Technical Report

1st Alexander Ziebell a.ziebell@oth-aw.de
2nd Anja Stricker a.stricker@oth-aw.de
3rd Annika Stadelmann a.stadelmann@oth-aw.de
4th Leo Schurrer l.schurrer@oth-aw.de
5th Philip Bartmann p.bartmann@oth-aw.de
6th Ronja Bäumel r.baeumel@oth-aw.de
7th Ulrich Stark u.stark@oth-aw.de

***Zusammenfassung*—Nach einem gemeinsamen Urlaub mit Freunden, bei denen man sich untereinander immer wieder Geld ausgelegt hat, ist die Verwirrung darüber, wer wem wie viel Geld schuldet, oft vorprogrammiert.**

Um dem entgegen zu wirken, bieten wir nun eine Lösung: Die Anwendung „Wo ist mein Geld?“ erlaubt den Benutzern für verschiedene Anlässe stets den Überblick über gemeinsame Ausgaben mit Freunden oder Kollegen zu bewahren.

Doch auch bei der Umsetzung dieser Anwendung kann es einige Hürden geben, die es zu überwinden gibt.

I. Einleitung

Um den Überblick zu bewahren, wer wem wie viel Geld nach einem Ausflug oder Urlaub mit Freunden schuldet, entwickeln wir eine Webanwendung. Mit „Wo ist mein Geld?” kann man stets genau sehen, wer eine Aktivität bezahlt hat, wer die Schuldner sind und die zugehörigen Zahlungsbeträge und die Ausgleichszahlungen an den Gläubiger.

Wie funktioniert aber nun genau die Anwendung?

Nach dem Starten und Eingeben des Namens kann man Gruppen für verschiedene Anlässe mit verschiedenen Personen erstellen. Auch ist es möglich, einer schon bestehenden Gruppe beizutreten oder neue Mitglieder hinzuzufügen.

Sobald Ausgaben eingetragen werden, berechnet die Anwendung die entsprechenden Ausgleichszahlungen für die Gruppenmitglieder. Diese können auch visualisiert dargestellt werden, um den Überblick zu behalten.

So können gemeinsame Ausgaben mit Freunden oder Kollegen individuell verwaltet werden.

Doch wie sieht es im System aus? Wie ist die Architektur? Welche Hindernisse könnten auftreten? All das sind Fragen, die im Folgenden beantwortet werden sollen.

II. Problemstellung

Die Problemstellung spiegelt sich in den User-Story wider. Für den Minimum Viable Product gibt es folgende User-Stories:

- Die Ausgaben in einer Liste, um Überblick zu bewahren.
- Berechnung von Ausgleichszahlungen.
- Eine Ausgabe anlegen oder löschen.

Auch die restlichen User-Stories, die umgesetzt werden sollen, beinhalten jeweils ein Teil der Problemstellung:

- Einen Benutzer-Account erstellen.
- Einloggen in den Benutzer-Account.
- Gruppen für verschiedene Anlässe und Personen anlegen.
- Gruppe löschen.
- Personen zu Gruppe hinzufügen oder löschen.

Die Akzeptanzkriterien finden sich im Fachkonzept. Wichtig ist hier also ein System zu entwickeln, welches nicht nur optisch ansprechend, sondern auch praktikabel ist. Dabei sollte man immer im Hinterkopf haben, dass der User die Benutzung der Anwendung nicht nur erlernen, sondern verstehen muss. Das heißt, dass die Benutzung so intuitiv und einfach wie möglich ist.

III. Architekturentscheidung

A. Lösungsstrategie

Die Ziele, die verfolgt werden, lassen sich mit den Technologien verknüpfen, die verwendet werden. In der Tabelle (vgl. Tabelle I) kann man erkennen, welche Technologie für die Erreichung eines Architekturziels eingesetzt wurde.

