**Scopul lucrarii** consta in cercetarea Frameworkurilor VueJs si Angular utilizate la crearea unei aplicatii web (sit) in care vor fi incadrate componentele acestor frameworkuri

1. **VueJs Framevork**



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, Sina Weibo etc. Este inclus în nucleul Laravel și PageKit. Recent, sistemul de gestionare a depozitelor gratuite GitLab a trecut, de asemenea, la Vue.js.

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. Pe de altă parte, 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:

app.js  
var demo = new Vue({   
 el: '#demo',   
 data: {   
 message: 'Hello, World!'   
 }   
})

cream o nouă instanță prin new Vue

în el, definim ce element este urmărit

în data avem un obiect de stare

**app.html**

<div id="demo">   
 <p>{{message ? message : "no content"}}</p>   
<input v-model="message"> </div>

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.

**1.1. Vue.js concepte**

Principalele concepte ale Vue sunt:

* Constructor
* Componentele
* Directive
* Router

Prima versiune a avut încă filtre, dar, din câte știu, în versiunea curentă sunt considerate depreciate.

**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.

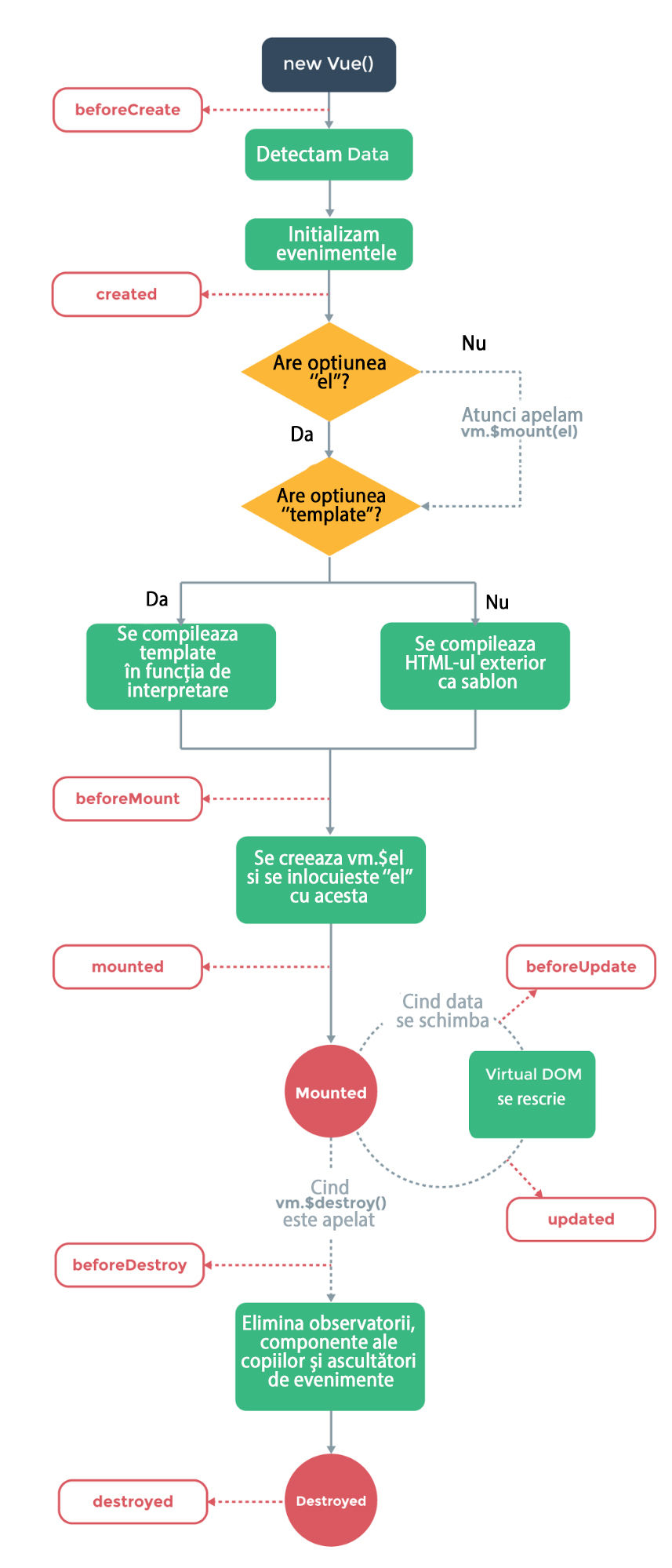
În methods se pot distinge următoarele metode și metode personalizate ale ciclului de viață Vue:

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**

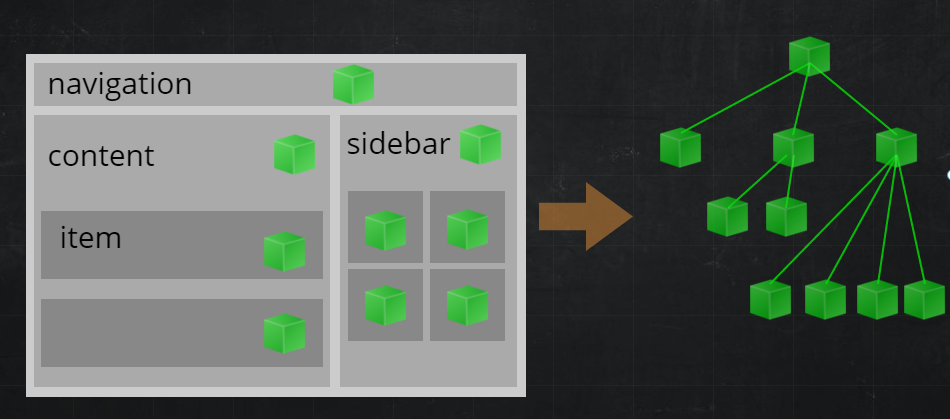
Directivele – atribute speciale pentru adăugarea elementelor html functionalitati noi.   
Iata exemple de directive care sunt similare cu cele din Angular:

* 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-pre — nu compileaza elementul si copii lui
* 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. Creați un fișier cu extensia .vue și scrieți stiluri, modele și logică acolo. Și puteți scrie 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) dacă se dorește, putem scrie aplicații pe Angular folosind limbi precum Dart sau JavaScript. Cu toate acestea, TypeScript este încă limba principală pentru Angular. 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 unghiulară 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 {

}

**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.

Dacă routerul determină că starea actuală a aplicației necesită o anumită funcționalitate și modulul care o definește nu a fost încărcat, routerul poate încărca leneș modul la cerere.

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.

Pentru a defini regulile de navigare, asociați căile de navigare cu componentele dvs. O cale utilizează o sintaxă asemănătoare unei adrese URL care integrează datele dvs. de program, în același mod în care sintaxa șablonului integrează vizualizările dvs. cu datele din program. Apoi puteți aplica logica programelor pentru a alege ce afișări să fie afișate sau ascunse, ca răspuns la introducerea de către utilizatori și la propriile reguli de acces.

**Lansarea aplicației**

Acum trebuie să-i spunem Angular cum să lanseze aplicatia. Pentru a face acest lucru, creați un fișier main.ts cu următorul conținut

import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';

const platform = platformBrowserDynamic();

platform.bootstrapModule(AppModule);

**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. Opțiunea "compilerOptions" stabilește opțiunile de compilare. Și opțiunea "exclude" exclude din compilare dosarul node\_modules, care conține toate pachetele descărcate. 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. De exemplu, în fișierul package.json, va fi posibil să specificați alte versiuni ale pachetelor dacă versiunile anterioare sunt învechite. Puteți schimba numele versiunii proiectului. Puteți să corectați setările TypeScript, dar, în general, organizația generală va fi aceeași. 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: