



**Universitatea  
Transilvania  
din Brașov**  
**FACULTATEA DE MATEMATICĂ  
ȘI INFORMATICĂ**

## LUCRARE DE LICENȚĂ

**Absolvent:** Nicorescu Teodor-Ionuț

**Coordonator:** Conf. dr. Livia SÂNGEORZAN

Brașov

Iulie 2021



Universitatea  
Transilvania  
din Brașov  
**FACULTATEA DE MATEMATICĂ  
ȘI INFORMATICĂ**

# LUCRARE DE LICENȚĂ

Aplicație web pentru călătorii internaționale

**Absolvent:** Nicorescu Teodor-Ionuț

**Coordonator:** Conf. dr. Livia SÂNGEORZAN

Brașov

Iulie 2021

## Cuprins

Introducere.....	5
1.1    Culturi și tradiții.....	5
1.1.1    Tradiții .....	5
1.1.2    Culturi.....	6
1.1.3    Schimbul cultural, de ce contează?.....	6
1.2    Călătoriile .....	7
1.2.1    Argumente pro .....	7
1.2.2    Argumente contra .....	8
1.3    Motivarea alegerii temei .....	8
Dezvoltarea Web .....	9
2.1    Scurtă istorie .....	9
2.2    Definiții .....	10
2.3    Site Web vs Aplicație web .....	10
2.4    Aplicațiile web .....	11
2.5    Servicii web .....	12
2.5.1    Tipuri de servicii web .....	13
Programe și tehnologii folosite .....	14
3.1    Angular .....	14
3.1.1    Concepțe de bază .....	14
3.1.2    Introducere .....	15
3.1.3    Componente Angular .....	16
3.1.4    Ciclurile de viață ale unei componente .....	18

3.1.5	Service și dependency injection .....	19
3.1.6	Directive .....	20
3.1.7	Pipes .....	24
3.1.8	Angular router și navigarea .....	26
3.1.9	Formulare .....	28
3.1.10	Comunicarea cu servicii back-end .....	29
3.1.11	Testarea aplicației .....	30
3.1.12	Internăționalizarea aplicației .....	30
3.1.13	Angular Material UI .....	31
3.1.14	DevExtreme Angular .....	33
3.1.15	Bootstrap .....	33
3.1.16	NgRx .....	34
3.2	.NET .....	36
3.2.1	Introducere .....	36
3.2.2	Concepțe .....	36
3.2.3	NuGet .....	38
3.2.4	Limbajul C# .....	39
3.2.5	ASP.NET Core WEB API .....	40
3.2.6	AutoMapper .....	40
3.2.7	SignalR .....	41
3.2.8	Principiile SOLID în C# .....	42
3.3	Baza de date .....	43
3.3.1	Introducere .....	43
3.3.2	MongoDB .....	44

Dezvoltarea Proiectului .....	45
4.1    Introducere .....	45
4.2    Dezvoltarea pe etape .....	46
4.2.1    Pagina de start .....	46
4.2.2    Pagina de autentificare .....	47
4.2.3    Pagina de înregistrare .....	48
4.2.4    Bara de instrumente a aplicației .....	51
4.2.5    Pagina de profil .....	53
4.2.6    Organizarea unei excursii .....	58
4.2.7    Vizualizarea excursiilor .....	60
4.2.8    Vizualizarea detaliată a unei excursii .....	62
4.2.9    Bara de navigare laterală .....	65
Concluzii și planuri de viitor .....	66
5.1 Concluzii .....	66
5.2 Planuri de viitor .....	66
Bibliografie .....	68

## Capitolul 1

### Introducere

#### 1.1 Culturi și tradiții

##### 1.1.1 Tradiții

O tradiție este o credință sau un comportament (obicei popular) transmis în cadrul unui grup sau societate cu semnificație simbolică sau semnificație specială cu origini în trecut. Exemplele obișnuite includ sărbătorile sau hainele impracticabile, dar semnificative din punct de vedere social (cum ar fi perucile avocaților sau pînenii ofițerilor militari), dar ideea a fost aplicată și normelor sociale, cum ar fi salutările. Tradițiile pot persista și evoluă 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, capacitatele 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 sablon pentru aşteptările 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 vîrstă poartă kimonos, aranjează flori în vase și au ceremonii de ceai. Unele țări se opun unor lucruri din cultura lor, cum ar fi discriminarea sau religia.

