IndexLive

MERN-Stack - Finanzdashboard Applikation

Abstract

IndexLive ist ein Finanz-Dashboard Webseite, das durch verschiedene Grafiken und Tabellen einen detaillierten Überblick über Unternehmensfinanzen liefert.

Mohammad Taiba

mohammad.taiba@fh-erfurt.de

Contents

1. Projektverlauf

1.1 Projektziel

Das Ziel des Projekts war es, eine umfassende Finanzverwaltungsanwendung zu entwickeln. Diese Anwendung sollte Benutzern die Möglichkeit bieten, ihre finanziellen Transaktionen, Produkte und wichtige Leistungsindikatoren (KPIs) zu verwalten. Die Anwendung wurde mit modernen Webtechnologien entwickelt und umfasst sowohl ein Frontend als auch ein Backend.

1.2 Meilensteine und Zeitplan

Planung und Anforderungserfassung (Woche 1):

- Erstellung des Projektplans und der Anforderungsanalyse.
- Festlegung der Technologien und Tools, die im Projekt verwendet werden sollen.
- Definition der Projektziele und -anforderungen.

Design (Woche 2):

- Entwurf der Anwendungsarchitektur.
- Design des Datenbankschemas und der API-Schnittstellen.

Entwicklung des Backends (Woche 3-6):

- Einrichtung des Express-Servers und der MongoDB-Datenbank.
- Implementierung der RESTful API für Produkte und KPIs.
- Implementierung der GraphQL-API für Transaktionen.
- Integration von Apollo Server für GraphQL.
- Erstellung der Datenbankmodelle mit Mongoose.

Entwicklung des Frontends (Woche 3-6):

- Einrichtung der React-Anwendung und des Redux-State-Managements.
- Implementierung der Benutzeroberflächenkomponenten.
- Integration der RESTful API und GraphQL-API in das Frontend.

Dokumentation (Woche 6)

2. Architektur

2.1 Gesamtübersicht

Die Architektur der Finanzverwaltungsanwendung besteht aus einem modularen Backend-Server, der mit einer MongoDB-Datenbank verbunden ist, und einem modernen React-Frontend. Die Hauptkomponenten umfassen:

- Express-Server: Der Hauptserver, der sowohl RESTful API-Endpunkte als auch eine GraphQL-API bereitstellt.
- Apollo Server: Handhabt GraphQL-Anfragen und integriert sich nahtlos mit dem Express-Server.
- React-Frontend: Eine Single-Page-Anwendung, die Benutzeroberfläche bereitstellt.
- Redux: State-Management-Bibliothek zur Verwaltung des globalen Zustands der Anwendung.

- Mongoose: Objekt-Datenmodellierungsbibliothek für MongoDB und Node.js, die die Datenbankinteraktion vereinfacht.

2.2 Backend-Architektur

Express-Server

Der Express-Server dient als zentraler Knotenpunkt für die Verarbeitung von HTTP-Anfragen. Er stellt sowohl RESTful API-Endpunkte als auch eine GraphQL-API bereit.

- RESTful API: Wird für CRUD-Operationen auf Produkten und KPIs verwendet.
- GraphQL API: Wird für flexible und effiziente Datenabfragen und -manipulationen von Transaktionen verwendet.
- Middleware: Helm, Cors, Body-Parser und Morgan werden zur Sicherheits- und Logging-Zwecken eingesetzt.

Apollo Server

Der Apollo Server ist in den Express-Server integriert und handhabt alle GraphQL-Anfragen. Er ermöglicht die Nutzung der GraphQL-Spezifikation zur flexiblen Abfrage und Manipulation von Daten.

Mongoose

Mongoose wird zur Modellierung der Anwendungsschemata und zur Interaktion mit der MongoDB-Datenbank verwendet. Es ermöglicht die Definition von Schemas und die Validierung von Daten vor dem Speichern.

2.3 Frontend-Architektur

React

React bildet das Herzstück des Frontends. Es ermöglicht die Erstellung einer dynamischen und reaktionsschnellen Benutzeroberfläche.

Redux

Redux wird zur Verwaltung des Anwendungszustands verwendet. Es stellt eine zentrale Datenquelle bereit, die von allen Komponenten abgerufen werden kann.

