# Dokumentation
# Modul 152
## Steepr

Valentino Rusconi – Elia Reutlinger

# Inhaltsverzeichnis

# Entwicklungsumgebung

| | |
|---|---|
| Betriebssystem: | Mac OS Mojave |
| Hardware: | Hardware unabhängig |
| Entwicklung IDE: | Visual Studio Code |
| Validierungstools: | ESLint |
| Coding Convention: | JavaScript & React Standards von Airbnb |
| | |
| Programmiersprachen: | Javascript & Reactjs |

# Programmierung

In folgenden Abschnitt sind die Funktionen und deren Nutzen der Applikation beschrieben.

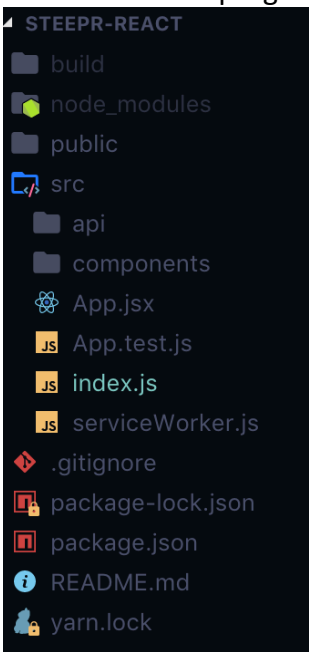
## Schnittstellen

- Pexels API
- React-Bootstrap
- Axios
- Create-React-App V2 Boilerplate

## Projektaufbau

Der Einstiegspunkt der Applikation befindet sich im Root-Verzeichnis unter index.html im Ordner Public. Jedoch ist dies für die. Programmierung in React nicht wichtig. Der eigentliche Einstiegspunkt ist das File index.js welches von index.html importiert wird. Da React sich einen eigenen virtuellen DOM baut, sind html-files nicht nötig. Mehr dazu unter https://reactjs.org/

Im Ordner **‚api'** sind die API-Keys und die Funktionen zur Pexels Schnittstellen Kommunikation programmiert.




Im Ordner **‚components'** sind Applikations-Komponenten (spezifisch die Searchbar) programmiert.

Die App.js Component wird initial aufgerufen und der Konstruktor initialisert den „State" der Applikation. Auch definiert er den Scope der selbst implementierten Funktionen.

### Funktionen

App.jsx

#### componentDidMount()

Diese Funktion wird aufgerufen sobald die Component inital gerendert wurde.
Hier wird die Funktion getImagesFromApi(url) auf.

#### getImagesFromApi(url)

Diese Funktion macht konfigurierte API-Calls, durch die Helper-Funktion welche von pexels.js importiert wird gegen die Pexels Schnittstelle und liefert Bilder als Resultat. Dieses Resultat wird verarbeitet und als Applikations-State gesetzt.

#### render()

Diese Funktion sorgt für den render-Teil der Applikation. Hier wird entschieden was visualisiert wird. Die Funktion render() wird automatisch getriggert wenn Teile des Applikation-States verwendet werden, was hier der Fall ist, und dieser sich ändert. Also im Fall eines erneuten API-Calls gegen aussen und diese Bilder werde als neuen State gesetzt. Danach wird in der render() Funktion das gesamte UI der Applikation zusammengebaut( mit Import der Searchbar Component welche Importiert wird).

Searchbar.jsx (Properties = getImagesFromApi(url), Funktionsübergabe von App.jsx)
Ich enthalte mir bereits erwähnte Funktionen erneut zu erklären.

#### search(e)

Diese Funktion wird getriggert sobald sich die Eingabe im Input Feld der Searchbar entwas ändert. Danach wird ein API-Call gegen die Pexels-API gemacht durch die Funktion getImagesFromAPI(url), welche als Property von App.jsx übergeben wurde.

## Validierung

### Coding Standards

Zur Validierung des reactjs Projektes wurden die Coding Standards von Airbnb gewählt. Diese wurden mithilfe eines weitverbreiteten VS Code Plugins ESLint eingehalten. Dazu wurden die durchdachten Coding Standards der Firma Airbnb gewählt welche in der Entwicklung von reactjs eine grosse Rolle spielen.

### HTML & CSS Validierung

Da Webpack & ESLint verwendet wurden ist eine solche Validierung nicht notwendig(da ESLint während der Implementierung auf Fehler hinweist & Webpack während des bundle-Prozesses diese(falls noch vorhanden) behebt. Aber Tools wie „W3C Validator“ sind bereits bekannt und wurden auch schon öfters verwendet.

## Gewählte Bildformate

Folgende Bilderformate werden dem Enduser zur Verfügung gestellt:

- Original Upload des Fotographen: per Download
- Medium compressed: Anzeige auf der Website + speichern unter..
- Small: per Download
- Tiny gecropped auf 200x280: per Download

Dem Endnutzer können nach Belieben weitere Formate wie:

- Landscape
- Portrait
- Large / Large 2x

Zur Verfügung gestellt werden. Die Enscheidung jedoch fiel bewusst, dass die Performance der Website nicht verloren geht, dem User jedoch eine relativ hohe Qualität geboten werden kann. Deswegen wird das Format Medium für die Anzeige gewählt, welche der Höhe von 350Pixel entspricht mit skalierter Breite.

Die Downloadmöglichkeiten wurden an Hand der Kompetenz «Stellen Sie mehrere mögliche Formate zur Verfügung» implementiert und kann nach Wunsch oder Usability variieren.

## Testing

Folgende Dokumentation beinhaltet lediglich das Testkonzept. Die folgenden Tests wurden weder implementiert noch ausgeführt.

### Unit Tests

Bei den Unit Tests geht es darum, einzelne Programmabschnitte in ihrer eigenen Funktion zu testen. Unit Tests sind die billigsten Tests und decken den grössten Teil der Tests ab.

Folgende Unit Tests sind für unser Projekt zu implementieren:

- Testen der Datenbankverbindung
- Testen der einzelnen Javascript Funktionen
- Testen der unabhängigen Fehlerbehandlung
- Testen API Calls / Backendverbindung
- Testen des Initalwertes
- Testen der Komprimierung

- 

## Integrations Tests

Bei den Integrationstests geht es darum die einzelnen Programmabschnitte in Verbindung zu einander zu testen. Da die einzelnen Funktionen bereits mit den Unit Tests gedeckt sind, ist es nichtmehr nötig die Funktionalität zu testen, sondern nur noch ob diese in Fusion funktionieren.

Folgende Integrationstests sind für unser Projekt zu implementieren:

- Testen der Datenbankfunktionen in Verbindung mit Frontend Interaktion
- Testen der Filterfunktionen in Verbindung mit Datenbankabfragen
- Testen der Komprimierung in Verbindung mit Frontend

## Akzeptanztests

Die Akzeptanztests sind die Überprüfung ob eine Software aus Sicht des Benutzers wie beabsichtigt funktioniert und dieser die Software akzeptiert.

Hier kommen unsere Test Cases ins Spiel, welche nach folgendem Muster aufgebaut werden:

| | |
|---|---|
| **Name vom Testfall** | |
| **Testfall ID** | **UC-001++** |
| **Datum** | |
| **Version** | **1.0** |
| **Autor** | |
| **Tester** | |
| **Betriebssystem** | |
| **Kurzbeschreibung** | |
| **Vorbedingungen** | |
| **Nachbedingungen** | |

| Schritt | Aktivität | Erwartetes Resultat |
|---|---|---|
| 1 | | |
| 2 | | |
| 3 | | |

## Imageupload

| | |
|---|---|
| **Name vom Testfall** | Imageupload |
| **Testfall ID** | **UC-001** |
| **Datum** | - |
| **Version** | **1.0** |
| **Autor** | Valentino Rusconi |
| **Tester** | **Valentino Rusconi** |
| **Betriebssystem** | Mac OS Mojave |
| **Kurzbeschreibung** | Ein Bild wird in der Applikation hochgeladen |
| **Vorbedingungen** | Fileupload wurde implementiert |
| **Nachbedingungen** | Image wurde in der Datenbank gespeichert und wird dem User angezeigt |

| Schritt | Aktivität | Erwartetes Resultat |
|---|---|---|
| 1 | Fileupload | Image wird in der Datenbank gespeichert |
| 2 | Verarbeitung durch Backend | Image wird in drei verschiedene Grössen komprimiert |
| 3 | Bereitstellung neues Bild | Bild wird den User angezeigt |

## Rating

| | |
|---|---|
| **Name vom Testfall** | Rating |
| **Testfall ID** | **UC-002** |
| **Datum** | - |
| **Version** | **1.0** |
| **Autor** | Valentino Rusconi |
| **Tester** | **Valentino Rusconi** |
| **Betriebssystem** | Mac OS Mojave |
| **Kurzbeschreibung** | Ein Bild wird in der Applikation bewertet |
| **Vorbedingungen** | Fileupload wurde implementiert<br>Rating wurde implementiert<br>Bild steht zur Verfügung |
| **Nachbedingungen** | Rating wird in der Datenbank gespeichert und dem User angezeigt |

| Schritt | Aktivität | Erwartetes Resultat |
|---|---|---|
| 1 | Bild wird angeklickt | Rating kann durch UI abgegeben werden |
| 2 | Rating wird abgegeben | Abgegebenes Rating ist sichtbar |

## File Download

| | |
|---|---|
| **Name vom Testfall** | File Download |
| **Testfall ID** | **UC-003** |
| **Datum** | - |
| **Version** | **1.0** |
| **Autor** | Valentino Rusconi |
| **Tester** | **Valentino Rusconi** |
| **Betriebssystem** | Mac OS Mojave |
| **Kurzbeschreibung** | Ein Bild wird heruntergeladen |
| **Vorbedingungen** | Fileupload wurde implementiert<br><br>Bild steht zur Verfügung |
| **Nachbedingungen** | Bild befindet sich auf User Rechner |

| Schritt | Aktivität | Erwartetes Resultat |
|---|---|---|
| 1 | Download wird angeklickt | Bild öffnet sich in neuem Tab (Browser abhängig) |
| 2 | Bild speichern unter… | Bild wurde gesichert |

Filterung

| | |
|---|---|
| **Name vom Testfall** | Filterung |
| **Testfall ID** | **UC-004** |
| **Datum** | - |
| **Version** | **1.0** |
| **Autor** | Valentino Rusconi |
| **Tester** | **Valentino Rusconi** |
| **Betriebssystem** | Mac OS Mojave |
| **Kurzbeschreibung** | Bilder können nach gewünschter Kategorie gefiltert werden |
| **Vorbedingungen** | Fileupload wurde implementiert<br>Bilder stehen zur Verfügung<br>Filterung wurde implementiert |
| **Nachbedingungen** | Gefilterte Bilder werden korrekt angezeigt |

| Schritt | Aktivität | Erwartetes Resultat |
|---|---|---|
| 1 | Filter wird gewählt | Bilder werden korrekt gefiltert und angezeigt |

### Initial Search Query - React

| | |
|---|---|
| **Name vom Testfall** | Initial Search Query |
| **Testfall ID** | **UC-React-001** |
| **Datum** | |
| **Version** | **1.0** |
| **Autor** | Valentino Rusconi |
| **Tester** | **Valentino Rusconi** |
| **Betriebssystem** | Mac OS Mojave |
| **Kurzbeschreibung** | Die initial query der API- Abfrage soll geladen werden |
| **Vorbedingungen** | Applikation läuft / Initial query ist gesetzt |
| **Nachbedingungen** | Bilder mit Query ‘Switzerland’ werden initial geladen |

| Schritt | Aktivität | Erwartetes Resultat |
|---|---|---|
| 1 | Applikation starten | Bilder werden geladen |

Download {dateigrösse} - React

| | |
|---|---|
| **Name vom Testfall** | Download {Dateigrösse} |
| **Testfall ID** | **UC-React-002** |
| **Datum** | |
| **Version** | **1.0** |
| **Autor** | Valentino Rusconi |
| **Tester** | **Valentino Rusconi** |
| **Betriebssystem** | Mac OS Mojave |
| **Kurzbeschreibung** | Bilder lassen sich in verschiedenen Grössen herunterladen / speichern -> browserabhängig |
| **Vorbedingungen** | Applikation läuft<br><br>Bilder sind geladen und Liste mit auswählbaren Grössen steht zur Verfügung |
| **Nachbedingungen** | Bilder sind in gewünschter Grösse auf Userrechner gespeichert |

| Schritt | Aktivität | Erwartetes Resultat |
|---|---|---|
| 1 | Applikation starten | Bilder werden geladen |
| 2 | Bild in gewünschter Grösse anklicken | Bild öffnet/ downloaded in neuem Tab |
| 3 | Speichern unter… | Bild ist in gewünschter Grösse auf Rechner gespeichert |

### Search Funktion - React

| | |
|---|---|
| **Name vom Testfall** | Search Funktion |
| **Testfall ID** | **UC-React-003** |
| **Datum** | |
| **Version** | **1.0** |
| **Autor** | Valentino Rusconi |
| **Tester** | **Valentino Rusconi** |
| **Betriebssystem** | Mac OS Mojave |
| **Kurzbeschreibung** | Bilder werden in Echtzeit an Hand der Suchquery geladen |
| **Vorbedingungen** | Applikation läuft<br><br>Suchfeld ist verfügbar |
| **Nachbedingungen** | Bilder mit Query werden initial geladen |

| Schritt | Aktivität | Erwartetes Resultat |
|---|---|---|
| 1 | Applikation starten | Bilder werden geladen |
| 2 | Individuelle Query wird ins Suchfeld eingetragen | Bilder an Hand der Query werden geladen |

### Pagination Next - React

| | |
|---|---|
| **Name vom Testfall** | Pagination Next |
| **Testfall ID** | **UC-React-004** |
| **Datum** | |
| **Version** | **1.0** |
| **Autor** | Valentino Rusconi |
| **Tester** | **Valentino Rusconi** |
| **Betriebssystem** | Mac OS Mojave |
| **Kurzbeschreibung** | Wenn mehr wie 15 Bilder und eine Next Page zur Verfügung stehen soll ein Next Button gerendert werden welcher bei betätigen die nächsten Bilder ladet |
| **Vorbedingungen** | Applikation läuft<br><br>Next Button ist verfügbar |
| **Nachbedingungen** | Nächste Bilder Seite wird geladen |

| Schritt | Aktivität | Erwartetes Resultat |
|---|---|---|
| 1 | Applikation starten | Bilder werden geladen |
| 2 | Next Button wird betätigt | Nächste Seite wurde erfolgreich geladen |

Pagination Previous - React

| | |
|---|---|
| **Name vom Testfall** | Pagination Previous |
| **Testfall ID** | **UC-React-005** |
| **Datum** | |
| **Version** | **1.0** |
| **Autor** | Valentino Rusconi |
| **Tester** | **Valentino Rusconi** |
| **Betriebssystem** | Mac OS Mojave |
| **Kurzbeschreibung** | Wenn eine andere Seite wie die initial Seite geladen ist, soll ein Previous Button gerendert werden welcher die Bilder der vorherigen Seite ladet und anzeigt |
| **Vorbedingungen** | Applikation läuft<br><br>Previous Button ist verfügbar |
| **Nachbedingungen** | Vorherige Bilder Seite wird geladen |

| Schritt | Aktivität | Erwartetes Resultat |
|---|---|---|
| 1 | Applikation starten | Bilder werden geladen |
| 2 | Previous Button wird betätigt | Vorherige Seite wurde erfolgreich geladen |

### Browser Kompatibilität - React

| | |
|---|---|
| **Name vom Testfall** | Browser Kompatibilität |
| **Testfall ID** | **UC-React-006** |
| **Datum** | |
| **Version** | **1.0** |
| **Autor** | Valentino Rusconi |
| **Tester** | **Valentino Rusconi** |
| **Betriebssystem** | Mac OS Mojave |
| **Kurzbeschreibung** | Applikation funktioniert in allen Browsern identisch und wie erwartet |
| **Vorbedingungen** | Applikation läuft |
| **Nachbedingungen** | Applikation wird in allen Browsern gleich gerendert und verhält sich wie erwartet |

| Schritt | Aktivität | Erwartetes Resultat |
|---|---|---|
| 1 | Applikation starten | Bilder werden geladen |
| 2 | Vorherige Tests werden in allen Browsern durchgeführt | Alle Test Cases sind erfolgreich |

### Mobil Kompatibilität - React

| | |
|---|---|
| **Name vom Testfall** | Mobil Kompatibilität |
| **Testfall ID** | **UC-React-007** |
| **Datum** | |
| **Version** | **1.0** |
| **Autor** | Valentino Rusconi |
| **Tester** | **Valentino Rusconi** |
| **Betriebssystem** | Mac OS Mojave |
| **Kurzbeschreibung** | Applikation funktioniert resposive und wie erwartet |
| **Vorbedingungen** | Applikation läuft |
| **Nachbedingungen** | Applikation wird auf allen Geräten (Mobile / Resposive) wie erwartet dargestellt und verhält sich resposive |

| Schritt | Aktivität | Erwartetes Resultat |
|---|---|---|
| 1 | Applikation starten | Bilder werden geladen |
| 2 | Vorherige Tests werden auf allen in Browser verfügbaren Mobile Versionen durchgeführt | Alle Test Cases sind erfolgreich |