MINISTERUL EDUCAŢIEI AL REPUBLICII MOLDOVA

UNIVERSITATEA DE STAT „ALECU RUSSO” DIN BĂLŢI

FACULTATEA DE ŞTIINŢE REALE, ECONOMICE ȘI ALE MEDIULUI

CATEDRA DE MATEMATICĂ ȘI INFORMATICĂ

ANALIZA COMPARATIVĂ A FRAMEWORKURILOR ANGULAR SI VUEJS PENTRU ELABORAREA APLICAȚIILOR WEB
TEZĂ DE AN

Autor:

Student al grupei IS21Z

Mihai CRIHAN

_____

Conducător științific:

Sergiu CHILAT

lector univ., magistru

_____

BĂLȚI, 2019

CUPRINS

INTRODUCERE

La momentul dat avem o mulțime de limbaje de programare însa concurența de a deveni lideri pe piață face aceste limbaje tot mai complexe și greu de inteles, însă domeniile de utilizare se extind și abilitățile acestora cresc precum crește și cerintele de la programatorul care dorește să lucreze in acest domeniu, pentru a ușura munca programatorilor și de a face limbajul mai ușor de înțeles a fost elaborat un set de componente și modele numite frameworkuri. Framework (din engleza - "cadru, structură") - în programarea web este o platformă software specială sau un set de componente și modele care facilitează procesul de dezvoltare web. Deși frameworcurile sunt cunoscute în alte domenii ale programării, și anume în dezvoltarea web, acestea au înregistrat o dezvoltare intensă în ultimii ani. Toate frameworkurile utilizează modelul de proiectare MVC (model-view-controller). MVC este o condiție prealabilă pentru organizarea codului sau a componentelor. Sarcina sa este de a rezolva problema de design care a apărut în soluția de lucru.

În plus, modelul de design împarte întreaga aplicație în trei elemente:

- Procese de afaceri (model).
- Controlul debitului (controler).
- Crearea de pagini HTML (vizualizare).

Folosind modelul MVC, cele mai multe frameworcuri ajută la organizarea codului astfel încât orice modificare a modelului, controlerului sau vizualizării să nu aibă o influență puternică asupra structurii aplicației. Frameworkurile au multe avantaje, dar există și dezavantaje. În special, dezavantajul este procesul de detectare a erorilor datorită particularităților configurației.

Actualitatea: tema cercetată este una actuală, deoarece astăzi există o varietate de frameworkuri. Teza se va referi la VueJs și Angular, deoarece sunt tehnologii acluale si sunt unele din cele mai utilizate in spatiul web. Pentru cele mai populare framework-uri, se creează comunități, se elaborează manuale și documentație.

Pentru funcționarea site-ului și promovarea acestuia, framework-urile pot fi chiar mai eficiente decât sistemele de gestionare a conținutului. Spre deosebire de cele din urmă, framework-urile oferă site-uri cu viteză și nu necesită foarte multe resurse. În cele din urmă, ceea ce trebuie folosit pentru dezvoltarea site-ului - un framework curat sau avansat - depinde de dezvoltator și de client. Scopul și sarcinile specifice determină metoda de soluționare.

Scopul lucrarii: constă în cercetarea frameworkurilor VueJs și Angular pentru elaborarea unei aplicații web în care vor fi utilizate componentele acestor frameworkuri.

Obiectivele lucrării:

- Analiza resurselor informative: literatură de specialitate, comunitățile și tutorialele;
- Analiza generală a VueJs;
- Analiza generală a Angular;
- Analiza comparativă a tehnologiilor și posibilitățile acestora;
- Proiectarea aplicației pe ambele Frameworkuri.

Utilitatea practica: au fost elaborate doua aplicații pe doua frameworkuri VueJs si Angular ele au scopul de a fi identice, însa de a prezenta aceeasș functțe prin instrumentele acestor frameworkuri, partea teoretică contine informații aprofundate despre ele, avantaje, dezavantaje și partea practică este utilizată pentru verificarea cunostințelor și lucrul cu diferite module.

Teza este structurata in doua capitole:

Primul capitol este prezentarea VueJs si Angular:

- Scurt istoric – contine o informație despre framework si instrumentele lui;
- Conceptele principale;
- Componente și directive;
- Utilizarea bibliotecilor adăugatoare.

Al doilea capitol prezintă aplicațiile elaborate in aceste medii:

- Ce avem nevoie pentru a incepe lucrul;
- Cum are loc crearea proiectului;
- Elaborarea aplicației .

Teza este expusă pe 24 pagini, conține 23 surse bibliografice și 9 figuri din care 2 figuri sunt impartite in două.

1. CERCETAREA FRAMEWORKURILOR VUEJS ȘI ANGULAR

În capitolul dat vor fi cercetate două frameworkuri renumite, cu posibilitați mari pentru elaborarea aplicațiilor web, acestea au rolul de a simplifica lucrul cu limbajele de programare, în continuare sunt prezentate frameworkurile cu scopul de a informa și de a face un imbold pentru alegerea ului framework conform cerințelor.

1.1 Cercetarea Vuejs

Creatorul Vue.js este Evan You, fost angajat al Google și Meteor Dev Group. El a început să dezvolte acest framework în 2013, iar în februarie 2014 a avut loc primul comunicat public. Vue este utilizat pe scară largă în rândul companiilor chinezești, de exemplu: Alibaba, Baidu, Xiaomi etc.

Fig. 1.1 Emblema VueJs

Vue este un framework Front End progresiv pentru construirea de interfețe utilizator. Spre deosebire de alte cadre monolitice, Vue este proiectat de la început pentru a fi adoptat treptat. Biblioteca de bază se concentrează numai asupra stratului de vizualizare și este ușor de preluat și integrat cu alte biblioteci sau proiecte existente.

Vue este, de asemenea, perfect capabil să propulseze aplicații sofisticate cu o singură pagină atunci când este utilizat în combinație cu instrumente moderne și biblioteci de sprijin. Vue indeplineste un lucru enorm executind o legatura reactiva intre date si DOM.
Sa luam in consideratie cel mai simplu exemplu:

- cream o nouă instanță prin new Vue;
- în el, definim ce element este urmărit;
- în data avem un obiect de stare.

În html, firește, ar trebui să existe un element cu selectorul portivit.

