Audit MS1

Projektidee:

- Kassenzettel-Scan um Zettelchaos zu vermeiden und Kaufverhalten analysieren zu können
- Aus gewonnenen Daten können Statistiken erstellt werden
- Kollaboration in einer Gruppe, Teilen von Kassenzetteln
- Abrechnungen innerhalb der Gruppe
- Statistiken können anonymisiert an Dritt-Interessenten verkauft werden

Alleinstellungsmerkmal:

- Scan einzelner Artikel statt nur Gesamtdaten des Kassenzettels
- Echtzeitsynchronisation und Einbindung aller Teilnehmer bei der Abrechnung
- Spezielles Design und Features für Nutzergruppe Backpacker
    - Unterstützung der Kommunikation
    - Aufbau von Vertrauen innerhalb der Reisegruppe durch Transparenz

Präskriptive Aufgabenmodellierung:

- Für Use Case Abrechnung muss ein Kassenzettel in der Datenbank vorliegen und Schuldner darin referenziert sein
- Kassenzettel kann mit OCR gescannt oder manuell eingetragen werden

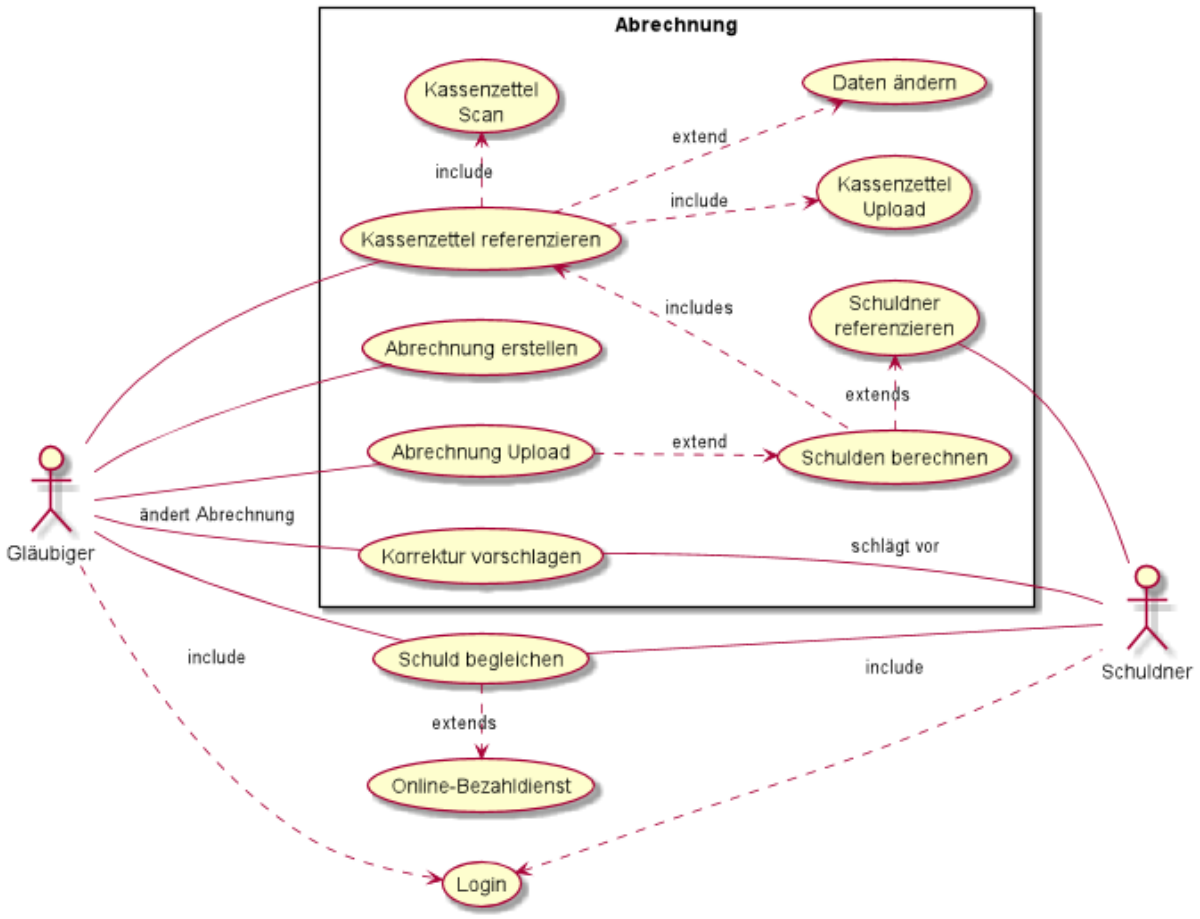

Figure 1: Use Case einer Abrechnung

- Dieses Schema beschreibt Datenschema eines Kassenzettels
- Teilnahme-Schema referenziert z.B. einen Nutzer mit seiner prozentualen Beteiligung am Artikel

```
const mongoose = require("mongoose");

const Schema = mongoose.Schema;

const participationSchema = new Schema ({
    participant: {type: Schema.Types.ObjectId, ref: "User"},
    percentage: Number
});

const articleSchema = new Schema ({
    name: {type: String, required: true},
    price: {type: Number, required: true},
    participation: [{type: participationSchema}],
    category: [{type: Schema.Types.ObjectId, ref: "Category"}],
    amount: Number,
    priceAll: Number
});

const addressSchema = new Schema ({
    street: {streetName: String, streetNumber: Number},
    location: {zipCode: Number, city: String, state: String},
    country: String
});

const receiptSchema = new Schema ({
    owner: {type: Schema.Types.ObjectId, ref: "User", required: true},
    store: {type: String, required: true},
    date: {type: Date, default: Date.now()},
    imagePath: {type: String, default: "folgt", required: true},
    address: {type: addressSchema},
    article: [{type: articleSchema, required: true}],
    total: {type: Number, required: true},
    payment: {type: Number, required: true},
    change: {type: Number, required: true},
});

module.exports = mongoose.model("Receipt", receiptSchema);
```

