Splittid

Eine Webapplikation im ME(~~A~~)RN Stack

Inhaltsangabe

1. Aufgabe
2. Konzept & Technologien
3. Architektur
4. Demo
5. Perspektiven

Inhaltsangabe

1. **Aufgabe**
2. Konzept & Technologien
3. Architektur
4. Demo
5. Perspektiven

Aufgabe

- Webapplikation
- MEAN Stack
- Paralleles und agiles Arbeiten in der Arbeitsgruppe durch Microservices
- Applikation zum Teilen von Schulden in einer Urlaubsgruppe

Inhaltsangabe

1. Aufgabe
2. **Konzept & Technologien**
3. Architektur
4. Demo
5. Perspektiven

Konzept & Technologien

User Story:

Als Gruppenmitglied (z.B. im Urlaub) möchte ich gemeinsam Geld ausgeben können, ohne mir Gedanken darüber machen zu müssen, wer mir was schuldet. Die Kosten sollen nach dem Urlaub gerecht aufgeteilt werden.

Konzept & Technologien

Standardfall:

1. Eine Gruppe mit 4 Personen war essen für 40€
2. Ein zahlungsfähiges Mitglied
3. Mitglied bezahlt und scannt die Rechnung oder gibt alternativ Rechnungsbetrag ein
4. Kosten werden gleichmäßig auf Gruppenmitglieder aufgeteilt
5. Jedes Gruppenmitglied bezahlt 10€

Konzept & Technologien

Randfall 1:

1. Eine Gruppe mit 4 Personen war essen für 40€
2. Zwei zahlungsfähige Mitglieder
3. Mitglied 1 & 2 bezahlt
4. Mitglied 1 scannt die Rechnung oder gibt alternativ Rechnungsbetrag ein
5. Mitglied 2 wird als Bezahler hinzugefügt
6. Kosten werden gleichmäßig auf Gruppenmitglieder aufgeteilt
7. Jedes Gruppenmitglied bezahlt 10€

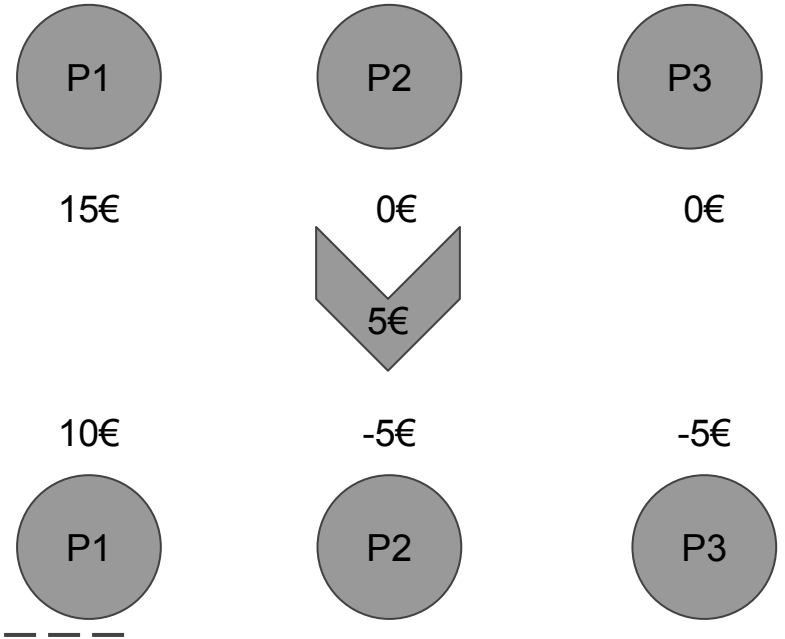
Konzept & Technologien

Randfall 2:

1. Eine Gruppe mit 4 Personen war essen für 40€
2. Ein zahlungsfähiges Mitglied
3. Mitglied bezahlt und scannt die Rechnung oder gibt alternativ Rechnungsbetrag ein
4. Eine Person wurde aus der “war beteiligt” Liste rausgenommen
5. Kosten werden gleichmäßig auf Gruppenmitglieder aufgeteilt
6. Drei Gruppenmitglieder bezahlen 13,33€

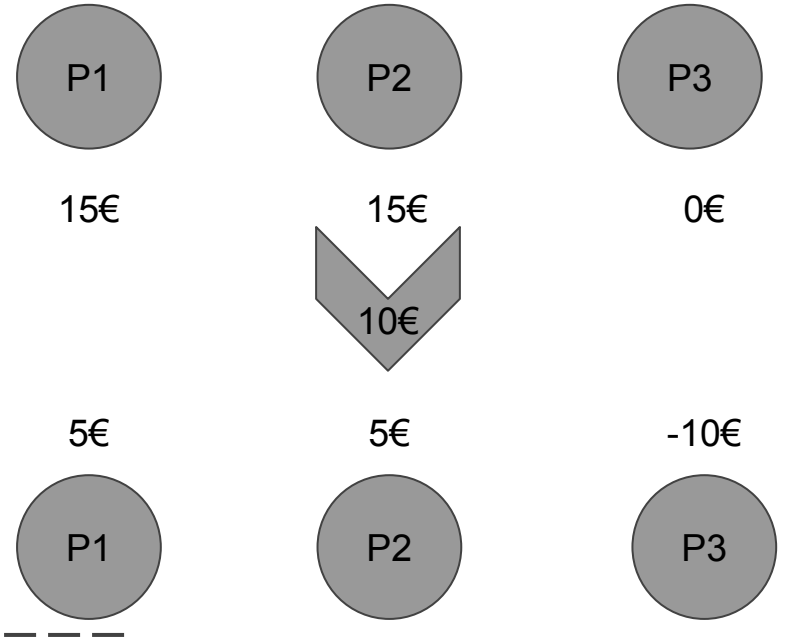
Konzept & Technologien

Algorithmus:



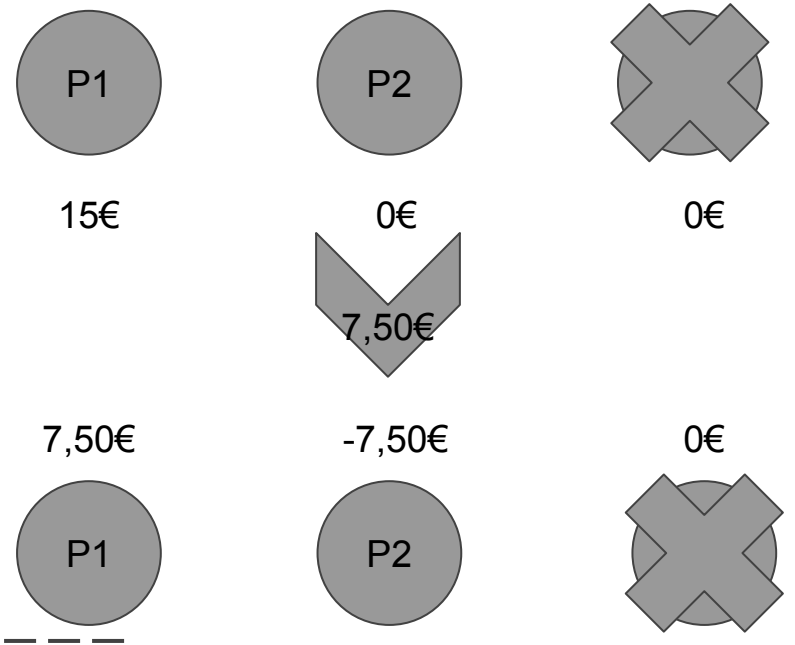
Konzept & Technologien

Algorithmus:



