

LUCRARE DE LICENȚĂ

**Absolvent**: Nicorescu Teodor-Ionuț

**Coordonator**: Dr. Livia SÂNGEORZAN

Brașov

Iulie 2021



LUCRARE DE LICENȚĂ

Aplicație web pentru călătorii internaționale

**Absolvent**: Nicorescu Teodor-Ionuț

**Coordonator**: Dr. Livia SÂNGEORZAN

Brașov

Iulie 2021

**Capitolul 1**

**Introducere**

* 1. **Culturi și tradiții**

**1.1.1** **Tradiții**

O traditie este o credinta sau un comportament (obicei popular) transmis in cadrul unui grup sau societate cu semnificatie simbolica sau semnificatie speciala cu origini in trecut. Exemplele obisnuite include sarbatorile sau hainele impracticabile, dar semnificative din punct de vedere social (cum ar fi perucile avocatilor sau pintenii ofiterilor militari), dar ideea a fost aplicata si normelor sociale, cum ar fi salutarile. Traditiile pot persista si evolua timp de mii de ani ( cuvântul tradiție în sine derivă din latina tradere care înseamnă literalmente a transmite, a preda, a da pentru păstrare ) . Deși se presupune în mod obișnuit că tradițiile au o istorie veche, multe tradiții au fost inventate în mod intenționat, indiferent dacă acestea sunt politice sau culturale, pe perioade scurte de timp. În contextele artistice, tradiția este folosită pentru a decide afișarea corectă a unei forme de artă. De exemplu, în interpretarea genurilor tradiționale (cum ar fi dansul tradițional), respectarea liniilor directoare care dictează modul în care ar trebui compusă o formă de artă are o importanță mai mare decât preferințele interpretului.

Cultura este un termen generic care cuprinde comportamentul social și normele găsite în societățile umane, precum și cunoștințele, credințele, artele, legile, obiceiurile, capacitățile și obiceiurile persoanelor din aceste grupuri. Oamenii dobândesc cultură prin procesele de învățare ale enculturării și socializării, lucru demonstrat de diversitatea culturilor din societăți.

O normă culturală codifică comportamentul acceptabil în societate; servește drept ghid pentru comportament, îmbrăcăminte, limbaj și comportament într-o situație, care servește drept șablon pentru așteptările într-un grup social. Acceptarea doar a unei monoculturi într-un grup social poate suporta riscuri, la fel cum o singură specie se poate ofili în fața schimbărilor de mediu, din cauza lipsei răspunsurilor funcționale la schimbare.

**1.1.2 Culturi**

Culturile sunt ceea ce face țara unică și interesantă. Fiecare țară are diferite activități culturale și ritualuri culturale. Cultura include bunuri materiale, lucrurile pe care oamenii le folosesc și le produc. Cultura este, de asemenea, credințele și valorile oamenilor și modurile în care aceștia gândesc și înțeleg lumea și propria lor viață.

Diferite țări au culturi diferite. De exemplu, unii japonezi mai în vârstă poartă kimonos, aranjează flori în vaze și au ceremonii de ceai. Unele țări se opun unor lucruri din cultura lor, cum ar fi discriminarea sau religia.

* + 1. **Schimbul cultural, de ce conteaza?**

„Schimburile simple pot rupe zidurile dintre noi, pentru că atunci când oamenii se reunesc și vorbesc între ei și împărtășesc o experiență comună, atunci umanitatea lor comună este dezvăluită.”

- **Barack Obama, fost presedinte al SUA**

Pe măsură ce ne conectăm cu cele de diferite medii, educații culturale și comunități decât ale noastre, devine din ce în ce mai clar că diferențele noastre contribuie nu numai la o societate globală unică și minunată, ci că există adesea mai multe asemănări între noi decât am face aştepta. Luați în considerare râsul - un semn universal și sunetul fericirii. Această reacție involuntară nu este unică pentru nicio cultură specifică, ci mai degrabă este un factor de unire la nivel mondial. Adesea, poveștile pe care le vedem în mass-media se concentrează foarte mult pe diferențele noastre; dar schimbul cultural provoacă în mai multe moduri:

1. Schimbul cultural arată mai degrabă importanța asemănărilor decât a diferențelor. În ciuda faptului că suntem înconjurați de un flux nesfârșit de medii de știri care perpetuează adesea teama de cei diferiți decât noi înșine dar, de fapt, vorbind și conectându-ne cu ceilalți, vedem clar că suntem mai asemănători decât diferiți. Iubim, râdem și plângem pentru aceleași valori umane universale.
2. Schimbul cultural evidențiază frumusețea diversității. Deși începem să vedem asemănările noastre mai clar, acest lucru nu înseamnă că unicitatea noastră cade pe marginea drumului. Mai degrabă, prin schimbul cultural, suntem capabili să ne sărbătorim diferențele și să folosim cunoștințele despre culturile celuilalt pentru a fi lideri globali eficienți.
3. Schimbul cultural evidențiază frumusețea diversității. Deși începem să vedem asemănările noastre mai clar, acest lucru nu înseamnă că unicitatea noastră cade pe marginea drumului. Mai degrabă, prin schimbul cultural, suntem capabili să ne sărbătorim diferențele și să folosim cunoștințele despre culturile celuilalt pentru a fi lideri globali eficienți.
   1. **Calatoriile**

Călătoriile pot fi locale, regionale, naționale (interne) sau internaționale. În unele țări, călătoriile interne non-locale pot necesita un pașaport intern, în timp ce călătoriile internaționale necesită de obicei un pașaport și o viză. Tururile sunt un tip obișnuit de călătorie. Exemple de tururi de călătorie sunt croaziere de expediție, tururi de grup mic și croaziere pe râu.

**1.2.1 Argumente pro**

* Puteti experimenta noi culture si va veti extinde cunostintele
* Calatoria va poate extinde viziunea asupra lumii
* Puteti invata limbi noi
* Va puteti face prieteni noi
* Amintiri pentru o viata
* Va poate ajuta sa scapati de o viata plictisitoare

**1.2.2 Argumente contra**

* Calatoria poate fi costisitoare
* S-ar putea să dai probleme
* Probleme de mediu
  1. **Motivarea alegerii temei**

Călătoria este forma perfectă de evadare din rutina zilnică și plictisitoare, mult mai bună decât citirea unei cărți sau vizionarea unui film. O schimbare de peisaj este, uneori, exact ceea ce mulți dintre noi avem nevoie pentru a trece peste problemele și stresul zilnic. De asemenea, a vedea părți ale lumii și a vă cufunda în culturi străine deschide noi căi de descoperire. Călătoria în sine poate fi educativă și îți poate lărgi orizontul și te poate transforma dintr -un cetățean al țării tale într -un cetățean al lumii.

Un aspect important al călătoriilor este faptul că ați putea învăța limbi noi sau , și mai important, de câte ori nu ne-au impresionat anumite locuri sau peisaje pe care le-am văzut doar în poze sau filme? De ce să nu vizităm acele locuri și, în același timp, să satisfacem cât mai multe curiozități despre țara respectivă sau modul de gândire al localnicilor.

Următorul aspect motivant este experientă acumulată pe parcursul unei călătorii, care poate aduce beneficii în diferite momente ale vieții noastre, de exemplu, în momentele grele vei fi mult mai capabil să te refaci deoarece știi cât de frumoasă poate fi viața.

În concluzie, prin această aplicație , doresc să ofer utilizatorilor șansa să cunoască cât mai mult din frumusețile acestei lumi, să se familiarizeze cu diferite personalități sau moduri de a gândi .

**Capitolul 2**

**Dezvoltarea web**

* 1. **Scurta istorie**

Creșterea internetului în masă este în mare parte corelată cu istoria sa ca mediu vizual. La fel ca multe alte sisteme și aplicații informatice, a fost nevoie de o interfață grafică pentru ca populația generală să înceapă să înțeleagă potențialul internetului. Computerul personal nu ar fi proliferat în gospodăriile noastre și în mediile de lucru fără monitorul de afișare și totuși nu ar fi folosit pe scară largă până când foile de calcul electronice, procesatoarele de text și jocurile video au început să atragă utilizatorii. În mod similar, utilizatorii nu au început să cumpere pe World Wide Web până când browserul web orientat vizual a început să apară la începutul anilor 1990. Și de atunci, chiar dacă au existat îmbunătățiri dramatice în tehnologia și estetica web, unele dintre cele mai vechi tehnici de design web au perseverat de-a lungul anilor.

Ideea de internet a existat într-o anumită formă timp de cel puțin o jumătate de secol înainte ca, în sfârșit, să devină o utilitate obișnuită în anii 1990 Conceput în anii 1980, World Wide Web a câștigat o tracțiune semnificativă odată cu introducerea browserului Mosaic în 1993. La scurt timp după aceea, companiile au început să recunoască potențialul comercial al web-ului, pe măsură ce infrastructura de rețea a crescut pentru a se adapta la ceea ce s-ar dovedi a fi un aflux masiv de activitate online. . Apoi, bula tehnologică a crescut și a izbucnit, ale cărei supraviețuitori (Google, Amazon și altele asemenea) au trecut de la a fi influenți tehnologici cheie la veritabili giganți corporativi în decurs de aproximativ un deceniu.

Web-ul a avansat foarte mult în anii care au urmat prăbușirii tehnologice din 2000-2001. În acest timp, guvernul a început să joace un rol din ce în ce mai influent pe web, în timp ce, în același timp, companii puternice de tehnologie au ieșit din cenușa marelui colaps pentru a stabili noul curs pentru comerțul digital și cultura. Și pe măsură ce a fost pusă această bază mai nouă și mai solidă, internetul a devenit din ce în ce mai mult canalul principal pentru telecomunicații în epoca modernă.

**2.2 Definiții**

Dezvoltarea web reprezinta munca efectuată în dezvoltarea unui site Web pentru Internet (World Wide Web) sau a unui intranet (o rețea privată). Dezvoltarea web poate varia de la dezvoltarea unei singure pagini statice simple de text simplu la aplicații web complexe, companii electronice și servicii de rețea socială. O listă mai cuprinzătoare de sarcini la care se referă în mod obișnuit dezvoltarea web poate include ingineria web, designul web, dezvoltarea conținutului web, legătura cu clientul, configurarea securității serverului web și a rețelei și dezvoltarea comerțului electronic.

* 1. **Site web vs Aplicatie web**

Cele mai multe persoane s-au obisnuit sa numeasca “site” orice pagina vizibila pe internet, dar exista cateva diferente intre un site web si o aplicatie web pe care ei le omit:

Primul punct pentru a începe diferențierea „aplicație web vs. site web” este interactivitatea. Un site web oferă conținut vizual și text pe care utilizatorul îl poate vedea și citi, dar nu afectează în niciun fel. În cazul unei aplicații web, utilizatorul nu numai că poate citi conținutul paginii, ci și poate manipula datele de pe această pagină. Interacțiunea ia forma unui dialog: utilizatorul face clic pe un buton sau trimite un formular și primește un răspuns de pe pagină. Acest răspuns poate lua forma unei descărcări de documente, chat online, plăți electronice și multe altele.

Un alt punct important in diferentierea acestora este autentificarea. Autentificarea este procedura care implică introducerea unui nume de utilizator sau email și parolei pentru a obține acces la sistem. Este o necesitate pentru software-ul web care necesită informații personale. Conturile de utilizator trebuie să fie securizate pentru a preveni accesul neautorizat și scurgerea datelor sensibile.

Aplicațiile web necesită în mare parte autentificare, deoarece oferă o gamă mult mai largă de opțiuni decât site-urile web. Luați în considerare un exemplu de rețele sociale. Când vă înregistrați, vă creați un cont și primiți un număr unic de identificare. Sistemul vă avertizează dacă datele dvs. de conectare și parola sunt slabe. Dacă le lăsați neschimbate, hackerii pot ajunge la contul dvs. și vă pot fura informațiile, precum și pot irita alți utilizatori cu e-mailuri nedorite sub numele dvs.

* 1. **Aplicațiile web**

O aplicație web reprezinta un software care ruleaza si este gazduit de un server web, spre deosebire de programele software de pe computer care sunt rulate local pe sistemul de operare (OS) al dispozitivului. Aplicațiile web sunt accesate de utilizator printr-un browser web cu o conexiune de rețea activă.

Aceste aplicații sunt programate utilizând o structură modelată client-server. Utilizatorului (clientul) îi sunt furnizate servicii printr-un server extern care este găzduit de o terță. Acest lucru se întâmplă prin câțiva pași simpli:

1. Utilizatorul realizeaza una sau mai multe cereri catre server prin protocolul HTTP
2. Serverul primeste acea cerere si este procesata
3. Serverul trimite inapoi catre utilizator un raspuns HTTP
4. Utilizatorul primeste raspunsul

Un utilizator poate fi conectat si trimite cereri catre mai multe servere, dar acestea au si ele un dezavantaj, si anume faptul ca pot cadea datorita unui numar mari de cereri simultan.

Daca mai sus am folosit termenii “server” si “HTTP”, urmeaza mai jos definitia acestora:

Un server web este un software de calculator și un hardware de bază care acceptă cereri prin HTTP, protocolul de rețea creat pentru a distribui pagini web, sau varianta sa securizată HTTPS. Un agent de utilizator, de obicei un browser web sau un crawler web, inițiază comunicarea făcând o cerere pentru o anumită resursă folosind HTTP, iar serverul răspunde cu conținutul resursei respective sau cu un mesaj de eroare. De asemenea, serverul poate accepta și stoca resursele trimise de la agentul utilizator, dacă este configurat pentru a face acest lucru.

Pe partea hardware, un server web este un computer care stochează software-ul serverului web și fișierele componente ale unui site web.

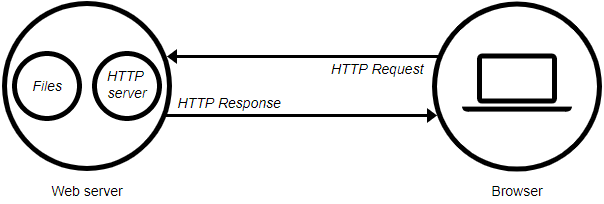


Figure 2.1: Exemplu client-server

Protocolul HTTP este un protocol pentru sisteme de informații distribuite, colaborative, hipermedia care permite utilizatorilor să comunice date pe World Wide Web.

* 1. **Servicii web**

Termenul de serviciu web (WS) poate fi:

* Un serviciu oferit de un dispozitiv electronic unui alt dispozitiv electronic, care comunică între ele prin intermediul World Wide Web
* un server care rulează pe un dispozitiv computerizat, care ascultă cereri la un anumit port printr-o rețea, care servește documente web (HTML, JSON, XML, imagini).

Într-un serviciu Web, o tehnologie Web, cum ar fi HTTP, este utilizată pentru transferul de formate de fișiere citibile de mașini, cum ar fi XML și JSON.

* + 1. **Tipuri de servicii web**
* **XML-RPC (Remote Procedure Call) este cel mai de bază protocol XML pentru schimbul de date între o mare varietate de dispozitive dintr-o rețea. Folosește HTTP pentru a transfera rapid și ușor date și comunicări alte informații de la client la server.**
* **UDDI (Universal Description, Discovery și Integration) este un standard bazat pe XML pentru detalierea, publicarea și descoperirea serviciilor web. Este practic un registru pe internet pentru companii din întreaga lume. Scopul este de a eficientiza tranzacțiile digitale și comerțul electronic între sistemele companiei.**
* **SOAP** este un protocol de serviciu Web bazat pe XML pentru schimbul de date și documente prin HTTP sau SMTP (Simple Mail Transfer Protocol). Permite proceselor independente care operează pe sisteme disparate să comunice utilizând XML.
* **Argumente pro:**
  + De obicei mai ușor de consumat
  + Mai multe standarde (WSDL etc.)
  + Calcul distribuit
* **Argumente contra:**
  + Configurare dificilă
  + Codificare mai complicate
  + Mai greu de dezvoltat
* **REST** oferă comunicare și conectivitate între dispozitive și internet pentru sarcini bazate pe API. Majoritatea serviciilor RESTful folosesc HTTP ca protocol de suport.
* Argumente pro:
  + Ușor
  + Lizibil de către om
  + mai ușor de construit
* Comunicare punt-la-punct
* Lipsa standardelor

**Capitolul 3**

**Programe si tehnologii folosite**

* 1. **Angular**
     1. **Concepte de baz**ă
* **Typescript**

TypeScript este un limbaj open-source care se bazează pe JavaScript, unul dintre cele mai utilizate instrumente din lume, prin adăugarea de definiții de tip static. Acesta este dezvoltat și menținut de Microsoft.

Tipurile de date oferă o modalitate de a descrie forma unui obiect, oferind o documentație mai bună și permițând TypeScript să valideze codul.

Desi in Angular este posibil sa scriem si cod javascript, acest lucru nu este recomandat. Angular a fost creat sa functioneze impreuna cu typescript.

* **HTML**

HTML (HyperTextMarkup Language)  este limbajul normalizat de marcare pentru proiectele destinate a fi afișate într -un browser web. Acesta poate fi utilizat împreună cu alte tehnologii precum Cascading Style Sheets ( CSS ) și limbaje de scriptare precum JavaScript.

HTML descrie structura unei pagini web, care este formată dintr -o serie de elemente. Elementele HTML îi spun browserului cum să afișeze conținutul și etichetează bucăți de conținut precum “ acesta este un antet”, “acesta este un paragraf”, “acesta este un link”, etc.

* **CSS**

CSS este un limbaj conceput pentru a permite separarea prezentării și a conținutului. Această separare poate îmbunătăți accesibilitatea conținutului, poate oferi mai multă flexibilitate și control în specificațiile caracteristicilor **prezentării ,** permite mai multor pagini web să partajeze formatarea prin specificarea CSS-ului relevant într -un fișier separat . css , care reduce complexitatea.

CSS-ul poate fi aplicat in 3 moduri**:**

1. **Inline –** Sunt fragmente scrise direct în cod HTML și aplicate unui singur element (cel pe care a fost aplicat).
2. **Internal –** Este scris direct în antetul unei pagini .html.
3. **External –** Cod scris în fi șiere externe, salvate cu extensia. css. Acestea pot fi utilizate pentru a determina aspectul unui întreg site web printr-un singur fișier.

Cea mai eficientă si folosită metodă estecea **external** deoarece, spre deosebire de celelalte, aceasta poate fi aplicată unei intregi pagini sau chiar mai multor pagini web, fără a repeta cod inutil.

* + 1. **Introducere**

Angular este o platformă și un cadru pentru crearea de aplicații client cu o singură pagină folosind HTML și TypeScript. Angular este scris în TypeScript si implementează funcționalități de bază și opționale ca un set de biblioteci TypeScript pe care le importați în aplicațiile dvs.

Arhitectura unei aplicații Angular se bazează pe anumite concepte fundamentale:

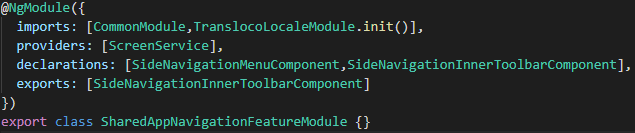
* Elementele de bază ale framework-ului Angular sunt componente care sunt organizate în NgModule.

Figure 3.1: Exemplu de NgModule

* NgModules colectează codul aferent în seturi funcționale;
* O aplicatie Angular este definită de un set de NgModule.
* O aplicație are întotdeauna cel puțin un modul rădăcină care permite bootstrapping-ul și are, de obicei, multe alte module de caracteristici.

Pentru a putea utiliza Angular, este nevoie de instalarea instrumentului Angular CLI prin comanda **npm install -g @angular/cli**. Odata ce acesta a fost instalat, putem genera o aplicatie Angular prin comanda **ng new <nume\_aplicație>**.

Rularea aplica**ț**iei se va face prin comanda **ng serve --open**.

* + 1. **Componente Angular**

Componentele sunt elementele care formează o aplicație. O componentă include o clasă TypeScript cu un decorator @Component(), un template HTML și stiluri. Fiecare clasa ce contine decoratorul @Component() este automat identificata ca fiind o componenta si declară următoarele informații specifice framework-ului Angular:

* Un selector care care definește modul în care este utilizată componenta într-un template. Elementele HTML dintr-un template care se potrivesc cu acest selector, devin instanțe ale componentei.
* Un template HTML care instruiește Angular cum să redea componenta.
* Un set opțional de stiluri CSS care definesc aspectul elementelor HTML ale template-ului.

Orice aplicație Angular conține cel puțin o componentă , aceasta fiind componența rădăcină a aplicației , cea care face conexiunea dintre celelalte componente și DOM (Document Object Model). Acestea oferă , de asemenea, o încapsulare puternică și o structură intuitivă a aplicației .

O componenta poate fi inclusa intr-o alta componenta prin selectorul unic, iar acestea isi pot transmite date intre ele cu ajutorul decoratorilor @Input() sau @Output():

* Decoratorul @Input se foloseste pentru a transmite date de la componenta parinte catre componenta copil.
* Decoratorul @Output se foloseste pentru a transmite date de la componenta copil catre componenta parinte.

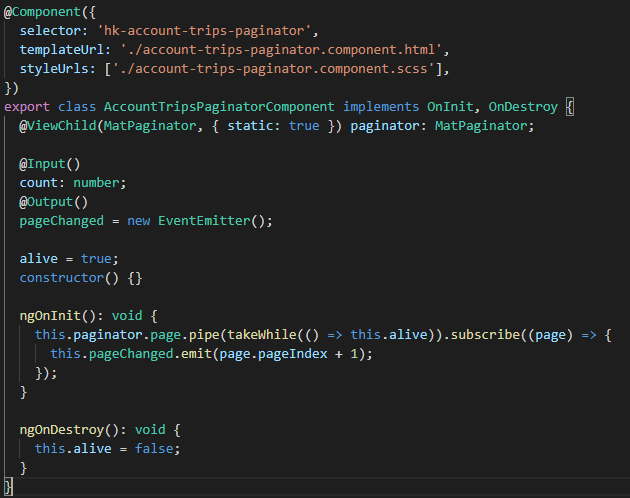


Figure 3.2: Fișier .ts dintr -o componentă Angular

În figura 3.2 se poate observa și noțiunea de @ViewChild. Cu ajutorul acesteia obținem o referință către un element HTML sau o componentă din fișierul .html, într-o variabilă din fișierul .ts al componentei curente.

Template-ul HTML este definit prin calea către fi șierul .html. Acesta poate fi extins cu sintaxă suplimentară care vă permite să inserați valori dinamice din fișierul .ts. Angular actualizează automat DOM-ul redat atunci când starea componentei dvs. se modifică. O aplicație a acestei caracteristici este inserarea textului dinamic sau declararea de **event listeners** pentru a asculta si raspunde la acțiunile utilizatorului, cum ar fi apăsarea tastelor, mișcările mouse-ului, clicurile și atingerile.



În exemplul de mai sus, textul afișat în interiorul elementului HTML <p> va fi conținutul variabilei **someText**, și anume “Nice text to render”.

* + 1. **Ciclurile de viata ale unei componente**

O instanță componentă are un ciclu de viață care începe atunci când Angular instanțiază clasa componentelor și redă vizualizarea componentă împreună cu vizualizările sale copil. Ciclul de viață continuă cu detectarea modificărilor, deoarece Angular verifică când se modifică proprietățile legate de date și actualizează atât vizualizarea, cât și instanța componentelor, după cum este necesar. Ciclul de viață se încheie atunci când Angular distruge instanța componentă și elimină șablonul redat din DOM. Angular realizeaza acest lucru cu ajutorul unor interfete pe care componenta noastra le poate implementa. Aceste interfețe conțin o singură metodă, a cărei denumire este numele interfeței cu prefixul ng. Acestea se apeleaza in felul urmator:

* **ngOnChanges**: Când se modifică o valoare a unei variabile ce conține decoratorul @Input() sau @Output(). Aceasta metoda nu se apeleaza daca componenta nu contine niciun element cu decoratorii @Input() sau @Output().
* **ngOnInit**: După primul **ngOnChanges**. Aceasta este apelată chiar dacă ngOnChanges nu este.
* **ngDoCheck**: Detectarea modificărilor personalizate de dezvoltator.
* **ngAfterContentInit**: După ce conținutul componentei s-a inițializat.
* **ngAfterContentChecked**: După fiecare verificare a conținutului componentei.
* **ngAfterViewInit**: După inițializarea view-ului unei componente.
* **ngAfterViewChecked**: După fiecare verificare a vizualizărilor unei componente.
* **ngOnDestroy**: Chiar înainte ca componenta să fie distrusă.

Aceste metode se apeleaza exact in ordinea declarata mai sus.

* + 1. **Service si dependency injection**

Un service este un obiect singleton care se instanțiază o singură dată pe durata de viață a aplicației . Acestea conț in metode care mențin datele pe tot parcursul vieții unei aplicații, adică datele nu sunt actualizat e și sunt disponibile tot timpul. Acest lucru ajută la împărțirea aplicației î n unități logice mici, diferite, care pot fi refolosite, și de a face conexiunea între business logic și diferite componente.