### 1.1.3 Schimbul cultural, de ce contează?

„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 aștepta. 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 concentreză 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 însine dar, de fapt, vorbind și conectându-ne cu ceilalți, vedem clar că suntem mai asemănători decât diferiți. Iubim, râdem și plângem pentru aceleasi 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.

## 1.2 Călătoriile

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 și va veti extinde cunoștințele
- Calatoria va poate extinde viziunea asupra lumii
- Puteti invata limbi noi
- Va puteti face prieteni noi
- Amintiri pentru o viață
- Va poate ajuta să scăpati de o viață plăcăsitoare

### 1.2.2 Argumente contra

- Calatoria poate fi costisitoare
- S-ar putea să dai probleme
- Probleme de mediu

## 1.3 Motivarea alegerii temei

Călătoria este forma perfectă de evadare din rutina zilnică și plăcătă, 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 căuta 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 ati 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 experiență 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

#### 2.1 Scurtă istorie

Creșterea internetului în masă este în mare parte corelată cu istoria sa ca mediu vizual. La fel ca multe alte sisteme și aplicații informatiche, 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.

## 2.3 Site web vs Aplicație web

Cele mai multe persoane s-au obisnuit să numească "site" orice pagina vizibilă pe internet, dar există câteva diferențe între un site web și o aplicație 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, plăti electronice și multe altele.

Un alt punct important în diferențierea 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 fură informațiile, precum și pot irita alți utilizatori cu e-mailuri nedorite sub numele dvs.

## 2.4 Aplicațiile web

O aplicație web reprezintă un software care rulează și 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 realizează una sau mai multe cereri către server prin protocolul HTTP
2. Serverul primește acea cerere și este procesată
3. Serverul trimite înapoi către utilizator un răspuns HTTP
4. Utilizatorul primește răspunsul

Un utilizator poate fi conectat și trimite cereri către mai multe servere, dar acestea au și ele un dezavantaj, și anume faptul că pot cadea datorită unui număr mare de cereri simultan.

Dacă mai sus am folosit termenii "server" și "HTTP", urmează 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.

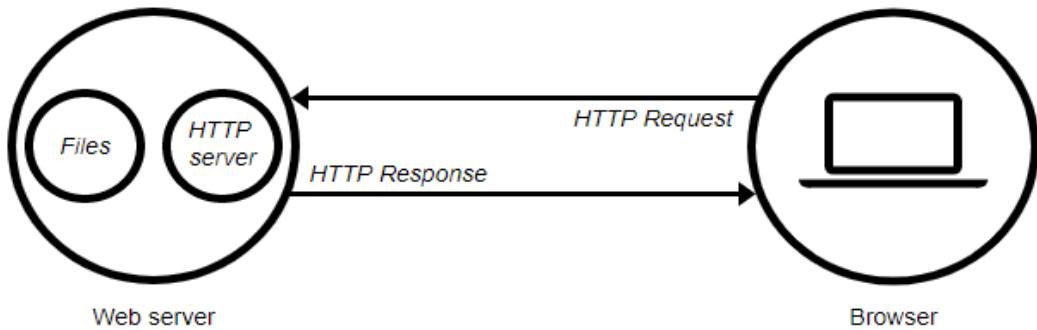


Figure 2.1: Exemplu client-server

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

## 2.5 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.

### 2.5.1 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 punct-la-punct
  - Lipsa standardelor

## Capitolul 3

# Programe și tehnologii folosite

### 3.1 Angular

#### 3.1.1 Concepte de bază

- **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 permitând TypeScript să valideze codul.