Um die Architekturentscheidungen, die getroffen wurden, möglichst detailliert herausarbeiten zu können, wurde das System in drei Bereiche unterteilt:

B. Frontend

Im Frontend wird React mit TypeScript genutzt und ein Vite-Server für das Hosting. Zusätzlich wird für das Design, wenn möglich, das Material UI-Theme verwendet. Findet sich keine gute Lösung in dem Theme, so wird mit SCSS nachgeholfen.

C. Backend

Das Backend besteht aus TypeScript mit einem Express-Server und Sequelize, die in einer NodeJS-Umgebung laufen. Diese besitzt eine Anbindung an den Docker-Container, in dem die Datenbank läuft.

D. Datenbank

Die MySQL-Datenbank läuft in einem Docker-Container. Hierbei werden die Tabellen automatisch beim Starten des Containers angelegt und mit mit vorher festgelegten Testdaten befüllt. Wie man im Entity-Relationship-Modell (vgl. Abbildung 1) erkennen kann, gibt es fünf Tabellen.

1) USER: In dieser Tabelle wird der User bei der Registrierung angelegt. Gespeichert werden sein eingegebener Name, der nicht in der Datenbank vorhanden sein darf, sowie der Hash-Wert seines Passworts. Außerdem erhält er eine ID als primären Schlüssel.

