Afbeelding met tekst

Automatisch gegenereerde beschrijving

Web 2.0 Application

Eventguard

Maxim Brabants | Kwinten De Rop

| Web Technologies | December 23 2022

Table of contents

[Introduction 2](#_Toc122692986)

[Frameworks & Technologies 3](#_Toc122692987)

[Database Management System 3](#_Toc122692988)

[Functional Requirements 3](#_Toc122692989)

[Home page/ landing page 3](#_Toc122692990)

[Registration process 3](#_Toc122692991)

[Frontend form validation 5](#_Toc122692992)

[Name – input field 6](#_Toc122692993)

[Telephone number – input field 6](#_Toc122692994)

[Email address – input field 6](#_Toc122692995)

[Username – input field 7](#_Toc122692996)

[Password – input field 7](#_Toc122692997)

[backend form validation 7](#_Toc122692998)

[Resources 10](#_Toc122692999)

# Introduction

This project report describes the full development of a Web 2.0 application around a topic of free choice. The ultimate goal of the website is to allow users to view, search, create, manage and share information.

The idea behind this particular project is an event organizing application where there exist two roles in the system. Either you have the users who are named as ‘organizers’ and this type of user is the one who is willing to plan an event or wants to organize an event in the future, but of course these users are in need of different kinds of materials or equipment in order to set up the particular big or small event.

This is where the companies or providers come into use because these are the type of users that offer or lend this kind of equipment to various organizers. The type of equipment can vary from sound installations, light equipment and even stage building constructions. Actually every sort of provider instance that can offer some kind of physical entity can be involved in the system. Briefly described: organizers can look or search up equipment or even companies on the website and providers can add new content to their page in the form of new products along with their information. Other functionalities are presented in the remaining part of this document.

# Frameworks & Technologies

TODO Kwinten

# Database Management System

TODO Kwinten

# Functional Requirements

This section of the document enlightens all the present functionalities that are included in the system. First, we will cover the registration and login process. How can users register for an account in the system? How can they log themselves in? What kinds of information do they need to provide in order to create an account? All these kinds of questions will be answered in this section.

## Home page/ landing page

## Registration process

This is almost every time the main entry point of nearly every website, out there on the internet. Users need to provide some personal information, they create their accounts with it, then they login in with this data and only from that moment on, the website can be used to its full potential.

Afbeelding met tekst

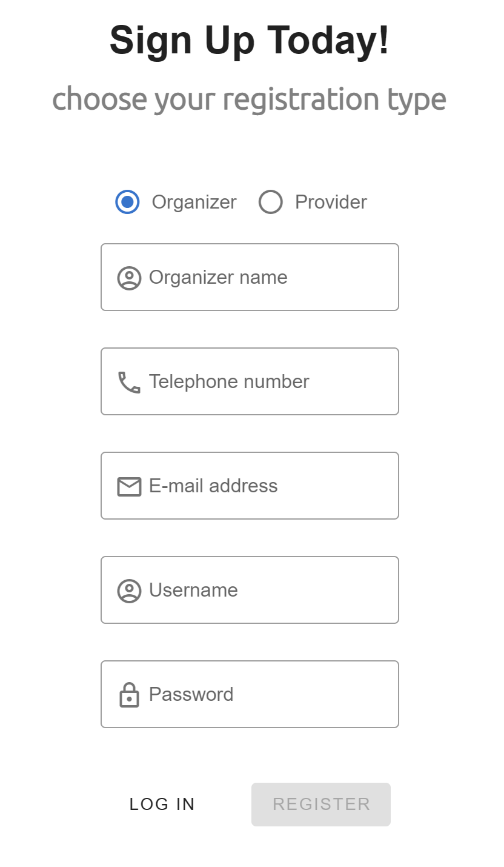
Automatisch gegenereerde beschrijving

Figure 1: organizer register form

Figure 2: provider register form

The above two snippets are taken from the official registration page of the website. On the left side, the registration form for the organizers is visible and on the right side, the exact same version but only for the providers/companies. Actually these two pages look exactly the same, except for the name fields where the label differs a bit. We chose to provide two separate forms, since we have a MongoDB collection for each of the user types. So we have a collection for all the organizers and one for the providers. This way it’s easier to store their information in the correct collection afterwards.

## Frontend form validation

Of course, all the provided user data in the form fields have to be checked upon potential dangers or false information. For example, some malicious people could abuse certain form fields in order to perform known SQL-injections in case of no validation or users could maybe input an email address that doesn’t correspond to an existing one. For this purpose we included some input rules for each field to prevent these situations from happening.

Afbeelding met tekst

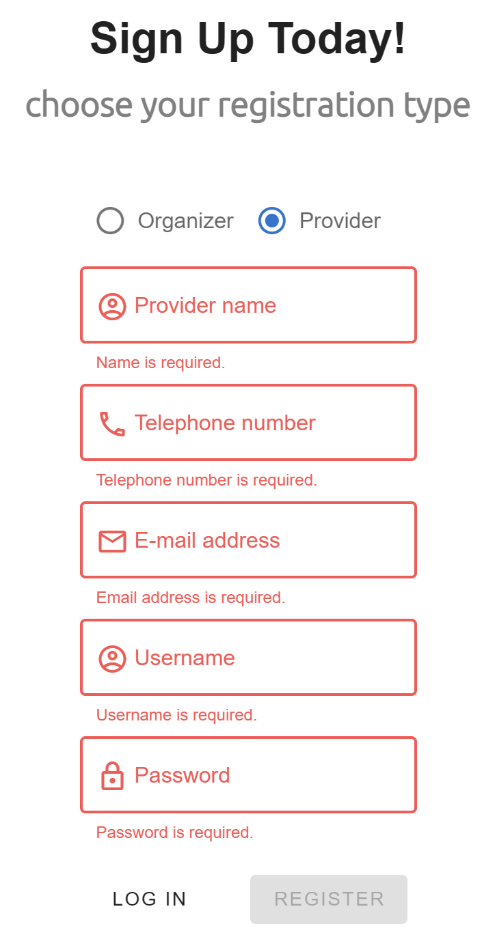
Automatisch gegenereerde beschrijving

Figure 4: disabled register button

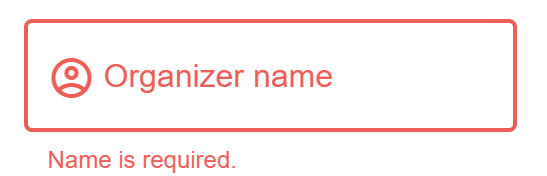
Figure 3: required validation

To begin with, all the form fields are required to be filled in, so no fields may be empty when submitting the form. In case there would be an empty field, it gets clearly highlighted by the system. On top of that, the system automatically disables the submit button (REGISTER button) when one or more of the fields are not filled in according to the rules.

Not only this condition is checked when the user inputs its data. Each field has an extra set of distinct validation rules defined as follows:

### Name – input field

For the name, we just **require it to be filled in**. No extra conditions are added here because it’s simply a name which doesn’t have to be checked against some format.



### Telephone number – input field

When it comes to the mobile phone number, we expect the user to provide just a **valid phone number**, meeting the following regular expression:

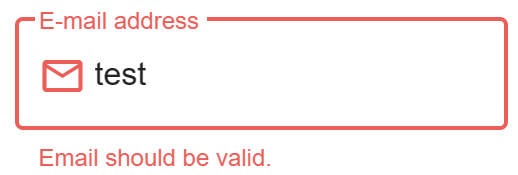
/[\+]?[(]?[0-9]{3}[)]?[-\s\.]?[0-9]{3}[-\s\.]?[0-9]{4,6}/

Afbeelding met tekst

Automatisch gegenereerde beschrijving

### Email address – input field

The user also has to submit an email address that is **valid** with respect to the following format: “**x@x.x**”.



### Username – input field

The username is a value that is required to be **unique** to every user. Although this is not checked real-time, furthermore the username has to be **longer than 6 characters**.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

### Password – input field

A password is needed as well to log in along with the username. The validation rule states that the password has to contain at minimum **a lowercase letter, a number and an uppercase letter**.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

We use the following regular expression for this purpose:

/(?=.\*\d)(?=.\*[a-z])(?=.\*[A-Z]).{6,}/

If all the conditions are met and the user has provided the right input in every field, the user gets redirected to the login page. Also a visual confirmation message is shown at the top so the user knows everything went fine.

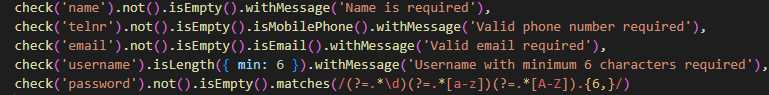
Afbeelding met tekst

Automatisch gegenereerde beschrijving

## backend form validation

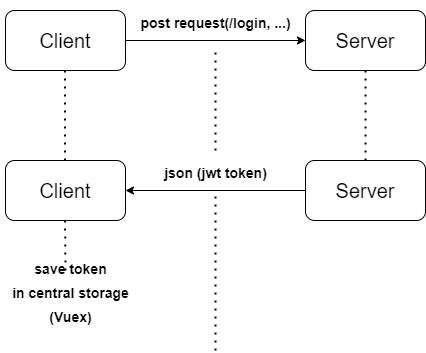
All the data, submitted by the user or any person that fills in the form, is eventually sent to the server and there it gets processed, but assume we wouldn’t have any form where the user could enter their credentials. Then the data would be entered through some web interface or request engine. This way, any data could be sent to the server without getting validated.

As a consequence of this, the server is also equipped with backend validation. Our server performs this with the help of Express middleware functions where we put one intermediate function for each value.



When the server side validation is performed, the data can be processed and put in the database. After all this is done, the user gets redirected to the login page where their email address and password can be entered. As mentioned earlier, form validation is performed on the login side as well. As soon as their credentials are sent to the server, these ones get checked upon their existence in database. So, the username, which is a unique field in the database, gets checked. When it exists, the password gets checked with the help of the **bcrypt ‘compare’ function** because if we just compare the normal password with the one that is encrypted in database, it won’t work.

When the user is finally authenticated, the server creates a **JSON Web Token** which gets sent back to the user. The logged in user can use this token to later authorize themselves with it if they want to access resources specific to that user account. The full process is actually as follows:



Of course this JWT token has to be stored somewhere on the client side. For this we used **Vuex.** It is a state management pattern + library for Vue.js applications. It serves as a centralized store for all the components in our application, with rules ensuring that the state can only be mutated in a predictable fashion.

As soon as we want to change something about this user state, we can just call methods in this state management to either get some user data, or set data about the user.

# Resources

* https://github.com/vue-leaflet/Vue2Leaflet/issues/476
* https://codepen.io/hesguru/pen/BaybqXv