Spliddit

Eine Webapplikation um Kosten gerecht in einer Gruppe aufzuteilen

Einführung

Spliddit ist eine Applikation, die im Web oder auf dem Smartphone läuft, um Kosten in einer Gruppe gerecht aufzuteilen.
Hierbei soll die Kerneigenschaft sein, dass man eine Rechnung, die in der Gruppe angefallen ist, hochlädt und die Applikation die Rechnungssumme erkennt. Diese Summe kann dann auf Gläubiger und Schuldner aufgeteilt werden. Standardmäßig ist derjenige, der die Rechnung hochgeladen hat, der Gläubiger des gesamten Betrags, sowie jedes Gruppenmitglied die Schuldner des gleichmäßig aufgeteilten Betrags.

Ablauf

Wenn man die App startet sieht man einen **Loginbildschirm**. Hier kann man auch auf einen **Registrierungsbildschirm** weitergeleitet werden. Nachdem man sich eingeloggt hat sieht man ein
Im Dashboard finden sich 3 Funktionen:
Eine **Rechnung abfotografieren und hochladen** oder eingeben (ab hier ein neuer Bildschirm)
Rechnung wird erkannt und Rechnungsbetrag wird angezeigt
Rechnung wird nicht erkannt und Rechnungsbetrag wird eingegeben
Man kann "**Bezahler**" der Rechnung **auswählen**.
Man kann "**Konsumenten**" der Rechnung **auswählen**.
Nachdem man die Rechnung bearbeitet hat und wieder ins Dashboard gelangt, sieht man **Schuldner und Gläubiger von der Person, die gerade eingeloggt ist.**

Features

Bei den Features haben wir die Core Features für einen MVP festgelegt. Die Nice to have Features sind für einen weiteren Ausbau angedacht worden.

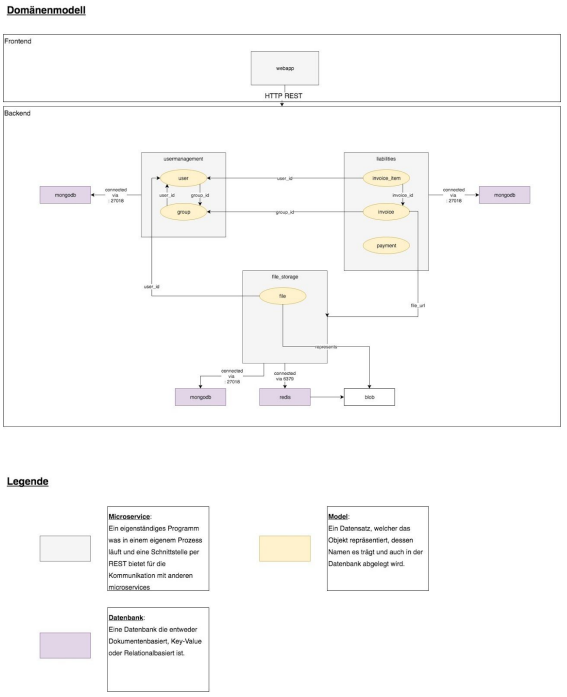
Core (Must) Features

Feature
Gruppe erstellen
Person zu einer Gruppe hinzufügen
Rechnung hochladen
OCR Auswertung der Rechnung
Eingabe von Rechnungsinformationen
Verteilung der Kosten auf Schuldner
Verteilung der Kosten auf Gläubiger
Dashboard, in dem ich sehe, wie vielen Leuten ich was schulde
Dashboard, in dem ich sehe, wie viele Leute mir was schulden

Nice To have Features

Feature
Zahle direkt mit Paypal den Schuldbetrag
Einloggen mit Facebook
Einloggen mit Google