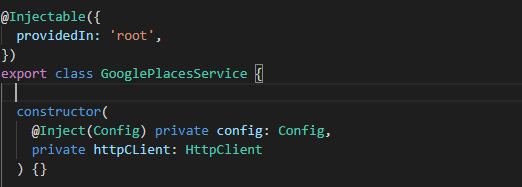


Figure 3.3: Exemplu de service in Angular

Exista 2 moduri pentru a face un serviciu sa fie singleton in Angular:

1. Setam proprietatea providedIn din interiorul decoratorului @Injectable ca ‘root’ (Metoda folosita in figura 3.3, care este si cea recomandata)
2. Includem service-ul in AppModule sau intr-un modul care este importat de AppModule (AppModule este modulul radacina al aplicatiei)

Dupa ce s-a executat o metoda din cele prezentate mai sus, service-ul va putea fi injectat in orice componenta.

Angular dependency injection are un sistem ierarhic de injecție, ceea ce înseamnă că injectoarele imbricate își pot crea propriile instanțe de serviciu. Ori de câte ori Angular creează o nouă instanță a unei componente care are furnizori specificați în @Component(), creează și un nou injector secundar pentru acea instanță. În mod similar, atunci când un NgModule nou este încărcat leneș în timpul rulării, Angular poate crea un injector pentru acesta cu proprii furnizori.

* + 1. **Directive**

Directivele sunt clase care adaugă un comportament suplimentar elementelor din aplicațiile Angular.

Acestea pot fi de 4 tipuri:

1. **Directive componente**

Sunt utilizate, în principal, pentru a specifica template-urile HTML. Aceastea sunt cele mai frecvent utilizate într -un proiect Angular.

1. **Directive atribut**

Directivele atribut sunt folosite pentru a modifica comportamentul unor elemente HTML în aspectul DOM. Cele mai utilizate directive ale atributelor sunt NgClass și NgStyle.

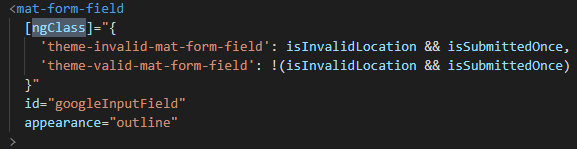


Figure 3.4: Exemplu de directiva ngClass

În figura 3.4, **isInvalidLocation** și **isSubmittedOnce** sunt valori de tip boolean. Cand acestea respecta conditia impusa, atunci clasa respective este adaugata elementului HTML.

1. **Directive structural**

Directivele structurale sunt utilizate pentru a adăuga sau elimina elementul HTML în aspectul Dom, de obicei prin adăugarea, eliminarea sau manipularea elementelor.

Exemple: NgIf, NgFor, NgSwitch

* **NgIf** – Este folosit pentru a crea sau a elimina o parte a elementelor din DOM în funcție de o condiție.
* **NgFor** - Este folosit pentru a personaliza afișarea datelor. Este utilizat în principal pentru afișarea unei liste de articole folosind bucle repetitive.
* **NgSwitch - Este ca switch-ul din JavaScript. Poate afișa un element dintre mai multe elemente posibile, pe baza unei condiții de comutare. Angular introduce doar elementul selectat în DOM.**

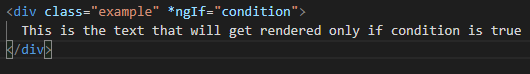
****

Figure 3.5: Exemplu directivă NgIf

În figura 3.5 avem un exemplu minimal ce pune în practică directiva structurală NgIf. Bineînțeles că în loc de textul din interiorul elementului div, putem adăuga alte elemente HTML, iar acestea vor fi afișate în concordanță cu condiția Fiecare element din interior poate avea, de asemenea, câte o directivă.

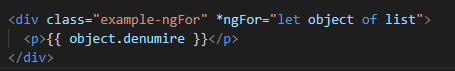


Figure 3.6: Exemplu directivă NgFor

In figura 3.6 se poate observa un exemplu de utilizare a directivei NgFor.

Variabila **list** provine din fișierul .ts, aceasta fiind o listă de obiecte. Pentru fiecare obiect din listă, se va adăuga în dom câte un element <p> care va conține valoarea proprietății **denumire** a obiectului.

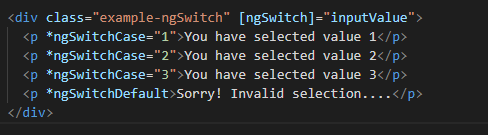


Figure 3.7: Exemplu directivă NgSwitch

In figura 3.7 avem, de asemenea, un exemplu al directivei NgSwitch.

În funcție de valoarea pe care inputValue o primește pe parcursul utilizării aplicației , Angular va introduce în DOM doar elementul selectat. Se poate vedea că valorile posibile sunt 1, 2 sau 3 ( reprezentate de \*ngSwitchCase). Dacă inputValue nu va avea niciuna dintre acestea , în DOM va fi adăugat elementul implicit ( acesta este reprezentat prin \*ngSwitchDefault).

**Observație: Un element nu poate conține mai mult de o directivă**

Pentru multiple directive, vom avea nevoie de mai multe elemente HTML imbricate, fiecare având cel mult o directivă.

1. **Directive personalizate**

În Angular, putem crea directive atribut și structurale folosind decoratorul @Directive.

Folosind directiva atribut personalizată putem schimba aspectele cum ar fi culoarea textului, culoarea fundalului și dimensiunea fontului unui element HTML, pe care il putem numi element gazdă. Pentru a schimba aspectul, Angular oferă clasa ElementRef care poate accesa direct DOM-ul.

**Care sunt beneficiile unei directive personalizate?** În momentul în care accesăm un element cu ajutorul clasei ElementRef direct într -o componentă , aplicația devine vulnerabilă la atacurile XSS. Prin urmare, se recomandă utilizarea ElementRef într -o directiva pentru a schimba aspectul sau comportamentul elementului gazdă.

**Cum putem crea o directiva?**

**Pasul 1:** Creăm o clasa și îi adăugăm decoratorul @Directive

**Pasul 2:** Îi alocăm un nume proprietății ‘selector’ a decoratorului @Directive.

**Pasul 3:** Creăm un constructor și adăugăm în acesta un parametru de tipul ElementRef folosind dependency injection pentru a accesa elementul gazdă.

**Pasul 4:** Declarăm o variabilă și îi adăugăm decoratorul @Input() pentru a putea primi conținut din componentă .

**Pasul 5:** Utilizăm decoratorul @HostListener() în directivă pentru a asculta evenimentele elementului.

**Pasul 6:** Adăugăm directiva în modulul aplicației , în metadatele declarațiilor.

Cu ajutorul pașilor de mai sus, putem crea, manual, o directivă , dar bineînțeles că Angular-CLI furnizează o comandă prin care putem genera o directiv ă minimală.

Comanda cu ajutorul careia putem face acest lucru este **ng generate directive <nume\_directiva>.**



Figure 3.8: Exemplu de directivă personalizată

* + 1. **Pipes**

În orice aplicație, prelucrarea sau manipularea datelor este un subiect destul de important. De exemplu, pentru prelucrarea unui string putem crea o funcție care primește ca parametru acel string și returnează valoarea prelucrată. Angular vine în ajutorul acestui aspect și introduce așa-numitele pipe-uri.

Putem utiliza pipe- uri pentru a transforma string -uri, date și alte informații pentru afișare. Pipe-urile sunt funcții simple pe care le putem utiliza în anumite expresii din template-uri pentru a accepta o valoare de intrare și a returna o valoare transformată.

Pipe-urile sunt utile și ușor de utilizat deoarece le declarăm în modulul aplicației o singură dată, după care le putem folosi în întreaga aplicație.

In Angular exista o serie de pipe-uri predefinite utilizate pentru formatarea datelor, cum ar fi DatePipe, UpperCasePipe, LowerCasePipe, CurrencyPipe, DecimalPipe si PercentPipe, care pot fi folosite in absolut orice template.

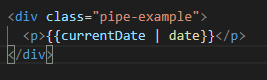


Figure 3.9: Exemplu utilizare pipe

Se poate observa în figura 3.9 că, pentru utilizarea pipe-ului, trebuie să plasăm operatorul ( | ) între conținutul ce dorim să îl transformăm și numele pipe-ului.

