

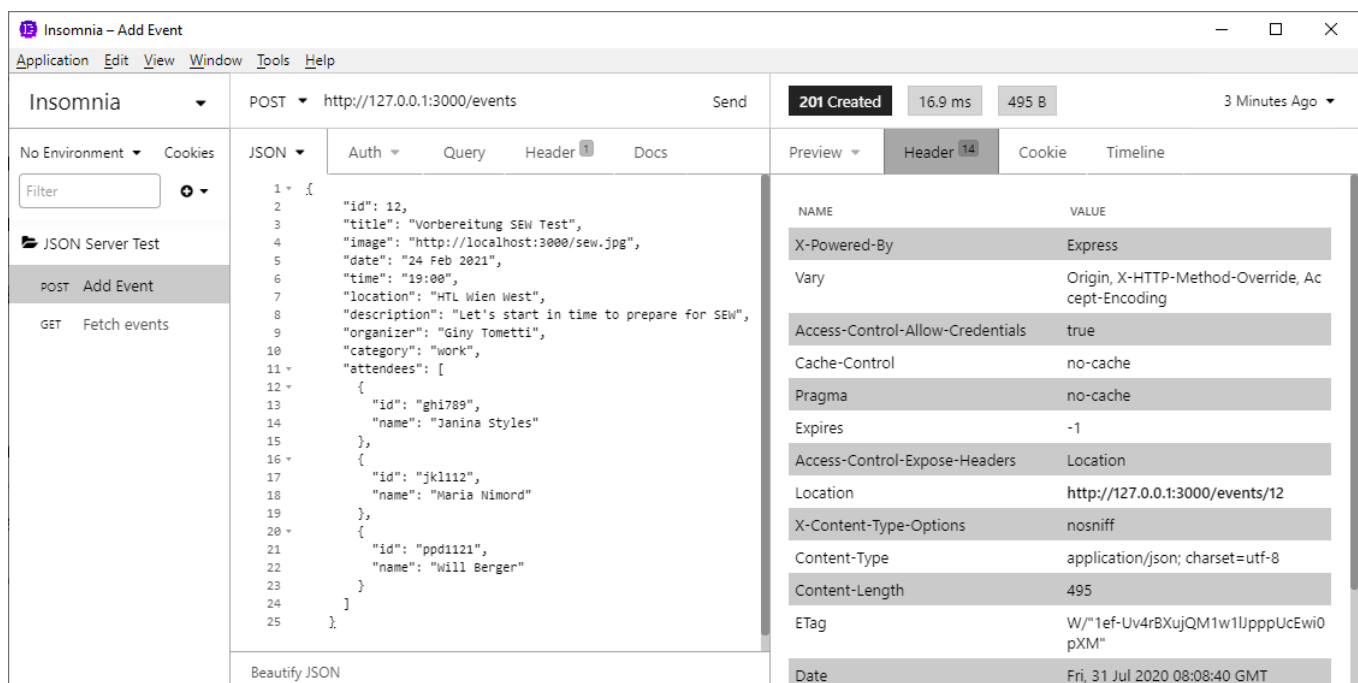
Warum clientseitige Validierung?

Bevor Daten an den Server übermittelt werden (und vielleicht später in einer Datenbank gespeichert werden), ist es sinnvoll, sicherzustellen, dass alle erforderlichen Formularfelder im richtigen Format ausgefüllt sind. Die Überprüfung und das Feedback erfolgen mittels JavaScript im Browser. Der Vorteile liegen auf der Hand:

- Unmittelbares Feedback an den User und damit bessere Benutzerfreundlichkeit
- Weniger unnötiger Datenverkehr zwischen Client und Server
- Geringere Serverlast

Ersparst du dir damit die Daten am Server nochmals zu überprüfen? **Nein**. Ein bössartiger User kann jederzeit mit einem Tool (**Isomnia**, **Post(wo)man**, **Rest Client**,...) den Client umgehen und einen Request senden (auch nützlich zum Testen von Server APIs). Tests am Server nennt man [serverseitige Validierung](#).

Beispiel **Insomnia**:



In erster Linie wollen wir die Eingabe von Formulardaten überprüfen. Es gibt dazu zwei Möglichkeiten:

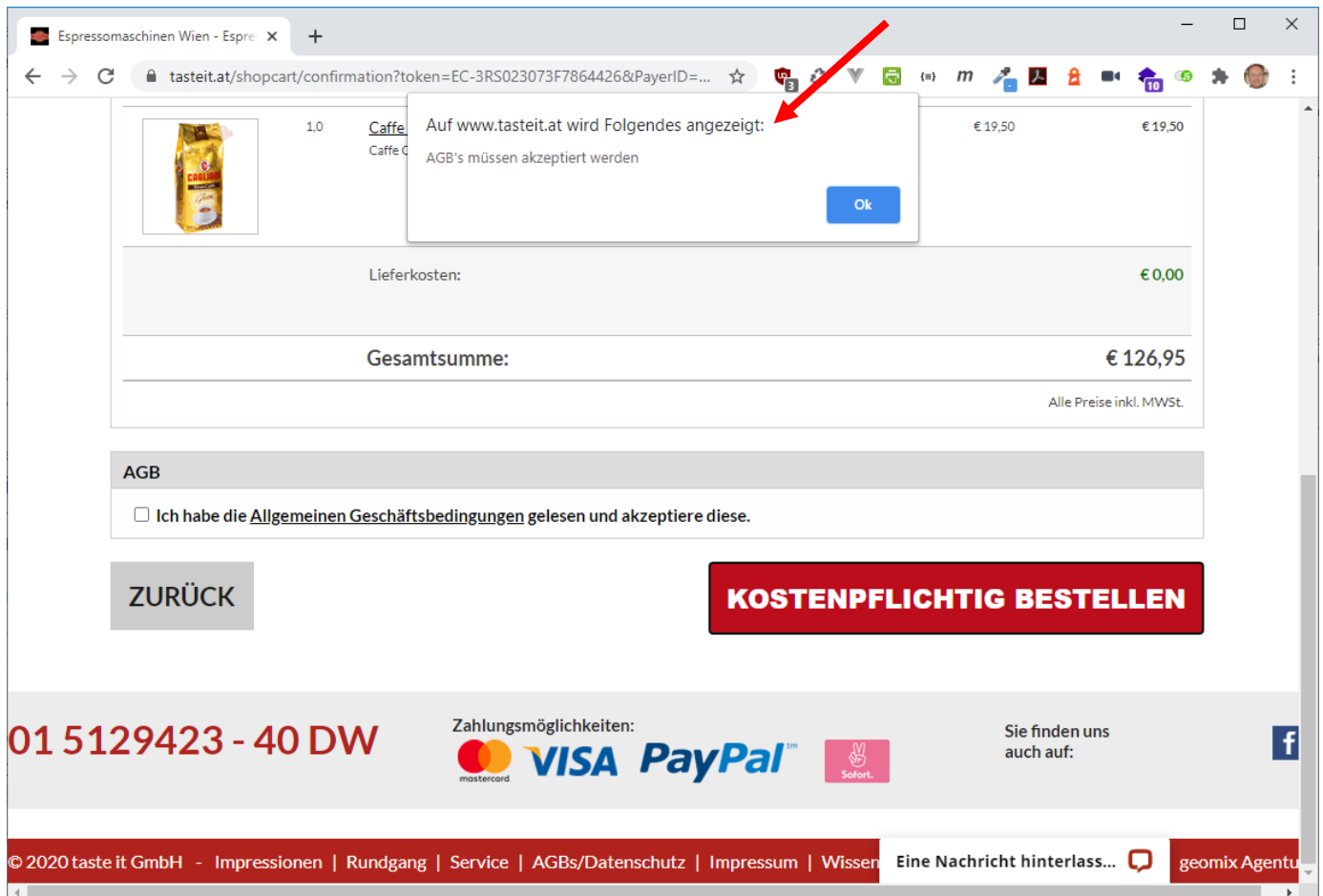
- Verwendung von HTML 5 Features wie zum Beispiel **required** (schnell, aber unflexibel).
- Verwendung von JavaScript (aufwendiger, aber flexibel).

Welche Arten von Fehlern wollen wir überprüfen?

Beispiele für Fehlerarten	
Fehlermeldung	Bedeutung
Dieses Feld muss ausgefüllt werden.	Dieses Feld darf nicht leer sein.
Bitte gib deine Telefonnummer im Format xxx-xxxx.	Ein bestimmtes Datenformat ist erforderlich, damit es als gültig betrachtet werden kann.
Bitte gib eine gültige E-Mail-Adresse ein.	die eingegebenen Daten sind nicht im richtigen Format.
Das Passwort muss zwischen 8 und 30 Zeichen lang sein und einen Großbuchstaben, ein Symbol und eine Zahl enthalten.	Für die eingegebenen Daten ist ein sehr spezifisches Datenformat erforderlich.

Wir werden in diesem Arbeitsblatt JavaScript verwenden, um Formulardaten zu überprüfen. Um die Implementierung zu erleichtern, gibt es zahlreiche Libraries, die wir verwenden können. Natürlich auch für Vue. In diesem Arbeitsblatt werden wir **Vuelidate** verwenden, da die Lib einfach zu verwenden ist und die Art und Weise, wie die Tests geschrieben werden, gut zu Vue.js passt.

Sehen wir uns aber zunächst ein schlechtes reales Beispiel an:



Die Problempunkte:

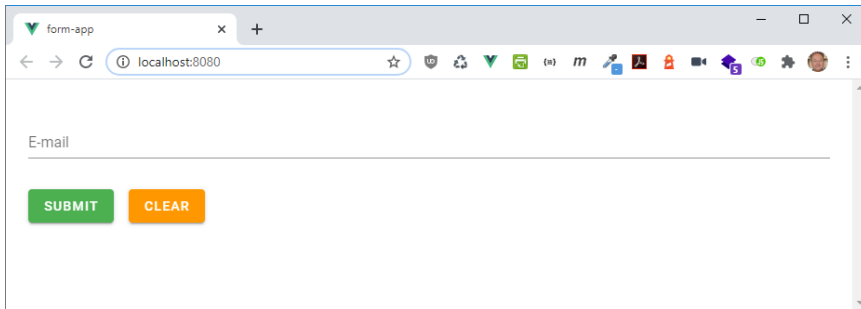
- Sieht einfach trashy aus!
- Der Benutzer bekommt das Feld mit dem Fehler nicht direkt im Formular markiert. Das Alert wird ganz oben auf der Seite angezeigt und kann leicht übersehen werden.
- Die Fehlermeldung sollte neben, über oder unter der fehlerhaften Eingabe stehen.
- Das Drücken des OK Buttons beim Alert ist lästig.
- Was, wenn es mehrere Fehler gibt? Diese sollten alle auf einmal angezeigt werden.

Ok, und wie geht es besser?

Das werden wir im nächsten Abschnitt behandeln!

Clientseitige Form Validierung mit Vuelidate

1. Entpacke die Beispiellapplikation **form-app 1 - Start.zip** und installiere die fehlenden Node-Module. Das GUI besteht zunächst nur aus einem Eingabefeld für eine E-Mail Adresse.



2. Installiere **Vuelidate** mit:

```
npm i vuelidate
```

in **Main.js** füge hinzu:

```
import Vuelidate from 'vuelidate';  
Vue.use(Vuelidate);
```

Vuelidate stellt sogenannte Validators zur Verfügung. Diese beinhalten bereits programmierte Prüfroutinen¹.

Beispiele: `required`, `minLength`, `minValue`, `maxValue`, `between`, `alpha`, `email`, `url`, `ipAddress`,...

