<template>

<section class="add-user">

<form

ref="formRef"

autocomplete="off"

class="row g-3 m-4"

novalidate

@submit.prevent

>

<InputContainer

:invalid-feedback="usernameErrorMessage"

input-id="username"

input-label="username"

>

<input

id="validationUsername"

v-model.trim="username"

:class="{

'is-invalid': !usernameMeta.valid && usernameMeta.validated,

}"

class="form-control"

name="username"

required

type="text"

/>

</InputContainer>

<InputContainer

:invalid-feedback="passwordErrorMessage"

input-id="password"

input-label="password"

>

<input

id="validationPassword"

v-model.trim="password"

:class="{

'is-invalid': !passwordMeta.valid && passwordMeta.validated,

}"

class="form-control"

name="password"

required

type="password"

/>

</InputContainer>

<InputContainer

:invalid-feedback="emailErrorMessage"

input-id="email"

input-label="email"

>

<input

id="validationEmail"

v-model.trim="email"

:class="{ 'is-invalid': !emailMeta.valid && emailMeta.validated }"

class="form-control"

name="email"

required

type="email"

/>

</InputContainer>

<InputContainer

:invalid-feedback="roleErrorMessage"

input-id="role"

input-label="role"

>

<select

id="validationRole"

v-model.trim="role"

:class="{ 'is-invalid': !roleMeta.valid && roleMeta.validated }"

class="form-select"

name="role"

required

>

<!-- <option :value="role" disabled selected>Please select a role</option>-->

<option

v-for="\_role in usersRoles"

:key="\_role"

:selected="role"

:value="\_role"

>

{{ startCase(\_role) }}

</option>

</select>

</InputContainer>

<InputContainer

:invalid-feedback="phoneErrorMessage"

input-id="phone"

input-label="phone"

>

<input

id="validationPhone"

v-model.trim="phone"

:class="{ 'is-invalid': !phoneMeta.valid && phoneMeta.validated }"

class="form-control"

name="phone"

required

type="tel"

/>

</InputContainer>

<hr class="my-3" />

<FormButtonsContainer>

<FormButton skin="primary" text="Add" @click="onAddAndNewClick" />

<FormButton

outline

skin="dark"

text="Add & View"

@click="onAddAndView"

/>

<FormButton

outline

skin="secondary"

text="Add & View all"

@click="onAddAndViewAll"

/>

<FormButton outline skin="danger" text="Clear" @click="onClear" />

</FormButtonsContainer>

</form>

<Teleport to="body">

<ToastContainer :placement="TOP\_CENTER">

<LiveToast

ref="toastSuccess"

skin="success"

@on-hidden-bs-toast="onHiddenBsToast"

/>

<LiveToast ref="toastError" skin="danger" />

</ToastContainer>

</Teleport>

</section>

</template>

<script lang="ts" setup>

import { TOP\_CENTER } from "@/constants/toasts";

import ToastContainer from "@/components/toast/ToastContainer.vue";

import LiveToast from "@/components/toast/LiveToast.vue";

import FormButton from "@/components/button/FormButton.vue";

import InputContainer from "@/components/form/InputContainer.vue";

import FormButtonsContainer from "@/components/form/FormButtonsContainer.vue";

import type { Ref } from "vue";

import { ref } from "vue";

import { useUsersStore } from "@/stores/app/users/users";

import moment from "moment";

import type { NewUserDto } from "@/stores/app/users/dto";

import startCase from "lodash/startCase";

import { useField } from "vee-validate";

import { useRouter } from "vue-router";

const router = useRouter();

const formRef = ref();

const toastSuccess = ref();

const toastError = ref();

const usersStore = useUsersStore();

const usersRoles: Ref<string[]> = ref([]);

try {

usersRoles.value = await usersStore.fetchUsersRoles();

} catch (error: any) {

toastError.value?.setupToast({

name: "Roles Error",

elapsedDuration: moment().startOf("second").fromNow(),

heading: "Fetch Roles Error",

text: "Failed to fetch roles from the server",

delay: 5000,

});

toastError.value?.show();

}

const usernameValidation = (value: string) => {

if (!value) {

return "This field is required";

}

return true;

};

const passwordValidation = (value: string) => {

if (!value) {

return "This field is required";

}

return true;

};

const emailValidation = (value: string) => {

if (!value) return "This field is required";

return true;

};

const phoneValidation = (value: string) => {

if (!value) return "This field is required";

if (value.length < 10)

return "The phone number should contain not less than 10 characters";

if (value.length > 10)

return "The phone number should contain not more than 10 characters";

return true;

};

const roleValidation = (value: string) => {

if (!value) return "This field is required";

return true;

};

const {

value: username,

errorMessage: usernameErrorMessage,

meta: usernameMeta,

} = useField("username", usernameValidation);

const {

value: password,

errorMessage: passwordErrorMessage,

meta: passwordMeta,

} = useField("password", passwordValidation);

const {

value: email,

errorMessage: emailErrorMessage,

meta: emailMeta,

} = useField("email", emailValidation);

const {

value: phone,

errorMessage: phoneErrorMessage,

meta: phoneMeta,

} = useField("phone", phoneValidation);

const {

value: role,

errorMessage: roleErrorMessage,

meta: roleMeta,

} = useField("role", roleValidation);

const validateForm = () => {

if (

usernameMeta.valid &&

passwordMeta.valid &&

emailMeta.valid &&

phoneMeta.valid &&

roleMeta.valid

)

return true;

else {

toastError.value?.setupToast({

name: "Add User Error",

elapsedDuration: moment().startOf("second").fromNow(),

heading: "Add User Error",

text: "Please fill in the required fields",

delay: 5000,

});

toastError.value?.show();

}

};

const createUserPayload = () => {

const payload: NewUserDto = {

username: username.value,

password: password.value,

email: email.value,

phone: phone.value,

role: role.value,

};

return payload;

};

const addUser = async (payload: NewUserDto) => {

try {

const user = await usersStore.addUser(payload);

toastSuccess.value?.setupToast({

name: "Add Success",

elapsedDuration: moment().startOf("second").fromNow(),

heading: "Add User",

text: "Added the user successfully!",

delay: 3000,

});

toastSuccess.value?.show();

return user;

} catch (error: any) {

console.error(error);

toastError.value?.setupToast({

name: "Add User Error",

elapsedDuration: moment().startOf("second").fromNow(),

heading: "Add User Error",

text: "Failed to add the user. " + error?.message,

delay: 5000,

});

toastError.value?.show();

}

};

const routeRedirect = ref("");

const onAddAndNewClick = async () => {

if (validateForm()) {

await addUser(createUserPayload());

// select the form using the formRef

const form = formRef.value as HTMLFormElement;

// call reset

form?.reset();

}

};

const onAddAndView = async () => {

if (validateForm()) {

const user = await addUser(createUserPayload());

routeRedirect.value = `/users/${user?.id}`;

}

};

const onAddAndViewAll = async () => {

if (validateForm()) {

await addUser(createUserPayload());

routeRedirect.value = "/users";

}

};

const onClear = () => {

// select the form using the formRef

const form = formRef.value as HTMLFormElement;

// call reset

form?.reset();

};

const onHiddenBsToast = () => {

router.push(routeRedirect.value);

};

</script>

<style scoped>

.btn-hover:hover {

transform: translateY(-3px);

/\*box-shadow: var(--bs-card-box-shadow);\*/

box-shadow: 0 10px 20px rgba(0, 0, 0, 0.2);

transition: all 0.2s ease-in;

}

.btn-hover:hover::after {

transform: scaleX(1.4) scaleY(1.6);

opacity: 0;

}

</style>