Figure 2Datenschema eines Kassenzettels

- Im Abrechnungs-Schema referenziert der Gläubiger seine abzurechnenden Kassenzettel und daraufhin wird das Schuld-Schema beim Start des Abrechnungsalgorithmus automatisch aktualisiert

```
const mongoose = require("mongoose");

const Schema = mongoose.Schema;

const articleDebtSchema = new Schema ({
    articleId: {type: Schema.Types.ObjectId, ref: "Article"},
    articleName: String,
    debt: Number
});

const debtSchema = new Schema ({
    userId:{type: Schema.Types.ObjectId, ref: "User"},
    articles: [{type: articleDebtSchema}],
    deptTotal: Number
});

const settlementSchema = new Schema ({
    creditor: {type: Schema.Types.ObjectId, ref: "User", required: true},
    debtor: [{type: debtSchema}],
    receipts: [{type: Schema.Types.ObjectId, ref: "Receipt"}],
    total: Number,
    finished: {type: Boolean, default: false}
});

module.exports = mongoose.model("Settlement", settlementSchema);
```

Figure 3Datenschema einer Abrechnung

Architekturskizze:

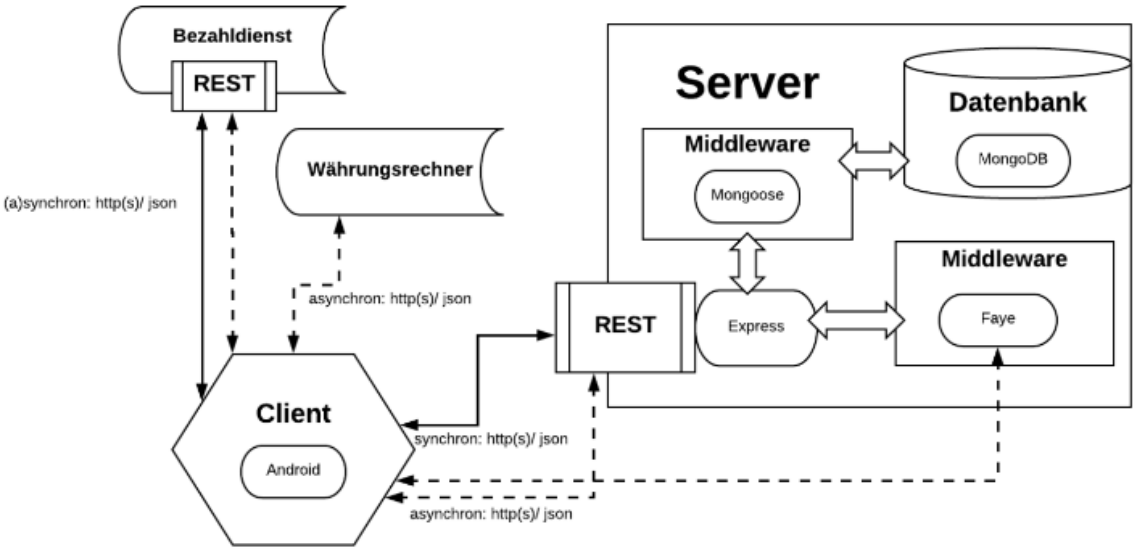

Figure 4Architekturskizze

- Client greift über ein REST-Interface auf die Datenbank und die Anwendungslogik des Servers zu
- Externe Webservices wie Online-Bezahldienst und Währungsrechner können über den Client eingebunden werden
- Für eine zeitlich entkoppelte, asynchrone Kommunikation zwischen Client und Server wird ein PubSub-Service implementiert
- REST wird verwendet um Server und Client zu entkoppeln, so dass es möglich ist verschiedene Clients zu implementieren
- Zudem lässt sich dadurch die Datenbank gut einbinden und es ist beliebig weiter skalierbar
- Als Datenbank wird MongoDB verwendet, da diese skalierbarer und effizienter ist als SQL und JSON verwendet
- Die Vorteile von SQL (feste Schemas, Referenzieren und und Querying) können mit der Erweiterung Mongoose dennoch genutzt werden
- Aus Sicherheitsaspekten sollen Authentifizierung und Bezahldienst synchron ablaufen
- Restliche Kommunikation sollte asynchron laufen, es sei denn die Dauer der Kommunikation ist so kurz, dass sich Asynchronität nicht lohnt

Proof of Concept:

- Der Use Case Abrechnung stellt den größten Teil unseres Alleinstellungsmerkmals dar und wurde daher vorgezogen für den Rapid Prototyp
    - Exit:
        - anhand der referenzierten Kassenzettel einer Abrechnung wird die Kostenverteilung pro Artikel ausgerechnet
        - die Abrechnung mit den Ergebnissen aktualisiert
        - alle Nutzer erhalten eine Mitteilung darüber
        - können das Ergebnis überprüfen.
    - Fail:
        - Kostenverteilung falsch ausgerechnet
        - ist für Nutzer nicht einsehbar bzw. kommen keine Mitteilung

        - falsche Nutzer wurden referenziert
        - Ergebnis wird von Nutzern nicht anerkannt
    - Fallback: Referenzen und Parameter müssen nachträglich änderbar sein