Tartu Ülikool
Arvutiteaduse instituut
Informaatika õppekava

Ingo Ignašov

Veebimaterjalide ja -saidi loomine Vue.js raamistiku jõudlusprobleemidest

Bakalaureusetöö (9 EAP)

Juhendaja: Helle Hein

Tartu 2022

Veebimaterjalide ja -saidi loomine Vue.js raamistiku jõudlusprobleemidest

Lühikokkuvõte:

Bakalaureusetöö eesmärk on luua Vue.js raamistiku jõudlusprobleeme kirjeldavad materjalid ning nende kuvamiseks luua Vue.js baasil veebisait. Töö sisaldab kasutatud mõistete definitsioone, taustainfo kirjeldust, loodava veebisaidi nõudeid, kasutatud tehnoloogiate ja keelte kirjeldust ning tehtud töö analüüsi.

Võtmesõnad:

Veebisait, veebileht, Vue.js, veebi jõudlus, veebirakendus

CERCS: P175 Informaatika

Creating materials and a website about Vue.js framework's performance problems

Abstract:

The purpose of the bachelor's thesis is to create materials describing Vue.js frameworks performance issues and to create a Vue.js-based website to display them. The work includes definitions of the terms used, a description of the background information, the requirements of the website to be created, a description of the technologies and languages used, and an analysis of the work done.

Keywords:

Website, web page, Vue.js, web performance, web application

CERCS: P175 Informatics

Sisukord

1. Mõisted ja terminid **5**

2. Taustainfo **6**

2.1. Veebi jõudlus 6

2.2. Veebi jõudluse tähtsus 6

2.3. Loodud materjal 7

2.3.1. Kuulajate ja arvutatud omaduste liigne kasutamine 7

2.3.2. Liigne komponentide ümber renderdamine ja uuesti paigaldamine 7

2.3.3. Probleemid andmepäringutega 9

2.3.4. Vuex ja Object.Freeze() 10

2.3.5. Funktsionaalsed komponendid loendi renderdamiseks (v-for) 12

3. Veebisaidi nõuded **14**

3.1. Funktsionaalsed nõuded 14

3.2. Mittefunktsionaalsed nõuded 15

4. Kasutatud tehnoloogiad ja keeled **16**

4.1. Vue.js 16

4.1.1. Deklaratiivne renderdamine 17

4.1.2. Reaktiivsus 17

4.1.3. Ühefailikomponendid 17

4.2. Vuetify 17

4.3. Atomaarse kujunduse metodoloogia 18

4.4. Üheleherakendus 19

4.4.1. Kliendipoolne marsruutimine 19

5. Veebisaidi analüüs **20**

5.1. Veebisaidi päis 20

5.2. Veebisaidi jalus 21

5.3. Veebisaidi koduleht 22

5.4. Veebisaidi infoleht 23

5.5. Veebisaidi juhiste leht 23

5.6. Veebisaidi kasulike linkide leht 24

6. Kokkuvõte **26**

7. Viidatud kirjandus **27**

Sissejuhatus

Veebirakendused on viimastel aastakümnetel kiiresti populaarsust kogunud [1] ning need hõivavad infotehnoloogias ja ärimaailmas üha tähtsamat kohta. Veebirakendused koosnevad mitmest suuremast osast ning selle bakalaureusetöö raames uuritakse täpsemalt ühte veebirakenduse osa - veebirakenduse eessüsteemi.

Tänapäeval kasutatakse eessüsteemi arendamiseks tihti vastavaid raamistikke. Eessüsteemi raamistik on tööriist, mis määrab ära, kuidas eessüsteem üles ehitada, lihtsustab eessüsteemi arendamist ning sisaldab erinevaid väljatöötatud abivahendeid [2]. Üks populaarsematest eessüsteemi raamistikest on Vue.js [3].

On leitud, et üheks peamiseks teguriks, mis mõjutab veebirakenduse kasutajakogemust, on veebilehe kiirus ja laadimiseks kuluv aeg [4]. See tähendab, et on oluline veebirakenduse eessüsteem arendada võimalikult efektiivne ja hea jõudlusega. Käesoleva töö teema valikul peamiseks motivatsiooniks oli anda Vue.js arendajatele informatsiooni, mis aitavad muuta Vue.js raamistikul arendatud veebirakendused võimalikult efektiivseks.

Töö eesmärgiks on koostada inglise- ja eestikeelsed juhised Vue.js kasutamiseks jõudluse ja efektiivsuse aspektist. Lisaks luuakse töö käigus materjale illustreerivad näidisprogrammid ning arendatakse loodud materjalide kuvamiseks Vue.js põhjal veebisait.

Bakalaureusetöö on jagatud kuueks peatükiks. Teises peatükis antakse projekti kohta taustainfot. Kolmandas peatükis selgitatakse veebisaidi nõudeid. Neljas peatükk kirjeldab kasutatud tehnoloogiaid. Viiendas peatükis analüüsitakse loodavat veebisaiti. Kuuendas peatükis on töö kokkuvõte.

1. Mõisted ja terminid

Veebileht (ingl *web page*) on dokument, mille poole saab pöörduda veebi kaudu [5]**.**

Veebisait (ingl *website*) loogiliselt ühendatud veebilehtede kogum, mis on hallatav ühe üksusena [6]**.**

Veebi jõudlus (ingl *web performance*) on veebisaidi või rakenduse objektiivne mõõde ja tajutav kasutuskogemus [7].

Eessüsteem (ingl *front end*) on "kliendipoolne, inimkasutajat või kasutavat süsteemi tagaosaga liidestav osa" [8].

Tagasüsteem (ingl *back end*) on "kasutajale nähtamatu töötlev, talletav, käitlev jne põhiosa" [9]**.**

Üheleherakendus (ingl *single page application*) on "veebileht või -sait mis suhtleb kasutajaga, kirjutades praeguse veebilehe dünaamiliselt ümber veebiserverist pärinevate uute andmetega, selle asemel, et veebibrauseri kaudu laadida terveid uusi lehti" [10]**.**

Brauser (ingl *browser*) on "rakendusprogramm liikumiseks ja veebilehtede kuvamiseks. Brauserid on ka failivorminguid näitava failivaaturid, millega saab vaadata tekste, pilte jpm " [11]**.**

Vaatekiht (ingl *view layer*) on MVVM (*model, view, view-model)* struktuuri komponent, mis defineerib kliendile nähtava sisu asetuse, struktuuri ja välimuse [12].

Puu raputamine (ingl *tree shaking*) on JavaScripti raames tihti kasutatav termin kasutamata koodi eemaldamiseks [13].

Elutsükli haak (ingl *lifecycle hook*) on Vue komponendi initsialiseerimisprotsessi jooksul mingi etapi läbimisel aktiveeriv funktsioon [14].

Konversioonimäär (ingl *conversion rate*) on protsent, mis näitab kui suur osa veebilehe külastajatest täidab lehe omaniku mingi soovitud tegevuse [15].

2. Taustainfo

2.1. Veebi jõudlus

Veebi jõudlus on aeg, mis kulub, et veebileht alla laadida ning kasutajale kuvada. Veebi jõudlus avaldub lehe laadimise kiiruses, kui kiiresti leht reageerib kasutaja sisendile, lehe sujuvuses ning üldises kiires kasutajakogemuses [7]. Veebi jõudlust võib jaotada kaheks tüübiks:

- Uuenduse jõudlus - kui kiiresti rakendus kasutaja sisendile reageerib. Näiteks kui kaua järgmine leht üheleherakenduses [10] laetakse, kui kaua täitub tabel pärast filtreerimist, lehe vahetamist jne. See on sisuliselt jõudlus, mis on seotud veebilehel toimuvaga peale selle esmast laadimist ning mis juhtub veebilehel kasutaja sisendi tõttu.
- Lehe laadimise kiirus - kui kiiresti rakendus veebilehe laeb ja kui kiiresti ilmub ekraanile sisu või muutub leht interaktiivseks. Kui räägime lehe laadimisest, on palju mõõdikuid, näiteks esimese sisendi viivitus (ingl *First Input Delay,* FID) [16], esimene sisuline värvimine (ingl *First Contentful Paint,* FCP) [17] ja interaktiivseks kuluv aeg (ingl *Time To Interactive,* TTI) [18].

2.2. Veebi jõudluse tähtsus

Veebi jõudlusel on otsene mõju kasutajakogemusele.