1.2 **Conceptele de baza a VueJs**

Principalele concepte ale Vue sunt:

- Constructor;
- Componentele;
- Directive;
- Router.

1.3 **Ce este un constructor**

Lucrul cu Vue.js începe cu crearea unei noi instanțe new Vue. În el se află un element pe care îl urmărește Vue. In template se selecteaza un element în care va fi afișat Vue. În data se stochează starea curentă a instanței și metoda computed ne furnizează proprietăți calculate. În methods se pot distinge următoarele metode și metode personalizate ale ciclului de viață Vue:

```
methods: {
    beforeCreate: function(){},
    created: function(){},
    beforeMount: function(){},
    mounted: function(){},
    beforeUpdate: function(){},
    updated: function(){},
    beforeDestroy: function(){},
    destroyed: function(){},
    }
```

Fiecare metoda este utilizată cu un anumit scop și exercită o funcție anumită:

- beforeCreate - urmărește datele și inițiază evenimente;
- created - caută el sau șablon. Dacă este așa, se face în ele; dacă nu, caută metoda render;
- beforeMount - creează vm.$el și le înlocuiește cu el;
- mounted - elementul este redat.

Când se schimbă starea:

- beforeUpdate - redă VDOM din nou și se compară cu DOM real, aplică modificările;
- updated - modificări redate;

- beforeDestroy - dezmembrarea completă a observatorilor, a componentelor interne și a ascultătorilor de evenimente;
- destroyed – se apeleaza când operația se oprește.

Pentru a vizualiza diagrama pentru ciclul de viață al instanței. (Anexa 1.)

1.4 **Lucrul cu directive**

Directivele sunt atribute speciale pentru adăugarea elementelor html și functionalitați noi. Iata exemple de directive care sunt similare cu cele din Angular:

- V-bind — Dinamic face legatura cu unul sau mai multe attribute;
- v-if — conditie de modificare a elementului;
- V-else — este blocul “else” pentru “if”;
- V-for — trece masivul de obiecte ciclic;
- V-model — leagă starea cu elementul input;
- V-on — leagă ascultătorul de evenimente cu elementul;
- V-once — denesează elementul doar la inceput si nu il mai urmarește;
- V-show — schimbă afișarea elementului, schimbînd proprietatea CSS display;
- V-text — reînnoieste elementul textContent.

Toate directivele Vue au prefixul "v-". În directivă este transmisă valoarea unei anumite valori de stare, iar argumentii pot fi atribute sau evenimente html.

1.5 **Rolul componentelor**

Componentele ajută la extinderea elementelor de bază html și la implementarea codului reutilizabil. În esență, componentele sunt părți reutilizabile ale User Interface. La etapa de proiectare, se împarte aplicația în părți independente și se obține structura copac a componentelor.

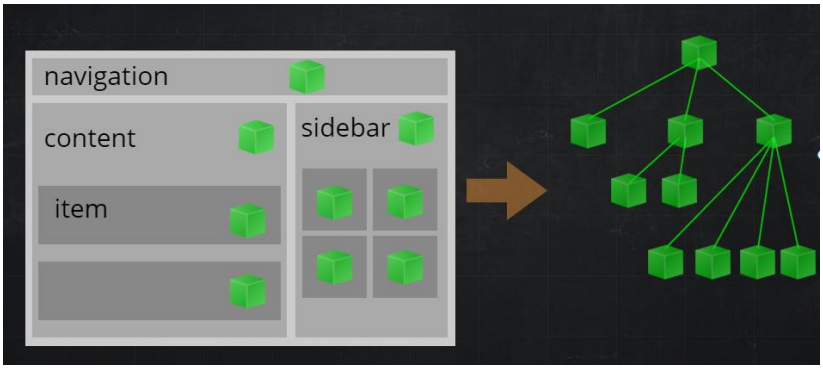


Fig. 1.2 Structura copac a Componentelor

Comunicarea între componentele vue se efectuează în conformitate cu principiul "Props in, Events out". Adică, de la elementul părinte la copil, informațiile sunt transmise prin props și înapoi - evenimentele sunt declanșate.
De asemenea, în Vue.js există așa-numitele componente cu un singur fișier. Se creeaza fisier cu extensia .vue și se scriu stiluri, modele și logică acolo pe orice preprocesor convenabil (SASS, Stylus, PostCSS, Jade, ...) și limba compilată în JS (CoffeeScript, TypeScript).

1.6 **Efecte și tranziții**

Vue oferă diferite modalități de a aplica efecte de animație atunci când elemente sunt desenate, actualizate sau șterse din DOM. Acestea includ instrumente pentru:

- aplicarea automată a claselor pentru tranziții și animații CSS;
- integrarea bibliotecilor straine pentru animațiile CSS, cum ar fi Animate.css;
- utilizarea JavaScript pentru a manipula DOM;
- integrarea bibliotecilor JavaScript de la terți pentru animații, cum ar fi Velocity.js.

Asta se întâmplă atunci când elementul este înfășurat într-o componentă de tranziție: Acesta va mișca în mod automat tranzițiile sau animațiile CSS aplicate. În caz contrar, CSS va fi adăugat / eliminat la cronometrele corespunzătoare.

Există șase clase aplicate pentru tranzițiile de intrare / ieșire.

- v-enter: Starea de pornire pentru introducere. Adăugat înainte de introducerea elementului, a fost eliminat un cadru după introducerea elementului.
- v-enter-active: Stare activă pentru intrare. Aplicată pe toată faza de intrare. Adăugat înainte ca elementul să fie inserat, eliminat atunci când tranziția / animația se termină. Această clasă poate fi utilizată pentru a defini curba de durată, întârziere și relaxare pentru introducerea tranziției.
- v-enter-to: Disponibil numai în versiunile 2.1.8+. Stare de sfârșit pentru introducere. A fost adăugat un cadru după introducerea elementului (în același timp, v-enter este eliminat), eliminat atunci când tranziția / animația se termină.
- v-leave: Starea de plecare. Adăugat imediat când este declanșată o tranziție de plecare, eliminată după un cadru.
- v-leave-active: stare activă pentru concediu. Aplicată pe toată faza de plecare. Adăugat imediat când declanșarea tranziției este declanșată, se elimină atunci când tranziția / animația se termină. Această clasă poate fi utilizată pentru a defini curba de durată, întârziere și relaxare pentru tranziția de plecare.