Konzept & Technologien

Algorithmus:



```
/*
 * Will go through all invoices and return each invoice_id
 * with the open debts to the respective creditors.
 * The key of the response is the receiving user and the value the money that user is getting.
 * If the number is negative, then the passed user_id owes that much to him.
 */
GetBalancesOfUser(user_id) {
  return new Promise(
    (resolve, reject) => {
      this.GetAnalysisObjectsWithUser(
        user_id
      ).then(
        (analysis_objects) => {
          let credits_debts = {};
          analysis_objects.forEach(
            (analysis_object) => {
              // Look if there are users otherwise it would be divide by zero
              if (analysis_object.invoice_items) {

                // break up if the user is not in the invoice_items
                if (!analysis_object.invoice_items.find((invoice_item) => invoice_item.user_id === user_id)) return;

                let price_per_person = analysis_object.total_price / analysis_object.invoice_items.length;
                let debitors = [];
                let creditors = [];

                // First find all debitors in one list and debitors in another list
                // Do a delta if a person is creditor to see if it receive or have to give money
                analysis_object.invoice_items.forEach(
                  (invoice_item) => {
                    if (invoice_item.role === 'creditor') {
                      let delta = invoice_item.advanced_price - price_per_person;
                      if (delta !== 0) {
                        if (delta > 0) {
                          creditors[invoice_item.user_id] = delta;
                        } else {
                          debitors[invoice_item.user_id] = delta;
                        }
                      }
                    } else {
                      debitors[invoice_item.user_id] = price_per_person;
                    }
                  }
                );

                this.StraightenDebitorsCreditors(debitors, creditors, user_id, credits_debts);
              }
            }
          );

          resolve(credits_debts);
```

```
StraightenDebitorsCreditors(debitors, creditors, user_id, credits_debts) {
  // Straighten the whole thing up
  /*
   First order all debitors and creditors from lowest to highest and then give the creditors their money
   */
  debitors.sort((a, b) => a - b);
  creditors.sort((a, b) => a - b);

  // Iterate over all debitors
  for (let debitor_key in debitors) {

    // Look if the current debitor still has money
    while (debitors[debitor_key] !== 0) {

      // Iterate over all creditors
      for (let creditor_key in creditors) {
```

```
// jump over if the creditor has all his money back
if (creditors[creditor_key] === 0) return;

// Give the creditors their money back
// Check if the charge is bigger than the creditor has
// so he can give him all his money
// otherwise you have to subtract the creditors charge from the money a debitor has
// and do the next round
if (creditors[creditor_key] - debitors[debitor_key] >= 0) {

  // If the user_id is in debitors than he has to give money to the creditor_key
  // We only need our information so check if the creditor_key is "me"
  if (user_id in debitors && user_id === debitor_key) {
    this.AppendToCreditsDebts(credits_debts, creditor_key, -debitors[debitor_key]);
  }

  // If the user_id is in creditors than the debitor_key has to give the user money
  // We only need our information so check if the creditor_key is "me"
  if (user_id in creditors && user_id === creditor_key) {
    this.AppendToCreditsDebts(credits_debts, debitor_key, debitors[debitor_key]);
  }

  creditors[creditor_key] -= debitors[debitor_key];
  debitors[debitor_key] = 0;
} else {

  // If the user_id is in debitors than he has to give money to the creditor_key
  // We only need our information so check if the creditor_key is "me"
  if (user_id in debitors  && user_id === debitor_key) {
    this.AppendToCreditsDebts(
      credits_debts,
      creditor_key,
      -(debitors[debitor_key] - creditors[creditor_key])
    );
  }

  // If the user_id is in creditors than the debitor_key has to give the user money
  // We only need our information so check if the creditor_key is "me"
  if (user_id in creditors && user_id === creditor_key) {
    this.AppendToCreditsDebts(
      credits_debts,
      debitor_key,
      debitors[debitor_key] - creditors[creditor_key]
    );
  }

  creditors[creditor_key] = 0;
  debitors[debitor_key] -= creditors[creditor_key];
}
```

Konzept & Technologien

Technologie Stack:

- Programmiersprachen
  - ES6
- Microservices
  - Express.JS (REST Service)
  - MongoDB (Datenhaltung)
  - Redis (FileStorage)
- Frontend
  - React
  - React Router (Routing)
  - Axios (HTTP Requests)

Konzept & Technologien

ES6

Vorteile:

- Klassenorientiert
- Scopes wurden verbessert
- Variablen (let, const)
- Modules

Konzept & Technologien

ExpressJS

Vorteile:

- Keine Umstellung der Programmiersprachen, bzgl. Front-, Backend
- Asynchrone Requests und Responses
- Enormes Angebot an Plugins für ExpressJS

Konzept & Technologien

React

…und warum wir kein Angular benutzt haben

- Angular ist mehr als nur ein HTML Renderer
- Angular bietet alle Funktionalitäten (“All in one”) von Haus aus an
  - React benutzt den “best of breed” Ansatz
- Es lässt sich somit schnell und einfach ein MVP erzeugen
- Das “drumherum” muss nicht programmiert werden