Apollo Client

Apollo Client wird für die Interaktion mit der GraphQL-API verwendet. Es bietet eine einfache Möglichkeit, Daten von der GraphQL-API zu holen und im Zustand der Anwendung zu speichern.

3. Anwenderdokumentation

3.1 Installation und Einrichtung

Voraussetzungen:

- Node.js und npm
- MongoDB

Schritte zur Installation

Um die Website "IndexLive" zu installieren und einzurichten, folgen Sie bitte dieser Schritt-für-Schritt-Anleitung:

1. Repository klonen:

```
git clone https://git.ai.fh-erfurt.de/mo7930ta/indexlive
cd indexlive
```

2. Backend-Installation:

```
cd server
npm init -y
npm i express body-parser cors dotenv helmet morgan mongoose mongoose-currency
npm i -D nodemon
npm install swagger-jsdoc swagger-ui-express
npm install apollo-server-express graphql
```

3. Frontend-Installation:

```
cd client
npm install
npm i react-redux @reduxjs/toolkit react-router-dom @mui/material @emotion/react @emotion/styled @mui/icons-material @mui/x-data-grid
npm i -D
npm i -D @types/react-dom
npm i -D eslint eslint-config-react-app
npm i -D @types/node
npm i recharts
npm install @apollo/client graphql
```

4. Datenbank starten:
   Verbinden Sie die App mit einer MongoDB-Server.
   - Installieren Sie die Plugin MongoDB
   - Installieren Sie auf dem MongoDB – Compass
   - Besuchen Sie die Webseite https://cloud.mongodb.com/ um ein Cluster für Ihren Server zu erstellen und dann mit der App anzuwenden.
     - Speichern Sie die Mongo-URL, die Sie erstellt haben und dann setzen Sie das in der .env (6.Schritt) ein!
       - Da müssen Sie auch Ihre Benutzername und das Passwort Ihrem Cluster-Server verwenden!

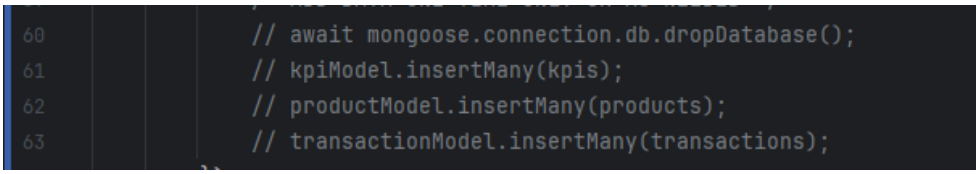
5. .env-Datei in Client erstellen:

```
VITE_BASE_URL=http://localhost:10081
```

6. .env-Datei i n Server erstellen:

```
MONGO_URL= <Ihre gespeicherte Mongo-URL>
PORT1=10081
PORT2=10082
PORT3=10083
DATABASE_PORT=10085
PROJECT_ID=1817
```

7. Server starten:
   Falls bis jetzt alles geklappt hat, dann before Sie den Server zum ersten Mal starten, beheben Sie die Kommentare in server/index.js:



   Damit die Daten in der Datenbank – Server gepusht werden, danach kommentieren Sie wieder die Zeilen, um unerwartet Fehlern zu vermeiden.

   Um den Server zu starten, führen Sie bitte diesen Befehl:

```
cd server
npm run dev
```