- v-leave-to: Disponibil numai în versiunile 2.1.8+. Starea de încheiere pentru concediu. Adăugat un cadru după declanșarea unei tranziții de plecare (în același timp, eliminarea v-leave), eliminată atunci când tranziția / animația se termină.

1.7 Ce este routing

În Vue.js, un pachet separat vue-router este responsabil pentru rutare. Crearea unei aplicații cu o singură pagină cu Router Vue + Vue este simplă. Cu Vue.js, deja compunem aplicația noastră cu componente. Acesta acceptă rute imbricate pentru componentele imbricate, oferă un API simplificat pentru cârligele de navigare, comportamentul de defilare gestionat și comenzile de navigare avansate. Atunci când adăugăm Vue Router la mix, tot ce trebuie să facem este să cartografiem componentele noastre pe rute și să lăsăm Vue Router să știe unde să le facă.

1.8 Cercetarea Angular

Angular este una dintre cele mai populare biblioteci pentru crearea de aplicații complexe Frontend. Angular (numit și "Angular 2+" sau "Angular v2 și mai nou") este un framework de dezvoltare web cu sursă deschisă bazată pe limbajul TypeScript.

Fig. 1.3 Emblema Angular

Proiectul este dezvoltat de Echipa Angular de la Google și de o comunitate de utilizatori individuali și companii. Angular este o rescriere completă, de către aceeași echipă, a frameworkului AngularJS. Inițial, versiunea rescrisă a AngularJS a fost numită "Angular 2" de echipă, însă acest lucru a provocat confuzie printre dezvoltatori. De aceea, echipa a anunțat că "AngularJS" se va referi la versiunile 1.X și "Angular" (fără "JS") la versiunile 2 și ulterioare.
De-a lungul anilor de evoluție, majoritatea defectelor din framework au fost eliminate, bibliotecile au fost aduse într-o stare foarte stabilă, iar mărimea comunității tinde spre infinit. Angular oferă o astfel de funcționalitate ca legare bidirecțională, care vă permite să modificați în

mod dinamic datele într-un singur loc al unei interfețe atunci când datele modelului se modifică într-un alt model, modele, rutare și așa mai departe.

1.9 **Crearea componentelor**

Componentele reprezintă elementele de bază ale aplicației Angular. Fiecare aplicație Angular are cel puțin o componentă. Prin urmare, vom crea un nou fișier în dosarul src / app, pe care îl vom numi app.component.ts și în care definim codul. La începutul fișierului, este definită directiva de import, care importă funcționalitatea modulului Angular / core, oferind acces la funcția de decorator @Component. Următorul este, de fapt, funcția decorator @Component, care asociază metadatele cu clasa de componente AppComponent. În această funcție, în primul rând, este definit parametrul selector sau selectorul css pentru elementul HTML care va reprezenta componenta. În al doilea rând, definește un parametru de șablon sau un șablon care specifică modul de redare a componentei.

```
import { Component } from '@angular/core';
@Component({
   selector: 'my-app',
})
export class AppComponent {
}
```

1.10 **Directivele in Angular**

Marchează o clasă ca directivă Angular. Putem defini propriile directive pentru a atașa comportamentul personalizat elementelor din DOM. Opțiunile oferă metadate de configurare care determină modul în care directiva ar trebui procesată, instanțiată și utilizată în timpul rulării.

- selector - Selectorul CSS care identifică această directivă într-un șablon și declanșează instanțiarea directive;
- inputs - Enumeră setul de proprietăți de intrare legat de date pentru o directive;
- Outputs - Enumeră setul de proprietăți de ieșire legate de eveniment;
- providers - Configurează injectorul din această directivă sau componentă cu un jeton care găsește un furnizor de dependență;
- exportAs - Definește numele care poate fi folosit în șablon pentru a atribui această directivă unei variabile;
- queries - Configurează interogările care vor fi injectate în directive;

- host - Caracterizează proprietățile clasei pentru a lega elementul gazdă pentru proprietăți, atribute și evenimente utilizând un set de perechi cheie-valoare;
- JIT - Dacă este adevărat, această directivă / componentă va fi omisă de compilatorul AOT și astfel va fi întotdeauna compilată folosind JIT.

Clasele de directivă, cum ar fi clasele componente, pot implementa cârligele ciclului de viață pentru a influența configurația și comportamentul acestora.

1.11 **Module funcționale**

Angular NgModules diferă și completează modulele JavaScript. Un NgModule își poate asocia componentele cu codul său, cum ar fi serviciile, pentru a forma unități funcționale. Fiecare aplicație Angular are un modul rădăcină, denumit convențional AppModule, care lansează aplicația. O aplicație conține de obicei multe module funcționale. La fel ca modulele JavaScript, nu este nevoie de niciunul mai nou, ci de NgModules. De exemplu, pentru a utiliza serviciul de router în aplicația dvs., importați Router NgModule. Proiectare pentru reutilizabilitate complexă. În plus, este o tehnică de încărcare leneșă - adică un modul de încărcare la cerere - trebuie să fie încărcat la pornire.

Un NgModule este definit de o clasă decorată cu @NgModule (). Decoratorul @NgModule () este o funcție care are un singur obiect de metadate. Cele mai importante proprietăți sunt după cum urmează.

- declarations: componentele, directivele și conductele care aparțin acestui NgModule;
- exports: subsetul de declarații care ar trebui să fie vizibile și utilizabile în șabloanele componente ale altor NgModule;
- imports: Alte module ale căror clase exportate sunt necesare prin șabloanele componentelor declarate în acest NgModule;
- providers: Creatorii de servicii pe care acest NgModule contribuie la colectarea globală de servicii; acestea devin accesibile în toate părțile aplicației. (De asemenea, puteți specifica furnizorii la nivelul componentelor, care este adesea preferat.);
- bootstrap: vizualizarea principală a aplicației, numită componentă rădăcină, care găzduiește toate celelalte vizualizări ale aplicației. Doar NgModule rădăcină ar trebui să stabilească proprietatea bootstrap.

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
@NgModule({
```

```
    imports: [ BrowserModule ],
    providers: [ Logger ],
    declarations: [ AppComponent ],
  exports: [ AppComponent ],
  bootstrap: [ AppComponent ]
  })
  export class AppModule { }
```

1.12 **Crearea unui modul de aplicație**

Aplicația Angular este formată din module. Structura modulară vă permite să încărcați și să utilizați cu ușurință numai acele module care sunt direct necesare. Și fiecare aplicație are cel puțin un modul rădăcină. Prin urmare, vom crea un fișier nou în folderul src / app, pe care îl vom numi app.module.ts cu următorul conținut:

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule }   from '@angular/forms';
import { AppComponent }   from './app.component';
@NgModule({
  imports:      [ BrowserModule, FormsModule ],
  declarations: [ AppComponent ],
  bootstrap:    [ AppComponent ]
})
export class AppModule { }
```

Cu ajutorul directivelor de import, un număr de module de care avem nevoie sunt importate aici. În primul rând, acesta este modulul NgModule. Modulul BrowserModule este, de asemenea, necesar să lucreze cu browserul. Deoarece componenta noastră folosește un element de intrare sau un element de formă, permitem și modulul FormsModule. Apoi, componenta creată anterior este importată.

1.13 **Serviciul router**

Routerul Angular NgModule oferă un serviciu care vă permite să definiți o cale de navigare între diferitele stări de aplicații și să vizualizați ierarhiile din aplicație. Este modelat pe convențiile de navigare a browserului:

- Se introduce o adresă URL în bara de adrese, iar navigatorul navighează către o pagină corespunzătoare;
- Executarea unui click pe link-uri din pagină și browser-ul navighează către o pagină nouă;
- Click pe butoanele din spate și înainte ale browserului, iar navigatorul navighează înapoi și înainte în istoricul paginilor pe care le-ați văzut.

Routerul prezinta o hartă a URL-urilor cu vizualizările în loc de pagini. Când un utilizator efectuează o acțiune, cum ar fi clic pe un link, care ar încărca o pagină nouă în browser, routerul interceptează comportamentul browserului și afișează sau ascunde ierarhiile de vizualizare.
Routerul interpretează o adresă URL de link în conformitate cu regulile de navigare din vizualizarea aplicației dvs. și cu starea datelor.

Este podibila navigarea la vizualizări noi atunci când utilizatorul face clic pe un buton sau selectează dintr-o căsuță drop sau ca răspuns la un alt stimulent din orice sursă. Router-ul înregistrează activitatea în istoricul browser-ului, deci funcționează și butoanele înnapoi și înainte.

1.14 **Configurarea**

Deoarece TypeScript este folosit pentru a determina codul aplicației, se va crea, de asemenea, un nou fișier tsconfig.json în folderul rădăcină al proiectului. Acest fișier definește setările pentru compilatorul TypeScript. Dacă este utilizat TypeScript pentru a lucra cu Angular, atunci aceste două fișiere vor fi de fapt prezente în fiecare proiect. Și pot fi transferate de la proiect la proiect cu modificări minime. Deoarece aplicația va fi împărțită în mai multe fișiere separate, se va folosi constructorul de pachete web pentru a le construi.

2. ANALIZA COMPARATIVĂ VUEJS ȘI ANGULAR

2.1 Aplicatia Vue

2.2 Node Package Manager (NPM)

Este recomandatîă utilizarea NPM atunci când se construiesc aplicații mari pe Vue. Această opțiune funcționează excelent cu instrumente de construire precum Webpack și Browserify. Vue are, de asemenea, instrumente compatibile pentru utilizarea componentelor cu un singur fișier.

Comanda de instalare: **npm install vue** ea instaleaza ultima versiune stabila.

2.3 Instrumente pentru linia de comandă (CLI)

Vue.js furnizează instrucțiuni (CLI) pentru crearea rapidă a unui framework pentru aplicații ambițioase de o singură pagină. În doar câteva minute veți obține o configurație de lucru cu reincarcare a modulelor, analiza erorilor în timpul configuratiei a ansamblului de producție.

CLI este un instrument pentru cei familiarizați cu Node.js și instrumentele corespunzătoare de construire. Comanda de instalare: **npm install vue-cli**

```
"devDependencies": {
  "@vue/cli-plugin-babel": "^3.5.0",
  "@vue/cli-plugin-eslint": "^3.5.0",
  "@vue/cli-service": "^3.5.0",
```

Fig. 2.1. Este aratat că CLI a fost instalat

2.4 Crearea aplicației

Dezvoltarea aplicației în contextul unui spațiu de lucru VueJs. Un spațiu de lucru conține fișierele pentru unul sau mai multe proiecte. Pentru a crea un nou spațiu de lucru și un proiect inițial pentru aplicație se ruleaza comanda CLI: **vue create proiect-vue.**

În urma acestor comenzi, in dosarul ales se creează dinamic toate fișierele necesare pentru a începe crearea aplicației si componentul principal App.Vue. si main.js,

```
import Vue from 'vue'
import Vuex from 'vuex'
import App from './App.vue'
import Router from 'vue-router'
import router from './Router'
import store from '../store'
import vueResource from 'vue-resource'
import BootstrapVue from 'bootstrap-vue'
import 'bootstrap/dist/css/bootstrap.css'
import 'bootstrap-vue/dist/bootstrap-vue.css'
Vue.use(Vuex)
Vue.use(BootstrapVue)
Vue.use(vueResource)
Vue.use(Router)

Vue.config.productionTip = false

new Vue({
  render: h => h(App),
  components: { App },
  router :router,
  store,
}).$mount('#app')
```

Fig. 2.2. Fisierul main.js cu toate bibliotecile importate

O componenta vue este impărțită in 3 secțiuni: prima parte este integrarea codului html, secțiunea script in care se execută scriptele necesare și interacțiunea cu componentele și ultima secventa este dedicată pentru aplicarea stilurilor.

Sunt create urmatoarele componente necesare pentru dezvoltarea aplicației

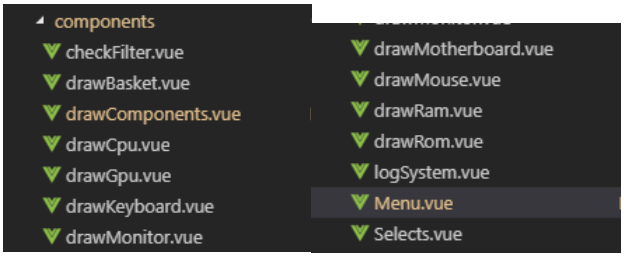


Fig. 2.3. Componentele create

Componenta Menu.vue este responsabilă pentru apariția barei Meniu in proiect si butoanele pentru interacțiune.

Apoi se creează componenta Select.vue care are rolul de a desena componenta pe care o alegem acestea sunt: draw(Cpu, Gpu, Keyboard, Monitor, Motherboard, Mouse, Rom).vue și drawRam.vue, aceste componente au conținut asemanator și sunt utilizate pentru a afișa continutul din JSON in card html pe pagina cu conținutul adecvat denumirii.

Pentru a primi datele și a le afișa in pagina este utilizat json-server din care sunt primite datele cu ajutorul plaginului pentru Vue.js ce oferă servicii pentru realizarea și primirea răspunsurilor (resource) și înscriem această informație primită într-un masiv de date.

```
created(){

this.$http.get('http://localhost:3000/ motherboard).then(function(data){
this.motherboard = data.body
})
},
```

```
<div v-for="item in motherboard">
 <b-card>
  <b-img :src="require('../' + item.src)" img-top tag="article"></b-img>
    <h4> {{ item.name}} </h4> <hr>
      <b-card-text>
      <p> Socket: {{ item.socket }} </p> <hr>
      <p> Memoryslot: {{ item.memoryslot }} </p><hr>
      <p class="price"> {{ item.price }} $</p><hr>
  </b-card-text>
```

```
    </b-card>
  </div>
```

Această secvență de cod este responsabila de a desena cardurile cu informația preluată din masiv cu ajutorul directivei v-for preluînd lincul imaginii, numele componentei, parametrii acesteia și prețul.

Apoi se creează butonul care adaugă in coș elementul selectat cu o verificare

```
<b-button variant="dark" @click="modifySetVar(item)" :disabled="(cpuFilter != '' && item.socket != cpuFilter) || (ramFilter != '' && ramFilter != item.memoryslot)">Add</b-button>
```

La apăsarea acestui buton are loc verificarea pentru coinciderea socketului si tipului de memorie ram aleasă, dacă elementele nu coincid are loc conditia :disabled si dezactivează butoanele pentru restul componentelor, apoi se executa directiva v-on click care trimite datele despre componentă în store si adaugarea produselor în filtru și in coș.

2.5 **Mecanismul Store**

Vuex oferă un mecanism de "implementare" a depozitului în toate componentele copilului componentei rădăcină, care are opțiunea store (inclusă în Vue.use (Vuex)).

```
import Vuex from 'vuex'
import Vue from 'vue'
Vue.use(Vuex)
const store = new Vuex.Store({
```

Prin store specificam opțiunii de stocare din instanța rădăcină, oferim accesul la depozit în toate componentele copilului prin intermediul this.$store.

Pentru aceasta initializam constructorul methods

```
methods: {
    modifySetVar(motherboard) {
      this.$store.commit('setSocket', motherboard.socket)
      this.$store.commit('setMemory', motherboard.memoryslot)
      this.$store.commit('addProduct', { type: this.type, component: { name: motherboard.name, price: motherboard.price } })
    }
  },
```

În componenta checkFilter pastrăm datele la selectarea unui element principal din cele trei (motherboard, ram, cpu) înscriem valorile primului element ales in variabile store pentru a cunoaște ce parametri căutam și pentru a putea interacționa cu ele si a modifica starea acestora este adaugat butonul X care șterge valoarea parametrului ales.

```
<h4> Cpu Socket: <span> {{ socketFromStore }}  </span><button class=" btn-danger"
@click="clearSocket">X</button> </h4>
    <h4> Ram Type: <span> {{ getMemory }}  </span><button class=" btn-danger"
@click="clearRam">X</button></h4>
```

În felul următor ale loc primirea parametrilor pentru componentelor alese.

```
socketFromStore () {
    return this.$store.getters.getSocket
  },
  getMemory(){
    return this.$store.getters.getMemory
  },
},
```

Și ștergerea unui parametru pentru posibilitatea de a alege altul

```
methods: {
  clearSocket(){
    this.$store.commit('setSocket', ' ')
  },
  clearRam()
    this.$store.commit('setMemory', ' ')
  }
}
```

Desenarea coșului este realizată de componenta drawBasket.vue. In coș este creată o tabela cu coloane în care se va înscrie informația

Type	Name	Quantity	Price

Total Price: 0

Fig 2.4. Interfața coșului

În coș extragem informația despre cumpărăturile efectuate si organizăm in tabela.

```
<tbody v-for="(item, index) in basket">
    <tr>
      <th> {{item.type}}</th>
      <td>{{item.name}}</td>
      <td>{{item.quantity}}</td>
      <td>{{item.price}}</td>
    <b-button size="sm" class="btn btn-danger" @click="delet(index)">X</b-button>
      </tr>
 </tbody>
```

Directiva parcurge masivul de componente și le înscrie in coș unde fiecare item corespunde cu coloana in care trebuie sa fie afișat, iar butonul X este utilizat pentru a elimina elementul din cos.

```
delet(i){
      let basket = this.basket
      basket.splice(i, 1);
      this.$store.commit('addProduct', basket)
   },
```

La adăugare in coș se execută verificarea in scriptul store care controleaza dacă avem deja un element de acest tip ales atunci se incrementează cantitatea cu 1.

```
addProduct(state, something){
    let type = something.type;
    let product = something.component;
    if (state.basket.some(el => el.name === product.name)) {
      const key = state.basket.findIndex(el => el.name === product.name);
      state.basket[key].quantity = state.basket[key].quantity + 1;
    } else {
      state.basket.push({ type: type, name: product.name, quantity: 1, price: product.price
});
      localStorage.setItem('key',JSON.stringify(state.basket));
    }
```

Și totodata are loc trimiterea elementului in coș si localStorage, tot în această secvență este utilizată incrementarea cantității care este folosită pentru a calcuca prețul total a componentelor in coș si afișarea acestuia.

```
getTotalPrice(state){
    let total = 0
    state.basket.forEach(el => {
      total += el.price * el.quantity
    })
    return total
  }
```

După selectarea componentelor necesare si deja adaugate in cos se efectuează procesul de cumpărare, pentru aceasta avem nevoie de datele personale în cazul dat numele, numarul de telefon și e-mail, insă pentru a fi convinși ca datele sunt valide aplicam o verificare .

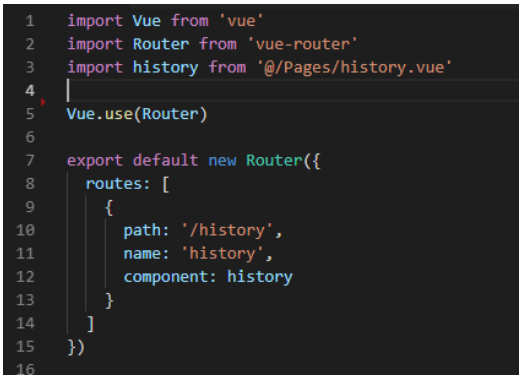
```
telState() {
        return this.tel.slice(0, 1) == "0" && this.tel.length > 8 && this.tel.length < 10 ||
           this.tel.slice(0, 4) == "+373" && this.tel.length > 11 && this.tel.length < 13 ?
true : false
      },
```

Numarul trebuie sa se inceapa cu 0 sau +373 si lungimea sa corespunda încă 8 cifre ce vor urma prefixul. Verificarea postei consta in faptul ca trebuie sa existe caractere innaintea semnului conventional @ o combinatie de litere, semnul punct si ultima combinatie de litere intre 2 la 4 litere.

```
emailState() {
        var reg = /^([A-Za-z0-9-.])+@([A-Za-z0-9-.])+.([A-Za-z]{2,4})$/;
        if (reg.test(this.email) == false) {
           return false
        } else {
           return true
        }
```

2.6 Utilizarea Routerului

Toată informația despre cumpărături se va afișa în pagina prin routing, unde datele sunt preluate din local storage



```
import Vue from 'vue'
import Router from 'vue-router'
import history from '@/Pages/history.vue'

Vue.use(Router)

export default new Router({
  routes: [
    {
      path: '/history',
      name: 'history',
      component: history
    }
  ]
})
```

Fig 2.5. Componenta router

```
historry(){
        let historry = []
         historry = JSON.parse(localStorage.getItem('orders'))
         console.log(historry)
         return  historry
   }
```

2.7 Aplicatia Angular

2.8 Instalarea frameworkului

Angular ajută la crearea aplicațiilor moderne pentru web, mobil sau desktop. Pentru a crea o aplicatie Angular simpla vom folosi instrumentul CLI Angular pentru a accelera dezvoltarea, respectând în același timp recomandările ghidului stilurilor care beneficiază de fiecare proiect Angular. Înainte de a începe, ne asiguram ca in mediul de dezvoltare utilizat avem incluse Node.js și un manager de pachete npm.

2.9 Node Package Manager (NPM)

Angular necesită Node.js versiunea 8.x sau 10.x. Pentru a verifica versiunea, introducem comanda **nod -v** într-o fereastră terminal / consola. Pentru a obține Node.js, Instalam serverul Node.js și managerul de pachete npm dacă sunt absente pe mașina gazdă. Pentru instalare, se utilizeaza programul de configurare node.js. Împreună cu serverul, se instalează și npm. În

același timp, nu sunt necesare cunoștințe speciale pentru a lucra cu NodeJS și npm. După instalarea instrumentelor necesare, se creaza o aplicație simplă. Pentru a face acest lucru, definim dosarul aplicației de pe hard disk. Indicam calea proiectului. În acest directoriu, se creaza un nou fișier package.json.

2.10 Instrumente pentru linia de comandă (CLI)

Se utilizeaza CLI Angular pentru a crea proiecte, a genera codul aplicațiilor și a bibliotecii și a efectua o varietate de sarcini de dezvoltare în curs de desfășurare, cum ar fi testarea, gruparea și implementarea. Instalam Angular CLI la nivel global. Pentru a instala CLI folosind npm, se deschide o fereastră terminal / consola și introducem următoarea comandă: **npm install -g @angular/cli**

Fig 2.6. Angular CLI este instalat

2.11 Crearea aplicatiei

Dezvoltarea aplicației în contextul unui spațiu de lucru Angular. Un spațiu de lucru conține fișierele pentru unul sau mai multe proiecte. O bibliotecă de tip end-to-end (e2e).Pentru a crea un nou spațiu de lucru și un proiect inițial se ruleaza comanda CLI: **ng new ang-pr.**

Crearea unei componente in Angular **ng g c <numele>**

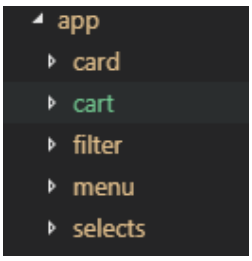


Fig.2.7 Componentele create

Aplicația pe angular are același scop ca aplicacția pe VueJs ca structura și funcțional. Se creează meniul similar aplicației pe Vue

```
<nav class="navbar navbar-dark bg-dark">
  <a class="navbar-brand"><img src="../../assets/img/logo.png"></a>
  <form class="form-inline">
```

```
    <button class="btn my-2 my-sm-0"><img src="../../assets/img/basket.png"
alt=""></button>
  </form>
</nav>
```

Sunt urmați aceeași pași și se creează selectul in care se vor pastra tipurile de componente la acegerea carora vor fi desenate cardurile corespunzatoare.

```
<div>
    <h3>Please select your component</h3> <br>
  <select class="custom-select custom-select-lg mb-3" (change) =
"drawCard($event.target.value)" >
    <option *ngFor="let item of data" ng-model="">{{item}}</option>
  </select>
</div>
```

Cu ajutorul directivei (change) se face referință la dataService și preluarea de acolo a valorilor care vor fi afișate în selecturi.

```
drawCard(key: string){
     this.dataService.setKey(key)
     this.store.getComponents();
}
```

Acestea lucreaza dupa ce a fost initializat angular si serviciile Store

```
import { Component, OnInit } from '@angular/core';
import { DateJsonService } from '../date-json.service';
import { StoreService } from '../store.service';
                              …
ngOnInit() {
   this.dataService.setKey('components')
   this.dataService.getData().subscribe(data => this.data = data);
   this.dataService.setKey('motherboard');
   this.store.getComponents();
```

Inițial în select este ales comonentul Motherboard și sunt desenate cardurile pentru el. Crearea cardului este efectuata conform alegerii obiectului dorit din componenta selects prin directiva *ngFor si inscriem in card informația primita.

```
<div class="card m-2" style="width: 18rem;" *ngFor="let component of components">
    <img src="{{component.img}}" class="card-img-top" alt="...">
    <h5 class="card-title">{{ component.name }}</h5>
    <div class="card-body text-center">
     <ul class="list-group list-group-flush">
      <li class="list-group-item" *ngFor="let item of fields"> {{ component[item].field }} : {{ component[item].value }}</li>
      <li class="list-group-item"> Price: {{ component.price }} </li>
```

Aici este integrată verificarea, daca proprietațile componentului ales nu coincide cu alte componente să nu poată fi posibilă cumpararea acestora.

```
<button *ngIf="type == 'motherboard'" class="btn btn-danger mt-2" (click)="add(component)" [disabled]="(cpuFilter != '' && cpuFilter != component.socket.value) || (ramFilter != '' && ramFilter != component.memoryslot.value)">Buy</button>
```

Cu ajutorul directivei (click) adaugam elementul selectat in cos si dacă acesta are component de verificare se adaugă și aceasta pentru a fi utilizata ulterior.

```
add(component) {
   this.setFilter(component);
   this.store.addToBasket(component, this.type);
  }
```

Verificăm dacă proprietațile alese satisfac condiția și atunci le pastram pentru verificarea ulterioara.

```
setFilter(key) {
   if(this.type == 'cpu') {
     this.store.setFilter('cpu', key.socket.value)
   } else {
     if(this.type == 'ram') {
       this.store.setFilter('ram', key.memoryslot.value)
```

```
} else {
  this.store.setFilter('motherboard', key.socket.value, key.memoryslot.value)
}
```

Pentru a ține cont de proprietățile alese dorite pastrăm pe pagina aceste date cu posibilitatea de a amîna alegerea și de a modifica parametrii doriți

```
<li class="list-group-item">
    CPU Filter: {{ cpuFilter }}
    <button class="btn btn-danger float-right" (click)="removeFilter('cpu', cpuFilter)">x</button>
  </li>
  <li class="list-group-item">
    RAM Filter: {{ ramFilter }}
    <button class="btn btn-danger float-right" (click)="removeFilter('ram', ramFilter)">x</button>
```

Cînd dorim sa eliminam filtrul de verificare ne referim la store si cerem valorile

```
removeFilter(type: string, value: string) {
    this.store.removeFilter(type, value);
  }
```

Astfel are loc stergerea

```
removeFilter(type: string, value: string) {
    if (type === 'cpu') {
      this.cpuFilter.next(' ');
    } else {
      this.ramFilter.next(' ');
    }
  }
```

Acum se creează componenta cart care are rolul de a afisa cosul si de a pastra in el piesele dorite selectate pentru a le procura. Se creeaza o tabela in care se înscriu elementele procurate

```
<tr *ngFor="let item of basket">
      <td *ngIf="item.name">
        <div>{{ item.type }}</div>
      </td>
      <td *ngIf="item.name">
        <div>{{ item.name }}</div>
      </td>
      <td *ngIf="item.name">{{ item.quantity }}</td>
      <td *ngIf="item.price">{{ item.price }} $
```

<button class="btn btn-danger float-right"

(click)="removeItem(item.name)">x</button>

Tot aici este butonul in dreptul fiecarui element pentru al îl elimina din coș si de alege altă opțiune dorită.

{{ totalPrice }} $

this.store.currentTotalPrice.subscribe(price => this.totalPrice = price);

2.12 **Componenta store**

În store se primesc datele din diferite componente pentru a fi repartizate si folosite în componentele necesare și pentru a face manipulațiile dorite.

```
getTotalPrice() {
  let temp = 0;
  this.basket.forEach(component => {
    temp += component.quantity * component.price;
  });
  this.totalPrice.next(temp);
 }
```

Astfel are loc calcularea prețului total care se va afisa in coș.

Și tot in store se pastrează variabila în care se primește datele despre cumparaturi si permite afișarea acestora din localStorage in coș.

```
constructor(private data: DateJsonService) {
  this.basket = JSON.parse(localStorage.getItem('basket')) || this.basket;
  this.sourceBasket.next(this.basket);
  this.getTotalPrice();
 }
```

După ce au fost alese piesele dorite ele pot fi procurate, la aceasta etapă cumparatorul este intrebat datele personale.

<form>

<input type="text" class="form-control mb-2" (keyup)="verifyAcces()"

placeholder="Name" name="name" [(ngModel)]="owner.name" required>

<input type="text" class="form-control mb-2" (keyup)="verifyAcces()"

placeholder="Surname" name="surname" [(ngModel)]="owner.surname" required>

<input type="number" class="form-control mb-2" (keyup)="verifyAcces()"

placeholder="Phone Number" name="phone" [(ngModel)]="owner.phone" required>

<input type="text" class="form-control mb-2" (keyup)="verifyAcces()" placeholder="E-Mail" name="email" [(ngModel)]="owner.email" email>

</form>

2.13 **Utilizarea Routerului**

De asemenea ca si in Vue a fost utilizat Routerul pentru a afisa datele din localStorage in pagina

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

const routes: Routes = [
  { path: 'history', component: hystory }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Fig. 2.8. Indicarea adresei pentru Router

CONCLUZII

În această lucrare au fost utilizate două frameworkuri VueJs și Angular, ele au facut o mare concurența între frameworkuri. Vue este condus în întregime de comunitatea open source, în timp ce Angular de o parte semnificativă a angajaților Google și Facebook.

Au fost elaborate două alicații pentru a arăta asemănarile și deosebirile acestora, cea mai observabila deosebire este dimensiunea construcțiilor. Angular este o alegere bună pentru companiile cu echipe și dezvoltatori care folosesc deja TypeScript, însă curba de studiere este mult mai abrutpă. Vue este mai nou, fără sprijinul unei companii importante. Însa, sa dovedit a fi bun în ultimii ani ca un concurent puternic pentru Angular. Acest lucru poate juca un rol cu o mulțime de giganti chinezi ca Alibaba și Baidu luând Vue drept principalul lor front-end JavaScript. Vue ar trebui să fie alegerea potrivita. dacă preferați simplitatea, dar și flexibilitatea. Vue este mai ușor sintactic ca Angular și susține multe sisteme de construcție diferite, fără a limita dezvoltatorii în ce structură să utilizeze pentru aplicație. Mulți programatori se bucură de această libertate, deși există aceia care preferă să aibă singura modalitate corectă de a construi o aplicație.

Tot necesarul pentru a începe Vue este de a cunoaște HTML și JavaScript. Cu aceste abilități de bază pot fi construite aplicații non-triviale. Dar, complexitatea sistemului Angular se datorează în mare parte faptului că se concentrează doar pe aplicații complexe mari.

La aceste aplicații au fost folostite aceleași componente, metode, biblioteci pentru a fi un ghid de ajutor pentru o persoana ce doreste să înceapă programarea pe unul din aceste frameworkuri, pentru a scoate la iveală deosebirile, unora le place cînd componenta conține în interiorul său html, scripturi și css atunci alegerea este Vue, iar daca prefera fișiere diferite atunci alegerea este Angular. In aplicații au fost folosite aceleași directive, deoarece ambele frameworkuri sunt asemănătoare.

Toate obiectivele înnaintate ca scop au fost îndeplinite:

- Analiza resurselor informative: literatură de specialitate, comunitățile și tutorialele;
- Analiza generală a VueJs;
- Analiza generală a Angular;
- Analiza comparativă a tehnologiilor și posibilitațile acestora;
- Proiectarea aplicației pe ambele Frameworkuri.

Aplicațiile au fost elaborate cu succes și pot fi folosite ca șablon pentru a creea alte aplicații de tipul dat în timp cit mai scurt utilizind componentele deja create, sau de a fi implementate in alte domenii combinate.

Răspunsul la dezbaterea Angular vs Vue este că nu există o alegere absolută corectă, pe care probabil că multi asteapta. Fiecare dintre aceste biblioteci are propriile avantaje și dezavantaje. Pe baza proiectului care trebuie elaborat și a cerințelor puse individuale, una dintre ele va fi mai potrivită decât cealalta. Este întotdeauna esențial să se analizeze posibilitatile lor si cerintele puse pentru a alege varianta optimală.

BIBLIOGRAFIE

1. Jacob Schats. *How we do Vue: one year later.* [on-line]. [citat 10.05.2019]. Disponibil: https://about.gitlab.com/2017/11/09/gitlab-vue-one-year-later/
2. *What is Vue.js?* . [on-line]. [citat 10.05.2019]. Disponibil: https://vuejs.org/v2/guide/
3. *Vue.js для сомневающихся.* [on-line]. [citat 10.05.2019]. Disponibil: https://habr.com/ru/post/329452/
4. *Vue-cli.* [on-line]. [citat 14.05.2019]. Disponibil:https://www.npmjs.com/package/vue-cli
5. *Comparison with Other Frameworks*. [on-line]. [citat 16.05.2019]. Disponibil: https://vuejs.org/v2/guide/comparison.html
6. *Getting started*. [on-line]. [citat 15.05.2019]. Disponibil: https://router.vuejs.org/guide/
7. *Enter/Leave & List Transitions* [on-line]. [citat 16.05.2019]. Disponibil: https://vuejs.org/v2/guide/transitions.html
8. *VueJs Directives* [on-line]. [citat 15.05.2019]. Disponibil: https://www.w3schools.com/whatis/whatis_vue.asp
9. *Почему 43% фронтенд-разработчиков хотят изучить Vue.js* [on-line]. [citat 16.05.2019]. Disponibil: https://medium.com/devschacht/%D0%BF%D0%BE%D1%87%D0%B5%D0%BC%D1%83-%D1%84%D1%80%D0%BE%D0%BD%D1%82%D0%B5%D0%BD%D0%B4-%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D1%87%D0%B8%D0%BA%D0%B8-%D1%85%D0%BE%D1%82%D1%8F%D1%82-%D1%83%D1%87%D0%B8%D1%82%D1%8C-vue-js-63aa3b456dfa
10. *The Vue Instance* [on-line]. [citat 16.05.2019]. Disponibil: https://vuejs.org/v2/guide/instance.html
11. *Почему мы используем Vue фреймворк.*[on-line]. [citat 17.05.2019]. Disponibil: https://www.vis-design.com/ru/blog/pochemu-vue-a-ne-react-ili-angular.htm
12. *Что такое Framework?* [on-line]. [citat 17.05.2019]. Disponibil: https://glossary-internet.ru/terms/F/3201/
13. *Getting Started* [on-line]. [citat 18.05.2019]. Disponibil: https://angular.io/guide/quickstart
14. *Structural directives* [on-line]. [citat 18.05.2019]. Disponibil: https://angular.io/guide/structural-directives
15. *Angular directives* [on-line]. [citat 18.05.2019]. Disponibil: https://www.w3schools.com/angular/angular_directives.asp
16. *Введение в Angular*. [on-line]. [citat 19.05.2019]. Disponibil: https://metanit.com/web/angular2/1.1.php
17. *Routing.* [on-line]. [citat 19.05.2019]. Disponibil: https://angular.io/tutorial/toh-pt5
18. *Services* [on-line]. [citat 19.05.2019]. Disponibil: https://angular.io/tutorial/toh-pt4
19. *React vs Angular vs VueJs.* [on-line]. [citat 21.05.2019]. Disponibil: https://medium.com/front-end-weekly/react-vs-angular-vs-vue-js-a-complete-comparison-guide-d16faa185d61
20. *Comparison with Other Frameworks.* [on-line]. [citat 21.05.2019]. Disponibil: https://vuejs.org/v2/guide/comparison.html
21. *React vs Angular vs VueJs*. [on-line]. [citat 21.05.2019]. Disponibil: https://dzone.com/articles/react-vs-angular-vs-vuejs-a-complete-comparison-gu
22. *What to choose in 2019?*. [on-line]. [citat 21.05.2019]. https://medium.com/@TechMagic/reactjs-vs-angular5-vs-vue-js-what-to-choose-in-2018-b91e028fa91d
23. *Part 4: Working with the frameworks* [on-line]. [citat 21.05.2019]. https://www.codeinwp.com/blog/angular-vs-vue-vs-react/

Anexa 1. Diagrama ciclului de viață