Um einen der eingebauten Validierer zu verwenden, musst du ihn zuerst importieren.

Du kannst auch eigene Validators programmieren.

3. Importiere in **App.vue** die Validators `required`, `email`.

required: das Feld muss ausgefüllt werden

email: prüft mit einer Regex, ob es sich um eine gültige E-Mail Adresse handelt

```
import {required, email} from 'vuelidate/lib/validators';
```

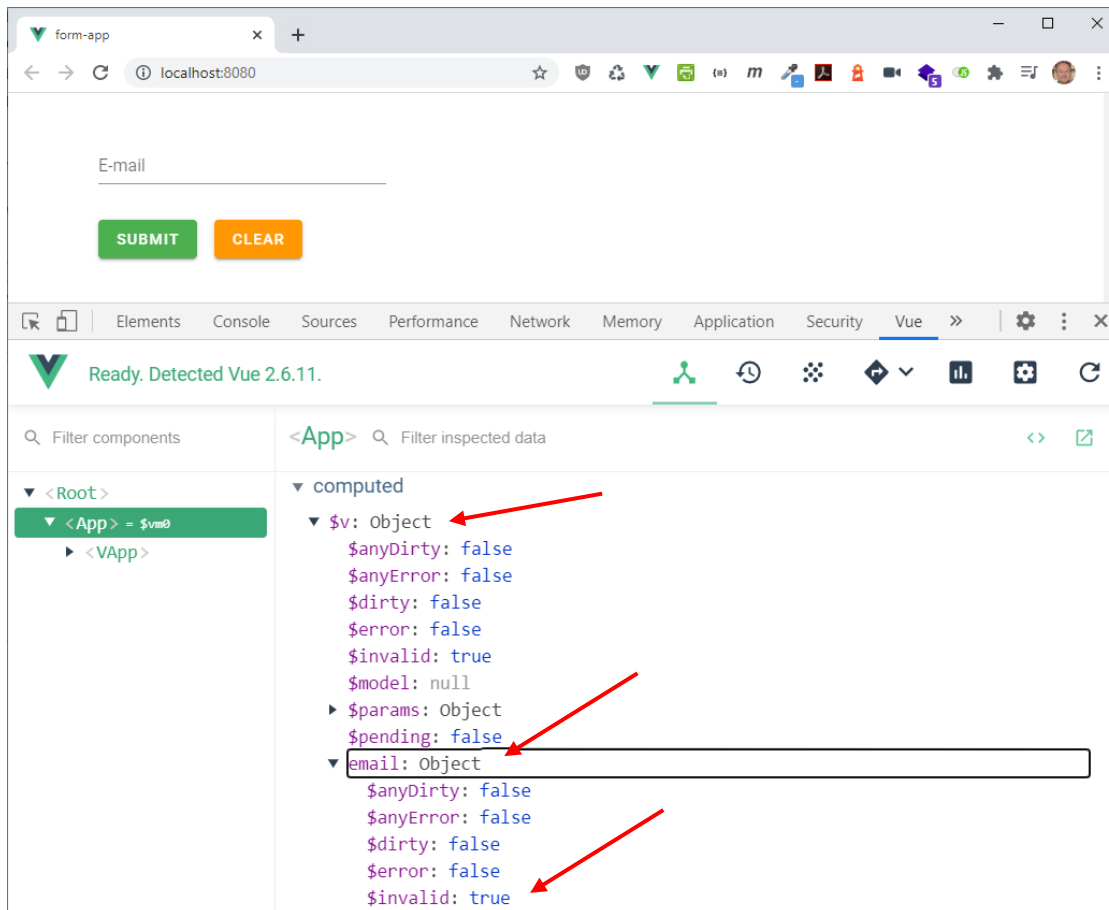
4. Füge ein Property `validations` mit einem Objekt hinzu. In diesem Objekt definiere die Validations für `email` mit `required` und `email`.

```
export default {  
  name: 'App',  
  
  data: () => ({  
    email: null,  
  }),  
  
  validations: {  
    email: { required, email },  
  },  
};
```



5. Prüfe im Browser, dass Vuelidate ein computed Property `$v` mit einem Property `email` angelegt hat!

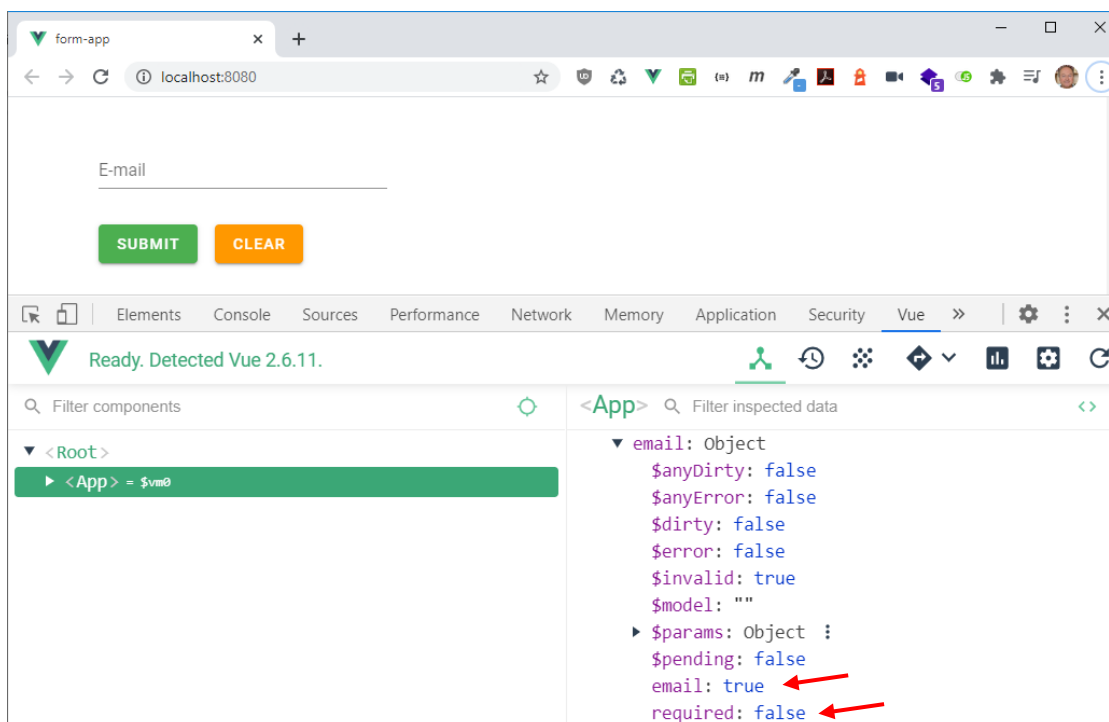
¹ <https://vuelidate.js.org/#sub-builtin-validators>



The screenshot shows a web browser at localhost:8080 with a form containing an 'E-mail' input field and two buttons: 'SUBMIT' (green) and 'CLEAR' (orange). Below the browser, the Vue DevTools component inspector is open, displaying the component tree on the left and the component's data on the right. The component tree shows the root component as <App> = \$vm0. The component's data is shown as a computed object with the following properties: \$anyDirty: false, \$anyError: false, \$dirty: false, \$error: false, \$invalid: true, \$model: null, \$params: Object, \$pending: false, and email: Object. The email object has properties: \$anyDirty: false, \$anyError: false, \$dirty: false, \$error: false, \$invalid: true, and \$model: null. Red arrows point to the \$invalid: true property in both the \$v object and the email object.

Vuelidate hat bereits begonnen das Property **email** zu überprüfen. **\$invalid** ist **true**!

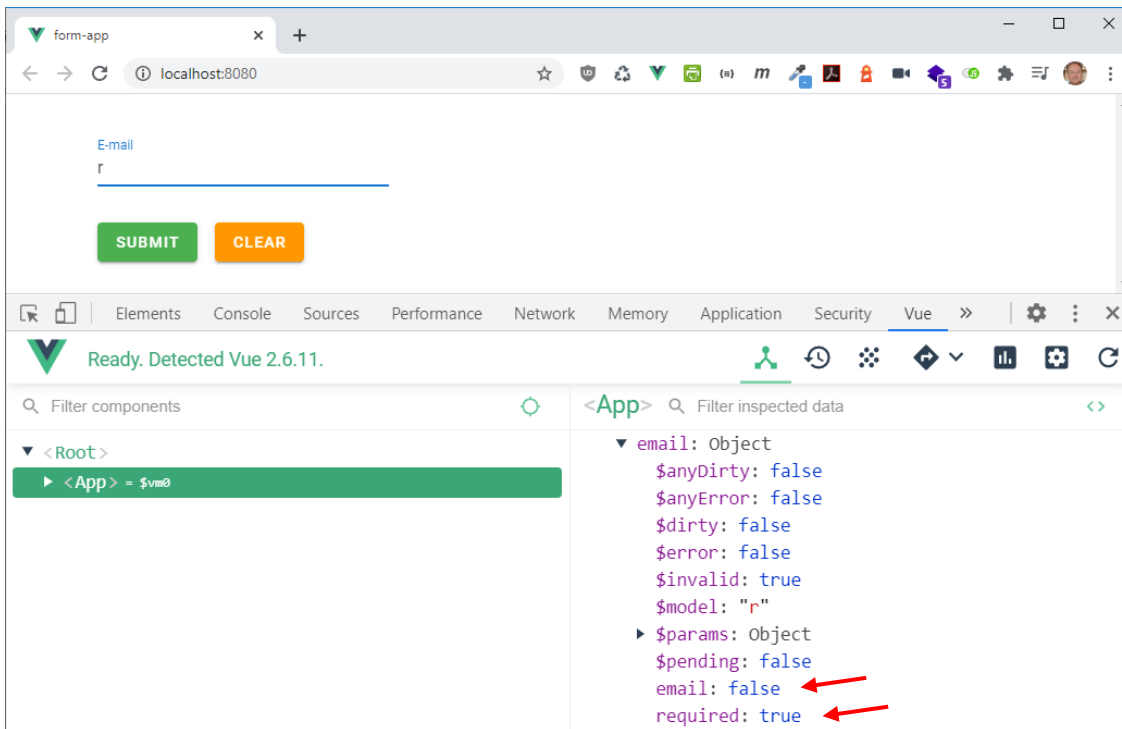
Es gibt im Objekt **email** zwei Properties (**required** und **email**), die das Ergebnis unserer Validators darstellen.



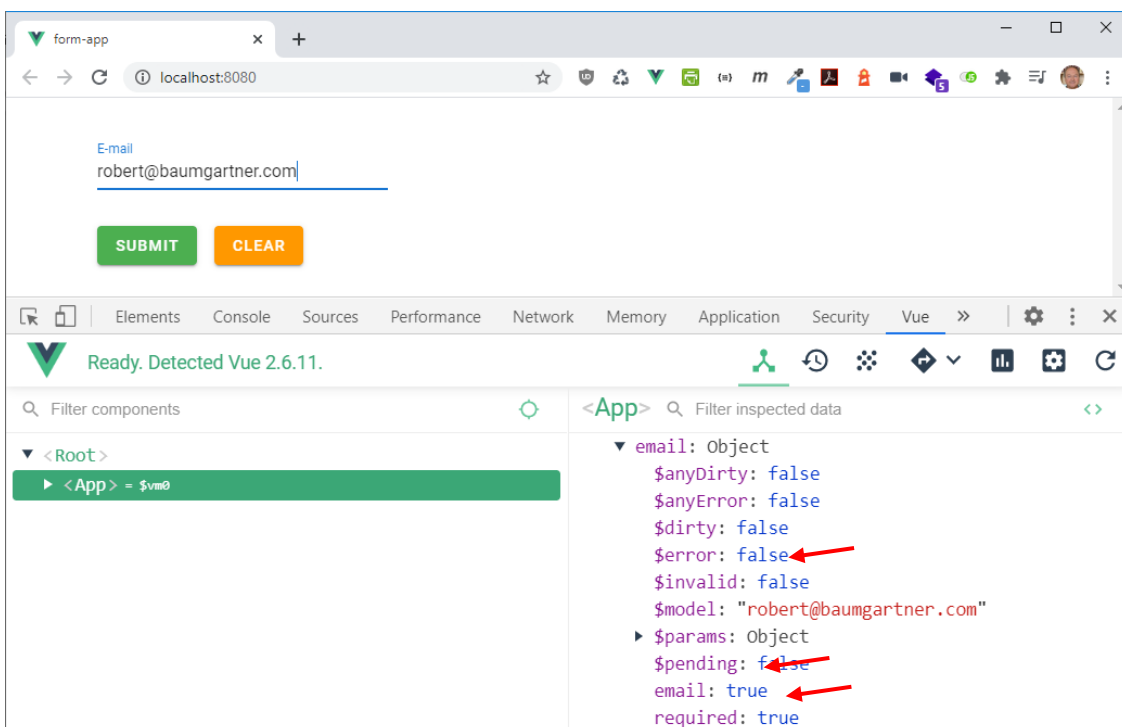
The screenshot shows the same web browser and Vue DevTools component inspector. The component tree on the left is the same. The component's data on the right is expanded to show the 'email' object. The 'email' object has properties: \$anyDirty: false, \$anyError: false, \$dirty: false, \$error: false, \$invalid: true, \$model: "", \$params: Object, \$pending: false, email: true, and required: false. Red arrows point to the email: true and required: false properties.

required ist **false**, da nichts ausgefüllt wurde!

Wenn der User nun beginnt eine E-Mail Adresse zu tippen, wird **required** **true** und **email** ist solange **false**, bis die Adresse gültig ist. Das ist ein Beispiel für eine Überprüfung während des Tippens.



6. Gib eine gültige E-Mail Adresse ein und checke die Properties von `email`.



Der nächste Schritt ist nun dem User eine Fehlermeldung anzuzeigen.

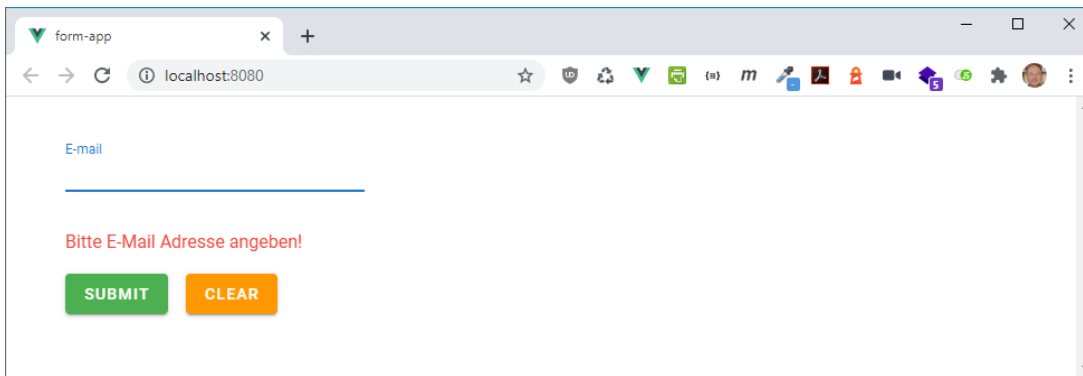
7. Erstelle dazu zwei `p` Tags mit den Texten:

Bitte E-Mail Adresse angeben!

Bitte E-Mail richtig ausfüllen!

Verwende die obigen Properties `email` und `required` mit `v-if`.

Die Fehlermeldung wird sofort angezeigt.



form-app

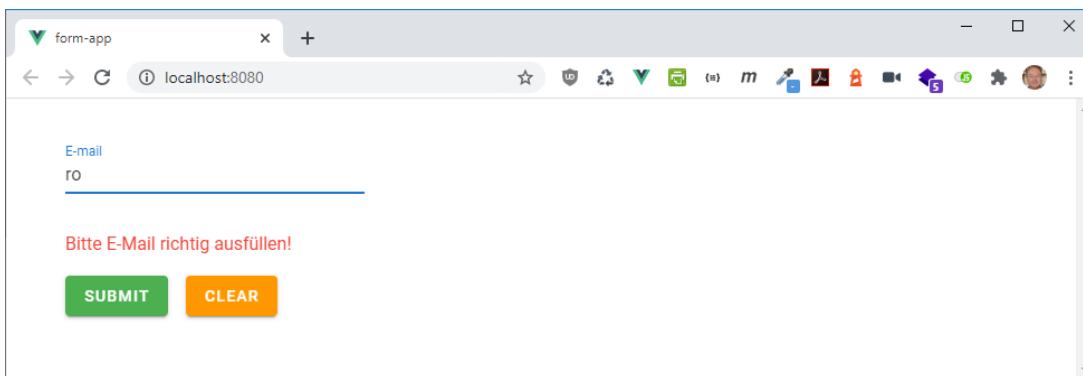
localhost:8080

E-mail

Bitte E-Mail Adresse angeben!

SUBMIT CLEAR

Sobald etwas eingetippt wird, ändert sich die Meldung:



form-app

localhost:8080

E-mail

ro

Bitte E-Mail richtig ausfüllen!

SUBMIT CLEAR

Das ist natürlich nicht optimal, da sich die App schon beschwert, bevor der User etwas eingetippt hat! Stelle dir das bei mehreren Eingabefeldern vor!

Wir müssen also die Fehlermeldung unterdrücken, bis der User etwas im Formular getan hat.

Vuelidate stellt dazu das Property `$dirty` zur Verfügung. Der Wert `true` bedeutet, der User hat mit dem Formularfeld etwas gemacht.

Also ist der Fehlerfall gegeben, wenn `$dirty true` ist und `$invalid true` ist.

Aber wie können wir `$dirty` auf `true` setzen? Dazu gibt es in Vuelidate die Methode `$touch`.

Wir rufen also im Fall, dass der User etwas mit dem Formularfeld gemacht hat diese Methode auf:

```
$v.email.$touch()
```

Aber bei welchen Events? Nun, es sind zwei Events für uns interessant:

input: Wird jedes Mal ausgelöst, nachdem ein Wert durch den Benutzer geändert wurde.

blur: Wird ausgelöst, wenn der User den Fokus auf ein Formularelement gesetzt hat und dann zu einem andern Teil der Seite wechselt.

8. Füge den unten stehenden Code dem Eingabefeld hinzu:

```
@input="$v.email.$touch() "
```

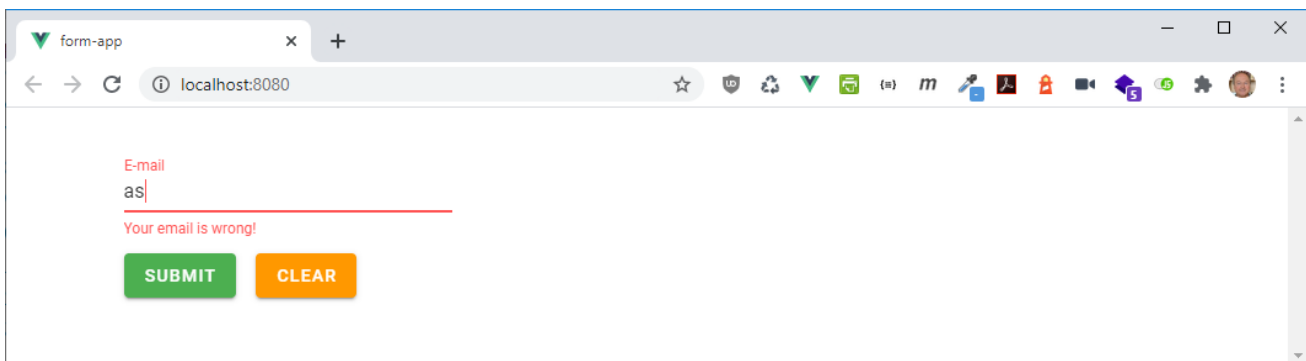
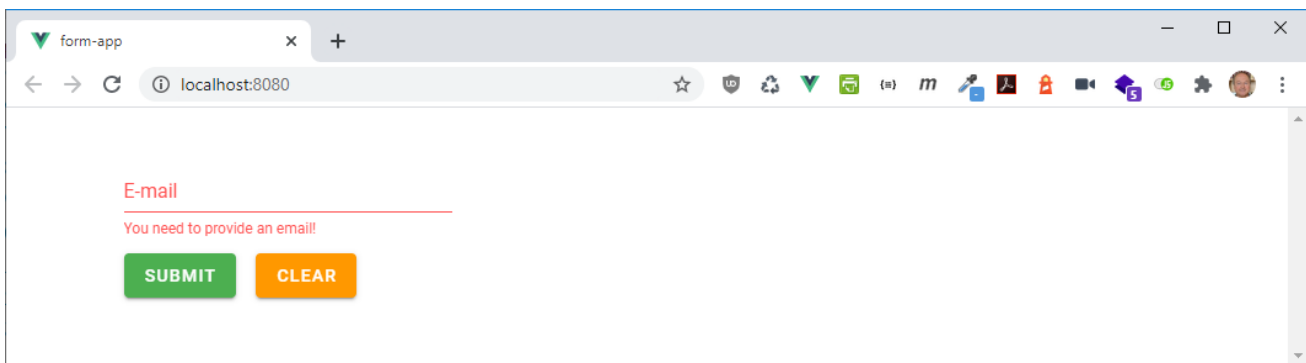
```
@blur="$v.email.$touch() "
```

Wir definieren nun ein computed Property welches ermittelt, wann welche Errormessage angezeigt werden soll.

Erstelle ein computed Property `emailErrors`:

```
computed: {  
  emailErrors() {  
    let errors = [];  
    if (!this.$v.email.$dirty) return errors;  
    if (!this.$v.email.email) errors.push('Your email is wrong!');  
    if (!this.$v.email.required) errors.push('You need to provide an email!');  
    return errors;  
  }  
}
```

9. Binde nun das computed Property an das Attribut `error-messages` des Eingabefeldes und voilà du hast eine animierte Fehlerausgabe!



Du kannst auch den `input` Event beim Eingabefeld weglassen, dann wird die E-Mail Adresse erst gecheckt, wenn der User das Feld verlässt.

10. Nun noch den Fall von `submit` behandeln. Wenn der User auf `submit` drückt, rufen wir eine Methode auf, die wiederum die `touch()` Methode aufruft. Ist danach `$v.$invalid == false`, simulieren wir ein Absenden der Daten durch die Ausgabe `submitted` auf der Console.

```
submit() {  
  this.$v.$touch();  
  if (!this.$v.$invalid) console.log('submitted');  
}
```

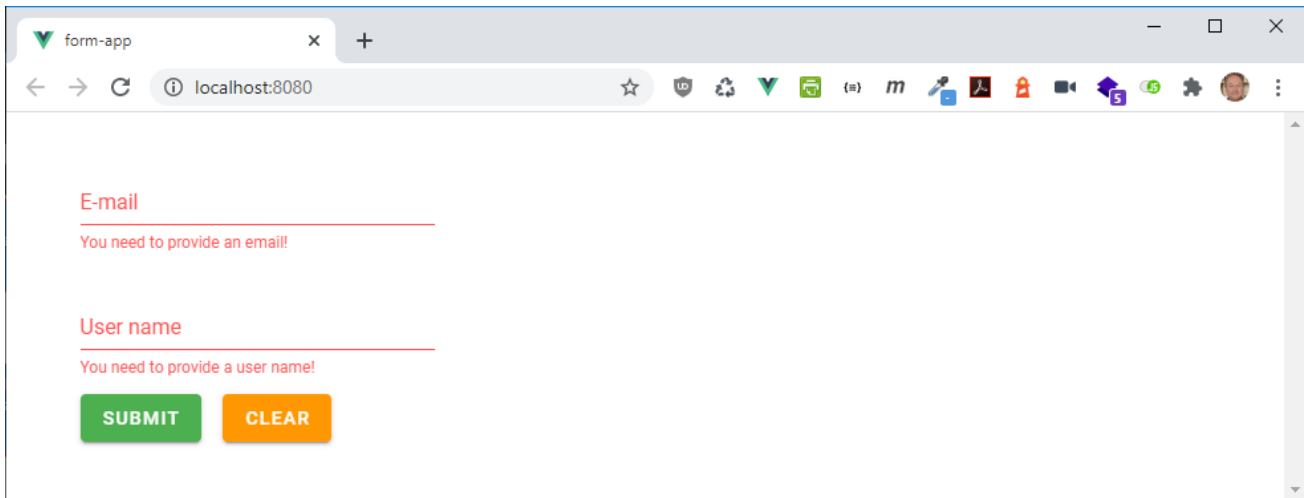
11. Füge nun ein Eingabefeld für den User Namen hinzu. Regeln:

- Muss ausgefüllt sein
- muss mindestens 8 Stellen lang sein
- darf nur aus Buchstaben und Zahlen bestehen

Lies dazu die Onlinedokumentation (<https://vuelidate.js.org/#sub-builtin-validators>)!

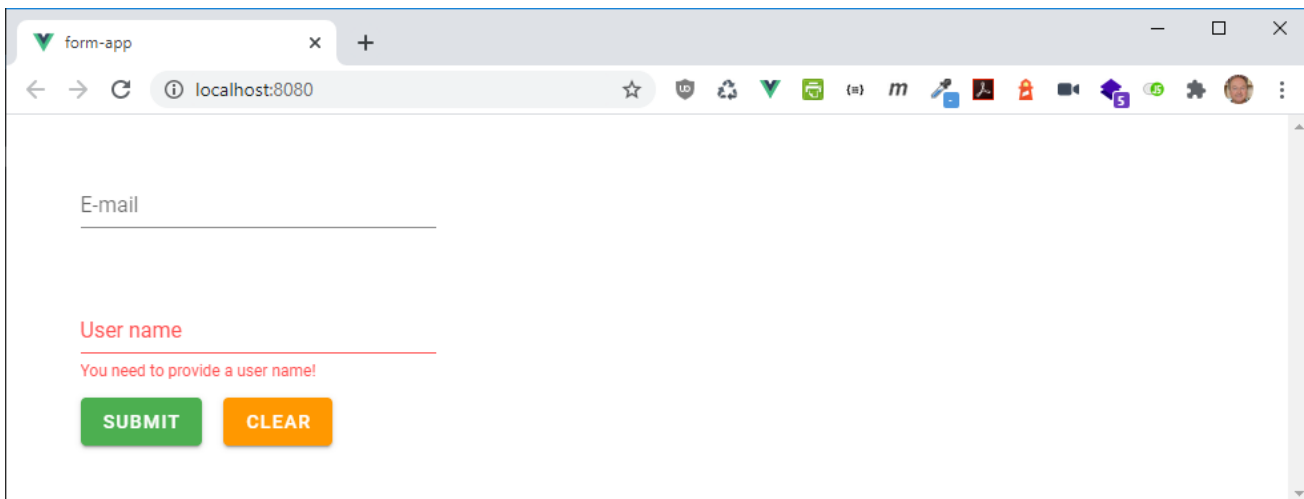
Tipp: Angabe der minimalen Länge bei Property `validations`: mit `minLength`: `minLength(8)`

Submit gedrückt:



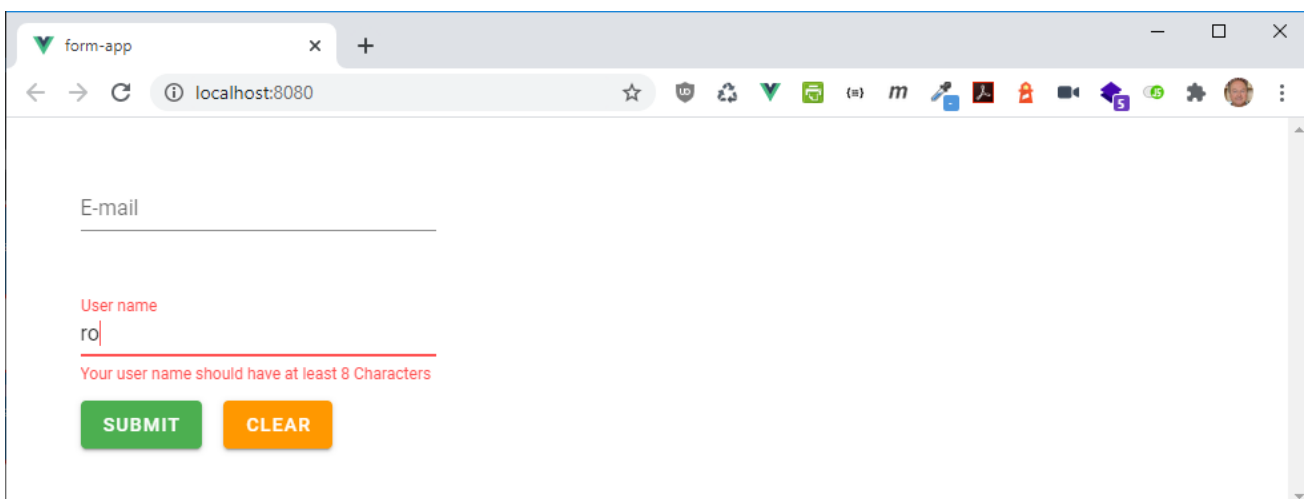
The screenshot shows a web browser window with the address bar at localhost:8080. The page contains a form with two input fields. The 'E-mail' field has a red border and a red error message below it: 'You need to provide an email!'. The 'User name' field also has a red border and a red error message below it: 'You need to provide a user name!'. Below the fields are two buttons: 'SUBMIT' (green) and 'CLEAR' (orange).