Un pipe poate avea și parametri optionali. De exemplu, putem utiliza CurrencyPipe cu un cod de țară cum ar fi EUR. Pentru utilizarea parametrilor trebuie să adăugăm (:) după numele pipe-ului.

De asemenea, putem crea un pipe personalizat care sa formateze sau sa transforme valoarea de intrare in orice mod dorim.

**Cum cream un pipe personalizat?**

**Pasul 1:** Creăm o clasa și îi adăugăm decoratorul @Pipe

**Pasul 2:** Îi oferim un nume, adăugând proprietatea name a decoratorului @Pipe.

**Pasul 3:** Din clasa pe care am creat-o, implementăm interfața **PipeTransorm**

**Pasul 4:** Din interfața **PipeTransform** implementăm metoda **transform**, care primește unul sau mai mulți parametri.

**Pasul 5:** Adăugăm pipe-ul în modulul aplicației, în metadatele declarațiilor.

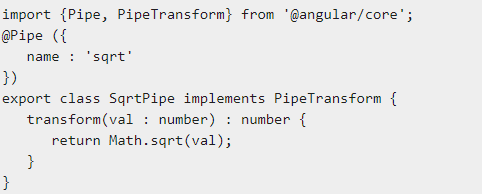


Figure 3.10: Exemplu pipe personalizat

**Observație: Se pot folosi mai multe pipe-uri împreună**

* + 1. **Angular router si navigarea**

Într-o aplicație cu o singură pagină, putem schimba ceea ce utilizatorul vede afișând sau ascunzând anumite elemente din DOM care corespund anumitor componente, în loc de a ieși la server pentru a obține o pagină nouă. Pe măsură ce utilizatorii efectuează sarcini ale aplicației, trebuie să se deplaseze între diferitele view- uri definite.

Pentru a gestiona navigarea de la un view la altul, utilizați Angular Router. Routerul permite navigarea interpretând o adresă URL a browserului ca o instrucțiune de modificare a view-ului.

În momentul creării unei noi aplicații Angular, se va genera automat și un modul responsabil de navigare în care putem configura rutele dorite. Ulterior, acest modul va trebui importat în modulul principal al aplicației.

**Exista 3 pasi fundamentali pentru crearea unei rute:**

**Pasul 1: Importam RouterModule si Routes în modulul de rutare**. Angular-CLI efectueaza acest pas automat si configureaza importurile si exportrueile din @NgModule()

**Pasul 2:** **Definim rutele în array-ul de rute**. Fiecare rută din această matrice este un obiect JavaScript care conține două proprietăți. Prima proprietate, calea, definește adresa URL pentru traseu. A două proprietate, componenta, definește componenta pe care Angular ar trebui să o folosească pentru calea corespunzătoare.

**Pasul 3:** **Adăugăm rutele în aplicație**. Rutele, odată definite, pot fi adăugate în aplicație. Mai întâi adăugăm atributul routerLink elementului HTML care este folosit pentru navigare, după care adăugăm în template-ul în care se dorește afișarea componentei care corespunde căii.

**Observație: Dacă adăugăm o rută de mai multe ori, aceasta va afișa componenta primei rute adăugate în matrice.**

Angular router foloseste o strategie “primul venit, primul servit” atunci se potriveste segmentul de ruta, astfel ca rutele mai specifice ar trebui plasate deasupra rutelor mai putin specifice.

Dacă doriți să transmiteți anumite date între 2 componente care nu sunt părinte-copil, putem face asta cu ajutorul router-ului. Pentru a face acest lucru, utilizăm interfața ActivatedRoute.

Pentru a obține informațiile de pe o rută, trebuie urmați câțiva pași:

1. Importăm ActivatedRoute și ParamMap în componentă.

2. Injectăm o instanță a interfeței ActivatedRoute în constructorul componentei.

3. În metoda ngOnInit() ne abonam la parametrii rutei și îi salvăm într -o variabilă.

De asemenea, se pot configura redirecționări . Pentru a face acest lucru, este nevoie să specificăm proprietatea “redirectTo” rutei de pe care dorim să redirecționăm utilizatorul.

Pentru a preveni accesul catre anumite rute, putem specifica un guard pentru acestea. Guard-urile disponibile: **CanActivate, CanActivateChild, CanDeactivate, Resolve, CanLoad.**

* + 1. **Formulare**

Gestionarea datelor introduse de utilizator cu formulare este fundamentul multor aplicații. Aplicațiile utilizează formulare pentru a permite utilizatorilor să se conecteze, să actualizeze un profil , să se înregistreze , să introducă informații sensibile și să efectueze multe alte sarcini de introducere a datelor.

Angular oferă două abordări diferite pentru gestionarea datelor introduse de utilizator prin intermediul unui formular: **Reactive** și **bazate pe** **template**. Amandoua captează evenimentele de intrare a utilizatorului, le validează , creează un model de formular și un model de date pentru actualizare și oferă o modalitate de urmărire a modificărilor.

* **Formulare reactive** – Ofera acces direct, explicit la modelul obiect, sunt mai scalabile, reutilizabile si testabile.
* **Formulare bazate pe template** – Se bazează pe directive în template pentru a crea și manipula modelul de bază . Sunt ușor de adăugat la o aplicație , dar nu se potrivesc la fel de bine ca cele reactive. Sunt utile când dorim să adăugăm un formular simplu la aplicație .

Diferente cheie intre formularele **reactive** si cele **bazate pe template**

|  |  |  |
| --- | --- | --- |
|  | **Formulare Reactive** | **Formulare Bazate pe template** |
| Configurarea modelului | Explicit, creat in componenta | Implicit, creat de directive |
| Model de date | Structurat si nevariabil | Nestructurat si variabil |
| Flux de date | Sincron | Asincron |
| Validare | Functii | Directive |

Dacă formularele sunt o parte centrală a aplicației ,scalabilitatea este foarte importantă.

Abilitatea de a reutiliza formulare între componente este esențială. Ce tip de formulare sunt mai scalabile? Formularele reactive, deoarece acestea oferă acces direct la API-ul formularului subiacent și utilizează fluxul de date sincron între view și modelul de date.

* + 1. **Comunicarea cu servicii back-end**

Majoritatea aplicațiilor front-end trebuie să comunice cu un server prin protocolul HTTP, pentru a descărca, încărca și accesa alte servicii back-end. Pentru acest aspect, Angular vine în ajutor cu un API HTTP client, clasa de servicii HttpClient ce poate fi importată din @angular/common/http.

Clientul HTTP oferă următoarele caracteristici:

* Abiltatea de a solicita ca răspuns un obiect de un anumit tip
* Tratarea simplificată a erorilor
* Caracteristici de testabilitate
* Interceptarea cererii și a răspunsului

De asemenea, putem accesa următoarele metode din clientul HTTP:

* **Metoda GET** – Solicitările folosind această metodă ar trebui doar să recupereze date.
* **Metoda HEAD** – La fel că metoda GET, dar această metodă nu primește și corpul de răspuns.
* **Metoda POST** – Această metodă este utilizată pentru a trimite o entitate la sursa specificată, cauzând adesea efecte secundare pe server.
* **Metoda PUT** – Înlocuiește toate reprezentările actuale ale sursei țintă cu corpul curent al cererii.
* **Metoda DELETE** – Șterge sursă specificată
* **Metoda CONNECT** – Stabilește un tunel către server identificat de resursa țintă
* **Metoda OPTIONS** – Este folosită pentru a descrie opțiunile comunicării pentru resursa țintă.
* **Metoda TRACE** – Efectuează un test de loop-back al mesajului de-a lungul rutei către resursa țintă.
* **Metoda PATCH** – Se folosește pentru a aplica modificări parțiale pentru resursă.
  + 1. **Testarea aplicației**

Testarea aplicației ne ajută să verificăm dacă această funcționează după așteptări.

Configurarea pentru teste este făcută de Angular CLI în momentul creării aplicației. Pentru a rula testele, folosim comanda ng test, care lansează instrumentul pentru teste, Karma.

Pentru a modifica configurarea, putem edita fișierul karma.conf.js din folderul rădăcină al proiectului și fișierele test.ts din folderul src/.

Cele mai folosite teste sunt:

* **Teste unitare** - Testând câte un lucru pe rând, testele nu ar trebui să aibă grijă de punctele de interacțiune dintre diferitele unități ale aplicației. Acestea constau în testarea metodelor și funcțiilor individuale ale claselor, componentelor sau modulelor utilizate de aplicație.
* **Teste de integrare** – Verifică dacă mai multe unități interacționează corect între ele. De exemplu, poate fi testarea interacțiunii cu baza de date sau asigurarea faptului că microserviciile funcționează împreună așa cum era de așteptat.

