Katedra za elektronsko poslovanje

ELAB **TODO**

SKRIPTA

Preporučujem **yarn,** posebno na Windows mašinama.

VSCode extenzije:

1. Vetur
2. sarah.drasner Vue VSCode Snippets

Korišćenje vue cli-a:

1. $ npm install -g @vue/cli

ili

$ yarn global add @vue/cli

*Ukoliko se prvi put instalira @vue/cli potreban je restart na Windows mašinama.*

1. $ vue create my-project-name

* Manually select features
* Navigiramo se do **Router** i **Vuex** i to uključite pritiskom na **SPACE**, zatim **ENTER**
* Y(es) za history mode
* ESLint + Prettier
* Lint on save
* In dedicated config files
* Save for future? N(o)
* Use Yarn

1. $ cd my-project-name
2. $ vue add bootstrap-vue

<https://bootstrap-vue.js.org/>

Vue komponentna implementacija Bootrsrap-a. Može i normalan Bootstrap da se uključi, ne mora ovaj plugin. Npr. može CDN da se zalepi u public/index.html ili da se uključe svi potrebni fajlovi. Dodavanje preko CLI-a ima prednosti što se tiče bundle size-a i reusability-a.

* Y(es) za babel polifil

**COMMIT-ujte promene**

1. $ yarn add axios vue-axios  
   **COMMIT-ujte promene**
2. Nakon ovih koraka otvorite iz VSCode-a folder u kome ste napravili projekat.
3. Uključiti VueAxios i axios i proslediti ih Vue.use(VueAxsios,axios) metodi u **main.js (todo>src>main.js)**

**main.js**

import "@babel/polyfill";

import "mutationobserver-shim";

import Vue from "vue";

import axios from 'axios';

import VueAxios from 'vue-axios';

import "./plugins/bootstrap-vue";

import App from "./App.vue";

import router from "./router";

import store from "./store";

Vue.config.productionTip = false;

Vue.use(VueAxios, axios);

**new** Vue({

  router,

  store,

  render: h => h(App)

}).$mount("#app");

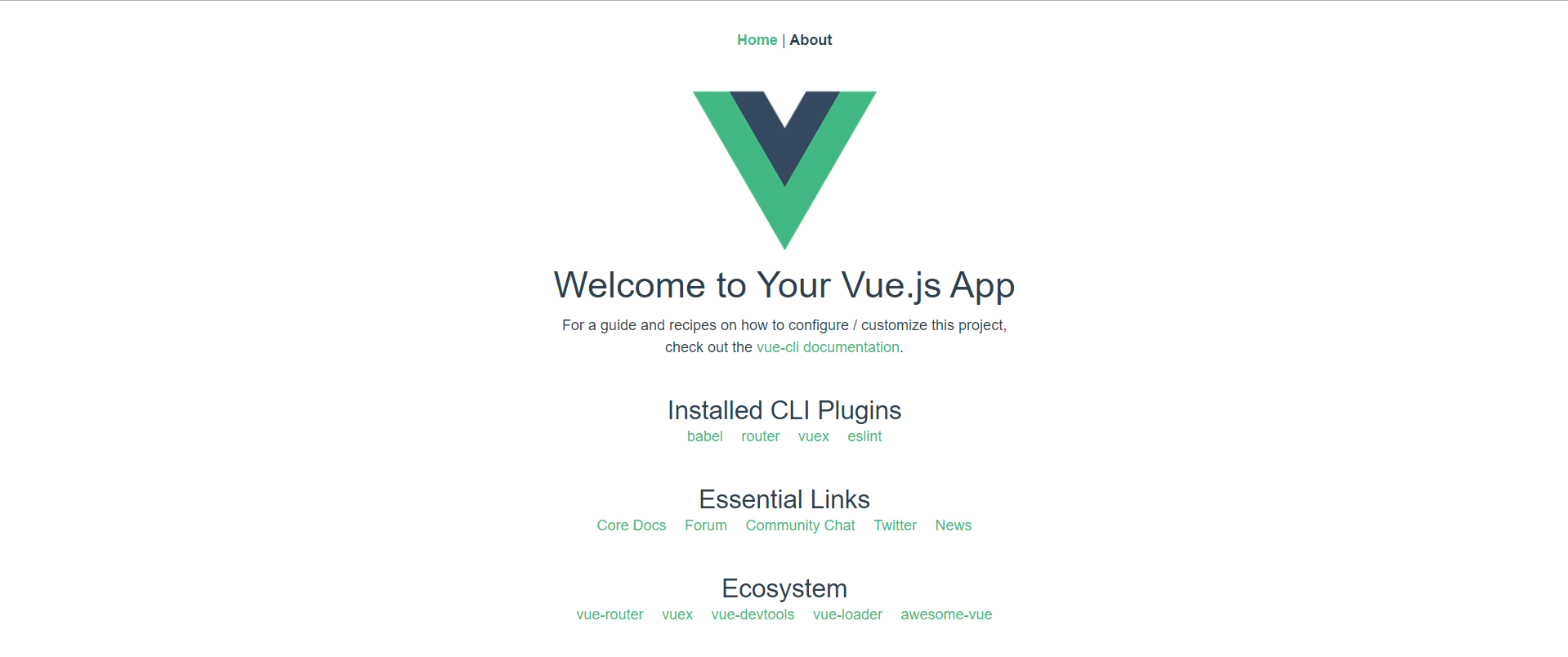
Da biste pokrenuli dev server:

$ yarn serve --port 8081

ili

$ npm run serve

da biste pokrenuli dev server, koji će se pokrenuti na portu 8081 (localhost:8081). Pogledajte sample aplikaciju.

Početnu aplikaciju možemo videti odlaskom na localhost:8081  


Ukoliko je sve proteklo kako treba, sada možete početi sa izradom aplikacije.

**store/index.js** - definišite prvo baseUrl do vašeg backend servera.

import Vue from "vue";

import Vuex from "vuex";

Vue.use(Vuex);

// Vuex je centralizovano skladiste state-a nase aplikacije, kao npr. Redux

// U ovom primeru imamo samo baseUrl, koji setujemo kako bismo iz cele aplikacije

// znali na kojoj adresi nam je backend server

export default **new** Vuex.Store({

  state: {

    baseUrl: 'http://localhost/todo'

  },

  mutations: {},

  actions: {},

  modules: {}

});

**App.vue** - obrišite sve, ostavite samo template tag sa router-view tagom

<template>

  <!--

Ovo je tipicina Vue komponenta.

Koristimo router za prikaz komponenti, tako da u ulaznom App.vue fajlu samo

definisemo router-view sto znaci da router ima kontrolu nad tim tagom

naravno ovde mozemo ukljuciti jos neke komponente, pre ili posle routera,

npr. ako zelimo da imamo navbar u okviru cele aplikacije, mozemo to uraditi ovde

tako sto bismo ucitali Navbar kao komponentu i renderovali bismo je pre router-view-a

  -->

  <router-view />

</template>

Kreirajte prazne komponente **Navbar.vue, List.vue, List.vue, ListItem.vue**

Kreirajte view **ListView.vue**

<template>

  <!-- views folder sluzi da se objedini nekoliko komponenti u celoviti prikaz -->

  <div>

    <!--3. Renderujemo komponentu -->

    <span>TEST 123</span>

    <Navbar />

    <List />

  </div>

</template>

<script>

// Redosled ukljucivanja komponentni je sledeci:

// 1. importujemo komponentu

import Navbar from "@/components/Navbar.vue";

import List from "@/components/List.vue";

export default {

  // 2. registrujemo komponentu

  components: {

    Navbar,

    List

  }

};

</script>

<style scoped>

</style>

U **router/index.js** fajlu menjamo rute na sledeći način:

Brišemo **about** view, a umesto **home** view-a uključujemo **ListView**, ovako:

import Vue from "vue";

import VueRouter from "vue-router";

import ListView from "../views/ListView";

Vue.use(VueRouter);

//Ovde definisemo nase rute kao objekte, path je putanja u browseru, npr. /items, /posts

//component je komponenta koja se prikazuje kada se ucita ta ruta.

const routes = [{

  path: "/",

  name: "home",

  component: ListView

}];

// Instancira se VueRouter, standardan nacin za uvezivanje routera.