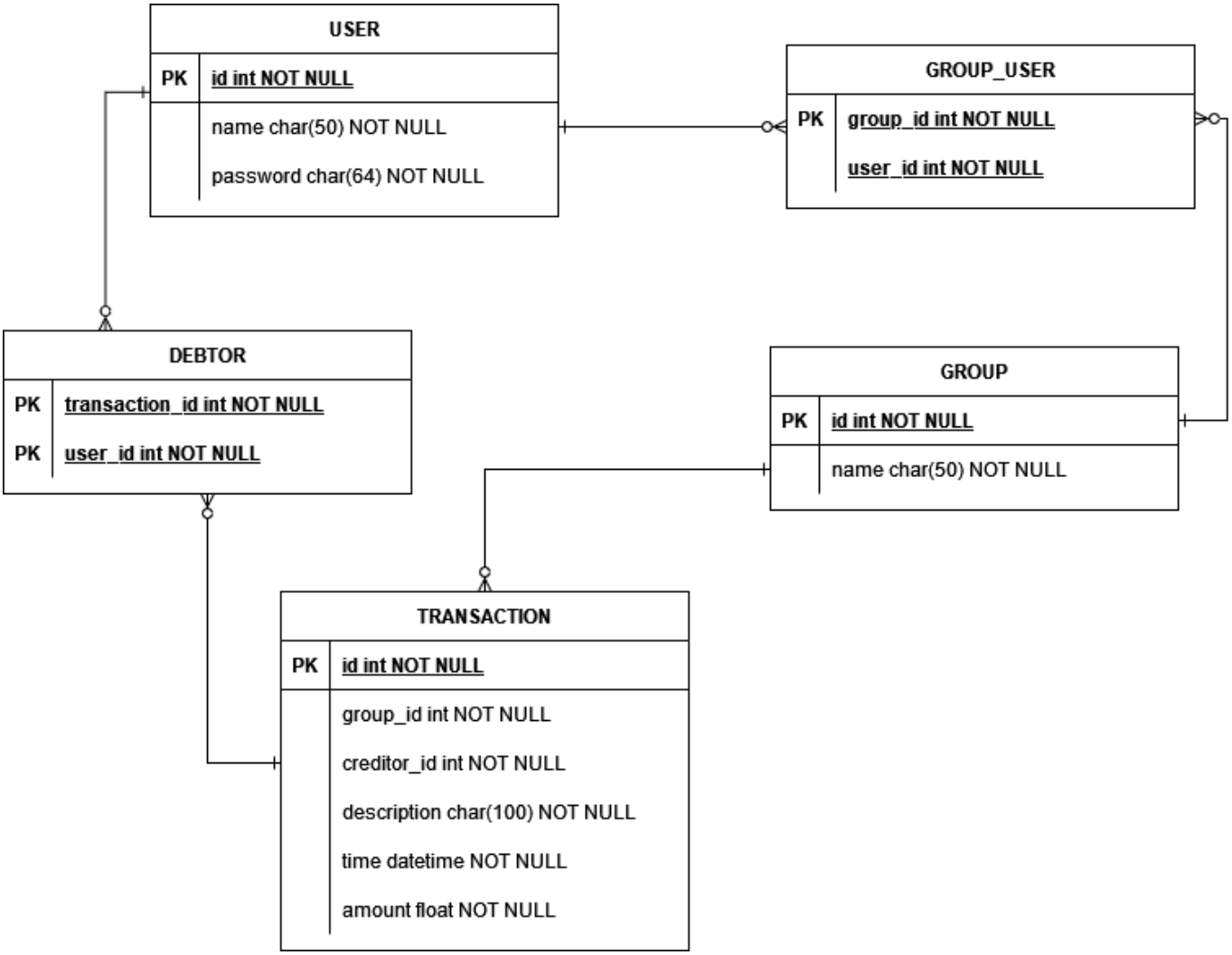

Abbildung 1. Das ER-Modell der Datenbank

2) TRANSACTION: Hier befinden sich die einzelnen Ausgaben. Eine Ausgabe enthält die ID der Gruppe, die ID des Gläubigers, eine Beschreibung, das Datum der Ausgabe, sowie einen Betrag.

3) DEBTOR: Um nun einem Schuldner eine Ausgabe eines Gläubigers zuzuweisen, gitb es die Debtor-Tabelle. Hier wird eine Ausgabe mit einem User verknüpft. So können zu einer Ausgabe mehere Schuldner zugewiesen werden.

4) GROUP: In der Group-Tabelle werden eine eindeutige ID, sowie der Name der Gruppe gespeichert.

5) GROUP-USER: Auch muss die Gruppenzugehörigkeit gespeichert werden. Hier finden sich die ID des Users mit der jeweiligen ID der Gruppe, in der er sich befindet. So kann zugeordner werden, welche User in welchen Gruppen ist.

IV. Randbedingungen

Um mit der Entwicklung starten zu können, müssen vorher Bedingungen besprochen werden, welche von allen Mitgliedern eingehalten werden sollen. So wird sichergestellt, dass das System auf jedem Rechner der Entwickler läuft und auch der Code ein einheitliches Erscheinungsbild hat.

A. Technische Randbedingungen

Als Entwicklungsumgebung wird VS Code verwendet. Zudem gibt es Richtlinien für das Testen der Software:

- Für jede Frontend-Komponente wird ein Unit-Test mit Vitest und dem React-Testing-Framework geschrieben.
- Für Backend-Endpunkte werden auch Unit-Tests mit Vitest und möglicherweise auch mit Postman-Anwendung erstellt.
- Um die einzelnen Aktionen, die mit der Datenbank in Verbindung stehen, zu testen, werden die Tabellen (vgl. Abbildung 1) mit Testdaten gefüllt.
- Wichtig ist auch, dass Komponenten untereinander getestet werden. Das heißt, dass der Autor sein Werk nicht manuell selbst testet, da hier die Wahrscheinlichkeit, Fehler zu finden, geringer ist.

Außerdem wird MUI als Komponenten-Bibliothek für das Frontend verwendet, um kostbare Zeit beim Design zu sparen. Falls Icons verwendet werden, werden die Material-Icons benutzt. Zu finden sind diese hier: https://fonts.google.com/icons?selected=Material+Icons.

B. Organisatorische Randbedingungen

Um einen reibungslosen Ablauf zu gewährleisten, wird jede Woche Montag im Weekly-Meeting nach der Feedback-Session besprochen, was es an Aufgaben gibt und was in der kommenden Woche getan werden muss. Anwesend sollten alle Mitglieder sein, sofern sie nicht durch Krankheit oder sonstige Notfälle verhindert sind. Da es noch Personen im Team gibt, die weniger Erfahrung mit dem Framework React haben, werden Mentee-Mentor-Beziehungen gebildet, sodass es einen konkreten Ansprechpartner bei Problemen oder Fragen gibt. Der Mentor ist dazu angehalten, sich bei Problemen um seinen Mentee zu kümmern. Wo es möglich ist, sollte Peer-Programming betrieben werden, da es zum Einen beim Lernen des Frameworks hilft und zum Anderen auch Austausch über mögliche alternative Lösungen stattfindet.

C. Codier-Richtlinien

Damit der Programm-Code ein einheitliches Bild abgibt, müssen Codier-Richtlinien eingehalten werden. Diese wurden von allen Gruppenmitgliedern erstellt. Jedes Mitglied ist dazu angehalten, zu prüfen, ob folgende Richtlinien eingehalten wurden und gegebenenfalls Änderungen vorzunehmen, um die Einhaltung sicher zu stellen.

- Format zur Benennung neuer Branch: „ ⟨Issue-Nummer⟩-⟨Name-des-Issues⟩”
- Die Format-Einstellungen für den Programm-Code sind in der Datei „prettierrc.json” festgehalten und werden automatisch von der Visual Studio Code-Erweiterung „Prettier” genutzt und bei jedem Speichern umgesetzt.
- Der Programm-Code wird auf Englisch geschrieben.
- Die Sprache der Benutzeroberfläche ist allerdings auf Deutsch.
- Es sollten möglichst spezifische Typen verwendet werden, das heißt kein „any”.
- Keine Verwendung von „var”. Es sollten lieber „const” oder „let” benutzt werden.
- Funktionen-, sowie Variablennamen sollten ohne Kommentar in ihrer jeweiligen Funktionalität nachvollziehbar sein.
- Nur, wenn Funktionen- und Variablennamen nicht eindeutig auf die Funktionalität hindeuten, sollten Kommentare verwendet werden.
- Der Umfang einer Methode wird auf ein Minimum beschränkt, das heißt, so viel wie nötig und so wenig wie möglich.
- Methoden sollten nicht mehr machen, als ihr Name vermuten lässt.
- Methoden, Variablen und Ordnernamen werden in „camelCase” geschrieben.
- Für Typen, Klassen und React-Komponenten gilt hingegen „PascalCase”.

V. Ausblick

wird noch eingefügt...

Architekturziel	Zugeordneter Architekturansatz
Im Backend sind keine anderen Programmierfähigkeiten als im Frontend notwendig.	NodeJS
Während der Entwicklungszeit gibt es eine statische Typisierung.	TypeScript
REST-API weist geringen Implementierungsaufwand auf.	Express.js
Die Daten werden in einer relationalen Datenbank gespeichert und bereitgestellt.	MySQL
Der Zugriff auf die MySQL-Datenbank ist schnell, sicher und ohne SQL-Befehle möglich.	Sequelize
Die Webseite hat einen modularen Aufbau.	React
Nutzer werden authentifiziert.	JSON Webtoken als Cookie im Frontend gespeichert
Effizientes Kompilieren und Hot-Reloading bei der Entwicklung des Frontends.	Vite
Frontend und Backend werden automatisiert getestet.	Vitest und Jest
Stylesheets sind übersichlicher definiert.	SCSS
Das Aussehen der Webseite ist einheitlich.	Material UI
Die Entwicklung der Webseite ist zeitsparender.	Material UI
Einheitliche Versionierung der Datenbank und Einspielen der Testdateien.	Docker
Das Codeformat ist einheitlich.	Prettier

Tabelle I
DEM ARCHITEKTURZIEL IST JEWEILS EINE TECHNOLOGIE ZUGEORDNET