Angular 2 Component

```
import { Component } from '@angular/core';

@Component({
  selector: 'my-component',
  template: `
    <div>Hello my name is {{name}}.
      <button (click)="sayMyName()">
        Say my name
      </button>
    </div>
  `
})
export class MyComponent {
  name: string;
  constructor() {
    this.name = 'Max'
  }
  sayMyName() {
    console.log('My name is', this.name)
  }
}
```

React Component

```
import React from ‘react’;

class Greeting extends React.Component {
  constructor(props) {
     super(props);
     this.state = {
       name: ‘Max’
     };
  }
  sayMyName() {
    console.log(this.state.name);
  }
  render() {
    return (
      <div>Hello my name is {this.state.name}.
        <button onClick={this.sayMyName}>
          Say my name
        </button>
      </div>
    );
  }
}
```

Konzept & Technologien

Microservices

- usermanagement (REST-Service)
  - CRUD Benutzer
  - CRUD Gruppen
- liabilities (REST-Service)
  - CRUD Rechnungen
  - CRUD Rechnungspositionen
  - Rechnungsscanner mit OCR
  - Berechnung der Schuldenaufteilung

Konzept & Technologien

Microservices

- file_storage (REST-Service)
  - CRUD Dateien
  - Speicherung im Redis
- webapp (React Application)
  - Zuständig für Frontend-Rendering mit Hilfe der Daten von unseren Microservices

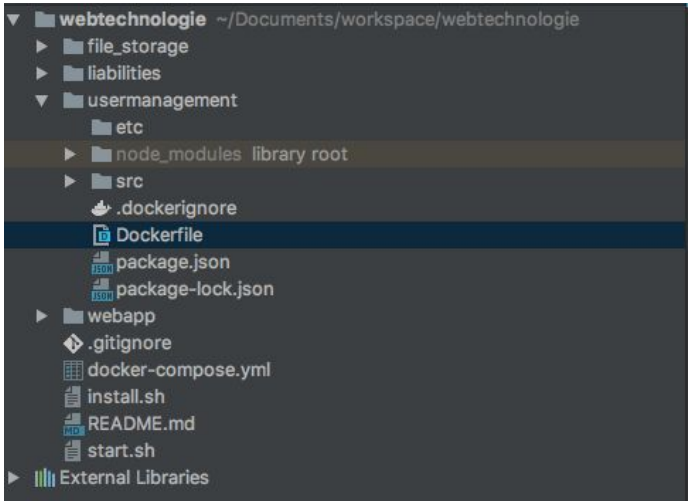
---

Konzept & Technologien

Docker

- Zum testen, bereitstellen, deployen der Applikation
- Definiert pro Webservice
- docker-compose für gesamte Applikation

Konzept & Technologien



Konzept & Technologien

```
FROM node:latest
RUN mkdir /usermanagement
WORKDIR /usermanagement
COPY package.json /usermanagement
RUN npm install

ENV MONGODB_URL localhost
ENV MONDODB_PORT 27017
ENV PORT 8002

EXPOSE 8002
CMD ["npm", "start"]
```

Konzept & Technologien

```
version: "2"
services:
  file_storage_mongo_db:
    image: mongo
    ports:
      - "27017:27017"
  file_storage_redis:
    image: redis
    ports:
      - "6379:6379"
  file_storage:
    build: file_storage
    ports:
      - "8000:8000"
    volumes:
      - "./file_storage:/file_storage"
    links:
      - file_storage_mongo_db
      - file_storage_redis
    depends_on:
      - file_storage_mongo_db
      - file_storage_redis
    environment:
      - MONGODB_URL=file_storage_mongo_db
      - MONGODB_PORT=27017
      - REDIS_URL=file_storage_redis
      - REDIS_PORT=6379
      - PORT=8000
```

Inhaltsangabe

1. Aufgabe
2. Konzept & Technologien
3. **Architektur**
4. Demo
5. Perspektiven

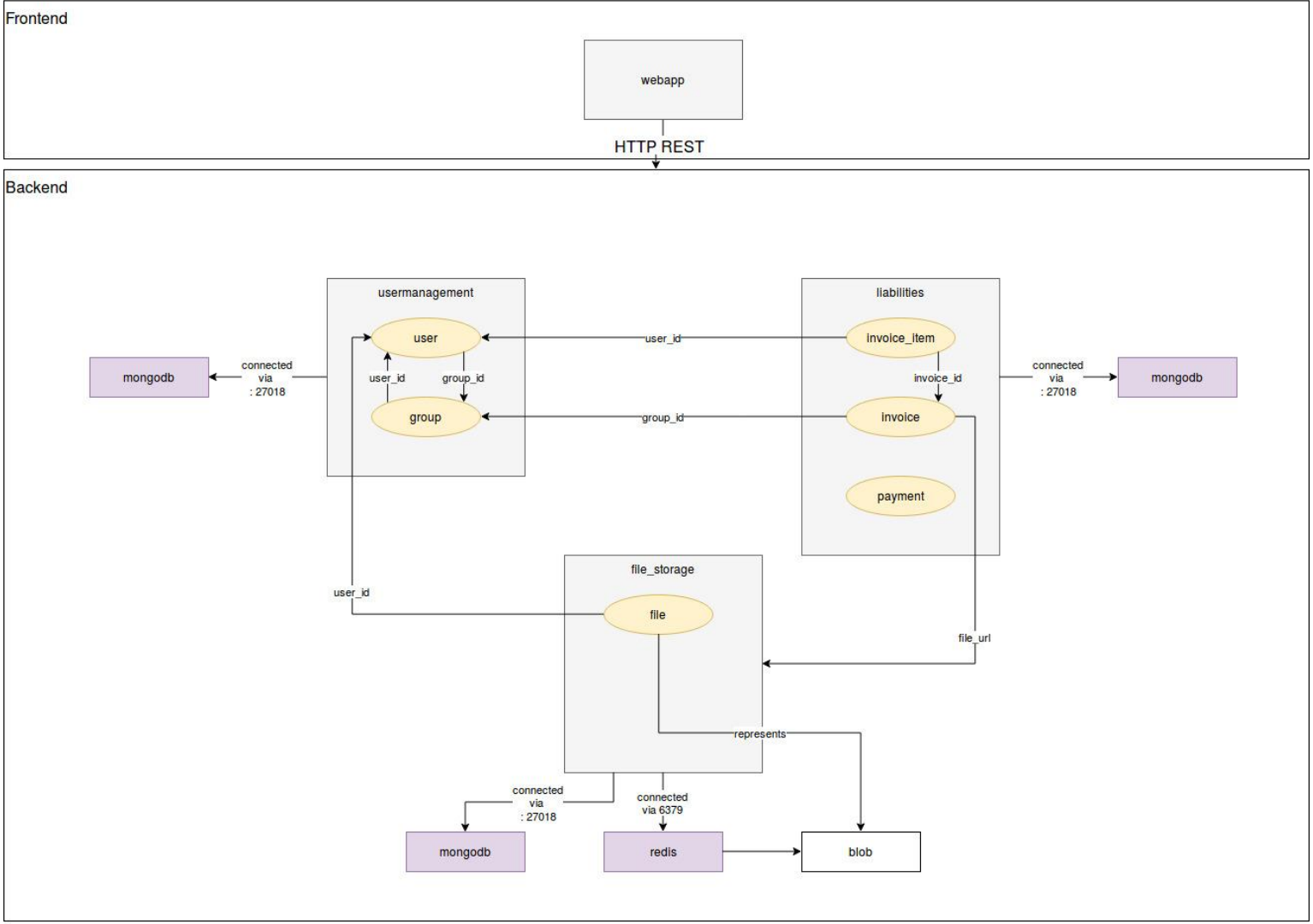
Architektur

Legende

Microservice:
Ein eigenständiges Programm was in einem eigenem Prozess läuft und eine Schnittstelle per REST bietet für die Kommunikation mit anderen microservices

Model:
Ein Datensatz, welcher das Objekt repräsentiert, dessen Namen es trägt und auch in der Datenbank abgelegt wird.