Pentru amandouă tipuri de teste de mai sus, se poate genera un raport de acoperire a codului. Acesta ne permite să estimăm cât din codul aplicației este testat.

* + 1. **Internaționalizarea aplicației**

Internaționalizarea (i18n) este procesul de proiectare și pregătire a aplicației pentru a putea fi utilizată în diferite locații din întreaga lume.

Localizarea este procesul de construire a versiunilor aplicației pentru diferite regiuni locale, inclusiv extragerea textului pentru traducere în diferite limbi și formatarea datelor pentru anumite regiuni locale.

Acest lucru poate fi realizat in felul urmator:

* Utilizarea pipe-urilor pentru a afișa date, numere, procente sau valute într -un format local.
* Marcăm textul în template-ul componentelor pentru traducere.
* Marcăm formele de plural al expresiilor pentru traducere.
* Marcăm text alternativ pentru traducere.

Pentru traducerea eficientă a textului, putem utiliza librăria **Transloco**. Aceasta poate fi instalată prin comandă **ng add @ngneat/transloco**.

**Transloco** permite definirea traducerilor în diferite limbi și să comutam între ele cu ușurință în timpul rulării aplicației. Această librărie expune un API bogat pentru a gestiona traducerile în mod eficient și curat. Acesta permite traducerea atât printr-un pipe cât și printr-o directivă.

**Observație: Se recomandă traducerea cu ajutorul unei directive**

* + 1. **Angular Material UI**

Angular Material este o bibliotecă de componente UI pentru dezvoltatorii Angular. Componentele Angular Material ajută la construirea de pagini web și aplicații web atractive, consistente și funcționale, respectând în același timp principiile moderne de design web, cum ar fi portabilitatea browserului, independența dispozitivelor și degradarea grațioasă.

Angular Material a fost introdus în 2014 de către Google ca un limbaj de proiectare. Inițial a fost aplicat pe AngularJS pentru a crea aplicații mult mai atractive și eficiente, apoi, în 2016, Google a rescris codul de la 0, eliminând JavaScript și l-a numit Angular.

**Caracteristici principale**:

* Design responsiv incorporat
* CSS standard cu amprentă minimă
* Noi versiuni ale controalelor interfeței utilizatorului
* Funcții îmbunătățite și specializate precum carduri , bară de instrumente, apelare rapidă , navigare laterală , glisare, etc.
* Cross-browser și poate fi utilizat pentru a crea componente web reutilizabile.

**Avantaje**:

* **Calitate superioara**:
  + Componente internaționale și accesibile pentru toată lumea. Bine testat pentru a asigura performanță și fiabilitate.
  + API-uri simple, cu comportament consecvent pe mai multe platforme.
* **Versatil**:
  + Oferă instrumente care îi ajută pe dezvoltatori să își construiască propriile componente personalizate cu modele de interacțiune comune.
  + Personalizabil în limitele specificațiilor de proiectare a materialelor.
* **Fără fricțiuni**:
  + Construit de echipa Angular pentru a se integra perfect cu Angular.
  + Începeți de la zero sau introduceți în aplicațiile existente.

Biblioteca Angular Material se poate adăuga într -o aplicație Angular prin comanda **ng add @angular/material.** Odată ce am rulat comanda , va trebui aleasă o temă din cele 4 predefinte sau putem seta una personalizată.

Această biblioteca este compusă dintr -o gama largă de componente Angular, enumerate mai jos:

* Controale de formular – select, input, sliders, checkbox, etc.
* Componente de aspect – card, grid, list, tabs
* Butoane
* Modele de navigare – side-nav, menu, toolbar
* Tabele de date cu antet, paginator,
* Modale si ferestre pop-up
* Indicatori – spinner, progress bar
  + 1. **DevExtreme Angular**

Devextreme este un set de componente UI gata de întreprindere. Acesta oferă o colecție cuprinzătoare de componente UI de inaltă performanță și receptive pentru utilizare în aplicații web și mobile de ultima ultimă generație.

Suita este livrată cu un data-grid complet, componente grafice interactive, editoare de date, diagrame interactive, navigare și multe alte widget-uri multifuncționale.

De asemenea, DevExtreme Angular ofera un mic suport in cazul in care se doreste crearea unui proiect de la 0. Acesta permite generarea unei aplicatii simple, cu un meniu de navigare si cateva exemple de view-uri. Aceasta aplicatie se poate genera in cativa pasi cu ajutorul DevExtreme CLI, rulând comanda urmatoare**: npx -p devextreme-cli devextreme new** **angular-app <nume\_aplicatie>.**

În cazul în care se dorește adăugarea DevExtreme la o aplicație angular existentă , trebuie rulată următoarea cumanda: **npx -p devextreme-cli devextreme add devextreme-angular.**

* + 1. **Bootstrap**

Boostrap este un framework front-end open source, gratuit pentru crearea de site - uri și aplicații web. Acesta este construit pe HTML, CSS și JavaScript pentru a facilita dezvoltarea de site-uri și aplicații responsive.

Design-ul responsiv face posibil ca o pagină web să detecteze dimensiunea și orientarea ecranului vizitatorului și să adapteze automat afișajul.

Bootstrap include componente ale interfeței de utilizator, machete și instrumente JS, împreună cu cadrul de implementare. Software-ul este disponibil precompilat sau ca cod sursă.

Acest framework a fost creat de Mark Otto și Jacob Thornton la Twitter pentru a îmbunătăți consistență instrumentelor utilizate pe site și de a reduce întreținerea. A servit drept ghid de stil pentru dezvoltarea instrumentelor interne la companie timp de peste un an înainte de lansarea sa publică și continuă să facă acest lucru și astăzi.

Instalarea acestuia se face cu ajutorul celebrului package manager, **npm**, ruland comanda **npm install bootstrap.**

* + 1. **NgRx**

NgRx este un framework pentru dezvoltarea aplicațiilor reactive în Angular. Acesta oferă biblioteci pentru:

* Gestionarea stării globale și locale
* Izolarea efectelor secundare pentru a promova o arhitectură mai curată a componentelor
* Gestionarea colectării entităților.
* Integrare cu Angular Router.
* Unelte pentru dezvoltatori care îmbunătățesc experiența dezvoltatorului atunci când construiesc multe tipuri diferite de aplicații.

Pachetele NgRx sunt împărțite în câteva categorii principale:

1. State

* **Store** - Gestionare globală a stării RxJS pentru aplicații Angular, inspirată de Redux.
* **Effects** - Model de efect secundar pentru @ngrx/store.
* **Router Store** - Legături pentru conectarea Angular Router la @ngrx/store.
* **Entity** - Adaptor de state pentru gestionarea colecțiilor de înregistrări.
* **ComponentStore** - Bibliotecă autonomă pentru gestionarea stării local/component.

1. Data

* Data - Extensie pentru gestionarea simplificată a datelor entității.

1. View

* Componentă - Extensie pentru aplicații Angular complet reactive.

1. Instrumente de dezvoltator

* Store Devtools - Instrumentare pentru @ngrx/store care permite urmărirea vizuală a stării și a depanării în timp.
* Schematics - Bibliotecă de schele pentru aplicații Angular care utilizează biblioteci NgRx.

**Când este recomandat să folosim NgRx?**

NgRx este o alegere bună atunci când aplicația creată este una cu o mulțime de interacțiuni cu utilizatorii și mai multe resurse de date sau când gestionarea stării cu ajutorul serviciilor nu mai este suficientă.

Pentru a ne putea da seama daca o aplicatie are nevoie de NgRx, putem lua in considerare principiul SHARI:

* **S**hared: state accesat de mai multe componente si servicii
* **H**ydrated: state care este persistat si rehidradat din storage-ul extern.
* **A**vailable: state care trebuie sa fie disponibil la reintrarea in rute.
* **R**etrieved: state care trebuie recuperat cu un efect secundar.
* **I**mpacted: state afectat de actiuni si alte resurse.