Technische Architektur

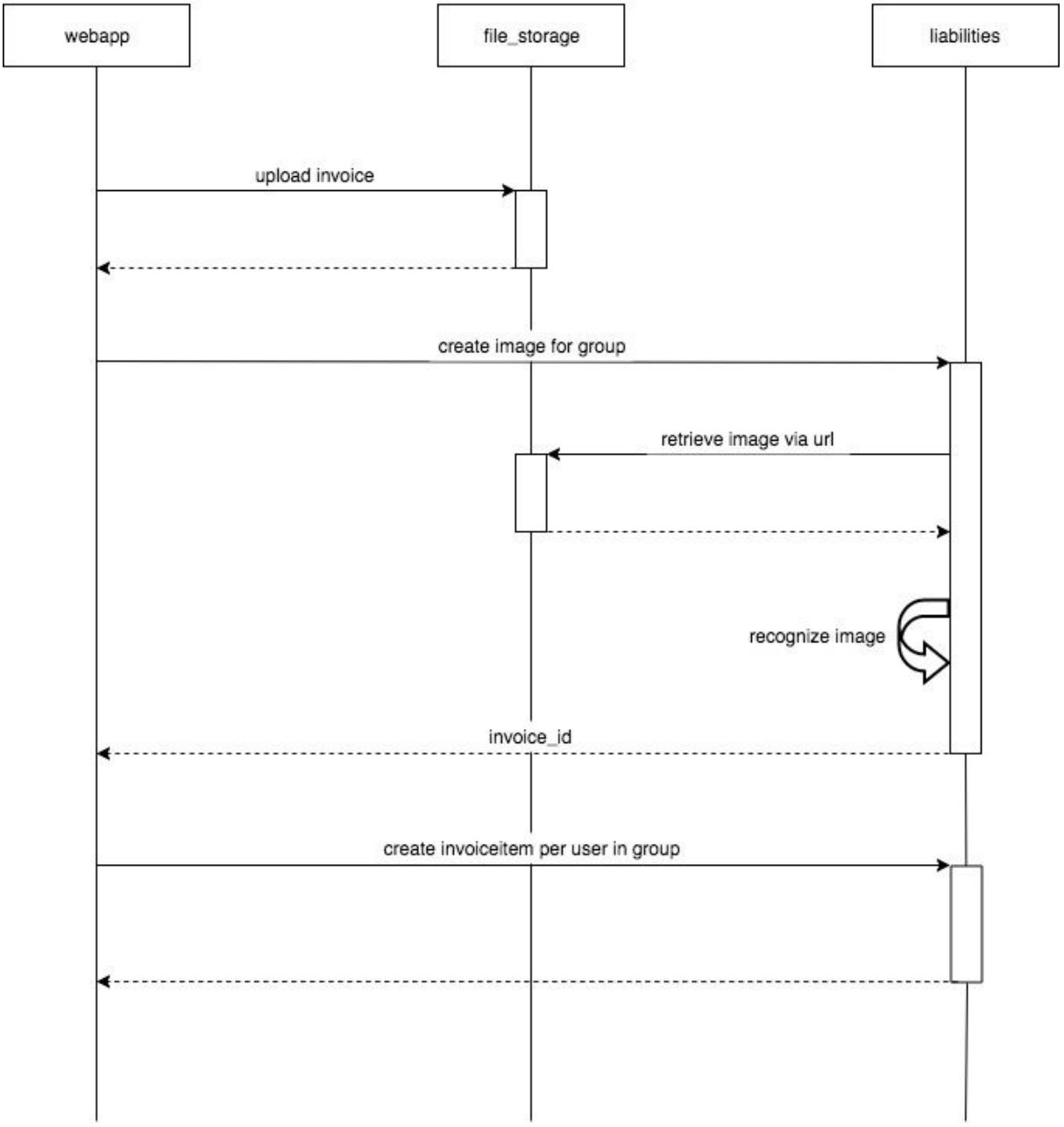


(Anhang: Technische Architektur.jpeg)

Bei der Architektur wurde beachtet, dass wir eine Microservice Architektur benutzen, die **Domänen-Orientiert** (und nicht Feature-Orientiert) ist.
Alle Microservices kommunizieren über eine eigene REST Schnittstelle. Jeder Microservice besitzt auch seine eigene Datenbank, um die Microservices wirklich voll voneinander abzukoppeln und sie auch auf eigenen Servern starten zu können. Über IDs können Relationen zwischen zwei Models aus verschiedenen Microservices hergestellt werden. Bei Bedarf wird dann ein Request an den jeweiligen Service geschickt.

Beispielablauf zwischen Microservices

Rechnungserstellung



Frontend

Entscheidung für React statt Angular 2 im Frontend

Die Entscheidung für Reactstatt Angular 2 fiel ziemlich schnell.
Angular 2 ist ein all-in-one JS Framework, was zahlreiche Funktionen anbietet und welche auch gut in das Framework integriert sind. Bei **React hat man hingegen einen best-of-breed Ansatz** und kriegt quasi nur einen “HTML Renderer”. Alle weiteren Funktionen, die man braucht, müssen selbst implementiert oder mit Bibliotheken gefüllt werden. Da somit die Lernkurve und das generelle Verständnis eines JS Frameworks bei React kürzer ist, liegt dies auf der Hand. Das hat dazu geführt, dass wir uns für React entschieden haben, um somit die Entwicklungszeit für Features in den Backends zu gewinnen um im Frontend lediglich schnell eine Lösung zu bauen und diese Features anzuzeigen.