Desi în Angular este posibil să scriem și cod javascript, acest lucru nu este recomandat. Angular a fost creat să funcționeze împreună 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 limbi 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 în 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 fișiere externe, salvate cu extensia .css.  
Acestea pot fi utilizate pentru a determina aspectul unui întreg site web printr-un singur fișier.

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

### 3.1.2 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 și 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.

```
@NgModule({
  imports: [CommonModule, TranslocoLocaleModule.init()],
  providers: [ScreenService],
  declarations: [SideNavigationMenuComponent, SideNavigationInnerToolbarComponent],
  exports: [SideNavigationInnerToolbarComponent]
})
export class SharedAppNavigationFeatureModule {}
```

Figure 3.1: Exemplu de NgModule

- NgModules colectează codul aferent în seturi funcționale;
- O aplicație 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`. Odată ce acesta a fost instalat, putem genera o aplicație Angular prin comanda `ng new <nume_aplicație>`.

Rularea aplicației se va face prin comanda `ng serve --open`.

### 3.1.3 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 conține decoratorul `@Component()` este automat identificată ca fiind o componentă și declară următoarele informații specifice framework-ului Angular:

- Un selector 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 optional de stiluri CSS care definesc aspectul elementelor HTML ale template-ului.

Orice aplicație Angular conține cel puțin o componentă, aceasta fiind componentă 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 componentă poate fi inclusă într-o altă componentă prin selectorul unic, iar acestea își pot transmite date între 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.

```
@Component({
  selector: 'hk-account-trips-paginator',
  templateUrl: './account-trips-paginator.component.html',
  styleUrls: ['./account-trips-paginator.component.scss'],
})
export class AccountTripsPaginatorComponent implements OnInit, OnDestroy {
  @ViewChild(MatPaginator, { static: true }) paginator: MatPaginator;

  @Input()
  count: number;
  @Output()
  pageChanged = new EventEmitter();

  alive = true;
  constructor() {}

  ngOnInit(): void {
    this.paginator.page.pipe(takeWhile(() => this.alive)).subscribe((page) => {
      this.pageChanged.emit(page.pageIndex + 1);
    });
  }

  ngOnDestroy(): void {
    this.alive = false;
  }
}
```

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 fișierul .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 și raspunde la

acțiunile utilizatorului, cum ar fi apăsarea tastelor, mișările mouse-ului, clicurile și atingerile.

```
someText = "Nice text to render"; <p>{{someText}}</p>
```

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

### 3.1.4 Ciclurile de viață ale unei componente

O instanță componentă are un ciclu de viață care începe atunci când Angular instantiază 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 distrugе instanțа componentă și elimină şablonul redat din DOM. Angular realizeaza acest lucru cu ajutorul unor interfețe pe care componența noastră le poate implementa. Aceste interfețe conțin o singură metodă, a cărei denumire este numele interfeței cu prefixul `ng`. Acestea se apelează în felul următor:

- **ngOnChanges**: Când se modifică o valoare a unei variabile ce conține decoratorul `@Input()` sau `@Output()`. Aceasta metodă nu se apelează dacă componenta nu conține 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 apelează exact în ordinea declarată mai sus.

### 3.1.5 Service și dependency injection

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

```
@Injectable({
  providedIn: 'root',
})
export class GooglePlacesService {
  constructor(
    @Inject(Config) private config: Config,
    private httpClient: HttpClient
  ) {}
```

Figure 3.3: Exemplu de service in Angular

Există 2 moduri pentru a face un serviciu să fie singleton în Angular:

1. Setăm proprietatea `providedIn` din interiorul decoratorului `@Injectable` ca 'root' (Metoda folosită în figura 3.3, care este și cea recomandată)
2. Includem service-ul în `AppModule` sau într-un modul care este importat de `AppModule` (`AppModule` este modulul radacina al aplicației)

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 injectie, ceea ce înseamnă că injectoarele imbicate 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 lenes în timpul rulării, Angular poate crea un injector pentru acesta cu proprii furnizori.