**Alte concepte cheie**:

* **Siguranta tipurilor de date**: Siguranța tipurilor este promovată în întreaga arhitectură, bazându-se pe compilatorul TypeScript pentru corectitudinea programului. În plus, strictețea NgRx în ceea ce privește siguranța tipului și utilizarea modelelor se pretează bine la crearea unui cod de calitate superioară.
* **Imuabilitate și performanță**: Store-ul este construit pe o structură de date unică, imuabilă, ceea ce face din detectarea modificărilor o sarcină relativ simplă folosind strategia OnPush. NgRx Store oferă, de asemenea, API-uri pentru crearea de funcții de selecție memoized care optimizează preluarea datelor din state.
* **Incapsularea**: Folosind NgRx Effects and Store, orice interacțiune cu efectele secundare ale resurselor externe, cum ar fi cererile de rețea sau socketurile web, precum și orice logică de afaceri, pot fi izolate de interfața de utilizare. Această izolare permite componente mai pure și simple și susține principiul responsabilității unice.
* **Serializabilitate**: Prin normalizarea modificărilor de stare și trecerea acestora prin observabile, NgRx oferă serializabilitate și asigură starea în mod previzibil stocată. Acest lucru permite starea să fie salvată în stocarea externă, cum ar fi localStorage.
  1. **.NET**
     1. **Introducere**

**.NET este o platformă de dezvoltator open source gratuită, multiplatform, pentru a construi mai multe tipuri diferite de aplicații. Aceasta poate fi utilizata prin intermediul mai multor limbaje de programare, precum:**

* **C#.NET**
* **VB.NET**
* **C++.NET**
* **J#.NET**
* **F#.NET**
* **JSCRIPT.NET**
* **Windows Powershell**
* **Iron Ruby**
* **Iron Python**
* **C Omega**
* **ASML (Abstract State Machine Language)**
  + 1. **Concepte**

**Componentele principale ale platformei .NET sunt:**

* **CLR (Common Language Runtime) -** Componenta de bază și mașina virtuală a .NET Framework. Acesta este mediul de execuție care rulează codul și ajută la facilitarea procesului de dezvoltare prin furnizarea diferitelor servicii și la gestionarea codului.
* **FCL (Framework Class Library) –** Este colecția de biblioteci și metode de clase reutilizabile, orientate spre obiecte, etc. care pot fi integrate cu CLR. Este la fel ca fișierele antet din C / C ++ și pachetele din java.
* **CTS (Common Type System)** – Descrie un set de tipuri de date care pot fi utilizate în diferite limbaje .Net în comun. CTS se asigură că obiectele scrise în diferite limbaje pot interacționa între ele.
* **CLS (Common Language Specification) –** Specifică un set de reguli care trebuie respectate sau satisfăcute de toți compilatorii de limbaje care vizează CLR. Ajută la moștenirea si depanarea între limbaje.

**De ce este multiplatform?**

Prin multiplatform se înțelege faptul că se pot crea aplicații .NET pentru diferite sisteme de operare, precum:

* Windows
* macOS
* Linux
* Android
* iOS
* tvOS
* watchOS

Arhitecturile procesorului suportate sunt**: x64, x86, ARM32, ARM64**.

Cateva dintre capacitatile limbajelor .NET sunt:

* Siguranța tipurilor de date
* Tipuri generice
* Delegați
* Lambda
* Evenimente
* Excepții
* Atribute
* Cod asincron
* Programare paralelă
* Analizatoare de cod

SDK-ul .NET este un set de biblioteci și instrumente pentru dezvoltarea și rularea aplicațiilor.

Descarcarea SDK-ului cuprinde urmatoarele componente:

* **.NET CLI** – Instrumente în linia de comandă care pot fi folosite pentru dezvoltarea locală și scripturile de integrare continuă.
* **.NET driver** – O comandă CLI care rulează aplicații dependente de framework.
* **Compilatoare pentru limbajele Roslyn si F#**
* **Motorul de construcție MSBuild**
* **Runtime-ul .NET –** Oferă un sistem de tip, încărcare de ansamblu, un colector de gunoi, interoperativ nativ și alte servicii de bază.
* **Biblioteci de execuție –** Oferă tipuri de date primitive și utilități fundamentale.
* **Runtime-ul ASP.NET Core** - Oferă servicii de bază pentru aplicații conectate la internet, cum ar fi aplicații web, aplicații IoT și backend-uri mobile.
* **Runtime-ul desktop** – Oferă servicii de bază pentru aplicațiile desktop Windows, inclusiv Windows Forms și WPF.
  + 1. **NuGet**

NuGet este un package manager open-source, conceput pentru .NET. Un pachet NuGet este un fișier .zip cu extensia .nupkg care conține cod compilat (DLL-uri), alte fișiere legate de acel cod și un manifest descriptiv care include informații precum numărul versiunii pachetului.

Dezvoltatorii care vor să partajeze cod, creează pachete și le publică pe nuget.org sau pe un host privat. Cei ce vor să utilizeze codul partajat, adaugă pachetul la proiectul lor și pot apela API-ul expus de pachet în codul de proiect.

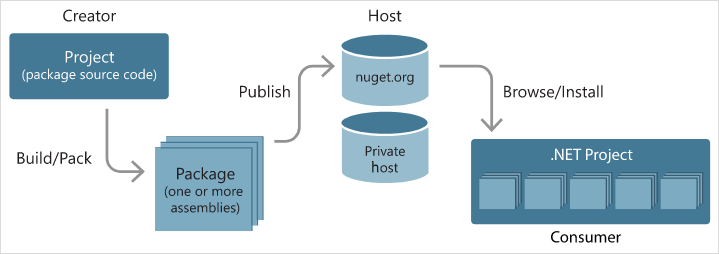


Figure 3.11: Fluxul de pachete între creatori, gazde și consumatori

De asemenea, pachetele trebuie sa fie si compatibile. Un pachet „compatibil” înseamnă că conține assembly-uri construite pentru cel puțin un target .NET framework compatibil cu un target framework al proiectului consumator.

* + 1. **Limbajul C#**

C# este un limbaj de programare modern, orientat pe obiecte, cu scop general, care poate fi utilizat pentru a îndeplini o gamă largă de sarcini și obiective care se întind pe o varietate de profesii. Acesta este utilizat în principal în cadrul Windows .NET, deși poate fi aplicat pe o platformă open source.

La fel ca alte limbaje de programare cu scop general, c# poate fi folosit pentru a crea o serie de programe și aplicații diferite: aplicații mobile, aplicații desktop, servicii bazate pe cloud, site-uri web, software de întreprindere și jocuri. Deși este extrem de versatil, există trei domenii în care este cel mai frecvent utilizat:

* Web development
* Aplicatii windows
* Jocuri video

Avantaje:

* **Simplitate și eficiență -** Programatorii petrec mai puțin timp scriind cod care este utilizat in mod repetat pe tot parcursul proiectului.
* **Curbă de învățare redusă –** Ușor de învățat pentru începători. Acest limbaj reprezintă un prim pas excelent în domeniu și oferă dezvoltatorilor aspiranți un mod confortabil de a se familiariza cu programarea.
* **Limbaj scalabil, ușor de menținut -** Datorită naturii stricte a modului în care trebuie scrise codurile statice, programele C# sunt consistente în mod fiabil, ceea ce ușor de ajustat și de întreținut.
* **Comunitate uriașă –** În lumea programării , existența unei comunități utile de care puteți depinde este importantă. Programatorii trebuie să se bazeze pe sprijinul altor persoane din același domeniu care au experimentat aceleași obstacole și frustrări.
* **Este orientat pe obiecte –** C# este complet orientat pe obiecte, principalele avantaje ale acestui aspect fiind eficiența și flexibilitatea.
  + 1. **ASP.NET Core Web API**

ASP. NET este un framework pentru construirea de servicii HTTP, care pot fi accesate de orice client, inclusiv browsere web și dispozitive mobile. Este o platformă excelentă pentru a construi aplicații RESTful.

**Ce este un API?**

În termeni simpli, un API este un fel de interfață ce conține un set de funcții care permit programatorilor să acceseze caracteristici sau date specifice ale unei aplicații sau alte servicii.

Un Web API este un API de pe web care poate fi accesat folosind protocolul HTTP. Acesta este ca un serviciu web sau WCF, dar excepția este că acceptă doar protocolul HTTP.