Näiteks Pinterest ehitas ümber oma veebilehed, mis vähendas ooteaega 40%. Peale seda kasvas Pinteresti otsingumootori optimeerimise (ingl Search Engine Optimization) [19] liiklus 15% ning kasutajate registreerimiste arv 15% [20].

Veebitoodet omavate firmade põhieesmärk on maksimeerida kliendi kasutajakogemust. On teada, et veebilehe kiirusel on mõju kasutajakogemusele [21]. Aeglane veebileht võib jätta klientidele firmast aeglase, lohaka ning ebakompetentse mulje. Seevastu kiire veebileht jätab firmast professionaalse ja kvaliteetse mulje [22].

Google on väitnud, et veebileht peaks laadima vähemalt kahe sekundiga [23]. Lisaks on leitud, et kliendil tekib peale kahte või kolme sekundit tunne, et ta ei kontrolli enam olukorda.

Lisaks, kiirus on ka tähtis külastajate hoidmiseks veebilehel (ingl *viewer retention*). Neil Patel väitis, et 40 protsenti kasutajatest lahkuvad lehelt, kui selle laadimiseks kulub rohkem kui 3 sekundit [4]. Kiiret ja sujuvat lehte on mugav kasutada ja see hoiab kliendi tähelepanu ning paneb ta lehele tagasi tulema.

Kiirel veebilehel on ka positiivne mõju müüginumbritele ja konversioonimäärale. Michael Wiegand väitis, et “Saidil, mis laaditakse 1 sekundiga, on konversioonimäär kolm korda kõrgem kui saidil, mis laaditakse 5 sekundiga ” [24]. Lisaks väitis ta, et iga sekund, mis lehe laadimisele kulub, vähendab konversioonimäära 5% võrra [24].

Kokkuvõttes võib väita, et veebi jõudlus on väga tähtis nii lõppkasutajatele kui ka äridele.

2.3. Loodud materjal

Käesolevas bakalaureusetöös loodi veebimaterjalid, mis koosnevad viiest peatükist.

Teemade valimisel lähtus autor isiklikust kogemustest ja internetis tehtud uuringust. Teemad on välja valitud selliselt, et anda veebisaidi kasutajale ülevaade raamistiku Vue.js vähem tuntud ja dokumenteeritud jõudlusprobleemidest. Kõik veebilehel käsitletud teemad ja vead on autor ise läbi kogenud, suur osa kirjutatust tugineb sellele.

2.3.1. Kuulajate ja arvutatud omaduste liigne kasutamine

See peatükk käsitleb Vue.js kuulajate ja arvutatud omaduste liigsel kasutamisel tekkivaid jõudlusprobleeme.

Arvutatud omadused (ingl *computed properties)* on Vue.js-is defineeritavad muutujad, mille abil on võimalik arvutada ja kuvada väärtusi mingi muu väärtuse või väärtuste hulga põhjal [25].

Kuulaja (ingl *watcher*) on arvutatud väärtus, mis jälgib mingit muutujat, mille muutumise korral käivitub kuulaja küljes defineeritud funktsioon [26].

Lühidalt, selles peatükis on kirjeldatud, kuidas kuulaja ja arvutatud omaduse funktsioonid on populaarsed viisid, kus arendajad oma koodi käivitavad. Tuleb arvestada sellega, et kui nendes funktsioonides kasutada kalleid arvutusi, siis võib see eessüsteemi jõudlust aeglustada.

2.3.2. Liigne komponentide ümber renderdamine ja uuesti paigaldamine

Teises peatükis kirjeldatakse, kuidas võib Vue.js komponentide liigne ümber renderdamine ning komponentide uuesti paigaldamine põhjustada jõudluse vähenemise.

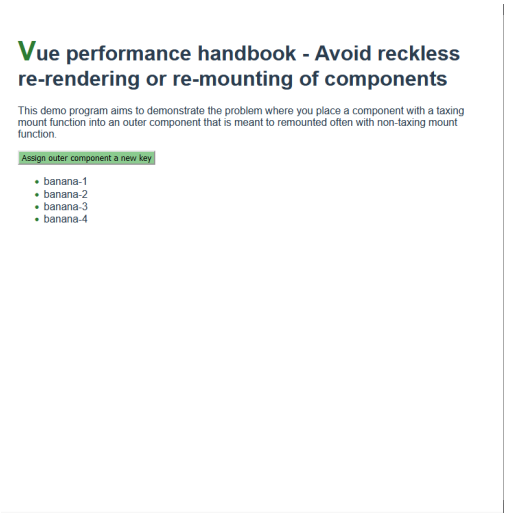
Vue.js on reaktiivne. See tähendab et, iga kord kui muutub mingi väärtus, mida kuvatakse mallis, siis komponent renderdatakse ümber [27]. Samal ajal aktiveerub Vue elutsükli haak *updated*. Kui *updated* funktsioonis on liiga kallis arvutus ning see haak aktiveerub liiga tihti, on tõenäoline, et see aeglustab veebilehe jõudlust.

Sarnane probleem on ka elutsükli haagiga *mounted* või *beforeMount*, mis aktiveeruvad komponendi loomisel.

Selles peatükis lõi autor ka komponendi uuesti paigaldamise probleemi illustreeriva näidisprogrammi, mis on kättesaadav aadressil: https://codesandbox.io/s/re-rendering-and-remounting-1xzztw.

See programm (vt Joonis 1) sisaldab lühikest kirjeldust ning nuppu, millel klikkides määratakse komponendile *outer-component.vue* (vt Joonis 2) uus võti, mille tõttu Vue

eemaldab ning paigaldab selle komponendi uuesti malli ning läbitakse komponendis *inner-component.vue* (vt Joonis 3) ajakulukas koodijupp.



Joonis 1. Peatükk 2 näidisprogrammi kuvatõmmis.

```vue
<template>
  <div>
    <InnerComponent />
    <ul>
      <li v-for="(banana, index) in bananaBox" :key="index">
        {{ banana.name }}
      </li>
    </ul>
  </div>
</template>

<script>
import InnerComponent from "./inner-component.vue";

const db = [
  { id: 1, name: "banana-1" },
  { id: 2, name: "banana-2" },
  { id: 3, name: "banana-3" },
  { id: 4, name: "banana-4" },
];

export default {
  name: "OuterComponent",
  components: {
    InnerComponent,
  },
  data() {
    return {
      bananaBox: [],
    };
  },
  async beforeMount() {
    //placeholder for a often reloaded code in a real application
    //for example, loading data to table, after filtering by some filter
    console.log("loading data for outer-component");
    this.bananaBox = await this.getBananas();
  },
  mounted() {
    console.log("re-mounted outer component!");
  },
  methods: {
    getBananas() {
      return new Promise((resolve, reject) => {
        setTimeout(() => {
          resolve(db);
        }, 2000);
      });
    },
  },
};
</script>
```

Joonis 2. Komponendi *outer-component.vue* lähtekood.

```vue
<template>
  <div></div>
</template>

<script>
export default {
  name: "InnerComponent",
  data() {
    return {
      exampleList: [],
    };
  },
  beforeMount() {
    //placeholder for a taxing code that is meant to only be done and
    //make with the idea that this component will not be remounted often
    this.exampleList = [...Array(30000000).keys()];
    this.exampleList.reverse();
    console.log(
      "re-mounted inner component with taxing code in beforeMount hook!"
    );
  },
};
</script>
```

Joonis 3. Komponendi *inner-component.vue* lähtekood.

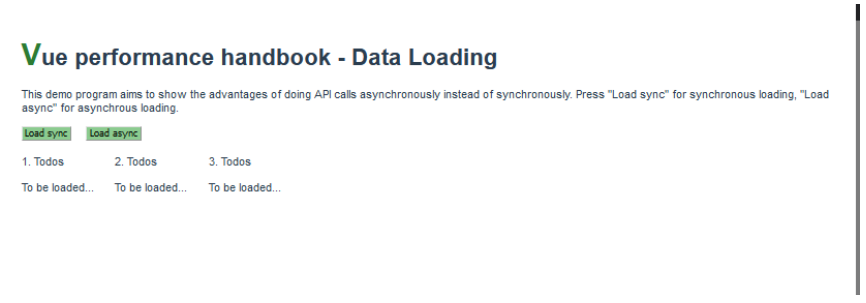
2.3.3. Probleemid andmepäringutega

Kolmas peatükk käsitleb probleeme päringute sünkroonsel jooksutamisel. Selles peatükis kirjeldatakse, kuidas saab Vue.js rakenduses andmepäringuid teha ning näidatakse, kuidas teha asünkroonseid päringuid kasutades *Promise* [28] objekti *Promise.all()* [29] meetodit.

