

Webbasierte Anwendungen

Web-Ueb-08 (12.06.2024)

Heute beginnen wir damit, für unser Spring-Backend ein schickes Vue3-Frontend zu bauen. Zunächst basteln wir uns dazu einen Rahmen mit Kopf- und Fußzeile, der für alle Ansichten stabil bleibt, und danach betten wir eine Ansicht mit einer Liste der Mitfahrgelegenheiten (Touren) ein. Nächste Woche holen wir uns die Echtdateien vom Server, diese Woche simulieren wir die Datenversorgung noch mit fest codierten Strukturen.

Aufgabe 1 (Frontend-Projekt anlegen)

Bitte legen Sie im Spring-Projektverzeichnis `projekt` mit `npm create vue@latest frontend` das Frontend-Projekt an (also auf derselben Ebene wie z.B. `gradlew`). Da der erzeugte Ordner `frontend` im `Spring-projekt`-Verzeichnis liegt, können Sie es leicht dem bisherigen Projekt-Git hinzufügen und dort mitverwalten. Denken Sie aber bitte daran, dass das Vue-Projekt aus Sicht von VSCode ein eigenes Projekt ist, und VSCode anhand der Inhalte des zu Beginn geöffneten Ordners seine Unterstützungsfunktionalität aktiviert. Wenn Sie also den `projekt`-Ordner in VSCode öffnen, bekommen Sie Spring-Unterstützung. Sie sehen den `frontend`-Unterordner (weil er eben da liegt), aber für VSCode ist das "irgendein Unterordner mit Dateien", so dass unter Umständen keine volle Vue-Unterstützung aktiviert wird. Es ist daher sinnvoll, den `frontend`-Ordner in einem separaten VSCode-Fenster zu öffnen oder einen VSCode Workspace ("Arbeitsbereich" in der deutschen Lokalisierung) mit beiden Ordnern, dem `projekt`- und dem enthaltenen `frontend`-Ordner, anzulegen, siehe <https://code.visualstudio.com/docs/editor/workspaces>).

Wählen Sie bitte bei `npm create` folgende Optionen aus (d.h. jeweils Auswahl `yes`)

- Add TypeScript
- Add Vue Router for Single Page Application development
- Add Pinia for state management
- Add ESLint for code quality
- Prettier for code formatting

Wie Sie wissen, sind in der Datei `package.json` die benötigten Abhängigkeiten gelistet. Bitte installieren Sie diese mit dem Kommando `npm install` (ohne Parameter) im `frontend`-Ordner.

Mit `npm run dev` (oder dem entsprechenden Eintrag in VSCodes "NPM-SKRIPTS"-Tab) sollten Sie danach den Entwicklungs-Server starten und sich unter `http://localhost:5173` das Projekt ansehen können.

Aufgabe 2 (App.vue – Rahmenkomponente)

Demo-Inhalte im Frontend-Projekt entfernen

Beim Aufsetzen des Frontendprojekts wurden einige Demo-Komponenten angelegt, die wir nicht brauchen. Bitte leeren Sie unter `src` die mit Demo-Inhalten gefüllten Verzeichnisse `components` und `views`, wir werden sie nachfolgend mit eigenen Inhalten füllen. Da Git keine leeren Verzeichnisse verwalten kann, können Sie jeweils eine (leere) Datei `.gitkeep` als Platzhalter in den Ordnern anlegen.

Wenn Sie Frontend-globale CSS-Styles definieren können, können Sie die vordefinierte Datei `assets/main.css` bearbeiten. Sie wird von `main.ts` importiert, hier können Sie den Import der Stylesheet-Datei auch löschen, wenn Sie diese nicht verwenden möchten.

Beim Anlegen des Projekts haben wir auch die “Vue Router”-Komponente aktiviert. Im generierten Demoprojekt ist diese für zwei Beispielkomponenten konfiguriert, die wir gerade gelöscht haben. Um Fehlermeldungen aus der Datei `router/index.ts` ruhig zu stellen, leeren Sie bitte das Array im `routes`-Attribut des Parameter-Objekt von `createRouter()`. Wir konfigurieren den Router nächstes Mal mit unseren eigenen Ansichten.

App.vue leeren

In einem ersten Schritt passen Sie die Rahmenkomponente `App.vue` bitte so an, dass eine geeignete Kopfzeile gezeigt wird, in der wir später Navigations-Links unterbringen können. Dazu räumen wir den vorgenerierten Beispieldinhalt erst einmal weg.

Bitte leeren Sie in `App.vue` vorerst das `<script setup lang="ts">`- und das `<template>`-Element. Auch den Inhalt des `<styles>`-Abschnitts können Sie entfernen. Die Komponente `App.vue` hat damit deutlich an Übersichtlichkeit gewonnen, zeigt aber nichts an.

App.vue füllen

Da `App.vue` den Rahmen für alle Ansichten unserer Anwendung bildet, ist dies auch eine schöne Stelle, um Kopf- und Fußzeilen, die auf allen Ansichten unserer Anwendung erscheinen sollen, anzulegen.

In der Spring/Thymeleaf-Anwendung hatten wir bereits eine Kopf- und ggf. Fußzeile als Fragmente angelegt. Sie können das entsprechende HTML in den `<template>`-Teil der `App.vue` übernehmen, müssten aber natürlich Thymeleaf-spezifische Attribute entfernen. Zur Übernahme von Frontend-weiten CSS-Klassen können Sie z.B. das oben erwähnte `assets/main.css` verwenden.

Starten Sie (sofern noch nicht geschehen) bitte den Frontend-Entwicklungsserver (`npm run dev`) und browse Sie ihn. Das Zwischenergebnis könnte wie folgt aussehen:



Fehler-Box bei Bedarf aufklappen

Perspektivisch möchten wir von überall in der Anwendung eine Statusmeldung setzen können (diese soll dann in der Statuszeile ausgegeben werden), und wenn die Statusmeldung auf den Leerstring gesetzt wird, soll die Statuszeile verschwinden. Für heute begnügen wir uns mit einem reaktiven Objekt in `App.vue`.

Ergänzen Sie Ihre `App.vue` bitte im `<script>` Teil provisorisch um eine Ref namens `info`, die mit dem String “Dies ist eine Nachricht” vorbelegt ist.


Sehen Sie nun bitte im `<template>` unter der Kopfzeile eine Infobox derart vor, dass der Inhalt der `info`-Ref darin ausgegeben wird, wenn deren Inhalt nicht leer ist (wenn keine Nachricht gesetzt ist, soll die Infobox also gar nicht erst erscheinen). Mit einem “x”-Button in der Infobox soll `info` nach einem Click auf den Leerstring gesetzt werden, woraufhin die Infobox (von selbst) verschwinden sollte.

Um die Funktionalität testen zu können, können Sie übergangsweise einen zusätzlichen Button “Hallo” in `App.vue` einbauen, der die Infonachricht wieder auf einen nicht-leeren String wie “Hallo” zu setzen. Allein das Setzen der `info` sollte die Infobox wieder aufploppen lassen.



Aufgabe 3 (TourenListView aus TourenListe mit TourenListeZeile)

Nun wollen wir unseren `App.vue`-Rahmen mit einem ersten Inhalt füllen. Hierzu möchten wir im Ordner `src/views` eine neue Vue-Komponente `TourenListView.vue` haben, welche die Überschrift “Das aktuelle Mitfahrangebot” ausgibt und danach eine Tabelle der Mitfahrgelegenheiten.

 **Touren**

Das aktuelle Mitfahrangebot

Abfahrt	von	nach	Entfernung	Plätze	frei	Preis	
1010-11-22 10:01:00	Wiesbaden	Schöngleina	264 km	1	1	10	Details
2023-11-17 19:30:00	Wiesbaden	Taucha	333 km	3	3	5	Details
2024-05-10 16:27:00	Wiesbaden	Bochum	173 km	2	2	12	Details
2033-02-01 18:00:00	Wiesbaden	Clenze	370 km	3	3	17	Details

Lizenzen

Kontakt

Im `read.MI` finden Sie unter dem Namen `Web-Ueb-08-TourenListView.sec` eine **Vorlage**, die Sie bitte unter dem Namen `TourenListView.vue` in den `src/views`-Ordner Ihres Frontend-Projekts übernehmen. Sie enthält eine `interface`-Definition für `ITourDTD` und ein kleines Array mit Beispieldaten als `Ref`. In einer späteren Ausbaustufe werden wir diesen `Behelfs`-Code durch einen Zugriff auf die “echte”, aktuelle Touren-Liste aus unserem `Spring-Backend` ersetzen.

Die in der View enthaltene Tabellen-Komponente `TourenListe.vue` bezieht ihren Inhalt aus einem Array von Datenobjekten des (Interface-)Typs `ITourDTD`, das ihr über die Property `touren` hereingereicht werden sollen, wie Sie in `TourenListView.vue` sehen können.

Bitte legen Sie im `src/components`-Ordner Ihres Frontend-Projekts einen Unterordner `tour` an und implementieren Sie darin

- die von dieser View eingebettete Komponente `TourenListe.vue`, welche eine in der Prop `touren` übergebene Array von Tourdaten ausgibt, indem sie
- für jeden Datensatz eine im selben Ordner zu entwickelnde `TourListeZeile.vue`-Komponente anzeigt (der im Screenshot gezeigte “Details”-Link in jeder Zeile ist zunächst ohne Funktion).

2033-02-01 18:00:00	Wiesbaden	Clenze	370 km	3	3	17	Details
---------------------	-----------	--------	--------	---	---	----	-------------------------

Während fast alle Daten direkt aus dem auszugebenden Datensatz übernommen werden können, ist die Spalte “frei” mit den noch zur Verfügung stehenden Plätzen nicht dort enthalten, lässt sich aber leicht aus der Gesamtzahl der Plätze `plaetze` und der Anzahl der bereits gebuchten `buchungen` bestimmen (für die Attributnamen bitte `Interface`-Definition von `ITourDTD` ansehen).

Wenn Sie testen wollen, wie die Anwendung bei Datenänderungen reagiert, können Sie gerne einen experimentellen `<button>` in die `TourenListView.vue` einbauen und eine Funktion auslösen lassen, welche die `tourenliste` verändert, beispielsweise, indem Sie für einen oder mehrere Einträge des Arrays den Preis erhöhen oder die Anzahl der gebuchten Plätze. Eine Änderung an den Daten müsste unmittelbar ein Update der Tabelle nach sich ziehen (bei Erhöhung der Anzahl gebuchter Plätze müsste z.B. die ausgegebene, errechnete Anzahl der freien Plätze automatisch abnehmen).