und aus Zutaten („Ingredients“).
>
> Ein Zutat („Ingredient“) bestehen aus einem Namen, einer Kategorie, einer Mengeneinheit und einer Menge.

Seiten:

- **Entry**: Die Entry-Page dient als Art „Landing-Page“ für den Nutzer. Von hier aus kann er sich entscheiden, entweder alle Rezepte anzusehen (also auf die Recipe-Masterview zu wechseln) oder zu dem Einkaufkorb („Shopping-Cart“) zu wechseln. Über den „Home“-Button im Header kommt er auch immer wieder auf diese Seite zurück. Diese Seite ist für eine spätere Skalierbarkeit Pflicht, hätte bei diesem kleineren Projekt jedoch auch vernachlässigt werden können.
- **Recipe-Masterview**: Auf der Recipe-Masterview werden dem Nutzer alle Rezepte aus dem Recipe-Service dargestellt, jeweils in einer Mat-Card mit dem Namen, dem Bild (bzw. Placeholder oder Error-Image, wenn ein invalider Pfad angegen wurde), einem „Add“-Button zum Hinzufügen der Zutaten des Rezeptes zum Einkaufskorb und einem „Details“-Button, um auf die Recipe-Detailpage des Rezeptes zu wechseln.
  Nachdem alle Rezepte dargestellt wurden, gibt es ein weiteren Button in der Auflistung, über welchen der Nutzer ein neues Rezept anlegen kann. Auf Knopfdruck wechselt der Nutzer zur RecipeCreation.
- **Recipe-Detailview**: Auf der Recipe-Detailview wird des jeweilige Rezept genauer dargestellt. Auf der linken Hälfte sind die jeweiligen Zutaten und die nötigen Schritte aufgelistet, die rechte Hälfte ist mit dem Bild gefüllt. Es gibt hier für den Nutzer ein

„Add“-Button und einen „Back“-Button, um zurück zur Recipe-Masterview zu wechseln.

- **CreateRecipe**: Auf der CreateRecipe-Seite kann der Nutzer ein neues Rezept anlegen. Hierfür wurde ein ng-form mit einer FormGroup und FormControls genutzt, damit per MultiSelect mehrere Zutaten aus einer dynamischen DropDown-Liste ausgewählt und dem Rezept zugewiesen werden können. Das ng-form hat zudem noch die Felder „name“, „image“ und „steps“. Neben dem Ingredients-Dropdown hat der Nutzer zudem die Möglichkeit, über den Button „Create new ingredient“ das Pop-Up für die Zutatenerstellung zu öffnen. Sobald der Nutzer mit seinen Eingaben zufrieden ist, kann er das Rezept über den „Submit“-Button hinzufügen, daraufhin wechselt er zu der (aktualisierten) Recipe-Masterview. Er hat zudem auch jederzeit die Möglichkeit, die Erstellung mit dem „Cancel“-Button abzubrechen.
- **CreateIngredient**: (als Pop-Up) besteht aus einem ng-form mit den Feldern „name“, „category“, „unit“ und „amount“. Der Name kann beliebig gewählt werden, bei „category“ und „unit“ hat der Nutzer nur die Möglichkeit, aus den bereits existierenden Werten (welche aus dem IngredientService kommen), auszuwählen. Das Feld „amount“ kann nur mit einer Zahl gefüllt werden. Per „Submit“-Button kann die Zutat angelegt werden, per „Escape“-Taste oder durch Klicken neben das Pop-Up kann die Erstellung abgebrochen werden.
- **Shopping-Cart**: Im Einkaufskorb werden alle hinzugefügten Zutaten angezeigt. Diese werden über den ShoppingCartService bereitgestellt und im Shopping-Cart nach den jeweiligen Kategorien geordnet. Der Nutzer hat hier (bisher) keine weiteren Interaktionsmöglichkeiten. Die Methode createDistinct() wird bei jedem Aufruf der Seite ausgeführt und erstellt ein Array, in welchem jede Zutat des ShoppingCartService nur einmal vorkommt, bei Dopplungen erhöht sich die Menge der Zutat. Als Dopplung gilt eine Zutat mit gleichem Namen, gleicher Kategorie und gleicher Mengeneinheit.

Services:

- **RecipeService**: Hier ist ein Mock-Array mit einigen Rezepten definiert. Per Methode getRecipes() wird ein Observable des Arrays geliefert. Mit der Methode addRecipe(inputRecipe:Recipe) kann ein neues Rezept an das Array angehangen werden. Um redundante Daten zu verhindern, importiert der RecipeService auch eine Instanz des IngredientsService und besitzt anstatt dort angelegten Zutaten nur Verlinkungen auf den jeweiligen Index des Ingredient-Array des IngredientsService.
- **IngredientsService**: Hier werden die Kategorien und Mengeneinheiten (jeweils in einem eigenen Array) festgelegt. Zudem gibt es hier auch ein Mock-Array mit Ingredients („ingredientArray“), welches mit der Methode addIngredient(ing:Ingredient) erweitert werden kann.
- **ShoppingCartService**: Der ShoppingCartService verfügt über ein leeres Array, welches mit der Methode addItem(ingredient) gefüllt werden kann. getItems() gibt die Einträge des Arrays wieder, emptyShoppingCart() leert das Array wieder. Dies war während der Entwicklung mit Mock-Daten im ShoppingCart sehr hilfreich, kommt nun jedoch nicht mehr zum Einsatz. Auch die Sortierung (compare(a,b)) ist mittlerweile nicht mehr von Nöten, da die createDistinct()-Methode des ShoppingCartComponents keine vorsortierte Liste mehr benötigt, so konnte Laufzeit gespart werden und dem Nutzer wird trotzdem das gleiche Ergebnis geliefert.

Abschluss:

1. Lessons learned:
    a. CSS-styling kann durch ein Framework bereits vordefiniert sein, sodass Befehle, welche unter „normalen“ Umständen funktionieren, eventuell keine oder nicht die gewünschte Auswirkung haben.
    b. Es ist hilfreich, vor der Entwicklung von Methoden seine Gedanken fundiert in Pseudocode festzuhalten und sich daran zu orientieren.
    c. Es gibt Fehler, die nicht einfach so per StackOverflow zu beheben sind, sondern auch mit dem Framework an sich zu tun haben können (siehe: https://github.com/angular/angular/issues/7845)
2. Known bugs:
    a. Dynamisch generierte Komponenten, in meinem Fall die Bilder auf der Recipe-Detailpage bekommen erst nach einem Page-Refresh den CSS-Style. Dies hängt mit Angular an sich zusammen und lässt sich nicht zu 100% beheben (siehe Ticket 7845, Link oben). Ein work-around wäre es, keine Placeholder-Images zu benutzen (nur hier tritt zu 100% der Fehler auf, bei den anderen nur gelegentlich) und die Bildgröße selbst zu limitieren, um nicht auf das CSS-Styling zurückgreifen zu müssen. Dies schränkt den Nutzer jedoch sehr ein und erfordert zudem noch ein Image-Upload anstelle der simplen Bereitstellung einer URL. Der Arbeitsaufwand ist an dieser Stelle für eine solche Kleinigkeit zu groß, als dass es sich rentieren würde, dies umzusetzen. Ich hoffe, dies stößt auf Verständnis.
3. Future Features (nicht mehr im Scope dieses Projektes):
    a. Firebase-Anbindung mit Observables, um nicht mehr nur auf lokale Arrays zurückgreifen zu müssen
    b. Input-Validierung bei allen ng-forms