Dokumentation

Erstellung einer WebApp mit React
im Modul ‚Mobile Anwendungen‘
Sommersemster 2019
FH Kiel

Frieder Griem (931044)
Jonas Daniel (931000)

Inhalt

1. Planung 1
1.1 Zielgruppe 1
1.2 Funktionalität 1
1.3 Design 2
2. Umsetzung 3
2.1 Vorteile von React für die Anwendung 3
2.2. Modularisierung der Anwendung 3
2.2 Funktionalitäten in React 4
2.3 Schwierigkeiten in der Umsetzung 8
3 Projekt auf GitHub 8

1. Planung

Zu Beginn der Anwendungsentwicklung wurde ein kompakter Planungsprozess durchgeführt. Anforderung war es, eine WebApp mit React zu entwickeln, in der Daten in Listenform gespeichert, bearbeitet, gelöscht und angezeigt werden können.

Um die Anwendung zu konkretisieren, wurde eine fiktive Zielgruppe erstellt, deren Anforderungen von der Anwendung erfüllt werden sollen.

1.1 Zielgruppe

Die Zielgruppe für unsere Anwendung besteht aus Mafiabossen, Kredithaien und anderen Geschäftsleute, die semilegale Dienstleistungen anbieten (z.B. Anbieter von Schutzgeldversicherungen, Kredite abseits des Fiskus etc.).

Die Anwendung soll ein in der Branche gängiges Problem lösen: Bei steigender Schuldnerzahl fällt der Überblick schwer. Die Anwendung löst dieses Problem und macht das Geldeintreiben sowie die Vergabe von Krediten unkompliziert möglich. Entstehen soll also eine Todo-Liste, welche auf die Anforderungen von Mafiabossen abgestimmt ist.

Unsere Zielgruppe stellt hohe Anforderungen an die Usability und wünscht ein aufgeräumtes Design.

1.2 Funktionalität

Aus den Ansprüchen der Zielgruppe ergeben sich folgende Kernfunktionalitäten: Anlegen und Löschen von Einträgen in einer Liste. Die Liste muss den Namen eines Schuldners und einen Betrag enthalten. Das Todo kann als erledigt abgehakt werden.

Um die Liste für unsere Zielgruppe attraktiver zu gestalten, werden folgende Zusatzfunktionen angeboten:

- Sortieren der Liste nach Betrag (aufsteigend und absteigend).
- Sortieren von erledigten Einträgen an das Ende der Liste.
- Eintrag der Gefahr, die vom Schuldner ausgeht und Visualisierung durch Icons.
- Suchfunktion, um bei vielen Schuldnern den Überblick zu behalten.
- Änderung bereits angelegter Einträge.

1.3 Design

Das Design soll schlicht gehalten werden und setzt somit auf Grau und Schwarz. Betont werden einzelne Elemente durch dem Milieu angepasste farbliche Akzente.

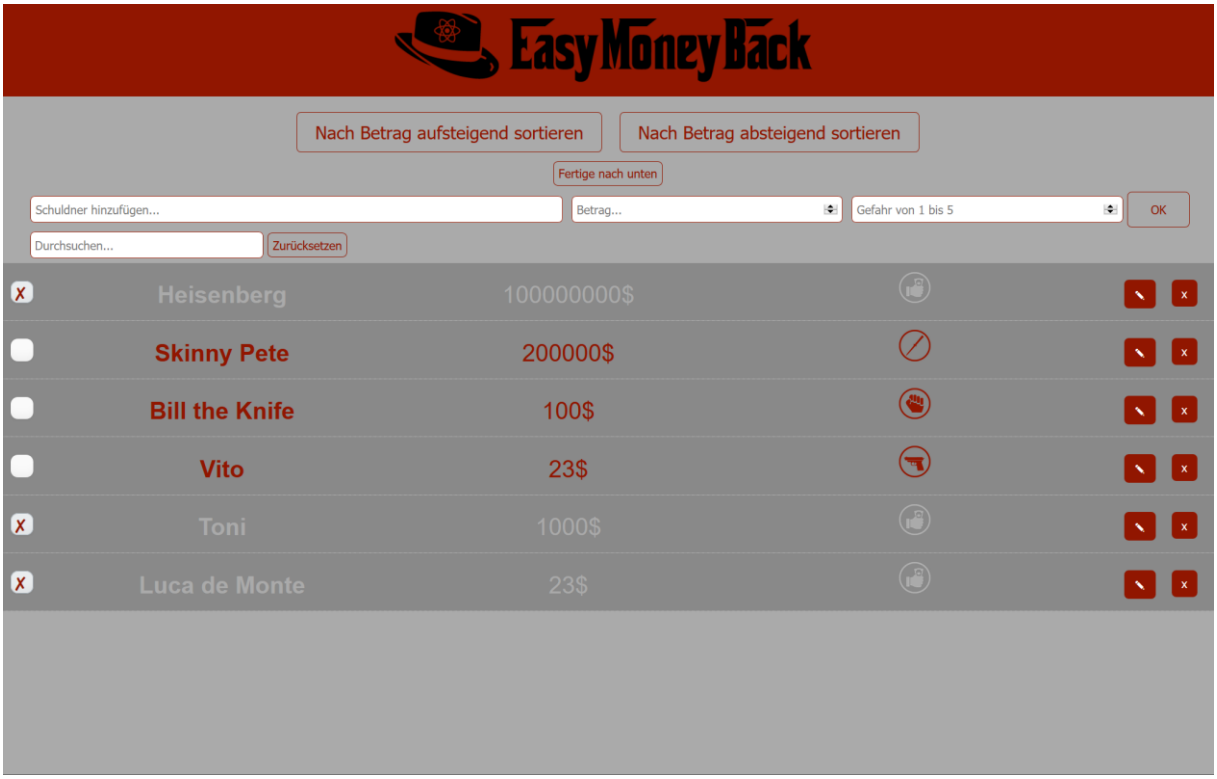

Abbildung 1: Design der Anwendung

Abbildung 2: Logo von EasyMoneyBack

2. Umsetzung

Die Umsetzung erfolgte mit der JavaScript-Library React. Im Folgenden werden einige Vorteile und Besonderheiten dieser Methode erläutert.

2.1 Vorteile von React für die Anwendung

Als JavaScript Library bietet React im Projekt den Vorteil, dass Seiteninhalte innerhalb des User Interfaces unkompliziert bedient werden können und ohne ein Refresh der Seite Änderungen dynamisch gerendert werden. Die Verwendung einer Datenbank ist durch Nutzung des Local Storage nicht nötig.

2.2. Modularisierung der Anwendung

Eine React-App ist in Komponenten aufgebaut, also in logisch trennbare Einheiten unterteilt. Eine Komponente kann wie ein Objekt in JavaScript Eigenschaften übernehmen (abzurufen mit *this.props*). Unsere App besteht aus folgenden Komponenten:

Die übergeordnete *App-Komponente* enthält alle weiteren Komponenten.
Die Komponente *Header* enthält das Logo.
Die Komponente *AddTodo* enthält eine Eingabemöglichkeit, um die Liste der einzutreibenden Schulden zu erweitern.
Die Komponente *Todos* enthält alle einzelnen Einträge der Liste.
Die Komponente *TodoItem* entspricht einem einzelnen Schuldeneintrag und hat die Eigenschaften id, title, amount, urgency und completed.

```
export class TodoItem extends Component {
  //Per Default ist der Edit-Mode deaktiviert
  state = {
    isInEditMode: false
  };

  //Styling ändert sich je nachdem, ob ein Listenounkt abgeschlossen ist
  getStyle = () => {
    if (this.props.todo.completed) {
      return {
        background: "#898989",
        color: "#aaa",
        padding: "10px",
        borderBottom: "1px #ccc dotted"
      };
    } else {
      return {
        background: "#898989",
        padding: "10px",
        borderBottom: "1px #ccc dotted",
        color: "#911600"
      };
    }
  };
```

Abbildung 3: Ausschnitt einer Komponente

2.2 Funktionalitäten in React

Im Folgenden werden die wichtigsten Funktionalitäten der Anwendung in React kurz vorgestellt, was auch eine kurze Einführung in die Funktionsweise von React bietet.

Aufbau der Einträge (Abb. 4): Die Einträge (debtors) sind Objekte mit den Eigenschaften id, title, amount, urgency und dem Wahrheitswert completed. Die ID wird mithilfe des Packages uuid vergeben.

```
state = {
  //Objekt Schuldner mit Eigenschaften und dazugehörigen Werten.
  //ID wird automatisch durch Universally Unique Identifier (uuid) vergeben
  debtors: [
    {
      id: uuid.v4(),
      title: "Toni",
      amount: "1000",
      urgency: "3",
      completed: false
    },
    {
      id: uuid.v4(),
      title: "Jack",
      amount: "100",
      urgency: "1",
      completed: false
    },
    {
      id: uuid.v4(),
      title: "Bill the Knife",
      amount: "100",
      urgency: "1",
      completed: false
    }
  ]
};
```

Abbildung 4: Aufbau von Einträgen

Hinzufügen von Einträgen (Abb. 5): Über einen Text-Input werden die Eigenschaftswerte eingetragen. Bei einem Submit wird die Methode onChangeTitle() aktiv. Diese überträgt die Eingaben des Feldes mit this.setState() an ein neues Objekt, das bei einem Submit ebenfalls aufgebaut wird. setState ist eine der elementaren React-Funktionen, um den Status eines Objekts zu ändern.

```
onSubmit = e => {
  e.preventDefault();
  //Titel und Dringlichkeit werden übergeben
  this.props.addTodo(this.state.title, this.state.amount, this.state.urgency);
  this.setState({ title: "" });
  this.setState({ amount: "" });
  this.setState({ urgency: "" });
};
//Bei Änderungen soll State geändert werden
onChangeTitle = e => this.setState({ title: e.target.value });
onChangeAmount = e => this.setState({ amount: e.target.value });
onChangeUrgency = e => this.setState({ urgency: e.target.value });

render() {
  return (
    <form onSubmit={this.onSubmit} style={{ display: "flex" }}>
      {/*Input für die Schuldner */}
      <input
        className="searchbar"
        minLength="0"
        maxLength="40"
        type="text"
        name="title"
        style={{ flex: "10", padding: "5px", border: "1px solid #911600" }}
        placeholder="Schuldner hinzufügen..."
        value={this.state.title}
        onChange={this.onChangeTitle}
      />
```

Abbildung 5: Hinzufügen von Einträgen unter der Verwendung von setState()

Sortierungsfunktion (Abb.6): Über einen Button wird die untenstehende Funktion zur Sortierung ausgelöst. Der Konstanten postList wird zunächst das komplette Array, indem sich die Daten der Liste befinden zugewiesen. Über die JS-Funktion .sort() wird die Reihenfolge des Arrays gemäß dem amount, also der Eigenschaft Schuldenmenge neu sortiert. Das Array mit der neuen Sortierung wird mit setState dem Objekt debtors zugewiesen.

```
//Liste nach Betrag absteigend sortieren
toggleListAmountDown = () => {
  const postList = [...this.state.debtors];
  postList.sort(function(a, b) {
    return parseInt(a.amount) < parseInt(b.amount) ? -1 : 1;
  });
  //postList.sort(numSort);
  this.setState({
    debtors: postList
  });
};
```

Abbildung 6: Sortierung der Liste unter Verwendung von .sort()

Speichern (Abb. 6): Für die Speicherfunktion werden die React-Lifecycle-Methoden componentDidUpdate() und componentWillMount() verwendet. Jedes Mal, wenn der State einer Komponente neu gesetzt wurde, wird (mittels componentDidUpdate()) eine Übetragung der Inhalte des Objekts debtors in den Local Storage eingeleitet. Wenn die Seite neu geladen wird, werden die Werte aus dem Speicher mit componentWillMount() für den initial Render abgerufen.

```
//Local Storage Speicherung ----------------------------------------------------------------------------------------
//Speichern der Werte der Todos in local Storage. Immer nachdem eine Komponente geupdated wurde (componentDidUpdate)
componentDidUpdate() {
  localStorage.setItem("debtors", JSON.stringify(this.state.debtors));
}

//auslesen aus Local storage, wenn es exisitert, wird es als state gesetzt
componentWillMount() {
  localStorage.getItem("debtors") &&
    this.setState({
      debtors: JSON.parse(localStorage.getItem("debtors"))
    });
}
```

Abbildung 7: Speicherung im Local Storage mit lifecycle methods

Suchfunktion (Abb. 8): Mit dem constructor() wird ein State initialisiert, der in der Eigenschaft search den Wert “ “ hat. In einem Eingabefeld wird die Suchanfrage eingegeben. Bei jeder Änderung im Eingabefeld (onChange) wird die Funktion updateSearch ausgeführt. Hierbei wird der Eigenschaft Search der Wert der Eingabe zugewiesen (mittels substr() von der nullten Stelle bis maximal 20 Zeichen). Innerhalb des render-Bereichs werden nur die Einträge angezeigt, welche den eingegebenen string enthalten.

Die Funktion resetSearch weist dem Suchfeld wieder den leeren Ausgangsstatus zu.

```
constructor() {
  super();
  this.state = {
    search: ""
  };
  this.searchEmpty = this.state;
}

//Bei jeder Eingabe wird die Suchanfrage geupdated
updateSearch(event) {
  this.setState({ search: event.target.value.substr(0, 20) });
}

//Suchleiste zurücksetzen auf Ausgangsstatus
resetSearch = () => {
  this.setState(this.searchEmpty);
};

render() {
  //Ausgabe nur gefilterte Elemente, die in der Such auch eingegeben wurden
  let filteredTitles = this.props.todos.filter(todo => {
    return (
      todo.title.toLowerCase().indexOf(this.state.search.toLowerCase()) !== -1
    );
  });
```

Abbildung 8: Suche und render()

2.3 Schwierigkeiten in der Umsetzung

Die .jsx-Syntax erwies sich zunächst als verhältnismäßig schwer zugänglich, insbesondere, da in den vorangegangen Modulen kein Javascript gelehrt wurde.

Außerdem gab es Probleme bei der Browserkompatibilität der Suchfunktion, da die JS-Methode sort() in Firefox und Chrome leicht unterschiedliche Ergebnisse liefert: Die Beiträge werden grundsätzlich korrekt nach Schuldenbetrag und Status (erledigt/nicht erledigt) sortiert.

In Chrome gibt es allerdings den Unterschied, dass bei einer Sortierung der erledigten Beiträge nach der Sortierung nach Schuldenbetrag die erledigten Beiträge zwar unten angezeigt werden, der Schuldenbetrag aber mit umgekehrter Reihenfolge, angezeigt wird.

3 Projekt auf GitHub

Das Projekt ist unter folgendem Link auf Github zu finden:
https://github.com/FriederG/notizapp_MOBAN/