**Caracteristici**:

* Este o platforma ideală pentru construirea de servicii RESTful.
* Este construit peste ASP. NET și suportă pipe-ul de solicitare/răspuns ASP.NET.
* Mapează verbele HTTP cu numele metodelor
* Acceptă diferite formate de date de răspuns. Suport încorporat pentru formatul JSON, XML, BSON.
* Poate fi găzduit în IIS, auto-găzduit sau alt server web care acceptă .NET 4.0+.
* Framework-ul include noul HttpClient pentru a comunica cu serverul API Web. HttpClient poate fi utilizat în partea server ASP.MVC, aplicația Windows Form, aplicația Console sau alte aplicații.
  + 1. **AutoMapper**

AutoMapper este un mapator obiect-obiect bazad pe convenții. Acesta utilizează un API de configurare fluent pentru a defini o strategie de mapare obiect-obiect. AutoMapper utilizează un algoritm de potrivire bazat pe convenții pentru a potrivi sursa cu valorile destinației si este orientat către scenarii de proiecție a modelelor pentru a aplatiza modele complexe de obiecte la DTO-uri și alte obiecte simple, al căror design este mai potrivit pentru serializare, comunicare, mesagerie sau pur și simplu un strat anticorupție între domeniu și stratul aplicației.

**De ce AutoMapper?** Deoarece oferă configurarea simplă a tipurilor, precum și testarea simplă a mapărilor.

* + 1. **SignalR**

ASP.NET Core SignalR este o bibliotecă open-source care simplifică adăugarea comunicării în timp real la aplicații.

**Buni pretendenți pentru SignalR:**

* Aplicații care necesită actualizări de înaltă frecvență de la server. De exemplu: jocuri, rețele sociale, votul, licitații, hărțile și aplicațiile GPS.
* Tablouri de bord și aplicații de monitorizare. Exemple includ tablouri de bord ale companiei, actualizări instantanee de vânzări sau alerte de călătorie.
* Aplicații colaborative. Aplicațiile pentru tablă albă și software-ul pentru întâlniri de echipă sunt exemple de aplicații colaborative.
* Aplicații care necesită notificări. Rețelele sociale, e-mail, chat, jocuri, alerte de călătorie și multe alte aplicații utilizează notificări.

SignalR oferă un API pentru crearea apelurilor de la distanță de la server la client (RPC). RPC-urile apelează funcțiile JavaScript pe clienți din codul .NET Core de pe partea serverului.

**Caracteristici**:

* Gestionează automat conexiunea.
* Trimite mesaje tuturor clienților conectați simultan. De exemplu, o cameră de chat.
* Trimite mesaje anumitor clienți sau grupuri de clienți.
* Scale pentru a face față traficului în creștere.

SignalR suporta o serie de tehnici pentru manipularea comunicarii in timp real si o alege automat pe cea care se incadreaza cel mai bine intre capacitatile serverului si a clientului:

* **WebSockets**
* **Server-Sent Events**
* **Long Polling**

SignalR urmează următorii pași pentru a decide modul de transport:

1. Dacă browserul este Internet Explorer 8 sau mai versiune mai recentă, se folosește **Long Polling.**
2. Dacă JSONP este configurat, se folosește **Long Polling**
3. Dacă se face conexiunea între domenii si sunt îndeplinite următoarele criterii: atât clientul cât și serverul suportă WebSocket iar clientul suportă CORS, atunci modul de transport folosit va fi **WebSocket**.
4. Dacă JSONP nu este configurat și conexiunea nu se realizează între domenii, **WebSocket** va fi folosit, dacă clientul și serverul suportă acest tip de transport.
5. Dacă clientul sau server-ul nu suportă WebSockets, se va folosi **Server Sent Events**.
6. Dacă Server Sent Events nu este disponibil, se încearcă modul de transport **Forever Frame**
7. Dacă Forever Frame eșuează, se va folosi **Long Polling**.
   * 1. **Principiile SOLID in C#**

Principiile de proiectare SOLID din C # sunt principii de bază.

SOLID reprezinta:

* **S:** Single Responsibility Principle (SRP) – Orice clasa sau structura ar trebui sa aiba un singur scop. Acest principiu oferă o modalitate bună de a identifica cursurile în faza de proiectare a unei aplicații și poate scoate în evidență toate modurile în care o clasă se poate schimba.
* **O:** Open closed Principle (OSP) - trebuie să proiectăm modulul / clasa noastră în așa fel încât noua funcționalitate să poată fi adăugată numai atunci când sunt generate noi cerințe.
* **L:** Liskov substitution Principle (LSP) - Se asigură că o clasă derivată nu afectează comportamentul clasei părinte, cu alte cuvinte, că o clasă derivată trebuie să poată fi substituită clasei sale de bază.
* **I:** Interface Segregation Principle (ISP) - În loc de o interfață uriașă, multe interfețe mici sunt preferate pe baza grupurilor de metode, fiecare servind un submodul.
* **D:** Dependency Inverstion Principle (DIP) - Modulele / clasele de nivel înalt nu ar trebui să depindă de modulele / clasele de nivel scăzut. Ambele ar trebui să depindă de abstracții. În al doilea rând, abstracțiile nu ar trebui să depindă de detalii. Detaliile ar trebui să depindă de abstracții.
  1. **Baza de date**

O bază de date este o colecție organizată de informații structurate sau date, de obicei stocate electronic într-un sistem informatic. O bază de date este de obicei controlată de un sistem de gestionare al bazelor de date (SGBD).

Un sistem de gestionare al bazelor de date oferă diverse funcții care permit introducerea, stocarea și recuperarea unor cantități mari de informații și oferă modalități de gestionare a modului în care aceste informații sunt organizate.

In prezent exista mai multe tipuri de baze de date:

* **Baze de date ierarhice** – Datele sunt stocate într-un nod de relație părinte-copii. Pe lângă datele reale, înregistrările conțin și informații despre grupurile lor de relații părinte/copil.
* **Baze de date de rețea –** Sunt utilizate in principal pe computere digitale mari. Sunt baze de date ierarhizate, dar spre deosebire de bazele de date ierarhice în care un nod poate avea un singur părinte, un nod de rețea poate avea o relație cu mai multe entități.
* **Baze de date relaționale** - Legătura dintre date este relațională și datele sunt stocate sub formă de tabel de coloane și rânduri. Fiecare coloană a unui tabel reprezintă un atribut și fiecare rând dintr-un tabel reprezintă o înregistrare.
* **Baze de date orientate obiect** - Oferă funcții complete de programare a bazelor de date, în timp ce conține compatibilitate în limbajul natural.
* **Baze de date grafice -** Sunt baze de date NoSQL și utilizează o structură grafică pentru interogări semantice. Datele sunt stocate sub formă de noduri, margini și proprietăți.
* **Baze de date ER model -** Fiecare rând al unui tabel reprezintă o instanță a unui tip de entitate și fiecare câmp dintr-un tabel reprezintă un tip de atribut.
* **Baze de date de documente -** Sunt, de asemenea, baze de date NoSQL care stochează date sub formă de documente.
* **Baze de date NoSQL -** Sunt baze de date care nu folosesc SQL ca limbaj principal de acces la date.
  + 1. **MongoDB**

MongoDB este o bază de date distribuită cu scop general, bazată pe documente, construită pentru dezvoltatorii de aplicații moderne și pentru era cloud, care oferă performanță crescută, o disponibilitate crescută și scalabilitate automată.

MongoDB stochează date în documente flexibile, de tip JSON, ceea ce înseamnă că câmpurile pot varia de la un document la altul, iar structura datelor poate fi modificată în timp.

Modelul de document se mapează la obiectele din codul aplicației, făcând datele ușor de utilizat.

MongoDB, fiind o bază de date non-relațională, oferă avantajul posibilitații creării și gestionării unui volum mare de date care poate avea atat un model bine structurat, cât și unul nestructurat. Acest lucru oferă flexibilitate, modificările modelelor din server neavând un impact semnificativ asupra modelelor din baza de date precum în bazele de date de tip relațional.

Avantajele cheie ale MongoDB sunt:

* **Lipsa schemei** – Faptul că MongoDB este o bază de date bazată pe documente, oferă programatorilor liberatea de a stoca și grupa datele în orice mod. Structura unui obiect este simplă și clară.
* **Flexibilitatea query-urilor** – Se pot executa filtrări complexe, datorită unui limbaj de query foarte versatil.
* **Este ușor de scalat** – Scalarea bazei de date poate fi configurată ușor.
* **Suport pentru diferite tipuri de date** – Datorită structurii datelor, nu mai este necesară conversia sau maparea datelor aplicației la un tip specific, acestea sunt convertite automat în BSON.
* **Este bine documentat** - Fiind foarte popular, MongoDB dispune de o documentație foarte bine definită
* **Indexarea** - Se pot crea indecși care contribuie la creșterea vitezei de căutare.