### 3.1.6 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.

#### 2. 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`.

```
<mat-form-field
  [ngClass]="{
    'theme-invalid-mat-form-field': isInvalidLocation && isSubmittedOnce,
    'theme-valid-mat-form-field': !(isInvalidLocation && isSubmittedOnce)
  }"
  id="googleInputField"
  appearance="outline">
```

Figure 3.4: Exemplu de directiva ngClass

În figura 3.4, `isInvalidLocation` și `isSubmittedOnce` sunt valori de tip boolean. Cand acestea respectă condiția impusă, atunci clasa respectivă este adăugată elementului HTML.

### 3. Directive structurale

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.

```
<div class="example" *ngIf="condition">
| This is the text that will get rendered only if condition is true
</div>
```

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.

```
<div class="example-ngFor" *ngFor="let object of list">
| <p>{{ object.denumire }}</p>
</div>
```

Figure 3.6: Exemplu directivă NgFor

În 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.

```
<div class="example-ngSwitch" [ngSwitch]="inputValue">
  <p *ngSwitchCase="1">You have selected value 1</p>
  <p *ngSwitchCase="2">You have selected value 2</p>
  <p *ngSwitchCase="3">You have selected value 3</p>
  <p *ngSwitchDefault>Sorry! Invalid selection....</p>
</div>
```

Figure 3.7: Exemplu directivă NgSwitch

În 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.

#### 4. 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 directivă pentru a schimba aspectul sau comportamentul elementului gazdă.

### **Cum putem crea o directivă?**

**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 directivă minimală.

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

```
@Directive({
  selector: '[appHighlight]'
})
export class HighlightDirective {

  constructor(private el: ElementRef) { }

  @HostListener('mouseenter') onMouseEnter() {
    this.highlight('yellow');
  }

  @HostListener('mouseleave') onMouseLeave() {
    this.highlight('');
  }

  private highlight(color: string) {
    this.el.nativeElement.style.backgroundColor = color;
  }
}
```

Figure 3.8: Exemplu de directivă personalizată

### 3.1.7 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 există 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.

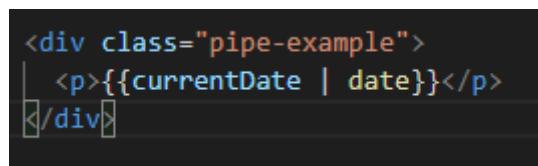


Figure 3.9: Exemplu utilizare pipe

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 să formateze sau să transforme valoarea de intrare în 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 PipeTransform

**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.

```
import {Pipe, PipeTransform} from '@angular/core';
@Pipe ({
  name : 'sqrt'
})
export class SqrtPipe implements PipeTransform {
  transform(val : number) : number {
    return Math.sqrt(val);
  }
}
```

Figure 3.10: Exemplu pipe personalizat

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

### 3.1.8 Angular router și 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.

**Există 3 pași fundamentali pentru crearea unei rute:**

**Pasul 1: Importam RouterModule și Routes în modulul de rutare.** Angular-CLI efectuează acest pas automat și configuraază importurile și exporturile 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 puțin 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.

### 3.1.9 Formulare

Gestionarea datelor introduse de utilizator cu formulare este fundamentalul 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**. Ambele 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** – Oferă acces direct, explicit la modelul obiect, sunt mai scalabile, reutilizabile și 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.

Diferențe cheie între formularele **reactive** și cele **bazate pe template**

	Formulare Reactive	Formulare Bazate pe template
Configurarea modelului	Explicit, creat în componentă	Implicit, creat de directive
Model de date	Structurat și nevariabil	Nestructurat și 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.

### 3.1.10 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:

- Abilitatea 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.

### 3.1.11 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 grija 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 aşa 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.

### 3.1.12 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 în 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 forme 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 comutăm între ele cu ușurință în timpul rulării aplicației. Această librărie expune un API bogat pentru a gestiona traducerile în mod eficient și curat. Aceasta permite traducerea atât printr-un pipe cât și printr-o directivă.

