Architekturdokumentation

Fachliche Modellierung

Die fachliche Modellierung der Splitter-Anwendung befindet sich im Package domain.model. Dabei ist ein Gruppen-Aggregat vorhanden, welches als Aggregat-Wurzel die Klasse “Gruppe” besitzt. Eine Gruppe wird über ihre ID identifiziert; darüber hinaus hat eine Gruppe einen Namen, eine Menge von Personen als Teilnehmer, eine Ausgabenliste und ist entweder offen oder geschlossen. Die Klasse “Ausgabe” ist eine Entität des Aggregats mit eindeutiger ID. Sie enthält eine Person als Geldgeber, welcher Geld für andere Teilnehmer der Gruppe vorstreckt, eine Beschreibung des Ausgabenzwecks, die Höhe der Ausgabe als Geldbetrag und eine Menge von Personen, die von der Ausgabe profitieren.
Die Wertobjekte “Person” und “Geld” sind Hilfsklassen des Aggregats. Ein weiteres Wertobjekt ist die Klasse “Schuld”, welche Zahler, Empfänger und Betrag einer Ausgabe definiert und die Klasse “Konto”, in der für eine Person ein positiver oder negativer Geldbetrag angegeben wird, der anzeigt ob die Person Geld schuldet oder geschuldet bekommt.

Die Geschäftslogik findet in domain.service statt in der Klasse “Berechnungsservice”. Dort werden die gesamten Schulden einer Gruppe berechnet, indem über die Ausgabenliste der Gruppe für jede Person ein Kontostand berechnet wird. Die Summe aller negativen und positiven Kontostände innerhalb einer Gruppe ergibt immer 0.

Komponentenstruktur

Die Anwendung ist als Onion-Architektur konzipiert. Die fachliche Modellierung befindet sich in der innersten Schicht im Package domain.model, gefolgt vom Berechnungsservice in domain.service. Der GruppenService im package application.service stellt den Übergang zum Frontend und über die Repositories eine Schnittstelle zur Persistenz/Datenbank bereit.

Das Frontend beinhaltet einen WebController, welcher POST- und GET-Requests verwaltet und einen REST-Controller, welcher eine API-Schnittstelle über das Serialisierungsformat JSON bereitstellt. Beide Controller greifen auf den Gruppenservice zu. Mit dem Unterpackage webdomain existiert eine Schnittstelle zwischen den fachlichen Klassen und den Anforderungen des Frontends, um eine zu hohe Kopplung zu vermeiden. Damit nur über GitHub eingeloggte User auf die Anwendung zugreifen können, wird OAuth eingebunden und in der Klasse WebSecurityConfiguration konfiguriert. Dabei ist die REST-Schnittstelle wie gefordert nicht abgesichert.

Im package repositories liegt die Schnittstelle zur Persistenzschicht. Dort wird das Interface GruppenRepository aus domain.service implementiert und ein DBGruppenRepo angelegt, welches vom CrudRepository erbt. Im Unterpackage dbdomain findet eine Übersetzung der fachlichen Klassen in die Datenbankkonzepte statt, um auch hier eine zu hohe Kopplung zu verhindern.

Kontextabgrenzung

Neben der Web-Schnittstelle gibt es eine REST-Schnittstelle, welche mittels JSON angesprochen werden kann. Über die REST-Schnittstelle können Clients neue Gruppen erzeugen, sowie Gruppen schließen, als auch Auslagen eintragen. Zudem kann man Informationen zu einzelnen Gruppen und mehreren Gruppen einer Personen anfragen und auch alle Ausgleichszahlungen abfragen.

Das folgende Diagramm stellt das beschriebene Modell dar: