Entwurfsbeschreibung
Nachrichtenkommunikation für das THW

na17b

Inhaltsverzeichnis

1 Allgemeines

Die Anwendung soll den gegebenen Nachrichtenfluss in einer Notfallzentrale des THW nachbilden.

2 Produktübersicht

Die Anwendung führt zunächst auf eine Landing Page,die nach Einsatz, Rolle, Name und Zeichen des Users fragt. Durch einen Klick auf Speichern gelangt man in die eigentliche UI. Im oberen Bereich befindet sich eine Kopfleiste, die einen Platzhalter für ein Logo, "Nachrichtenkom. für das THW", den Einsatznamen, Handzeichen, Rollenzeichen und -bezeichnung anzeigt. Über den Komplex Rollenzeichen und -bezeichnung besteht die Möglichkeit auf die Option des Nutzerwechsels zugreifen zu können, die der Lading Page entspricht. Am linken Bildrand befindet sich ein Menü, welches den Wechsel zwischen Übersicht, Erstellung eines Formulars,sowie einem Entwurf ermöglicht. Die Übersichtsseite vermittelt einen schnellen Überblick über alle hinterlegten Vordrucke, eingeschränkt auf Status, Technische Betriebsbuchnummer, Verfasser, Sichter, Datum, Uhrzeit, Empfänger und eine gekürzte Version des Inhalts. In der ersten Spalte befindet sich ein Button, über den der Vordruck zur genaueren Ansicht ausgewählt werden kann. Über der Formularübersicht befindet sich eine Leiste Filter, die durch Klick ausgeklappt werden kann. Dort können Filter so gewählt werden, dass jedes beliebige Formular einsehbar ist. Es kann nach Zeichenketten im Dokument, Kürzel des Sichters und der Betriebsbuchnummer gesucht werden. Die Formularmaske bildet den Vierfachvordruck realitätsnah nach. Je nach Status in dem sich die Nachricht befindet wird entweder ein blauer, grüner, roter oder gelber Rand angezeigt. Die Felder, die nicht zum ausfüllen vorhergesehen sind, sind ausgegraut, trotzdem jedoch beschreibbar. Ebenfalls vom Status der Nachricht sind die Buttons am rechten Rand. Stets ist ein Button zum leeren des Formulars existieren, wobei nur eigene Eingaben gelöscht werden. Es besteht die Möglichkeit einen Entwurf zuspeichern, wenn man die Formularseite ohne weiterschicken verlassen will. Es findet derzeit auch noch keine Überprüfung der Daten hinsichtlich Konsistenz und Vollständigkeit statt.

3 Grundsätzliche Struktur

Der Quelltext der Anwendung ist serviceorientiert aufgebaut. Hierzu werden Container verwendet welche die jeweiligen Services kapseln. Zwingende Voraussetzung in der Entwicklungsumgebung ist:

- Docker
  - Bevor die Anwendung ausgeführt werden kann müssen auf dem ausführenden Computer (Host) die folgenden Programme installiert sein:
    - docker
    - docker-compose
  - Die Anwendung ist auf zwei Docker-Container verteilt:
    - Server- Container der das Frontend und Controller-Logik implementiert
    - Daten-Container der den SPARQL-Endpoint implementiert

4 Struktur- und Entwurfsprinzipien einzelner Pakete

Frontend

Das Frontend ist in JavaScript geschrieben und verwendet das Vue.js-Framework. Als Erweiterungen kommen Vuex, sowie Vue-Router zum Einsatz. Die Verwaltung des Quelltextes sowie das kompilieren geschieht via webpack. Zur Vereinfachung wird auf die Bibliothek von Element-UI zurückgegriffen. Der Quelltext des Frontends befindet sich im Ordner `src/frontend`. In diesem Ordner befinden sich

neben einem weiteren Ordner `src` die von webpack benötigten Dateien und Ordner; diese werden im Folgenden nicht weiter beschrieben und sind weitgehend Standard. Der Ordner `src` unterteilt sich weiter in folgende Struktur:

```
./src/frontend/src/
├── api/
├── assets/
├── components/
├── router/
├── sparql_help/
├── store/
│   ├── actions.js
│   ├── getters.js
│   ├── index.js
│   ├── mutations.js
│   └── state.js
├── test-examples/
├── App.vue
├── main.js
└── status.config.js
```

- `api` Quistore-Adapter
- `assets` Mediendateien
- `components` Vue-Komponenten
- `router` Konfiguration von vue-router
- `sparql_help` SPARQL Umwandler
- `store` Konfiguration von Vuex
  - `actions.js` formuliert einen POST Request an Quitstore
  - `getters` stellt Funktionen zum Abrufen der Variablen bereit
  - `index.js` fügt die obigen Dateien zusammen zu einem globalen Store
  - `mutations` enthält Funktionen zum Bearbeiten des Zustandes
  - `state.js` deklariert globale Zustandsvariablen
- `test-examples` Beispieltests
- `App.vue` Wrapper-Komponente für das Frontend
- `main.js` Einstiegspunkt für das Programm, enthält alle Importe
- `status.config.js` regelt das Formularverhalten bezüglich der jeweiligen Status

```
Frontend - Arbeitsverzeichnis von Webpack
├── dist - Kompilierte und minimierte Webapp
└── restliche - Konfiguration des Webpack oder node-module, wird in der Regel
    nicht geändert
```

5 Datenmodell

Als Datenmodell wird RDF verwendet. Die Eingaben aus der UI werden über eine SPARQL-Schnittstelle im Quitstore in Tripel-Form gespeichert. Der Quitstore wird über `Projekt/quitstore.sh` lokal gestartet. Jedem Dokument wird eine ID, bestehend aus einer sechsstelligen Zufallszahl, zur Identifizierung zugeordnet. Der Quitstore stellt die Daten auf Anfrage als JSON-Response zur Verfügung.
Bei Aufrufen der Übersichtsseite werden alle für die Übersicht wichtigen Daten aus dem QuitStore über eine SPARQL Select Query ausgelesen und angezeigt. Wird ein Dokument geöffnet, werden, ebenfalls über eine Select Query, alle für dieses Formular gespeicherten Informationen ausgelesen um angezeigt werden zu können. Ebenfalls können nun Einsätze angelegt werden, die im Quitstore gespeichert werden, wodurch es möglich ist jedem Dokument den aktuellen Einsatz zuzuordnen.