8. Frontend starten:

```
cd client
npm run dev
```

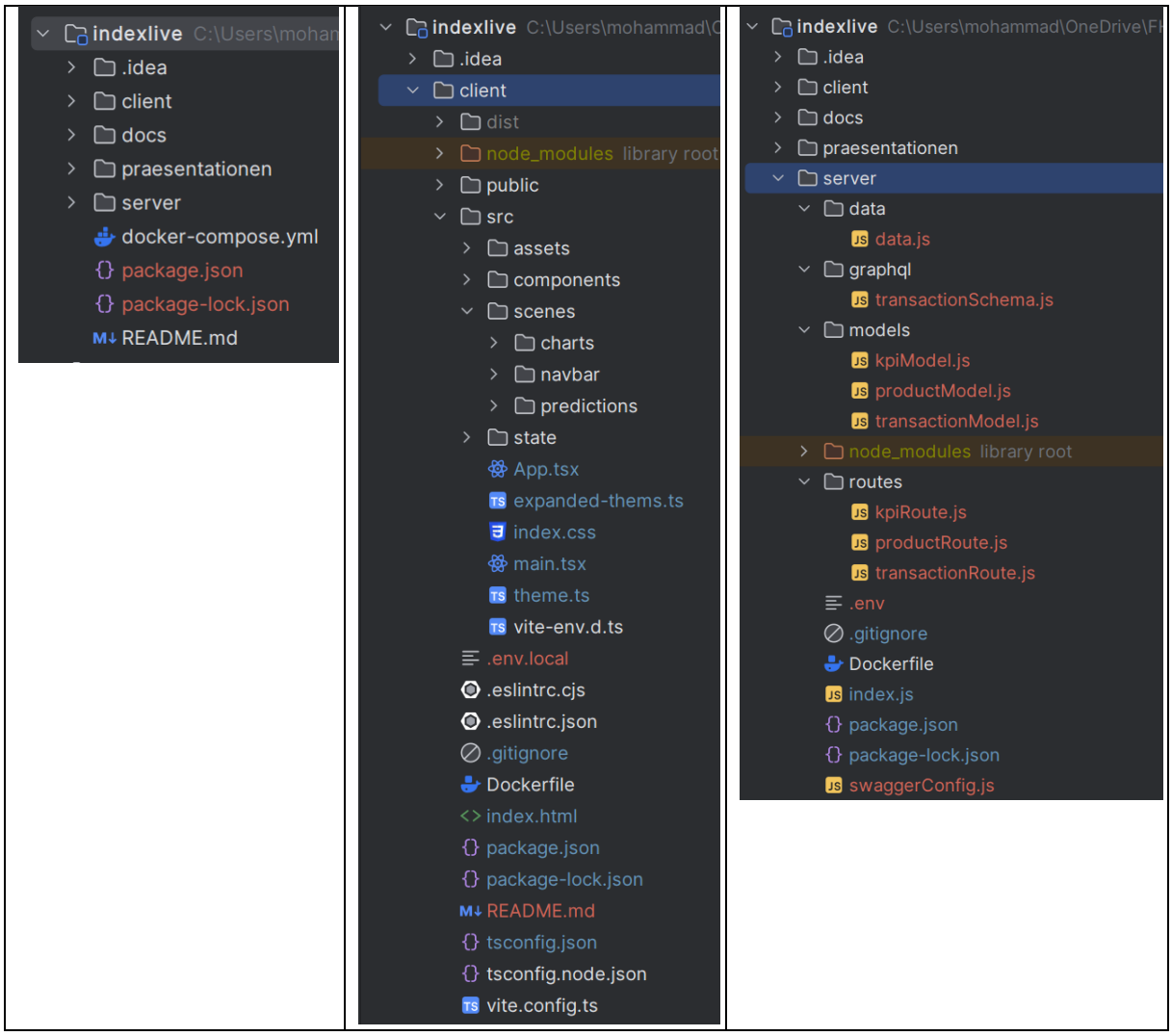
3.2 Nutzung der Anwendung

Dashboard: Zeigt eine Übersicht über die KPIs und kürzliche Aktivitäten.
Produkte: Zeigt eine Übersicht von allen Produkten als Tabelle.
Transaktionen: Zeigt eine Übersicht von letzten 50 Transaktionen als Tabelle.

4. Entwicklerdokumentation

4.1 Projektstruktur

Geamt	Für Client	Für Server



4.2 Backend-Services

REST-API-Endpunkte

GET, Holt alle Produkte, Transaktionen oder KPIs:

- http://localhost:10081/product/products
- http://localhost:10081/transaction/transactions
- http://localhost:10081/kpi/kpis

POST, Erstellt neue Produkte oder Transaktionen:

- http://localhost:10081/product/products
- http://localhost:10081/transaction/transactions

PUT, Aktualisiert die Produkte oder Transaktionen:

- http://localhost:10081/product/products
- http://localhost:10081/transaction/transactions

DELETE, Löscht die Produkte oder Transaktionen:

- http://localhost:10081/product/products
- http://localhost:10081/transaction/transactions

GraphQL-API

Verwenden Sie diese URL http://localhost:10081/graphql, um die Queries und die Mutationen durchführen zu können.

Queries:

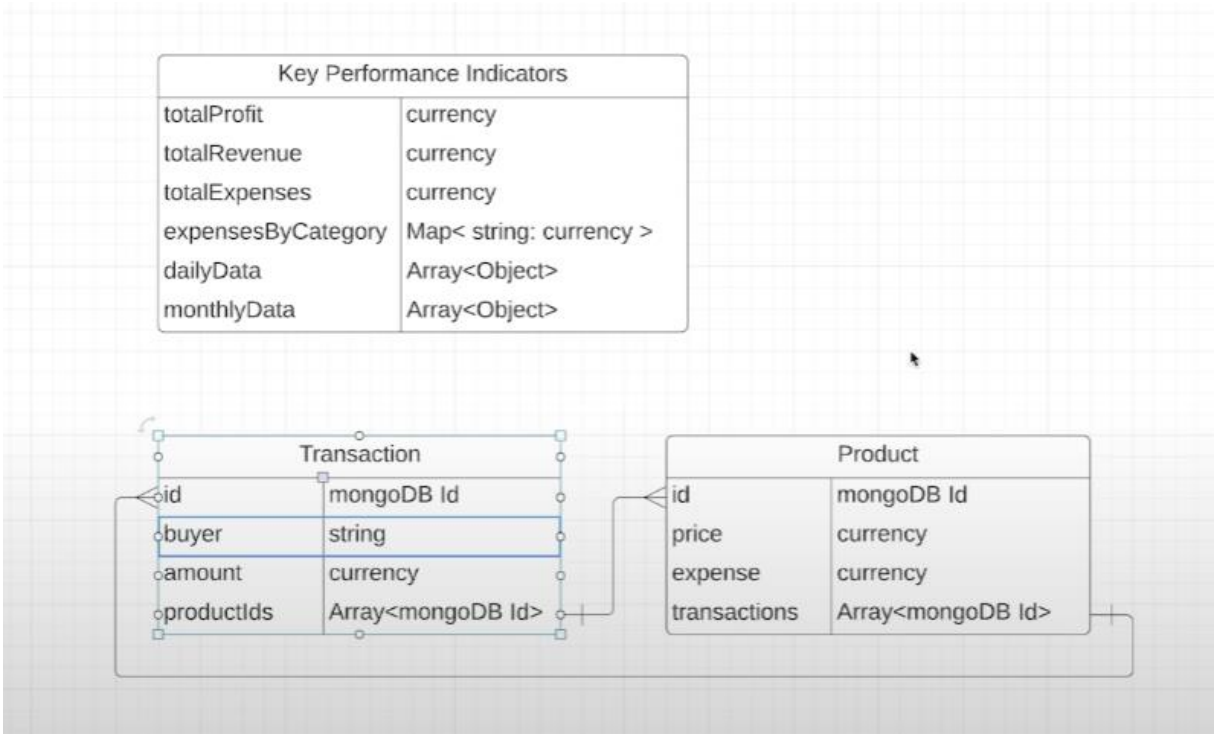
```
query {
transactions {
       id
       buyer
       amount
       productIds
  }
}
```

Mutationen

```
mutation {
createTransaction(buyer: "Leonel Messi", amount: 150.0, productIds:
["60d9f7f0f0f0f0f0f0f0f0f0", "60d9f7f0f0f0f0f0f0f0f0f1"]) {
       id
       buyer
       amount
       productIds
  }
}
```

4.3 Datenbankschema

ER-Diagramm:



Beschreibung der Entitäten:

Transaction:

Schema:

```
const TransactionSchema = new Schema(
    {
        buyer: {
            type: String,
```

```
            required: true,
        },
        amount: {
            type: mongoose.Types.Currency,
            currency: "USD",
            get: (v) => v / 100,
        },
        productIds: [
            {
                type: mongoose.Schema.Types.ObjectId,
                ref: "Product",
            },
        ],
    },
    { timestamps: true, toJSON: { getters: true } }
);
```

Erklärung:

buyer: Ein String, der den Namen des Käufers repräsentiert. Dieses Feld ist erforderlich.

amount: Ein Währungsfeld, das den Betrag der Transaktion in USD darstellt. Der Wert wird beim Abrufen durch 100 geteilt, um die korrekte Darstellung der Währung zu gewährleisten.

productIds: Ein Array von ObjectId-Referenzen, die auf die `Product`-Entitäten verweisen. Dies ermöglicht die Zuordnung mehrerer Produkte zu einer Transaktion.

timestamps: Automatisch hinzugefügte Felder, die das Erstellungs- und Aktualisierungsdatum der Transaktion speichern.

Product:

Schema:

```
const ProductSchema = new Schema(
    {
        price: {
            type: mongoose.Types.Currency,
            currency: "USD",
            get: (v) => v / 100,
        },
        expense: {
            type: mongoose.Types.Currency,
            currency: "USD",
            get: (v) => v / 100,
        },
        transactions: [
            {
                type: mongoose.Schema.Types.ObjectId,
                ref: "Transaction", // ref: to make a relationship with other
schemas
            },
        ],
    },
    { timestamps: true, toJSON: { getters: true } }
);
```

Erklärung:

name: Ein String, der den Namen des Produkts repräsentiert. Dieses Feld ist erforderlich.

price: Ein Währungsfeld, das den Preis des Produkts in USD darstellt. Der Wert wird beim Abrufen durch 100 geteilt, um die korrekte Darstellung der Währung zu gewährleisten.
description: Ein optionaler String, der eine Beschreibung des Produkts enthält.
timestamps: Automatisch hinzugefügte Felder, die das Erstellungs- und Aktualisierungsdatum des Produkts speichern.

KPI:

Schema:

```
const KPISchema = new Schema(
    {
        totalProfit: {
            type: mongoose.Types.Currency,
            currency: "USD",
            get: (v) => v / 100
        },
        totalRevenue: {
            type: mongoose.Types.Currency,
            currency: "USD",
            get: (v) => v / 100
        },
        totalExpenses: {
            type: mongoose.Types.Currency,
            currency: "USD",
            get: (v) => v / 100
        },
        expensesByCategory: {
            type: Map,
            of : {
                type: mongoose.Types.Currency,
                currency: "USD",
                get: (v) => v / 100
            }
        },
        monthlyData: [{
            month: String,
            revenue: {
                type: mongoose.Types.Currency,
                currency: "USD",
                get: (v) => v / 100
            },
            expenses : {
                type: mongoose.Types.Currency,
                currency: "USD",
                get: (v) => v / 100
            },
        }], // [] --> for an array
        dailyData: [{
            date: String,
            revenue: {
                type: mongoose.Types.Currency,
                currency: "USD",
                get: (v) => v / 100
            },
            expenses : {
                type: mongoose.Types.Currency,
                currency: "USD",
                get: (v) => v / 100
            },
        }],
```

```
    },
    {  timestamps: true, toJSON: { getters: true }} // gives us, when is a
particular one is created/updated
);
```

Erklärung:

totalProfit: Ein Währungsfeld, das den gesamten Gewinn in USD darstellt. Der Wert wird beim Abrufen durch 100 geteilt, um die korrekte Darstellung der Währung zu gewährleisten.

totalRevenue: Ein Währungsfeld, das den gesamten Umsatz in USD darstellt. Der Wert wird beim Abrufen durch 100 geteilt.

totalExpenses: Ein Währungsfeld, das die gesamten Ausgaben in USD darstellt. Der Wert wird beim Abrufen durch 100 geteilt.

expensesByCategory: Ein Map-Feld, das die Ausgaben nach Kategorien speichert. Jede Kategorie enthält einen Währungswert in USD, der beim Abrufen durch 100 geteilt wird.

monthlyData: Ein Array von Objekten, das die monatlichen Daten enthält. Jedes Objekt enthält:

month: Ein String, der den Monat darstellt.

revenue: Ein Währungsfeld, das den Umsatz des Monats in USD darstellt. Der Wert wird beim Abrufen durch 100 geteilt.

expenses: Ein Währungsfeld, das die Ausgaben des Monats in USD darstellt. Der Wert wird beim Abrufen durch 100 geteilt.

dailyData: Ein Array von Objekten, das die täglichen Daten enthält. Jedes Objekt enthält:

date: Ein String, der das Datum darstellt.

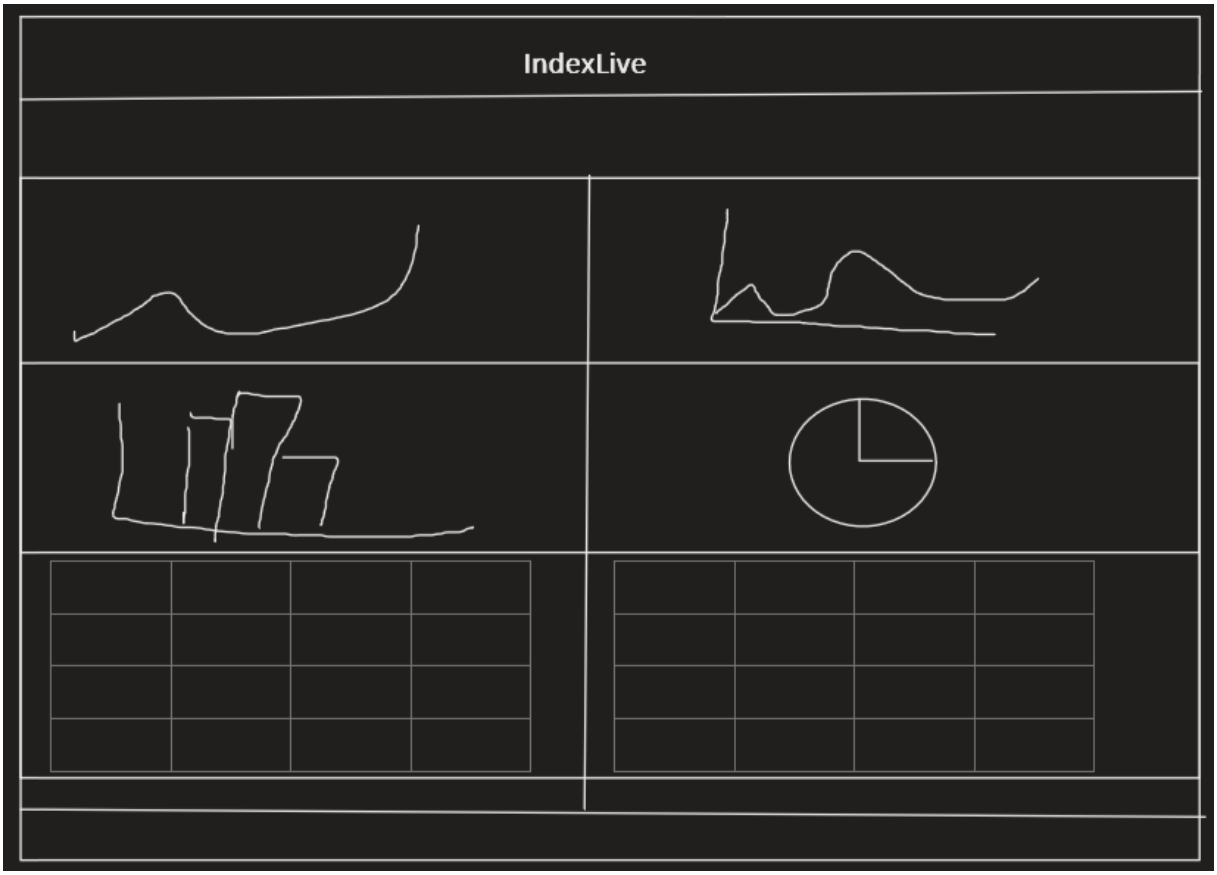
revenue: Ein Währungsfeld, das den Umsatz des Tages in USD darstellt. Der Wert wird beim Abrufen durch 100 geteilt.

expenses: Ein Währungsfeld, das die Ausgaben des Tages in USD darstellt. Der Wert wird beim Abrufen durch 100 geteilt.

timestamps: Automatisch hinzugefügte Felder, die das Erstellungs- und Aktualisierungsdatum des KPI-Dokuments speichern.

5. Architekturzeichnung

5.1 Projektentwurf:



5.2 Webseite - Architekturzeichnung