Fokus auf **User name** und dann auf Background:

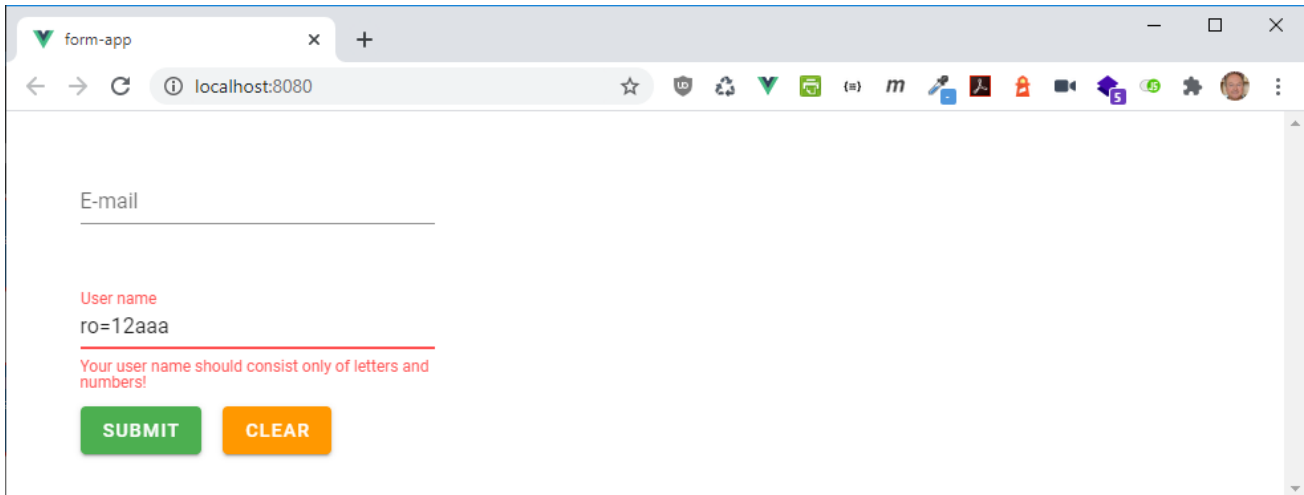


The screenshot shows the same web browser window. The 'User name' field now has a blue border, indicating it has focus. The 'E-mail' field has a light blue background. The error message for 'User name' is still present: 'You need to provide a user name!'. The 'SUBMIT' and 'CLEAR' buttons are still visible.

Eingaben:



The screenshot shows the same web browser window. The 'User name' field now contains the text 'ro'. The error message below it has changed to: 'Your user name should have at least 8 Characters'. The 'SUBMIT' and 'CLEAR' buttons are still visible.



form-app x +

localhost:8080

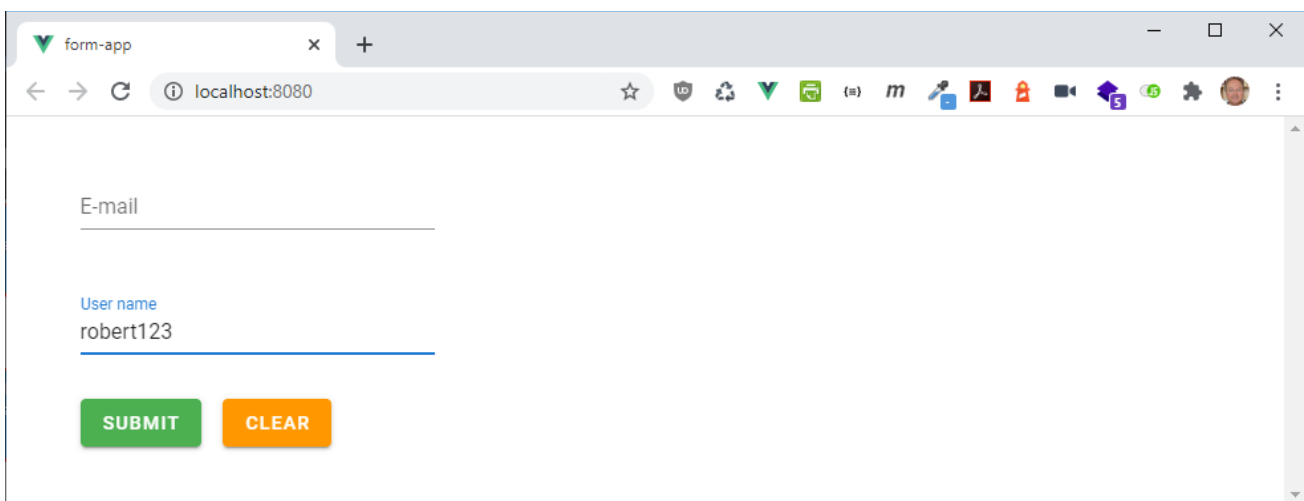
E-mail

User name

ro=12aaa

Your user name should consist only of letters and numbers!

SUBMIT CLEAR



form-app x +

localhost:8080

E-mail

User name

robert123

SUBMIT CLEAR