// Cli sam odradi ovo za nas

const router = **new** VueRouter({

  mode: "history",

  base: process.env.BASE\_URL,

  routes

});

export default router;

Ukoliko je sve prošlo kako treba, pogledajte output u browseru:



Napravite **Navbar.vue** komponentu:

<template>

  <b-navbar type="dark" variant="info">

    <b-navbar-brand href="#">

      <h5>

        ELAB

        <b-badge variant="dark">TODO</b-badge>

      </h5>

    </b-navbar-brand>

  </b-navbar>

</template>

i obrišite TEST 123 iz **ListView.vue,** pogledajte browser:



Obrišite sledeće view fajlove:

* **About.vue**
* **Home.vue**

Obrišite sledeće componente:

* **HelloWorld.vue**

Kreirajte ListItem.vue komponentu

<template>

  <div>

    <!-- @click odnosno v-on:click je Vue direktiva koja se okida kada se klike misem

      na element, addEventLister je analogni koncept u js-u.

      mi kada kliknemo pozivamo metodu iz methods objekta u script sekciji i prosledjujemo joj item,

      itemm koji dobijamo iz v-for direktive prolaskom kroz iterabilnu listu

    v-bind:key ili :key sluzi samo da jedinstveno identifikuje element u v-for-u

    :variant je primary, success, danger... bootstrap helper klasa

    -->

    <!-- v-for dirketiva sluzi da renderuje nesto sto je itreabilno bez kopiranja koda, npr. -->

    <!-- Pokazati ovaj primer kako radi: <div v-for="item in 5" :key="item.id">{{ item }}</div> -->

    <b-list-group-item

      style="cursor: pointer;"

      @click="changeStatus(item)"

      v-for="(item, id) in list.itemi"

      :key="id"

      id="list-item"

      class="d-flex justify-content-between align-items-center"

      :variant="(item.status == '0')?'info':'success'"

    >

      <!-- :syle ili v-bind:syle, sluzi da se u prop ubaci neki dinamicki content,

      npr. :style = '"color": stildugmeta', gde bi stilldugmeta bio neki prop ili data objekat-->

      <div

        :style="(item.status == '0')?'textDecoration: none;':'textDecoration: line-through'"

      >{{id + 1}}. {{item.content}}</div>

      <div>

        <!-- click.stop sluzi da spreci Event bubbling -->

        <b-badge @click.stop="deleteItem(item)" href="#" variant="danger" pill>X</b-badge>

      </div>

    </b-list-group-item>

    <b-input-group class="mt-3" style="z-index: 0" prepend="New item">

      <!-- v-model direktiva sluzi za 2way data binding, sve sto se unese u inputu, automatski

      je uneto u newListTitle data objektu, vazi i obrnuto-->

      <b-form-input v-model="newItem"></b-form-input>

      <b-input-group-append>

        <b-button @click="addItem(list)" size="sm" text="Add" variant="success">Add</b-button>

      </b-input-group-append>

    </b-input-group>

  </div>

</template>

<script>

export default {

  // Props, saljemo property-je iz roditelja u dete, ovo je dete komponenta

  props: {

    list: {

      type: Object

    }

  },

  // Data funkcija, sadrzi nase reaktivne objekte, svaka promena nad ovim objektima se automatski

  // reflektuje i u template

  data() {

    return {

      newItem: ""

    };

  },

  methods: {

    async deleteItem(item) {

      await this.$http.delete(`${this.$store.state.baseUrl}/itemi/${item.id}`);

      // Signal roditelju da se desio neki event

      this.$emit("update-lists");

      // VAZNO primetiti da se iz metoda mora sve pristupati preko this, jer Vue sve binduje na this

    },

    async changeStatus(item) {

      const resBody = {

        status: item.status == "0" ? "1" : "0",

        content: item.content,

        list\_id: item.list\_id

      };

      await this.$http.put(

        `${this.$store.state.baseUrl}/itemi/${item.id}`,

        resBody

      );

      // Signal roditelju da se desio neki event

      this.$emit("update-lists");

    },

    async addItem(list) {

      // Obicna validacija

      if (this.newItem.length < 1) return;

      const resBody = { content: this.newItem, status: 0, list\_id: list.id };

      await this.$http.post(`${this.$store.state.baseUrl}/itemi`, resBody);

      this.newItem = "";

      // Signal roditelju da se desio neki event

      this.$emit("update-lists");

    }

  }

};

</script>

<style scoped>

</style>

Kreirajte **List.vue** komponentu

<template>

  <div>

    <b-container>

      <b-row>

        <!-- v-for dirketiva sluzi da renderuje nesto sto je itreabilno bez kopiranja koda, npr. -->

        <!-- Pokazati ovaj primer kako radi: <div v-for="item in 5" :key="item.id">{{ item }}</div> -->

        <!-- :key sluzi da jedinstveno indentifikuje item u listi -->

        <b-col xs="12" sm="6" md="4" v-for="list in lists" :key="list.id">

          <div class="controls">

            <!-- Kada se klikne na ovaj badge koji sluzi za brisanje, poziva  se metoda delteList, .prevent sluzi da

            ne bi se izvrsio href-->

            <b-badge @click.prevent="deleteList(list)" href="#" variant="danger" pill>X</b-badge>

          </div>

          <b-card-group deck>

            <b-card class="mt-5 mx-1" bg-variant="light" no-body :header="list.title">

              <b-list-group class="mt-3" flush>

                <!-- Ovde prosledjujemo listu itema ListItem komponenti koja u sebi

                renderuje stavke u odnosu na listu i osluskujemo na event

                update-lists koji ako se desi pozivamo updateUi metodu-->

                <ListItem @update-lists="updateUi" :list="list" />

              </b-list-group>

            </b-card>

          </b-card-group>

        </b-col>

        <b-col xs="12" sm="6" md="4">

          <b-card-group deck>

            <b-card class="mt-5 mx-1" no-body>

              <b-input-group prepend="New list">

                <!-- v-model direktiva sluzi za 2way data binding, sve sto se unese u inputu, automatski

                je uneto u newListTitle data objektu, vazi i obrnuto-->

                <b-form-input v-model="newListTitle"></b-form-input>

                <b-input-group-append>

                  <b-button @click="addList" size="sm" text="Add" variant="success">Add</b-button>

                </b-input-group-append>

              </b-input-group>

            </b-card>

          </b-card-group>

        </b-col>

      </b-row>

    </b-container>

  </div>

</template>

<script>

import ListItem from "@/components/ListItem.vue";

export default {

  components: {

    ListItem

  },

  methods: {

    // this.$store.state.baseUrl uzimanje URL-a do backa

    async updateUi() {

      const response = await this.$http.get(

        `${this.$store.state.baseUrl}/liste.json`

      );

      this.lists = response.data;

    },

    async deleteList(list) {

      await this.$http.delete(`${this.$store.state.baseUrl}/liste/${list.id}`);

      this.updateUi();

    },

    async addList() {

      if (this.newListTitle.length < 1) return;

      await this.$http.post(`${this.$store.state.baseUrl}/liste`, {

        title: this.newListTitle

      });

      this.newListTitle = "";

      this.updateUi();

    }

  },

  async mounted() {

    this.updateUi();

  },

  data() {

    return {

      lists: [],

      newListTitle: ""

    };

  }

};

</script>

<style scoped>

.controls {

  display: flex;

  justify-content: space-between;

  align-items: flex-start;

  position: absolute;

  width: 100%;

  top: 40px;

  left: 0;

  z-index: 1;

}

@media (max-width: 576px) {

  .controls {

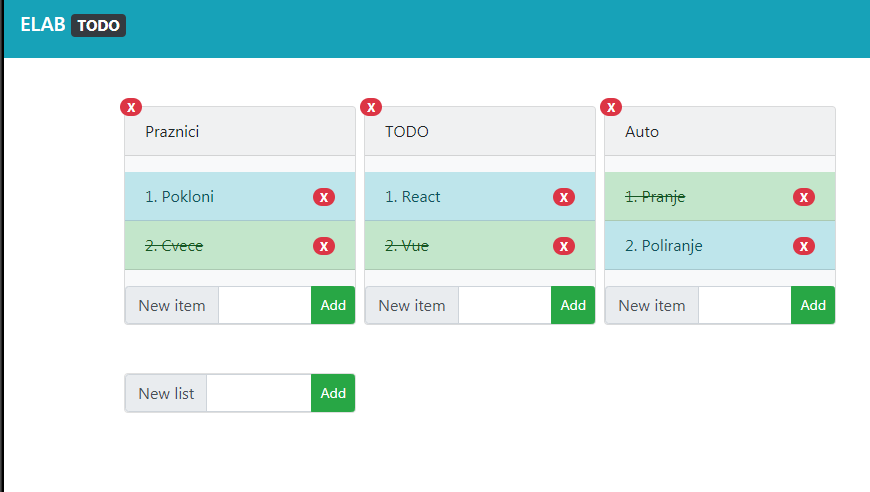
    padding: 0 10px;

  }

}

</style>

Aplikacija sada izgleda ovako, ukoliko imate neke stavke u bazi.



Ostalo je još uključiti edit opciju, da bi mogli da menjamo sadržaj već kreiranih listi i stavki.

Kreirajte **Dropdown.vue** komponentu:

<template>

  <!-- click.stop sluzi da spreci Event bubbling, jer je ovaj element ugnjezden u list item,

  bez click.stop pritiskom na edit dugme, promenio bi se status list itema-->

  <span @click.stop>

    <!-- @click ili v-on:click setuje enabled na true, time dobijamo renderovan div ispod koji na sebi ima v-if dirketivu,

    koja ukoliko dobije true vrednost  redneruje komponentu, a ako dobije false onda ne. Postoji v-else dirketiva takodje

    class mt-2 je helper bootstrap klasa da doda margin-top: 2,

    variant je primary, danger, success, info, bootstrap opet-->

    <b-badge @click="enabled = true" class="mt-2" href="#" variant="info" pill>Edit</b-badge>

    <!-- :class ili v-bind:class, sluzi da se u prop ubaci neki dinamicki content,

    npr. :style = '"color": stildugmeta', gde bi stilldugmeta bio neki prop ili data objekat

    v-if uslovno renderovanje-->

    <div :class="type == 'item'? 'item-fix': '' " id="form" v-if="enabled">

      <b-form-input

        autofocus

        @focusout="enabled = false"

        class="mr-1"

        v-model="content"

        size="sm"

        :placeholder="label"

      ></b-form-input>

      <!-- @mousedown.prevent sluzi da spreci promenu fokusa na dugme,

      jer bi se onda aktivirala @focusout metoda i zatvorila modal sa input poljem-->

      <b-button @mousedown.prevent @click="changeTitle" variant="info" size="sm">✔</b-button>

    </div>

  </span>

</template>

<script>

export default {

  // Props, saljemo property-je iz roditelja u dete, ovo je dete komponenta

  props: {

    label: {

      type: String

    },

    type: {

      type: String

    }

  },

  methods: {

    changeTitle() {

      // Obicna validacija

      if (this.content.length < 1) return;

      // Signal roditelju da se desio neki event i prosledjivanje odredjenog payload-a

      // odnosno nekih podataka roditelju

      // VAZNO primetiti da se iz metoda mora sve pristupati preko this, jer Vue sve binduje na this

      this.$emit("changeItem", this.content);

      this.content = "";

      this.enabled = false;

    }

  },

  data() {

    return {

      enabled: false,

      content: ""

    };

  }

};

</script>

<style scoped>

/\* scoped atribut znaci da se ovaj stil nece 'prelivati'

na druge komponente koje mozda imaju istu klasu ili id \*/

#form {

  width: 150px;

  background-color: rgb(240, 240, 240);

  position: absolute;

  z-index: 2;

  border-radius: 5px;

  padding: 5px;

  display: flex;

}

.item-fix {

  left: 50%;

}

@media (max-width: 576px) {

  .item-fix {

    left: 70%;

  }

}

@media (max-width: 480px) {

  .item-fix {

    left: 50%;

  }

}

</style>

I izmenite ostale komponente tako da uključuju **Dropdown.vue** komponentu:

**List.vue**

<template>

  <div>

    <b-container>

      <b-row>

        <!-- v-for dirketiva sluzi da renderuje nesto sto je itreabilno bez kopiranja koda, npr. -->

        <!-- Pokazati ovaj primer kako radi: <div v-for="item in 5" :key="item.id">{{ item }}</div> -->

        <!-- :key sluzi da jedinstveno indentifikuje item u listi -->

        <b-col xs="12" sm="6" md="4" v-for="list in lists" :key="list.id">

          <div class="controls">

            <!-- Edit dugme za listu

               Ovde renderujemo ucitanu komponentu, prosledjujemo joj prop label i postavljamo eventListener

            koji se $emituje iz te komponente i pozivamo metodu kad se to desi, ...arguments je payload koji nam salje

            $emit, a list je argument iz v-fora-->

            <Dropdown label="Title" @changeItem="changeTitle(list, ...arguments)" />

            <!-- Kada se klikne na ovaj badge koji sluzi za brisanje, poziva  se metoda delteList, .prevent sluzi da

            ne bi se izvrsio href-->

            <b-badge @click.prevent="deleteList(list)" href="#" variant="danger" pill>X</b-badge>

          </div>

          <b-card-group deck>

            <b-card class="mt-5 mx-1" bg-variant="light" no-body :header="list.title">

              <b-list-group class="mt-3" flush>

                <!-- Ovde prosledjujemo listu itema ListItem komponenti koja u sebi

                renderuje stavke u odnosu na listu i osluskujemo na event

                update-lists koji ako se desi pozivamo updateUi metodu-->

                <ListItem @update-lists="updateUi" :list="list" />

              </b-list-group>

            </b-card>

          </b-card-group>

        </b-col>

        <b-col xs="12" sm="6" md="4">

          <b-card-group deck>

            <b-card class="mt-5 mx-1" no-body>

              <b-input-group prepend="New list">

                <!-- v-model direktiva sluzi za 2way data binding, sve sto se unese u inputu, automatski

                je uneto u newListTitle data objektu, vazi i obrnuto-->

                <b-form-input v-model="newListTitle"></b-form-input>

                <b-input-group-append>

                  <b-button @click="addList" size="sm" text="Add" variant="success">Add</b-button>

                </b-input-group-append>

              </b-input-group>

            </b-card>

          </b-card-group>

        </b-col>

      </b-row>

    </b-container>

  </div>

</template>

<script>

import ListItem from "@/components/ListItem.vue";

import Dropdown from "@/components/Dropdown.vue";

export default {

  components: {

    ListItem,

    Dropdown

  },

  methods: {

    // asinhrono menjamo naslov liste, potom updatujemo UI

    // analogno i za ostale requestove

    // this.$store.state.baseUrl uzimanje URL-a do backa

    async changeTitle(list, title) {

      await this.$http.put(`${this.$store.state.baseUrl}/liste/${list.id}`, {

        title

      });

      this.updateUi();

    },

    async updateUi() {

      const response = await this.$http.get(

        `${this.$store.state.baseUrl}/liste.json`

      );

      this.lists = response.data;

    },

    async deleteList(list) {

      await this.$http.delete(`${this.$store.state.baseUrl}/liste/${list.id}`);

      this.updateUi();

    },

    async addList() {

      if (this.newListTitle.length < 1) return;

      await this.$http.post(`${this.$store.state.baseUrl}/liste`, {

        title: this.newListTitle

      });

      this.newListTitle = "";

      this.updateUi();

    }

  },

  async mounted() {

    this.updateUi();

  },

  data() {

    return {

      lists: [],

      newListTitle: ""

    };

  }

};

</script>

<style scoped>

.controls {

  display: flex;

  justify-content: space-between;

  align-items: flex-start;

  position: absolute;

  width: 100%;

  top: 40px;

  left: 0;

  z-index: 1;

}

@media (max-width: 576px) {

  .controls {

    padding: 0 10px;

  }

}

</style>

**ListItem.vue**

<template>

  <div>

    <b-list-group-item

      style="cursor: pointer;"

      @click="changeStatus(item)"

      v-for="(item, id) in list.itemi"

      :key="id"

      id="list-item"

      class="d-flex justify-content-between align-items-center"

      :variant="(item.status == '0')?'info':'success'"

    >

      <div

        :style="(item.status == '0')?'textDecoration: none;':'textDecoration: line-through'"

      >{{id + 1}}. {{item.content}}</div>

      <div>

        <Dropdown

          type="item"

          id="change-item"

          @changeItem="changeItem(item, ...arguments)"

          label="Content"

        />

        <b-badge @click.stop="deleteItem(item)" href="#" variant="danger" pill>X</b-badge>

      </div>

    </b-list-group-item>

    <b-input-group class="mt-3" style="z-index: 0" prepend="New item">

      <b-form-input v-model="newItem"></b-form-input>

      <b-input-group-append>

        <b-button @click="addItem(list)" size="sm" text="Add" variant="success">Add</b-button>

      </b-input-group-append>

    </b-input-group>

  </div>

</template>

<script>

import Dropdown from "@/components/Dropdown.vue";

export default {

  components: {

    Dropdown

  },

  props: {

    list: {

      type: Object

    }

  },

  data() {

    return {

      newItem: ""

    };

  },

  methods: {

    async changeItem(item, content) {

      const reqBody = {

        content: content,

        status: item.status,

        list\_id: item.list\_id

      };

      await this.$http.put(

        `${this.$store.state.baseUrl}/itemi/${item.id}`,

        reqBody

      );

      this.$emit("update-lists");

    },

    async deleteItem(item) {

      await this.$http.delete(`${this.$store.state.baseUrl}/itemi/${item.id}`);

      this.$emit("update-lists");

    },

    async changeStatus(item) {

      const resBody = {

        status: item.status == "0" ? "1" : "0",

        content: item.content,

        list\_id: item.list\_id

      };

      await this.$http.put(

        `${this.$store.state.baseUrl}/itemi/${item.id}`,

        resBody

      );

      this.$emit("update-lists");

    },

    async addItem(list) {

      if (this.newItem.length < 1) return;

      const resBody = { content: this.newItem, status: 0, list\_id: list.id };

      await this.$http.post(`${this.$store.state.baseUrl}/itemi`, resBody);

      this.newItem = "";

      this.$emit("update-lists");

    }

  }

};

</script>

<style scoped>

</style>

Sada aplikacija ima opciju izmene sadržaja i time ste završili izradu ove aplikacije.

Ceo projekat možete pogledati na sledećem linku: <https://github.com/NikolaStojicic/ELAB-TODO-VUE>