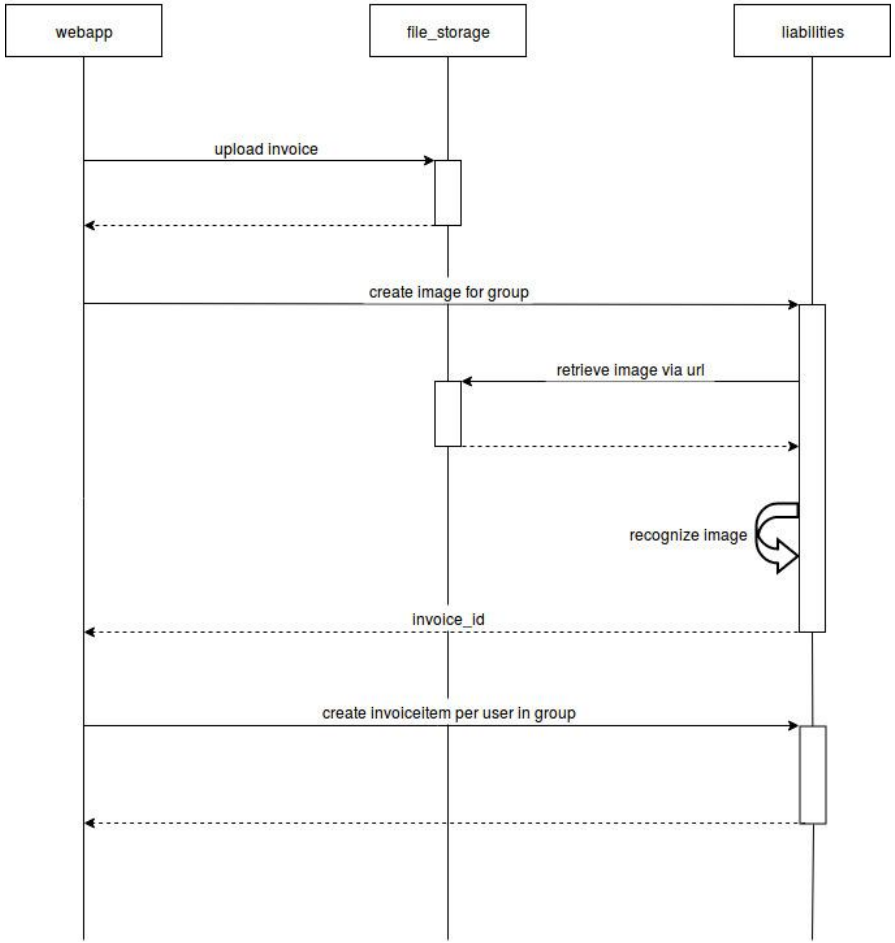
Datenbank:
Eine Datenbank die entweder Dokumentenbasiert, Key-Value oder Relationalbasiert ist.


Frontend
webapp
HTTP REST
Backend
usermanagement
user
group
user_id
group_id
liabilities
invoice_item
invoice
payment
invoice_id
user_id
group_id
mongodb
connected via : 27018
mongodb
connected via : 27018
file_storage
file
user_id
file_url
represents
connected via : 27018
connected via 6379
mongodb
redis
blob

Architektur

Rechnungserstellung

Rechnungserstellung



Architektur

OCR:

1. Bild wird im liabilities-service vom filestorage-service heruntergeladen
2. OCR Scan wird initialisiert und erkennt das Bild(tesseract.js)
3. Größter erkannter Wert wird zurückgegeben
4. Falls falsch erkannt, wird der Rechnungsbetrag vom Frontend aus geändert

Inhaltsangabe

1. Aufgabe
2. Konzept & Technologien
3. Architektur
4. **Demo**
5. Perspektiven

Inhaltsangabe

1. Aufgabe
2. Konzept & Technologien
3. Architektur
4. Demo
5. **Perspektiven**

Perspektiven

1. Autom. Bezahlung
    a. Paypal
    b. Bitcoin
    c. Kreditkarte
2. Benachrichtugung
    a. SMS
    b. E-Mail
3. Telegram Bot
4. Drucken
5. Erinnerung für offenstehende Posten

Perspektiven

Portierung in eine “native” mobile App auf iOS und Android

(schon gegeben durch Responsive Design)

– – –

Ende