**Definition des Scopes und der Schnittstellen**

**Unser Scope**

Unser Scope ist es, unseren Integrated Protoype möglichst genau umzusetzen. Also eine App zu bauen, wo Kunden/Bürger und Bauern zusammenkommen. Bürger und Bauern haben unterschiedliche Berechtigungen. In unserem ER-Diagramm ist dies wie folgt umgesetzt:

* Es gibt die Tabelle „User“ mit einer entsprechenden userID, einem password und dem Fremdschlüssel roleID.
* Ein User hat genau eine Rolle. Mehrere User können die gleiche Rolle haben
* Die Entität „Role“ fasst die existierenden Rollen mit ihren Bezeichnungen zusammen.
* Mehrere Rollen können mehrere Berechtigungen haben.
* Berechtigungen (die Entität „Permission“) haben eine ID sowie eine Bezeichnung. Die m:n-Beziehung zwischen „Role“ und „Permission“ müsste in der Praxis in einer Verbindungstabelle aufgelöst werden.
* Damit besteht auch die Möglichkeit „speziellere“ Rollen als „Bauer“ und „Bürger“ zu erstellen und diesen Rollen die entsprechenden Berechtigungen zuzuteilen.

Diese Änderungen am ER-Diagramm und am Scope haben wir nach dem Feedback mit Patrick und Britta umgesetzt, um das daraus entstandene ActionItem zu erfüllen.

Diese Änderungen bestehen lediglich im ER-Diagramm. In der Implementierung wird nur zwischen „Bauer“ und „Bürger“ unterschieden und die Berechtigungen können noch verschwimmen.

Ich zähle kurz auf, wie der Scope in unserer jetzigen Implementierung umgesetzt ist und wie sich das durch das Einführen der oben genannten Maßnahmen ändern würde.

* Es gibt einen Login/Register Screen bei dem man bei der Registrierung angeben muss, ob man Bauer oder Bürger ist.
  + Nach der Umsetzung des aktualisierten Scopes sollte der User, der sich registrieren möchte, aus mehr als diesen beiden Möglichkeiten wählen können.
* Ein Home Screen zur Übersicht über Stellenangebote und Marktplatz Angeboten, sowie Neuigkeiten. Dieser Screen ist für beide (alle) Rollen gleich.
* Den Marktplatz, wo Bauern ihre Erzeugnisse direkt an die Kunden verkaufen können, d.h. Bauern können hier Angebote reinstellen. Sowohl Bauern wie auch Bürger können sich Angebote anderer Bauern anschauen wie auch kaufen.
  + Die Möglichkeit Angebote zu erstellen und zu kaufen könnte als jeweils eigene Berechtigung in der Tabelle „Permission“ umgesetzt werden.
* Wir haben davon abgesehen einen Warenkorb zu implementieren, dieser soll mit einem Pop-up Fenster simuliert werden.
* Es gibt Stellenanzeigen, wo nur Bauern Jobangebote reinstellen können, auf die sich dann alle Benutzer der App bewerben können. Das Bewerben soll hier auch nicht ausimplementiert werden.
* Es gibt einen Profil Screen, mit dem eigenen Profil und passenden Funktionen. Ausschließlich Bauern sollten zudem die Möglichkeit haben, ihre Angebote zu löschen. Hier wird noch keine Unterscheidung zwischen Bürger und Bauer getroffen, beide haben in der jetzigen Implementierung die Sicht auf „Beispiel-Angebote“. Das Löschen wird durch ein Pop-Up simuliert.

**Schnittstellendefinition**

Controller

Startet Funktionalität mit startProcess

Datenbankschnittstelle

Kann lesen und schreiben



DB-Java-Client

Kann Benutzer-Eingaben lesen.



MariaDB

View

Model\*



Wir haben die Architektur unserer Anwendung nach dem Model-View-Controller-Entwurfsmuster gerichtet. Unsere Architektur hat die folgenden Eigenschaften:

* UI´s werden durch das Interface „View“ abgebildet. Gibt es in einer View zusätzlich Eingaben, die der Benutzer tätigen muss, so muss das Interface „DataView“ ausimplementiert werden. DataView erbt von View.
  + Das Interface „DataView“ schreibt vor, dass die Methode getData ausimplementiert werden muss. Die Methode getData liest Benutzer-Eingaben aus und gibt diese in einem String Array zurück.
* Die Funktionalität des Programms wird in dem Interface „Controller“ abgebildet. Jeder Controller muss die Methode startProcess ausimplementieren, die die Funktionalität des Controllers startet. Das Interface „DataController“ ist das Gegenstück zur „DataView“. Es erbt von Controller.
  + Mit Hilfe der Methode getData kann der DataController sich die User-Eingaben von der View geben lassen.
  + Mit der entsprechenden Methode aus der Klasse „Validator“ kann sich der Controller diese Eingaben auf Validität überprüfen lassen (z.B. Datentyp).
  + Ein DataController muss die Methode createModel ausimplementieren. Damit kann der Controller ein Tupel für eine Entität erstellen und diese Werte in die Datenbank schreiben.

Es ergeben sich folgende Abhängigkeiten:

* Eine View wird zuerst erzeugt (z.B. beim Start der Applikation oder durch einen anderen Programmteil).
* Die View erzeugt einen Controller und ruft dessen startProcess-Methode auf. Somit ist der Controller in der View austauschbar und die View kennt die dahintersteckende Funktionalität nicht.
* Der startProcess-Methode wird die View mitgegeben. Dabei kann der Controller aber keine UI-Elemente verändern, er kann allerdings das Fenster (die View) schließen und sich auch die Benutzer-Eingaben mit der getData-Methode zurückgeben lassen ohne die jeweiligen UI-Elemente zu kennen.
* Es gibt Klassen, die das Model repräsentieren sollen. Das Model wird durch den Controller mit Hilfe von SQL-Statements ausgelesen und verändert.