**ANALIZA COMPARATIVĂ A FRAMEWORKURILOR ANGULAR SI VUEJS PENTRU ELABORAREA APLICATIILOR WEB  
TEZA DE AN**

**Autor:**

Student al grupei IS31Z

Popa Dan

\_\_\_\_\_\_\_\_\_\_\_\_

**Conducători științifici:**

Sergiu CHILAT

lector univ., magistru

\_\_\_\_\_\_\_\_\_\_\_\_

**BĂLȚI, 2019`**

**Introducere**

Framework (din engleza framework - "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. Frameworcurile 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. Pentru limbile de programare, cum ar fi Java, PHP, Ruby și altele, s-au dezvoltat sau se dezvoltă mai multe framework-uri. Proiecte deosebit de populare pentru Java și PHP. Eu ma voi referi la VueJs si Angular, deoarece sunt tehnologii acluale si sunt unele din cele mai utilizate in spatial web. Pentru cele mai populare framework-uri, se creează comunități, se elaborează manuale și documentație.

Pentru funcționarea site-ului și promovarea acestuia, framework-uri pot fi chiar mai eficiente decât sistemele de gestionare a conținutului. Spre deosebire de cele din urmă, framework-uri 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** consta in cercetarea Frameworkurilor VueJs si Angular utilizate la crearea unei aplicatii web (sit) in care vor fi incadrate componentele acestor frameworkuri

*Obiectivele lucrării:*

1. Analiza resurselor informative: literatură de specialitate, comunitățile și tutorialele;
2. Analiza generală a VueJs
3. Analiza generală a Angular
4. Analiza tehnologiilor și posibilitatile acestora
5. Proiectarea aplicației pe ambele Frameworkuri
6. **VueJs Framework**



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.  
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 avem un element cu selectorul de care avem nevoie

Noi derivăm datele de la stat în sintaxa {{ }}. Aici avem expresia ternară obișnuită

input este asociat cu valoarea message a obiectului de stare folosind directiva v-model.

**Vue.js concepte**

Principalele concepte ale Vue sunt:

* Constructor
* Componentele
* Directive
* Router

**Constructor**

Lucrul cu Vue.js începe cu crearea unei noi instanțe new Vue. În el, avem un element pe care îl urmărește Vue. In template se selecteaza un element în care va fi afișat Vue. In 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(){},  
 }

* 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

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Directive**

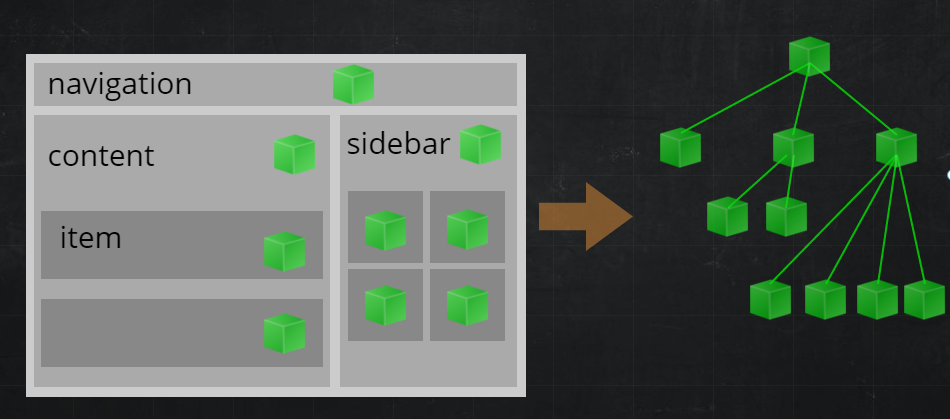
Directivele – atribute speciale pentru adăugarea elementelor html functionalitati noi.   
Iata exemple de directive care sunt similare cu cele din Angular:

* V-bind — Dinamic face legatura cu unul sau mai multe atribute
* v-if — conditie de modificare a elementului
* V-else — este blocul “else” pentru “if”
* V-for — trece masivul de obiecte ciclic
* V-model — leaga starea cu elementul input
* V-on — leaga ascultatorul de evenimente cu elementul
* V-once — deneseaza elementul doar la inceput si nu il mai urmareste
* V-show — schimba afisarea elementului, schimbind proprietatea CSS display
* V-text — reinnoieste elementul textContent

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

**Componente**

**C**omponentele 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, am împărțit aplicația noastră în părți independente și am obținut 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).

**Tranzitii**

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.

**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. **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](https://ro.wikipedia.org/wiki/Dezvoltare_web) [cu sursă deschisă](https://ro.wikipedia.org/wiki/Open-source) bazată pe limbajul [TypeScript](https://ro.wikipedia.org/wiki/TypeScript). Proiectul este dezvoltat de Echipa Angular de la [Google](https://ro.wikipedia.org/wiki/Google) și de o comunitate de utilizatori individuali și companii. Angular este o rescriere completă, de către aceeași echipă, a frameworkului [AngularJS](https://ro.wikipedia.org/w/index.php?title=AngularJS&action=edit&redlink=1). 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 cadru 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.

**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 {

}

**Directive**

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 directivei.
* inputs - Enumeră setul de proprietăți de intrare legat de date pentru o directiva.
* 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 directivă.
* 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

**Module**

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](https://angular.io/api/core/NgModule) } from '@angular/core';

import { [BrowserModule](https://angular.io/api/platform-browser/BrowserModule) } from '@angular/platform-browser';

@[NgModule](https://angular.io/api/core/NgModule)({

imports: [ [BrowserModule](https://angular.io/api/platform-browser/BrowserModule) ],

providers: [ Logger ],

declarations: [ AppComponent ],

exports: [ AppComponent ],

bootstrap: [ AppComponent ]

})

export class AppModule { }

**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.

**Routing**

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 aplicația dvs. Este modelat pe convențiile convenționale de navigare a browserului:

* Introduceți o adresă URL în bara de adrese, iar navigatorul navighează către o pagină corespunzătoare.
* Faceți clic pe link-uri din pagină și browser-ul navighează către o pagină nouă.
* Faceți clic 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. Puteți naviga 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.

**Configurarea**

Deoarece TypeScript este folosit pentru a determina codul aplicației, vom crea, de asemenea, un nou fișier tsconfig.json în folderul rădăcină al proiectului. Acest fișier definește setările pentru compilatorul TypeScript. Dacă folosim 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 noastră va fi împărțită în mai multe fișiere separate, vom folosi constructorul popular de pachete web pentru a le construi. Prin urmare, definim un alt fișier webpack.config.js în dosarul rădăcină al proiectului.

1. **Analiza Comparativa**

**Vue**

**Instalarea**

! Atentie Vue nu suportă IE8 și este mai mică deoarece utilizează caracteristici ECMAScript 5 care nu pot fi emulate în IE8. În caz contrar, toate browserele compatibile ECMAScript 5 sunt acceptate. Ultima versiune este 2.6.10.

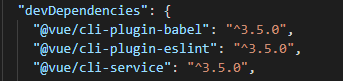
**NPM**

Este recomandata utilizarea NPM atunci când construiți 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.

**Instrumente pentru linia de comandă (CLI)**

Vue.js furnizează instrucțiuni (CLI) pentru crearea rapidă a unui cadru 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.

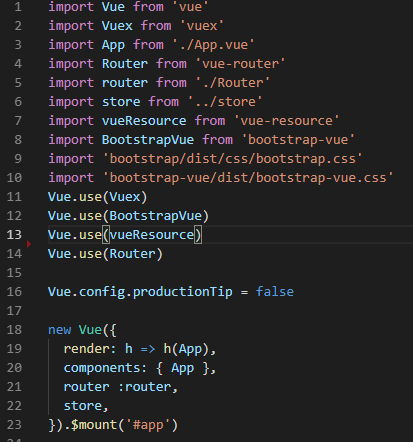
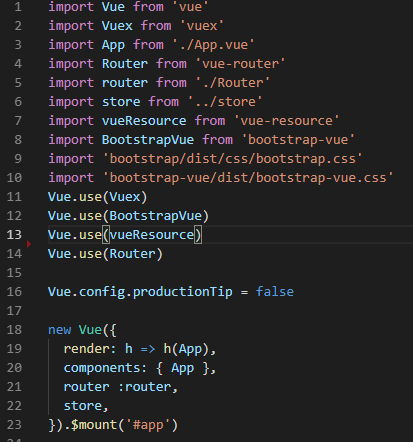
CLI este un instrument pentru cei familiarizați cu Node.js și instrumentele corespunzătoare de construire. Comanda de instalare: **npm install vue-cli**



**Crearea aplicatiei**

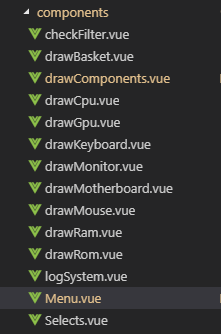
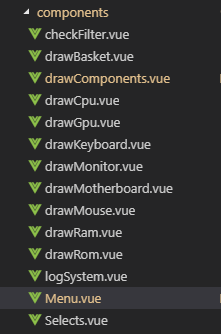
Dezvoltați aplicații în contextul unui spațiu de lucru VueJs. Un spațiu de lucru conține fișierele pentru unul sau mai multe proiecte. Pentru a crea un nou spațiu de lucru și un proiect inițial pentru aplicație se ruleaza comanda CLI: **vue create proiect-vue.**

In urma acestor comenzi, in dosarul ales se creeaza dinamic toate fisierele necesare pentru a incepe crearea aplicatiei si component principal App.Vue. si main.js,

O componenta vue este impartita in 3 sectiuni: prima parte este integrarea codului html, sectiunea script in care se executa scriptele necesare si interactiunea cu componentele si ultima secventa este deditata pentru aplicarea stilurilor.

Am creat urmatoarele componente necesare pentru dezvoltarea aplicatiei

Componenta Menu.vue este responsabila pentru aparitia barei Meniu in proiect si butoanele pentru interactiune.

Apoi am creat 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 si drawRam.vue, aceste componente au continut asemanator si sunt utilizate pentru a afisa continutul din JSON in card html pe pagina cu continutul adecvat denumirii.

Pnentru a primi datele si a le afisa in pagina am utilizat json-server din care am primit informatia cu ajutorul plaginului pentru Vue.js ce oferă servicii pentru realizarea și primirea răspunsurilor (resource) si inscriem aceasta informatie primita intr-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>*

Aceasta secventa de cod este responsabila de a desena cardurile cu informatia preluata din masiv cu ajutorul directivei v-for preluind lincul imaginii, numele componentei, parametrii acesteia si pretul.

Apoi se creeaza butonul care adauga in cos 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 apasarea acestui buton are loc verificarea pentru coinciderea socketului si tipului de memorie ram aleasa daca elementele nu coincid are loc conditia :disabled si dezactiveaza butoanele pentru restul componentelor, apoi se executa directiva v-on click care trimite datele despre componenta in store si adaugarea produselor in filtru si in cos.

**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 } })*

*}*

*},*

In componenta checkFilter pastram datele la selectarea unui element principal din cele trei (motherboard, ram, cpu) inscriem valorile primului element ales in variabile store pentru a cunoaste ce parametri cautam si pentru a putea interactiona cu ele si a modifica starea aceestore este adaugat butonul X care sterge 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>*

In felul urmator ale loc primirea parametrilor componentelor alese

*socketFromStore () {*

*return this.$store.getters.getSocket*

*},*

*getMemory(){*

*return this.$store.getters.getMemory*

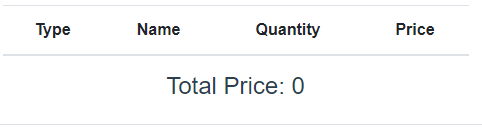
*},*

*},*

Si stergerea unui parametru pentru posibilitatea de a alege

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

Desenarea cosului este realizata de componenta drawBasket.vue. In cos este creata o tabela cu coloane in cere se va inscrie informatia



In cos extragem informatia despre cumparaturile efectuate si organizam 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 si le inscrie in cos unde fiecare item corespunde cu coloana in care trebuie sa fie afisat, 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 adaugare in cos se executa verificarea in scriptul store care controleaza daca avem deja un element de acest tip ales atunci incrementam 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));*

*}*

Si totodata are loc trimiterea elementului in cos si localStorage tot in aceasta secventa este utilizata incrementarea cantitatii care este folosit apentru a calcuca pretul total a componentelor in cos si afisarea acestuia.

*getTotalPrice(state){*

*let total = 0*

*state.basket.forEach(el => {*

*total += el.price \* el.quantity*

*})*

*return total*

*}*

Dupa selectarea componentelor necesare si deja adaugate in cos se efectueaza procesul de cumparare, pentru aceasta avem nevoie de datele personale in cazul dat numele, numarul de telefon si e-mail si pentru a fi convinsi ca datele sunt corecte aplicam verificari .

*telState() {*

*return this.tel.slice(0, 1) == "0" && this.tel.length > 8 && this.tel.length < 10 ||*

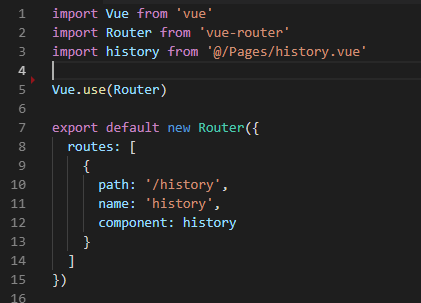
*this.tel.slice(0, 4) == "+373" && this.tel.length > 11 && this.tel.length < 13 ? true : false*

*},*

Numarul trebuie sa se inceapa cu 0 sau +373 si lungimea sa corespunda inca 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  
 }  
 },*

Iar toata informatia despre cumparaturi se va afisa in pagina prin routing



**Angular**

**Instalare**

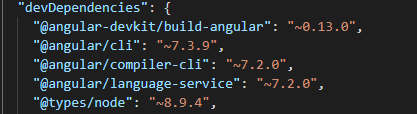
Angular ajută la crearea aplicațiilor moderne pentru web, mobil sau desktop. Pentru a crea o aplicatie Angular simpla vom folosi instrumentul CLI Angular pentru a accelera dezvoltarea, respectând în același timp recomandările ghidului stilurilor care beneficiază de fiecare proiect Angular. Înainte de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

**Node.js**

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.

**CLI**

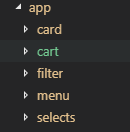
Se utilizeaza CLI Angular pentru a crea proiecte, a genera codul aplicațiilor și a bibliotecii și a efectua o varietate de sarcini de dezvoltare în curs de desfășurare, cum ar fi testarea, gruparea și implementarea. Instalam Angular CLI la nivel global. Pentru a instala CLI folosind npm, se deschide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



**Crearea** **aplicatiei**

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

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



Aplicatia pe angular are același scop ca aplicactia pe VueJs ca structura si functional.  
Am creat același meniu

*<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>*

Acum urmam aceeasi pasi si creem 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) ne referim la dataService si preluam de acolo valorile care vor fi afisate in 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();*

}

Initial in select este ales comonentul Motherboard si sunt desenate cardurile pentru el.

Crearea cardului este efectuata conform alegerii obiectului dorit din componenta selects prin directiva \*ngFor si inscriem in card informatia 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 am integrat verificarea daca proprietatile componentului ales nu coincide cu alte componente sa nu poata fi posibila 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 directive (click) adaugam elementul selectat in cos si daca acesta are component de verificare se adauga si aceasta pentru a fi utilizata ulterior.

*add(component) {*

*this.setFilter(component);*

*this.store.addToBasket(component, this.type);*

*}*

Verificam daca proprietatile alese ne satisfac 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 tine cont de proprietatile alese dorite pastram pe pagina aceste date cu posibilitatea de a amina alegerea si de a modifica parametrii doriti

*<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>*

Cind 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 creem 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 inscriem 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>*