Lisaks näidatakse, miks ja millal eelistada asünkroonselt tehtud päringuid sünkroonsetele.

Autori poolt loodud asünkroonsete päringute eeliseid demonstreeriv näidisprogramm asub aadressil: https://codesandbox.io/s/vueperformancehandbook-dataloading-kw8wbz.

Programmi lehel (vt Joonis 4) kuvatakse lühike programmi kirjeldus ja kaks nuppu: “Load sync” ja “Load async”, mida vajutades demonstreeritakse andmete sünkroonset või asünkroonset laadimist. Joonisel 5 kuvatakse demoprogrammis päringute illustreerimiseks kasutatud meetodid.



Joonis 4. Peatükk 2 näidisprogrammi kuvatõmmis.

```
},
methods: {
  async reloadAsync() {
    this.intializeTodos();

    const [toDos1, toDos2, toDos3] = await Promise.all([
      this.getToDos(1),
      this.getToDos(2),
      this.getToDos(3),
    ]);

    this.toDos1 = toDos1;
    this.toDos2 = toDos2;
    this.toDos3 = toDos3;
  },
  async reloadSync() {
    this.intializeTodos();

    this.toDos1 = await this.getToDos(1);
    this.toDos2 = await this.getToDos(2);
    this.toDos3 = await this.getToDos(3);
  },
  intializeTodos() {
    this.toDos1 = [];
    this.toDos2 = [];
    this.toDos3 = [];
  },
  getToDos(userId) {
    return new Promise((resolve, reject) => {
      setTimeout(() => {
        resolve(this.getToDo(userId));
      }, 2000);
    });
  },
  getToDo(userId) {
    return db[userId];
  },
 },
};
```

Joonis 5. Peatükk 3 näidisprogrammi päringute meetodid.

2.3.4. Vuex ja Object.Freeze()

Selles peatükis on toodud välja, kuidas Vue.js reaktiivsus võib *vuex* kasutades suuri jõudlusprobleeme tekitada.

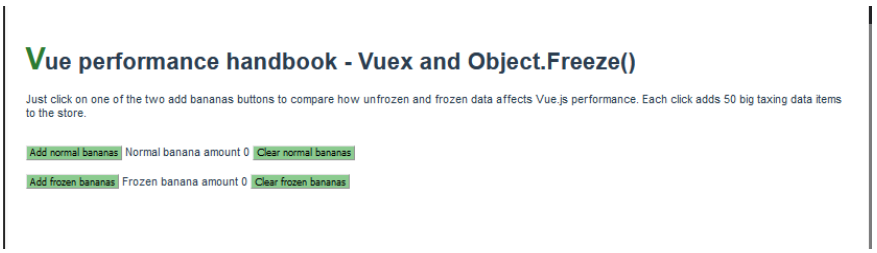
Vuex on Vue.js-i rakenduste olekuhaldusmuster ning -teek. See toimib komponentide tsentraliseeritud laona, mille reeglid tagavad, et olekut saab muuta ainult ettenähtud viisil [30].

Vaikimisi käsitleb Vue.js *vuex* laos hoitud objekte ja massiive reaktiivsetena, mille tõttu võivad need rakenduses palju mälu hõivata. See võib aga jõudlusele halvasti mõjuda. Selle parandamiseks on olemas JavaScripti meetod *Object.Freeze()* [31]. *Object.Freeze()* muudab objekti või massiivi mittemureeritavaks. Vue mittemuteeritavaid objekte reaktiivselt ei käsitle, seega kulub palju vähem mälu.

Probleemi ja lahendust illustreeriv näidisprogramm on leitav aadressil: https://codesandbox.io/s/object-freeze-cdvmxb.

Selles programmis (vt Joonis 6) on kuvatud lihtne kasutajaliides, mis sisaldab programmi eesmärgi kirjeldust ning nuppe, mille abil saab lisada mällu tavalisi või *Object.Freeze()*'ga töödeldud mahukaid objekte (vt Joonis 7 ja Joonis 8). Lisaks kuvatakse nende arvu mälus ning mälu saab ka tühjendada.

Programmi kasutades on märgata, et *Object.Freeze()*'ga töödeldud objektide lisamine toimib palju kiiremini ja efektiivsemalt.



Joonis 6. Peatükk 4 näidisprogrammi kuvatõmmis.

```
object-freeze.vue
export default {
  name: "ObjectFreeze",
  computed: {
    normalBananaCount() {
      return this.$store.state.normalBananas.length;
    },
    freezeBananaCount() {
      return this.$store.state.freezeBananas.length;
    },
  },
  methods: {
    addMoreNormalBananas() {
      this.$store.commit(
        "addMoreNormalBananas",
        Array(50)
          .fill()
          .map(() => this.createMemoryHoggingItem(3))
      );
    },
    clearNormalBananas() {
      this.$store.commit("clearNormalBananas");
    },
    addMoreFreezeBananas() {
      this.$store.commit(
        "addMoreFreezeBananas",
        Array(50)
          .fill()
          .map(() => this.createMemoryHoggingItem(3))
      );
    },
    clearFreezeBananas() {
      this.$store.commit("clearFreezeBananas");
    },

    //idea came from https://github.com/Kasheftin
    //file https://github.com/Kasheftin/vue-rerendering-optimization/blob/master/src/views/Example2.vue
    createMemoryHoggingItem(it, childrenCount = 10) {
      return {
        hashes: Array(10).fill("Lorem ipsum dolor sit amet."),
        children: it
          ? Array(childrenCount)
              .fill()
              .map(() => this.createMemoryHoggingItem(it - 1, childrenCount))
          : [],
      };
    },
  },
};
</script>
```

Joonis 7. Peatükk 4 näidisprogrammi põhikomponendi lähtekood.

```
store.js
import Vue from "vue";
import Vuex from "vuex";

Vue.use(Vuex);

const state = {
  freezeBananas: [],
  normalBananas: []
};

const mutations = {
  clearFreezeBananas(state) {
    state.freezeBananas = [].map(Object.freeze);
  },
  addMoreFreezeBananas(state, bananas) {
    state.freezeBananas = [
      ...state.freezeBananas,
      ...bananas.map(Object.freeze)
    ];
  },
  clearNormalBananas(state) {
    state.normalBananas = [];
  },
  addMoreNormalBananas(state, bananas) {
    state.normalBananas = [...state.normalBananas, ...bananas];
  }
};

export default new Vuex.Store({
  state,
  mutations
});
```

Joonis 8. Peatükk 4 näidisprogrammi objektide mällu lisamise meetodid.

2.3.5. Funktsionaalsed komponendid loendi renderdamiseks (v-for)

Viies peatükk käsitleb funktsionaalseid komponente ning nende kasulikkust suurte loendite renderdamisel.

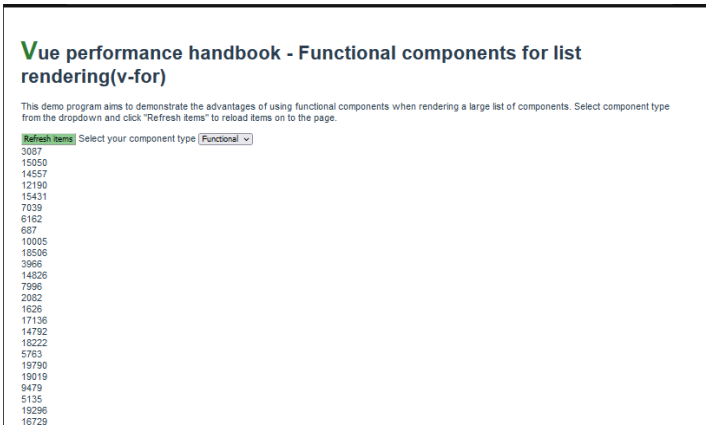
Funktsionaalne komponent (ingl *functional component)* on Vue.js komponent, millel pole olekut [32]. See tähendab et funktsionaalseks määratud komponendis ei saa kasutada *this* võtmesõna ning Vue.js käsitleb seda komponenti mittereaktiivsena.

Vue.js'is on suurte loendite renderdamise jaoks direktiiv *v-for* [33]. See on direktiiv, mille abil saab andmemassiivi läbi käia ning iga elemendi kohta luua komponendi.

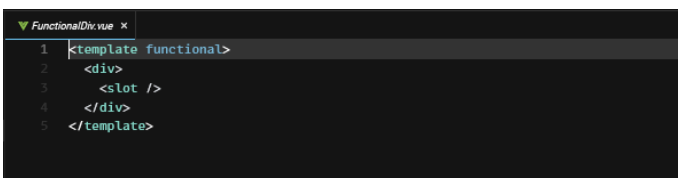
Funktsionaalset komponenti on soovitatav kasutada, kui on vaja renderdada palju komponente, näiteks kasutades *v-for*'i. Funktsionaalne komponent teeb sel juhul eessüsteemi jõudluse kiiremaks, kuna funktsionaalne komponent võtab mittereaktiivsuse ja oleku mitte omamise tõttu mälus vähem ruumi.

Funktsionaalsete komponentide kasutamist loendi renderdamisel võrreldes olekuga komponentidega demonstreerib autori poolt loodud näidisprogramm: https://codesandbox.io/s/functional-component-vs-non-dlyd3v.

Selles programmis (vt Joonis 9) kuvatakse lihtne veebileht, mis sisaldab programmi kirjeldust ning nuppu ja rippmenüüd. Rippmenüüst saab valida funktsionaalse (vt Joonis 10) ja mittefunktsionaalse (vt Joonis 11) komponendi vahel ning nuppu vajutades genereeritakse (vt Joonis 12) 10000 valitud tüüpi komponenti. Mõlemat varianti proovides võib väita, et funktsionaalsete komponentide renderdamine on kiirem.



Joonis 9. Peatükk 5 näidisprogrammi kuvatõmmis.



Joonis 10. Peatükk 5 näidisprogrammi funktsionaalse komponendi lähtekood.

WithStateDiv.vue

```vue
<template>
  <div>
    <slot />
  </div>
</template>
```

Joonis 11. Peatükk 5 näidisprogrammi mitte-funktsionaalse komponendi lähtekood.

```
data() {
  return {
    exampleItems: [],
    componentName: "FunctionalDiv",
  };
},

methods: {
  addItems() {
    this.exampleItems = [];
    while (this.exampleItems.length < 10000) {
      var r = Math.floor(Math.random() * 20000) + 1;
      if (this.exampleItems.indexOf(r) === -1) this.exampleItems.push(r);
    }
  },
},
```

Joonis 12. Peatükk 5 näidisprogrammi meetod *addItems()* komponentide genereerimiseks.

3. Veebisaidi nõuded

Käesolevas peatükis esitatakse veebisaidi nõuded. Kuna autor on selle veebilehe tellija, siis nõuete loomisel lähtuti peamiselt autori enda sisendist. Autori arvamus tugines nii enda subjektiivsetele eelistustele ja üldistele levinud veebidisaini parimatele tavadele [34]–[36].

3.1. Funktsionaalsed nõuded

Järgnevalt on toodud välja toodud funktsionaalsed nõuded. “Funktsionaalsed nõuded vastavad küsimusele "Mida peab tarkvara tegema?" (näiteks, süsteem

peab võimaldama kauba tellimist).”[37, p. 16]

Üldised nõuded:

1. Mitteeksisteeriva aadressi sisestamise puhul navigeeritakse kasutaja kodulehele.
2. Veebisait peab toimima üheleherakendusena; lehtede vahel navigeerides ei tohi brauser uut lehte laadida.
3. Veebisaidil peab olema päis, mis püsib alati lehe ülaosas.
4. Veebisaidil peab olema jalus, kus asub autori nimi ja kontaktinfo.
5. Kasutaja saab lehe päise pealkirjale vajutades navigeerida kodulehele.
6. Kasutaja saab lehe päises asuvale “HOME” lingile vajutades navigeerida kodulehele.
7. Kasutaja saab lehe päises asuvale “ABOUT” lingile vajutades navigeerida infolehele.
8. Kasutaja saab lehe päises asuvale “PERFORMANCE TIPS” lingile vajutades navigeerida juhiste lehele.
9. Kasutaja saab lehe päises asuvale “USEFUL LINKS” lingile vajutades navigeerida kasulike linkide lehele.
10. Kasutaja saab lehe päises asuvas “EESTI” ja “ENGLISH” nuppude kaudu vahetada veebisaidi keelt, eesti keele ja inglise keele vahel.

Koduleht:

- Kasutajale kuvatakse sissejuhatavat tervitusteksti.
- Peab sisaldama klikitavaid linke, mis viivad kasutaja juhiste lehele .

Infoleht:

- Kasutajale kuvatakse taustainfot veebilehest ja selle loojast.

Juhiste leht:

- Kasutajale kuvatakse vähemalt 5 peatükki Vue.js jõudluse kohta.
- Kasutaja saab veebilehe vasakus küljes asuva külgriba kaudu peatükkide vahel navigeerida.

Kasulike linkide leht:

- Kasutajale kuvatakse koos kirjeldava kommentaariga vähemalt 4 linki, mis viitavad veebi jõudlusega seotud artiklitele.

3.2. Mittefunktsionaalsed nõuded

“Mittefunktsionaalsed nõuded vastavad küsimusele "Kuidas tarkvara peab vajalikke funktsioone täitma?”. Näiteks, süsteemi vastuse aeg peab jääma etteantud piiridesse (tõhusus); süsteem peab teatud ajavahemike jooksul tõrgeteta töötama (töökindlus) jne.”

[37, p. 16]

Järgnevalt on loetletud projekti planeerimisel määratletud mittefunktsionaalsed nõuded:

1. Veebisait peab olema mobiilisõbralik.
2. Veebisait peab toetama inglise ja eesti keelt.
3. Veebisait peab olema kasutajasõbralik ja intuitiivne.
4. Veebisait peab reageerima brauseri akna suuruse muutustele.
5. Veebilehtede vahel navigeerimisel peab sisu kaduma ja ilmuma sujuva üleminekuga.
6. Veebisait peab olema kasutatav järgmiste brauseritega: Google Chrome, Mozilla Firefox, Safari ning Edge.

4. Kasutatud tehnoloogiad ja keeled

Käesolev peatükk kirjeldab veebisaidi arendusel kasutatud tehnoloogiaid ja keeli ning põhjendab nende valikuid. Kuna loodi täiesti uus veebisait, ei olnud tehnoloogiate ja keelte valimisel piiranguid.

Veebisaidi loomisel kasutatud keeled:

1. JavaScript (JS);
2. Hypertext Markup Language (HTML);
3. Cascading Style Sheets (CSS).

Eelnevad keeled on kasutusel enamikel kaasaegsetel veebisaitidel [38].

Veebisaidi loomisel kasutatud tehnoloogiad:

1. Vue.js;
2. Vuetify.

Veebisaidi loomisel kasutatud disaini strateegiad:

1. Atomaarse kujunduse metodoloogia (ingl *atomic design methodology*) [39];
2. Üheleherakendus (ingl *Single Page Application, SPA)* [10].

4.1. Vue.js

Vue.js [40] on eessüsteemi raamistik, mida kasutati käesoleva töö raames valminud veebilehe eessüsteemi loomiseks.

Vue.js on iseseisev, avatud lähtekoodiga JavaScript raamistik, mis on loodud 2014. aastal Evan You poolt [41]. Vue.js'i kasutatakse koos HTML-i, CSS-i ja JavaScriptiga ning see pakub deklaratiivset ja komponendipõhist programmeerimismudelit, mis aitab üheleherakendusi tõhusalt arendada [42]. Komponendipõhine programmeerimismudel on abstraktsioon, mis võimaldab muuta suured rakendused väikesteks, iseseisvateks ja taaskasutatavateks tükkideks [43].

Käesoleva töö jaoks Vue.js valimise peamisteks põhjusteks oli autori varasem kogemus raamistikuga, Vue.js-i lihtsus sobivus üheleherakenduste jaoks.

Vue.js-il on mitmeid olulisi omadusi ja funktsionaalsuseid: deklaratiivne renderdamine, reaktiivsus ja ühefailikomponendid. Järgnevalt kirjeldatakse neid täpsemalt.

4.1.1. Deklaratiivne renderdamine

Vue.js laiendab tavalist HTML-i malli süntaksiga, mis lubab deklaratiivselt kirjeldada HTML väljundit, sõltuvalt JavaScripti olekust [42]. Teisisõnu, JavaScripti dünaamiliselt muutuvat infot saab asetada topelt loogeliste sulgude vahele ning see info renderdatakse veebilehe malli.

4.1.2. Reaktiivsus

Vue.js jälgib automaatselt JavaScript oleku muutusi ning vastavalt muutustele uuendab dokumendi objektimudelit (ingl *Document Object Model, DOM*) [42].

4.1.3. Ühefailikomponendid

Käesoleva töö raames valminud veebisaidi loomisel kasutati ühefailikomponente (vt Joonis 13) (ingl *Single File Components, SFC*). Ühefailikomponent ühendab loogika (JavaScript), malli (HTML) ja stiili (CSS) ühte faili. Ühefailikomponendid esinevad projekti failistruktuuris *.vue struktuurina, kus * on faili nimi [44].

Ühefailikomponentidel on mitmeid eeliseid:

1. Võimalus muuta suur projekt väikesteks HTML-i, CSS-i ja JavaScripti sisaldavateks modulaarseteks tükkideks;
2. Komponendi taseme skoopiga CSS;
3. IDE toetus koos automaatse lõpetamise (ingl *auto-completion)* ja tüübikontrolliga (ingl *type-check).*

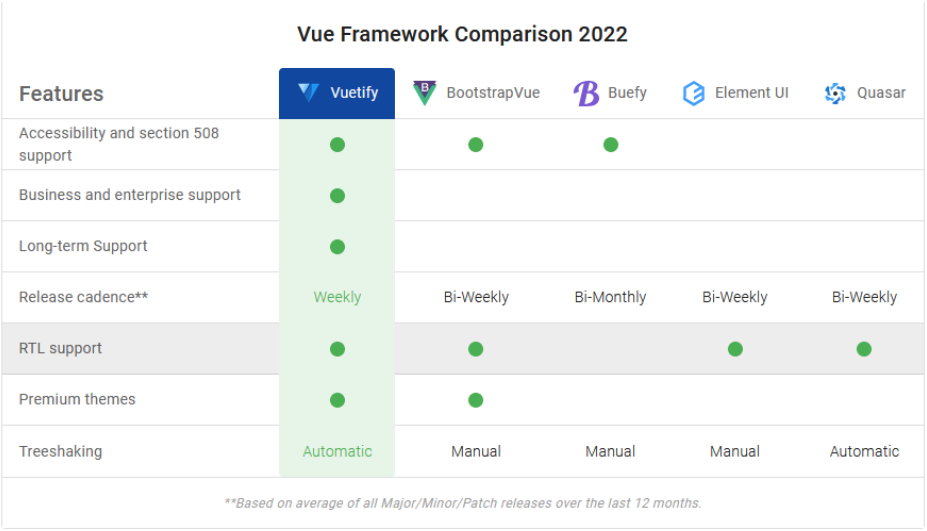
Joonis 13. Ühefailikomponent *useful-link.vue* loodud veebisaidi lähtekoodist.

4.2. Vuetify

Vuetify [45] on Vue.js põhjal loodud kasutajaliidese raamistik. See sisaldab erinevaid eelvalmistatud komponente ja tööriistu, mis aitavad arendajal luua lihtsalt rikkalikku kasutajakogemust [46].

Vuetify jälgib Material Design spetsifikatsiooni. See on Google poolt 2014. aastal väljatöötatud disainikeel, mille eesmärgiks on läbi päriselulisi esemeid meenutavate komponentide luua kvaliteetset kasutajakogemust [47].

Vuetify kasutamise põhjuseks on eelkõige selle lähedane seos Vue.js-iga, rikkalik valmis tehtud komponentide valik ja Material Designi tugi. Lisaks pakub Vuetify eeliseid teiste konkurentide ees. Näiteks pakub Vuetify pikaajalist toetust ja automaatset puu raputamise teenust (vt Joonis 14).



Vue Framework Comparison 2022

Features	Vuetify	BootstrapVue	Buefy	Element UI	Quasar
Accessibility and section 508 support	●	●	●		
Business and enterprise support	●				
Long-term Support	●				
Release cadence**	Weekly	Bi-Weekly	Bi-Monthly	Bi-Weekly	Bi-Weekly
RTL support	●	●		●	●
Premium themes	●	●			
Treeshaking	Automatic	Manual	Manual	Manual	Automatic

**Based on average of all Major/Minor/Patch releases over the last 12 months.

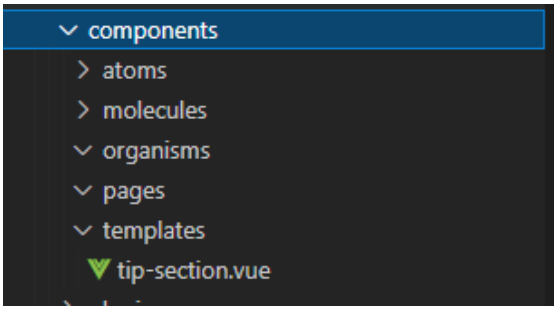
Joonis 14. Vuetify võrdlus konkurentidega [46].

4.3. Atomaarse kujunduse metodoloogia

Loodud veebilehe valmistamisel kasutati projekti komponentide organiseerimiseks atomaarse kujunduse metodoloogiat (vt Joonis 15).

See on metodoloogia, mis koosneb viiest erinevast astmest: aatom, molekul, organism, mall ja lehekülg. Need etapid on inspireeritud loodusest ning töötavad koos, et luua liidese süsteeme rohkem läbimõeldumal ja hierarhilisemal viisil [39].

Sisuliselt tähendab see seda, et veebisaidi kõige väiksemad komponendid tuleb paigutada kausta nimega *aatomid*, nendest suuremad komponendid, mis võivad sisaldada aatomi suuruseid komponente, tuleb paigutada kausta nimega *molekulid* ja nii edasi.



Joonis 15. Näide atomaarse kujunduse kasutusest loodud veebisaidi lähtekoodis.

4.4. Üheleherakendus

Bakalaureusetöö raames loodud veebisait arendati üheleherakendusena.

Üheleherakendus on veebirakendus või -sait mis suhtleb kasutajaga kirjutades veebilehte dünaamiliselt üle uute andmetega, selle asemel, et serverist uusi lehti laadida. Lehtede vahetudes mõned veebilehe elemendid jäävad muutumata (nagu päis, jalus jne) [48]. Arvestades, et veebilehe funktsionaalsed nõuded sisaldasid veebisaidil päise ja jaluse olemasolu, on otsus veebisait arendada ühelerakendusena põhjendatud.

4.4.1. Kliendipoolne marsruutimine

Üheleherakendustes kasutatakse kliendipoolset marsruutimist. See on tingitud asjaolust et serveripoolne marsruutimine hõlmab serverilt uute lehtede küsimist ja seejärel brauseris nende laadimist [49]. Üheleherakendused seevastu kasutavad kliendipoolsed marsruutimist, kus sisulaadimist vastavalt aadressile haldab veebisaidile laetud JavaScript [50].

Käesolevas töös arendatud veebilehel kasutati vue-routerit, mis on Vue.js-i poolt ametlikult toetatud kliendipoolse marsruutimise teek [51].

Vue-router'i kasutamiseks loodi projekti kaust *router*, kuhu loodi fail *index.js*. See fail sisaldab muutujat *routes*, kus on defineeritud veebisaidi marsruudid objektike massiivina. Veebisaidi veebilehtede aadressid on selles massiivis defineeritud järgmiselt:

1. Koduleht: */;
2. Infoleht: */about*;
3. Juhiste leht: */performance-tips*;
4. Kasulike linkide leht: */more-information*.

Vue-router'i rakendamiseks kasutati selle teegi *router-link* ja *router-view* komponente. *Router-link* komponent on praktiliselt <*a*> element, mis vahetab veebisaidi aadressi ilma lehte uuesti laadimata [52]. *Router-view* on komponent, mis kuvab vastavalt aadressile ja *routes* failis defineeritud marsruutidele lehekülje sisu [53]. Loodud veebisaidi lähtekoodis kasutati seda *src/App.vue* komponendis, päise ja jaluse vahel, dünaamilise sisu kuvamiseks (vt Joonis 16).

```
<v-main>
  <v-container class="pl-7 pr-7" fluid>
    <transition name="fade-custom" mode="out-in">
      <router-view></router-view>
    </transition>
  </v-container>
</v-main>
```

Joonis 16. Näide *router-view* kasutusest lähtekoodis.

5. Veebisaidi analüüs

Käesolev peatükk analüüsib valminud veebisaiti. Sellega saab tutvuda aadressil https://vueperformancehandbook.com/

5.1. Veebisaidi päis

Päis on visuaalne pilt või tüpograafiline element mis paikneb kogu veebilehe ülaosas, ideaalselt kõikidel veebisaidi veebilehtedel. Veebisaidi päis on väga tähtis, kuna see on esimene asi mida külastaja lehele sattudes vaatab [54].

Päise kujundades ja arendades lähtuti parimatest tavadest [2]. Autori eesmärk oli luua minimaalne intuitiivne päis, mis oleks mobiilisõbralik. Joonisel 17 on kujutatud veebisaidi päis, vaadatuna 1920x1080 resolutsiooniga ekraanilt. Päis on kleepuv (ingl *sticky)* ehk see paikneb lehekülje ülaosas ja on alati nähtav. Päise vasakus ääres paikneb veebisaidi pealkiri, millele klikkides suunatakse kasutaja kodulehele. Päise paremas ääres asub navigeerimisriba, mis koosneb neljast klikitavast osast.

Navigeerimisriba struktuur:

1. HOME (veebisaidi koduleht);
2. ABOUT;
3. PERFORMANCE TIPS;
4. USEFUL LINKS.

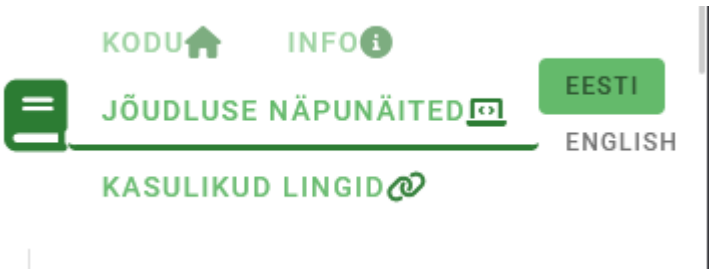
Nendele klikkides suunatakse kasutaja vastavale veebilehele. Lisaks asub navigeerimisribast paremal keele valiku osa, mis koosneb kahest nupust:

1. EESTI (muudab veebilehe eestikeelseks);
2. ENGLISH (muudab veebilehe ingliskeelseks).

Päises on ka kasutatud erinevaid ikoone, mis pärinevad Font Awesome ikooniteegist [55].

Joonis 17. Kuvatõmmis veebisaidi päisest, suurel ekraanil vaadatuna.

Joonisel 18 on kujutatud veebisaidi päis väikesest ekraanist vaadatuna, seadme resolutsioon on 375x812. On näha, et sõltuvalt brauseri akna suurusest päises olev pealkiri peidetakse ja navigatsiooniriba elemendid nihkuvad üksteise alla. Päis on mobiilisõbralik.



Joonis 18. Kuvatõmmis veebisaidi päisest, väikesel ekraanil vaadatuna.

5.2. Veebisaidi jalus

Jalus on veebilehe element, mis paikneb lehe allosas ja tavaliselt sisaldab tähtsat informatsiooni nagu autoriõiguse teatist, lahtiütlusi (ingl *disclaimers)* ja asjakohaseid linke. Jalus lisab veebilehele järjepidevuse tunnet, kuna see on alati veebilehel ja kuvab alati sama informatsiooni [56].

Jaluse loomisel lähtuti ideest luua võimalikult selge ja väike jalus. Oli tähtis, et jaluse värv sobiks muu lehekülje disainiga ning sisaldaks funktsionaalsetes nõuetes välja toodud informatsiooni. Päise vasakus ääres asub tekst, mis näitab, kes on veebisaidi autor. Päise paremas ääres on autori kontaktandmed. Jaluses kasutati Material Design ikoone [57].

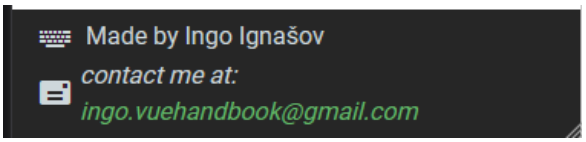
Joonisel 19 on pilt veebisaidi jalusest suurel ekraanil. Joonisel 20 on selle sama jaluse suurendatud vasak osa, joonisel 21 parem osa.

Joonis 19. Kuvatõmmis veebisaidi jalusest, ekraani resolutsioon: 1920x1080.

Joonis 20. Kuvatõmmis veebisaidi jaluse vasakust poolest, ekraani resolutsioon: 1920x1080.

Joonis 21. Kuvatõmmis veebisaidi jaluse paremast poolest, ekraani resolutsioon: 1920x1080.

Joonisel 22 on kujutatud veebisaidi jalus, väikesel ekraanil. On näha, et väikese seadme puhul nihkuvad jaluse elemendid üksteise alla ja midagi katki ei lähe.



Joonis 22. Kuvatõmmis veebisaidi jalusest, ekraani resolutsioon 375x812.

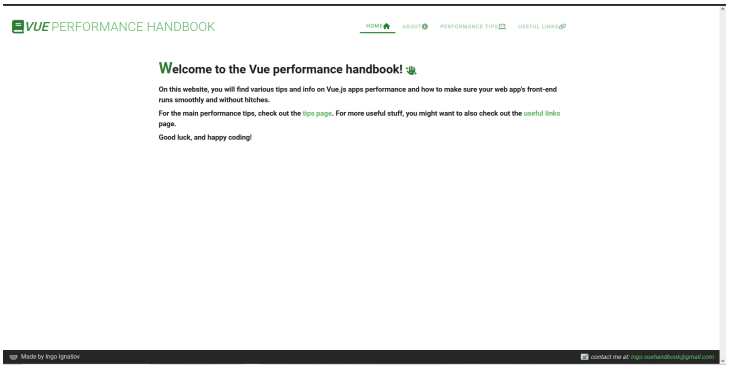
5.3. Veebisaidi koduleht

Joonisel 23 on kujutatud veebisaidi avaleht suurel ekraanil. Sellel on kuvatud tervitustekst, mis kirjeldab lühidalt, missuguse veebisaidiga on tegu ja mida see sisaldab. Tervitustekst sisaldab tekstisiseselt kahte linki mille kaudu on kasutajal võimalik liikuda juhiste lehele või kasulike linkide lehele.

Kodulehte disainides lähtus autor põhimõttest teha võimalikult kergekaaluline ja lihtne koduleht, et veebisaidi laadimise kiirus võimalikult madal hoida.

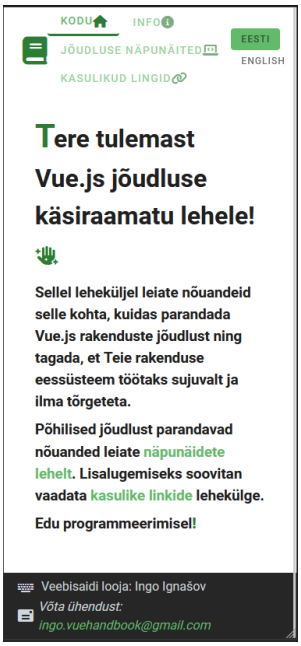
Kui kasutaja proovi navigeerida aadressile, mida loodud veebisaidil ei leidu, siis suunatakse kasutaja vaikimisi kodulehele.

Kodulehel on ka kasutatud ikooni, mis on pärit Font Awesome ikooniteegist [55].



Joonis 23. Kuvatõmmis veebisaidi kodulehest,ekraani resolutsioon 1920x1080.

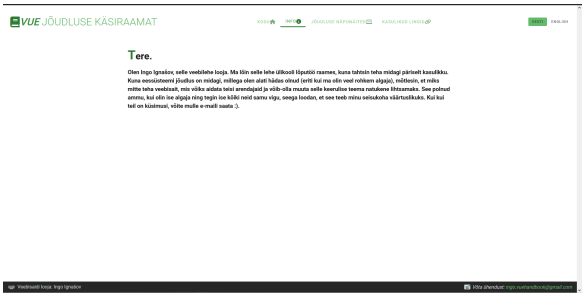
Joonisel 24 on kuvatud avaleht väikesel ekraanil. Kogu lehe sisu mahub ekraanile ära ja ühtegi väljapaistvat viga ei ole.



Joonis 24. Kuvatõmmis veebisaidi kodulehest, ekraani resolutsioon 375x812.

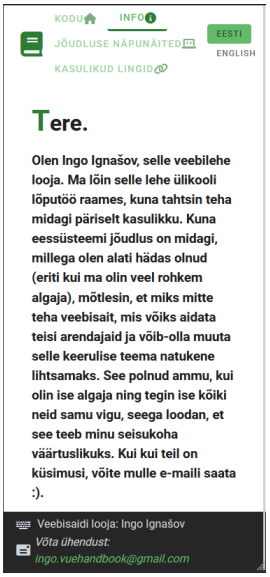
5.4. Veebisaidi infoleht

Joonisel 25 on veebisaidi infoleht suure resolutsiooniga ekraanil. Selle lehe mõte on anda kasutajatele huvi korral võimaluse saada informatsiooni veebisaidi autori ja selle loomise kohta. Kujundusele erilist tähelepanu ei pööratud, põhieesmärk oli lihtsalt kuvada selgitavat teksti.



Joonis 25. Kuvatõmmis veebisaidi infolehest, ekraani resolutsioon 1920x1080.

Joonisel 26 on kuvatud infoleht väikesel ekraanil.



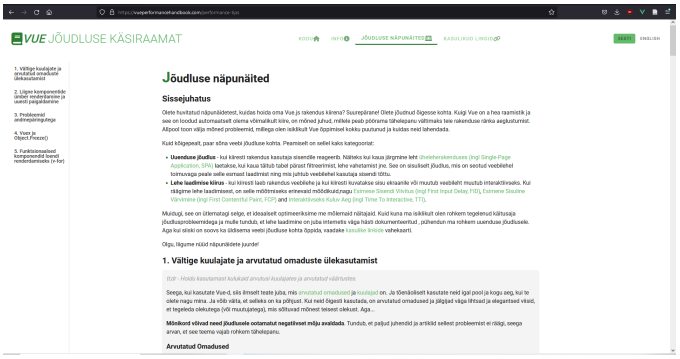
Joonis 26. Kuvatõmmis veebisaidi infolehest, ekraani resolutsioon 375x812.

5.5. Veebisaidi juhiste leht

Joonisel 27 kujutatud juhiste leht on loodud veebisaidi kõige sisukam leht. See koosneb põhiosast, mis sisaldab viit Vue.js jõudlusega seotud peatükki ning külgriba, mille abil on kasutajal võimalik peatükkide vahel navigeerida.

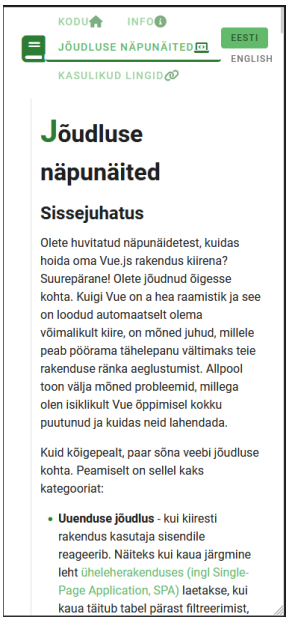
Lehe alguses paikneb sissejuhatus mis kirjeldab lühidalt lehe sisu. Veebilehe sisu on jaotatud viieks peatükiks, iga peatükk kirjeldab ühte viiest autori poolt välja valitud Vue.js jõudluse teemast.

Leht on arendatud blogi stiilis, tekst sisaldab linke mõistetele ja koodijuppe. Vajadusel lõi autor ka keskkonnas CodeSandbox [58] peatüki probleemi illustreeriva näidisprogrammi.



Joonis 27. Kuvatõmmis veebisaidi juhiste lehest. Ekraani resolutsioon, 1920x1080.

Veebileht on ka võimalikult mobiilisõbralik. Lehe suuruse kahanedes kaotatakse vasakul ääres asuv külgriba, et sisule jääks rohkem ruumi(vt Joonis 28).



Joonis 28. Kuvatõmmis juhiste lehe sissejuhatusest. Ekraani resolutsioon 375x812.

5.6. Veebisaidi kasulike linkide leht

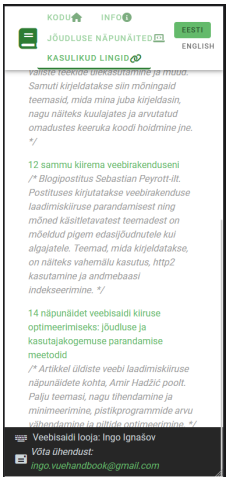
Joonisel 29 on kuvatud loodud veebisaidi kasulike linkide leht. Selle lehe ülesanne on kuvada kasutajale linke valitud ressurssidele koos autori kirjeldava kommentaariga. Autori eesmärk nende linkide lisamisel oli anda kasutajatele võimalus uurida põhjalikumalt veebi jõudluse teemasid mille kohta loodud veebisaidi juhiste lehel peatükki ei ole.

Lehel on kuvatud vertikaalselt üksteise all neli linki. Iga lingi all on autori poolt lisatud helehalli värviga kommentaar.

Joonis 29. Kuvatõmmis veebisaidi kasulike linkide lehest. Ekraani resolutsioon 1920x1080.

Joonis 30 kujutab eelmainitud infolehte, üles keritult, väikesel ekraanil. Veebileht näeb mobiilivaates korrektne välja. Üks puudus mida autor märkas: lehe täiesti alla kerides on märgata, et alumise tekstirea alumine äär peitub osaliselt jaluse taha (vt Joonis 31).

Joonis 30. Loodud veebisaidi infoleht, üles keritult. Ekraani resolutsioon 375x812.



Joonis 31. Kuvatõmmis veebisaidi infolehest, alla keritud. Ekraani resolutsioon 375x812.

6. Kokkuvõte

Käesoleva bakalaureusetöö eesmärgiks oli luua eesti- ja ingliskeelsed materjalid Vue.js raamistiku jõudluse kohta ning luua nende materjalide kuvamiseks veebisait. Need materjalid on kättesaadavad aadressil https://vueperformancehandbook.com/.

Käesolev töö koosneb veebi jõudluse ning selle tähtsuse kirjeldusest, loodud materjali peatükkide kokkuvõtetest, arendatud veebisaidi funktsionaalsete ja mittefunktsionaalsete nõuete täpsustamisest, veebisaidi loomiseks kasutatud keelte ja tehnoloogiate ülevaatest ning valminud veebisaidi analüüsist.

Töö jooksul valmis Vue.js baasil neljast veebilehest koosnev ühelerakendus, mis sisaldab loodud materjale. Kõik töös esitatud funktsionaalsed ja mittefunktsionaalsed nõuded said täidetud.

Veebisaidi lähtekood on saadaval
https://gitlab.com/IngoIgnasov/thesis/-/tree/main/thesis-project.

7. Viidatud kirjandus

[1] "Powerful apps fueled by the web: How developers are engaging an expanding Chrome OS audience," *chromeOS.dev*, Mar. 01, 2022. https://chromeos.dev/en/posts/powerful-apps-fueled-by-the-web/ (accessed Apr. 11, 2022).

[2] M. Pekarsky, "Does your web app need a front-end framework?," *Stack Overflow Blog*, Feb. 03, 2020. https://stackoverflow.blog/2020/02/03/is-it-time-for-a-front-end-framework/ (accessed Apr. 14, 2022).

[3] "The Most Popular Front-end Frameworks in 2022," *Stack Diary*, Feb. 19, 2022. https://stackdiary.com/front-end-frameworks/ (accessed Apr. 21, 2022).

[4] Neil Patel, "How Loading Time Affects Your Bottom Line," *Neil Patel*, Apr. 28, 2011. https://neilpatel.com/blog/loading-time/ (accessed Apr. 09, 2022).

[5] "AKIT - Andmekaitse ja infoturbe leksikon." https://akit.cyber.ee/term/4139-veebileht (accessed May 01, 2022).

[6] "AKIT - Andmekaitse ja infoturbe leksikon." https://akit.cyber.ee/term/4137-veebisait (accessed May 01, 2022).

[7] "What is web performance? - Learn web development | MDN." https://developer.mozilla.org/en-US/docs/Learn/Performance/What_is_web_performance (accessed May 03, 2022).

[8] "AKIT - Andmekaitse ja infoturbe leksikon." https://akit.cyber.ee/term/2413-front-end (accessed May 01, 2022).

[9] "AKIT - Andmekaitse ja infoturbe leksikon." https://akit.cyber.ee/term/2412-taga-2 (accessed May 01, 2022).

[10] "Single-page application," *Wikipedia*. May 01, 2022. Accessed: May 03, 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Single-page_application&oldid=1085668171

[11] "Brauser," *Vikipeedia*. Feb. 25, 2022. Accessed: May 03, 2022. [Online]. Available: https://et.wikipedia.org/w/index.php?title=Brauser&oldid=6057694

[12] "What is MVVM (Model-View-ViewModel)?," *Webopedia*, Aug. 06, 2020. https://www.webopedia.com/definitions/mvvm-model-view-viewmodel/ (accessed Apr. 19, 2022).

[13] "Tree Shaking," *webpack*. https://webpack.js.org/guides/tree-shaking/ (accessed Apr. 15, 2022).

[14] "Lifecycle Hooks | Vue.js." https://vuejs.org/guide/essentials/lifecycle.html (accessed May 04, 2022).

[15] dreammarketing, "3 Lihtsat sammu, kuidas tõsta oma kodulehe konversioonimäära?," *Dreammarketing*, Mar. 05, 2018. https://dreammarketing.ee/3-lihtsat-sammu-kuidas-tosta-oma-kodulehe-konversioonimaara/ (accessed May 07, 2022).

[16] "First input delay - MDN Web Docs Glossary: Definitions of Web-related terms | MDN." https://developer.mozilla.org/en-US/docs/Glossary/First_input_delay (accessed May 05, 2022).

[17] "First contentful paint - MDN Web Docs Glossary: Definitions of Web-related terms | MDN." https://developer.mozilla.org/en-US/docs/Glossary/First_contentful_paint (accessed May 05, 2022).

[18] "Time to interactive - MDN Web Docs Glossary: Definitions of Web-related terms | MDN." https://developer.mozilla.org/en-US/docs/Glossary/Time_to_interactive (accessed

May 05, 2022).
[19] "SEO Starter Guide: The Basics | Google Search Central | Documentation," *Google Developers*. https://developers.google.com/search/docs/beginner/seo-starter-guide (accessed May 05, 2022).
[20] "Rebuilding Pinterest pages for performance resulted in a 40% decrease in wait time, a 15% increase in SEO traffic and a 15% increase in conversion rate to signup." https://wpostats.com/2017/03/10/pinterest-seo.html (accessed May 05, 2022).
[21] uxplanet.org, "How page speed affects Web User Experience," *Medium*, Dec. 29, 2019. https://uxplanet.org/how-page-speed-affects-web-user-experience-83b6d6b1d7d7 (accessed May 05, 2022).
[22] J. Juviler, "The Ultimate Guide to Website Performance." https://blog.hubspot.com/website/website-performance (accessed May 05, 2022).
[23] "You and site performance, sitting in a tree... | Google Search Central Blog," *Google Developers*. https://developers.google.com/search/blog/2010/05/you-and-site-performance-sitting-in (accessed May 05, 2022).
[24] "Portent," *Portent*, Apr. 20, 2022. https://www.portent.com/blog/analytics/research-site-speed-hurting-everyones-revenue.htm (accessed May 05, 2022).
[25] D. A. Says, "Understanding computed properties in Vue.js," *LogRocket Blog*, May 04, 2021. https://blog.logrocket.com/understanding-computed-properties-in-vue-js/ (accessed Apr. 20, 2022).
[26] "Watchers | Vue.js." https://vuejs.org/guide/essentials/watchers.html (accessed Apr. 20, 2022).
[27] "Reactivity Fundamentals | Vue.js." https://vuejs.org/guide/essentials/reactivity-fundamentals.html (accessed May 04, 2022).
[28] "Promise - JavaScript | MDN." https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise (accessed May 05, 2022).
[29] "Promise.all() - JavaScript | MDN." https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise/all (accessed May 05, 2022).
[30] "What is Vuex? | Vuex." https://vuex.vuejs.org/ (accessed May 05, 2022).
[31] "Object.freeze() - JavaScript | MDN." https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/freeze (accessed May 05, 2022).
[32] "Render Functions & JSX — Vue.js." https://v2.vuejs.org/v2/guide/render-function.html?redirect=true#Functional-Components (accessed May 05, 2022).
[33] "v-for Directive in Vue.js," *GeeksforGeeks*, Jun. 25, 2020. https://www.geeksforgeeks.org/v-for-directive-in-vue-js/ (accessed May 05, 2022).
[34] J. Juviler, "9 Guidelines & Best Practices for Exceptional Web Design and Usability." https://blog.hubspot.com/blog/tabid/6307/bid/30557/6-guidelines-for-exceptional-website-design-and-usability.aspx (accessed Apr. 28, 2022).
[35] "The 2022 Web Design Best Practices You Need to Know [WDATX]," Feb. 06, 2019. https://websitedesignaustintexas.com/web-design/web-design-best-practices/ (accessed Apr. 27, 2022).
[36] "12 Web Design Best Practices for 2022," *Tiller*, Apr. 27, 2022. https://tillerdigital.com/blog/12-web-design-best-practices-for-2022/ (accessed Apr. 28, 2022).

[37] "tks-loeng.pdf." Accessed: May 07, 2022. [Online]. Available: http://tepandi.ee/tks-loeng.pdf
[38] "Usage Statistics of JavaScript as Client-side Programming Language on Websites, May 2022." https://w3techs.com/technologies/details/cp-javascript/ (accessed May 01, 2022).
[39] "Atomic Design Methodology | Atomic Design by Brad Frost." http://atomicdesign.bradfrost.com/chapter-2/ (accessed Apr. 15, 2022).
[40] "Vue.js - The Progressive JavaScript Framework | Vue.js." https://vuejs.org/ (accessed Apr. 21, 2022).
[41] "Vue.js - The Progressive JavaScript Framework | Vue.js." https://vuejs.org/about/faq.html#who-maintains-vue (accessed Apr. 21, 2022).
[42] "Vue.js - The Progressive JavaScript Framework | Vue.js." https://vuejs.org/guide/introduction.html#what-is-vue (accessed Apr. 20, 2022).
[43] "Introduction — Vue.js." https://v2.vuejs.org/v2/guide/?redirect=true#Composing-with-Components (accessed Apr. 20, 2022).
[44] "Vue.js - The Progressive JavaScript Framework | Vue.js." https://vuejs.org/guide/introduction.html#single-file-components (accessed Apr. 20, 2022).
[45] "Vuetify — A Material Design Framework for Vue.js," *Vuetify*. https://vuetifyjs.com/ (accessed Apr. 15, 2022).
[46] "Why you should be using Vuetify," *Vuetify*. https://vuetifyjs.com/en/introduction/why-vuetify/ (accessed Apr. 15, 2022).
[47] "What is Material Design?," *The Interaction Design Foundation*. https://www.interaction-design.org/literature/topics/material-design (accessed Apr. 15, 2022).
[48] bloomreach.com, "What is a Single Page Application? | Bloomreach." https://www.bloomreach.com/en/blog/2018/what-is-a-single-page-application (accessed Apr. 27, 2022).
[49] N. kingsley, "Server Side Routing vs Client-Side Routing | Codementor." https://www.codementor.io/@chinemeremnwoga/server-side-routing-vs-client-side-routing-yne57eq9a (accessed Apr. 27, 2022).
[50] W. Schepenaar, "Server-side vs Client-side Routing," *Medium*, May 29, 2017. https://medium.com/@wilbo/server-side-vs-client-side-routing-71d710e9227f (accessed Apr. 27, 2022).
[51] "Vue Router." https://router.vuejs.org/ (accessed Apr. 27, 2022).
[52] "Vue Router." https://router.vuejs.org/guide/#router-link (accessed Apr. 27, 2022).
[53] "Vue Router." https://router.vuejs.org/guide/#router-view (accessed Apr. 27, 2022).
[54] "What Is A Website Header And Why You Should Use It," *SISTRIX*. https://www.sistrix.com/ask-sistrix/getting-started-seo/what-is-a-website-header-and-why-you-should-use-it/ (accessed Apr. 27, 2022).
[55] "Font Awesome." https://fontawesome.com (accessed May 01, 2022).
[56] W. com / A. Inc, "Everything You Should Know About a Website Footer," *Jetpack*, Aug. 22, 2018. https://jetpack.com/blog/website-footer-tips/ (accessed May 01, 2022).
[57] "Material Design Icons." https://materialdesignicons.com/ (accessed Apr. 20, 2022).
[58] "CodeSandbox: Online Code Editor and IDE for Rapid Web Development." https://codesandbox.io/?from-app=1 (accessed May 04, 2022).

Lisad

I. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, Ingo Ignašov,

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose "Veebimaterjalide ja -saidi loomine Vue.js raamistiku jõudlusprobleemidest", mille juhendaja on Helle Hein, reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.
2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi litsentsiga CC BY NC ND 3.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.
3. Olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile.
4. Kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

Ingo Ignašov

09.05.2022