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

### 3.1.13 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, independentă dispozitivelor și degradarea grăioasă.

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 predefinite sau putem seta una personalizată.

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

- Controle 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 și ferestre pop-up
- Indicatori – spinner, progress bar

### 3.1.14 DevExtreme Angular

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

Suia 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 în cazul în care se dorește crearea unui proiect de la 0. Aceasta permite generarea unei aplicații simple, cu un meniu de navigare și cîteva exemple de view-uri. Aceasta aplicație se poate genera în câțiva pași cu ajutorul DevExtreme CLI, rulând comanda următoare: `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 comandă: `npx -p devextreme-cli devextreme add devextreme-angular`.

### 3.1.15 Bootstrap

Bootstrap 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 consistența 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, rulând comanda `npm install bootstrap`.

### 3.1.16 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.
- UTELTE 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.
2. Data
  - Data - Extensie pentru gestionarea simplificată a datelor entității.
3. View
  - Componentă - Extensie pentru aplicații Angular complet reactive.
4. 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 dacă o aplicație are nevoie de NgRx, putem lua în considerare principiul SHARI:

- Shared: state accesat de mai multe componente și servicii
- Hydrated: state care este persistat și rehidratat din storage-ul extern.
- Available: state care trebuie să fie disponibil la reintrarea în rute.
- Retrieved: state care trebuie recuperat cu un efect secundar.
- Impacted: state afectat de acțiuni și alte resurse.

### Alte concepte cheie:

- **Siguranța 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 interfață 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.

## 3.2 .NET

### 3.2.1 Introducere

.NET este o platformă de dezvoltator open source gratuită, multiplatform, pentru a construi mai multe tipuri diferite de aplicații. Aceasta poate fi utilizată prin intermediul mai multor limbi 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)

### 3.2.2 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 limbi .Net în comun. CTS se asigură că obiectele scrise în diferite limbi pot interacționa între ele.
- **CLS (Common Language Specification)** – Specifică un set de reguli care trebuie respectate sau satisfăcute de toți compilatorii de limbi care vizează CLR. Ajută la moștenirea și depanarea între limbi.

### 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**.

Câteva dintre capacitatile limbajelor .NET sunt:

- Siguranță 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 următoarele 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 și 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.

### 3.2.3 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.

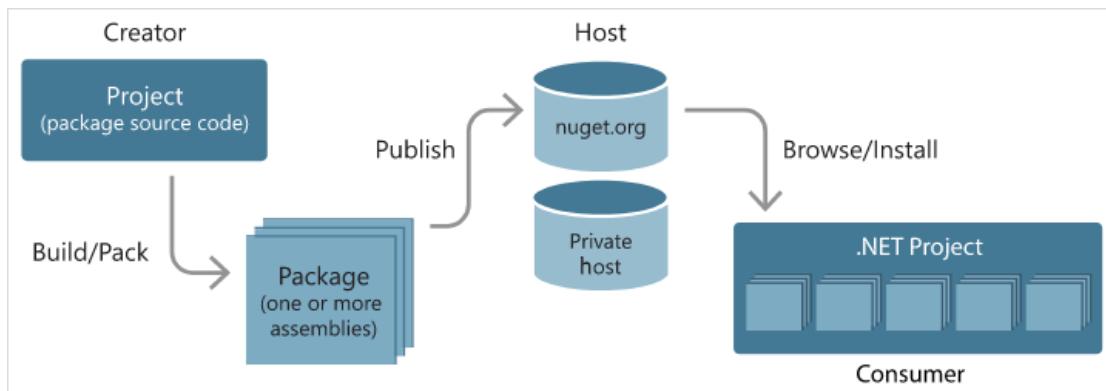


Figure 3.11: Fluxul de pachete între creatori, gazde și consumatori

De asemenea, pachetele trebuie să fie și 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.

### 3.2.4 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
- Aplicații windows
- Jocuri video

Avantaje:

- **Simplitate și eficiență** – Programatorii petrec mai puțin timp scriind cod care este utilizat în mod repetat pe tot parcursul proiectului.
- **Curbă de învățare redusă** – Ușor de învățat pentru începători. Acest limbaj reprezintă un prim pas excelent în domeniul 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 eficiență și flexibilitatea.

### 3.2.5 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 platformă 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.

### 3.2.6 AutoMapper

AutoMapper este un mapator obiect-obiect bazat 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 și este orientat către scenarii de proiecție a modelelor pentru a aplativa 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.

### 3.2.7 SignalR

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

**Buni pretendanți 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:**

- Gestioneză automat conexiunea.
- Trimit mesaje tuturor clientilor conectați simultan. De exemplu, o cameră de chat.
- Trimit mesaje anumitor clienti sau grupuri de clienti.
- Scale pentru a face față traficului în creștere.

SignalR suportă o serie de tehnici pentru manipularea comunicării în timp real și o alege automat pe cea care se incadrează cel mai bine între capacitatele serverului și 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 ulterioară, va fi utilizat **Long Polling**.
2. Dacă JSONP este configurat, se va utiliza **Long Polling**.
3. Dacă conexiunea dintre domenii este stabilită și sunt îndeplinite următoarele criterii: Dacă clientul și serverul acceptă WebSockets și clientul acceptă CORS, modul de transport utilizat va fi **WebSockets**.
4. Dacă JSONP nu este configurat și nu este stabilită o conexiune între domenii, va fi utilizat **WebSockets**, dacă clientul și serverul acceptă acest tip de transport.
5. Dacă clientul sau serverul nu acceptă WebSockets, va fi utilizat **Server Sent Events**.
6. Dacă Server Sent Events nu este disponibil, se va încerca modul **Forever Frame**
7. În cazul în care Forever Frame eșuează, se va utiliza **Long Polling**.

### 3.2.8 Principiile SOLID în C#

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

SOLID reprezinta:

- **S:** Single Responsibility Principle (SRP) – Orice clasa sau structura ar trebui să 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 astfel încât să putem adăuga noua funcționalitate doar când apar noi cerințe.
- **L:** Liskov substitution Principle (LSP) - Se asigură că clasa derivată nu afectează comportamentul clasei părinte, altfel spus, că clasa derivată trebuie să poată fi substituită clasei 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 Inversion Principle (DIP) - Modulul / clasa de nivel înalt nu ar trebui să depindă de modulul / clasa de nivel scăzut. Amandouă ar trebui să se bazeze pe abstracții. În al doilea rând, abstractizarea nu ar trebui să depindă de detalii. Detaliile ar trebui să depindă de abstractizare.

### 3.3 Baza de date

#### 3.3.1 Introducere

O bază de date este o colecție organizată de informații sau date structurate, stocate în general în formă electronică într-un sistem informatic. Bazele de date sunt în general controlate de un sistem de gestionare a 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.

În prezent există 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 în 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 relationale** – 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.

### 3.3.2 MongoDB

MongoDB este o bază de date distribuită bazată pe documente cu scop general, concepută pentru cei mai noi dezvoltatori de aplicații și pentru era cloud, oferind performanțe mai mari, o disponibilitate mai mare și o capacitate de expansiune automată.

MongoDB stochează date în documente JSON flexibile, adică, câmpurile variază de la un document la altul, iar structurile de date se pot schimba în timp.

Şablonul de document se potrivește cu obiectele din codul aplicației, facilitând astfel lucrul cu datele.

Bazele de date relationale oferă avantajul de a putea crea și gestiona cantități mari de date care pot avea atât modele bine structurate, cât și nestructurate. Aceasta oferă flexibilitate, deoarece modificările aduse modelului din server nu au un impact semnificativ asupra modelelor din baza de date, cum ar fi într-o bază de date relatională.

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 indecsi care contribuie la creșterea vitezei de căutare.

## Capitolul 4

# Dezvoltarea Proiectului

### 4.1 Introducere

Aplicația dezvoltată este o aplicație web ce oferă informații despre diferite locuri ale lumii și oportunitatea clientului să organizeze excursii sau drumeții în locurile respective, urmând că alți utilizatori să se alăture acestora.

Clientul efectuează o căutare cu ajutorul a diverse filtre, urmând că acestuia să i se afișeze cele mai relevante excursii bazate pe filtrele alese de el. Acesta poate vizualiza cele mai importante informații despre excursia respectivă, după care va trebui să selecteze una. În momentul selecție , utilizatorul va fi redirectionat pe o pagină cu excursia mult mai detaliată , urmând că acesta să se alăture sau să utilizeze opțiunea "alege altă excursie" , ceea ce îl va redirectiona înapoi pe pagina cu excursii. Dacă clientul nu a găsit nimic relevant, acesta poate organiza o excursie utilizând opțiunea " Creează o excursie" , unde va trebui să specifice diverse detalii despre aceasta, să o previzualeze, apoi să o finalizeze, urmând că excursia respectivă să fie publicată și disponibilă pentru ceilalți utilizatori.

Aplicația poate fi accesată cu ușurință, cu ajutorul unui browser web, ceea ce simplifică utilizarea utilizarea acesteia.

Aplicația a fost dezvoltată astfel încât să poată fi accesată de pe orice fel de browser și orice fel de sistem de operare. Mai mult decât atât, este o aplicație responsivă la ecranul clientului, astfel că paginile vor fi afișate întotdeauna corect, indiferent de dimensiunea ecranului sau dispozitivul utilizat (smartphone, tabletă, laptop, televizor).

Datorită faptului că aplicația se accesează printr-un browser, clientul nu are nevoie de spațiu de stocare adițional.

## 4.2 Dezvoltarea pe etape

### 4.2.1 Pagina de start

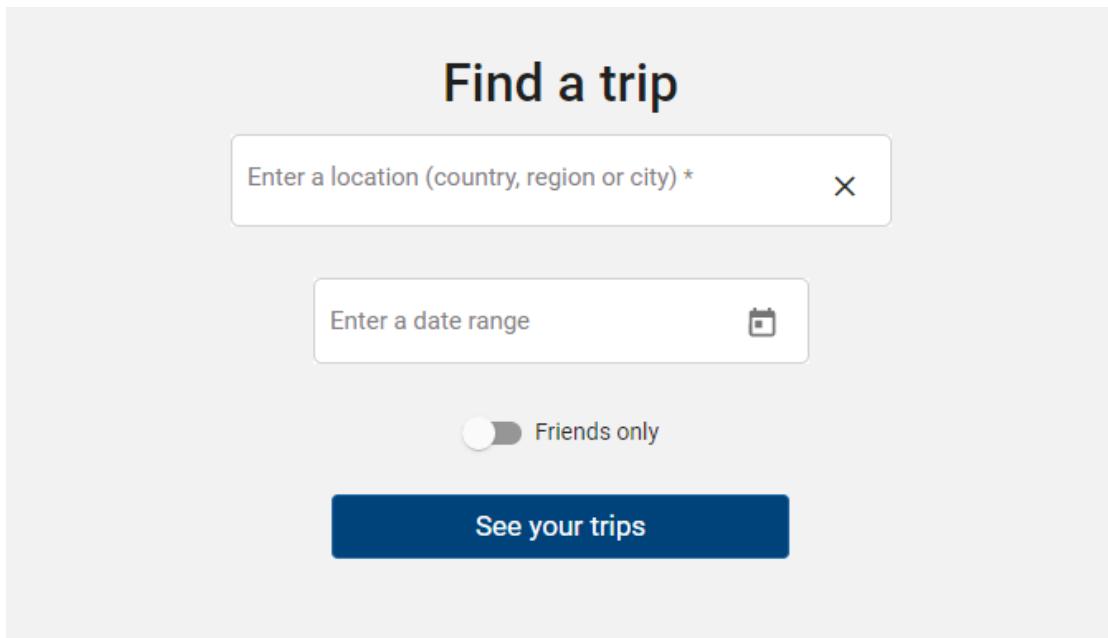


Figure 4.1: Pagina principală

Pagina principală a aplicației este și pagina de căutare a excursiilor. În această se află un formular cu ajutorul căruia utilizatorului îi este permis să caute excursii cu următoarele filtre: **Locație**, **data de început**, **data de încheiere** și **butonul de comutare "Doar prietenii"**.

- **Câmpul pentru locație** – Excusiile se organizează într-o anumită localitate, oraș sau regiune a unei țări. Prin urmare, acest câmp permite utilizatorilor filtrarea exactă a acestora.
- **Intervalul de date** – Bineînteleas ca o excursie trebuie să aibă o data de început și o data de încheiere. De aceea, acest filtru permite utilizatorului să caute doar excusiile care se află în intervalul respectiv.
- **Butonul de comutare** – Activarea acestui filtru va face ca utilizatorului să ii fie afisate doar excusiile create de cei din lista prietenii. Acest buton este disponibil doar dacă utilizatorul este autentificat. În caz contrar, acesta nu va fi afisat în pagina.

Pentru a putea efectua o căutare, utilizatorul este nevoit să completeze fiecare câmp, altfel i se va afișa un mesaj de eroare sugestiv.

The screenshot shows a search interface with two error messages. The first message, 'Enter a location (country, region or city) \*', is displayed in red at the top of the input field. Below it, a smaller message says 'You need to select a location from the list'. The second message, 'Enter a date range', is also displayed in red at the top of another input field, with the sub-message 'Select a valid date range' below it. Both error messages are enclosed in red-bordered boxes.

Figure 4.2: Mesaje de eroare în formularul de căutare

#### 4.2.2 Pagina de autentificare

The screenshot shows a 'Sign-in' page. It features two input fields: 'Email' and 'Password', each with a placeholder text and a corresponding icon (person for email, lock for password). Below these fields is a large blue 'Sign-in' button. At the bottom of the page, there is a link 'You don't have an account? [Sign-up](#)'.

Figure 4.3: Autentificarea în aplicație

În pagina de autentificare se află un formular simplu, cu două câmpuri. Pentru a se autentifica, utilizatorul trebuie să introducă adresa de email și parola. Formularul este unul reactiv, ce include validatori atât pentru câmpul de email, cât și pentru cel pentru parolă.

Dacă utilizatorul introduce date eronate, formularul va afișa în jurul câmpurilor o serie de mesaje de eroare sugestive, astfel încât utilizatorul să poată corecta greșelile.

The screenshot shows a login form with two fields. The first field is labeled 'Email' and contains the value 'fsafsa@nothing'. To its right is a user icon. Below the input field, a red error message reads 'Please provide a valid email!'. The second field is labeled 'Password' and has a lock icon to its right. Below it, another red error message reads 'Please provide your password'.

Figure 4.4: Validare în formularul de autentificare

În figura de mai sus este exemplificat un formular eronat. Dacă câmpul pentru adresa de email nu ar fi fost completată deloc, atunci ar fi apărut o altă eroare care să specifică acest lucru. De asemenea, Formularul nu poate fi trimis cât timp acesta conține erori.

După autentificarea cu succes, utilizatorul este redirectionat către pagina solicitată anterior și care nu a putut fi accesată fără logare sau, dacă nu a fost solicitată o astfel de pagină, atunci utilizatorul va fi redirectionat către pagină principală.

În cazul în care utilizatorul nu are un cont în aplicație, atunci acesta poate fi redirectionat către pagina de înregistrare folosind opțiunea din partea de jos.

#### 4.2.3 Pagina de înregistrare

Deși utilizatorul poate utiliza aplicația atât autentificat, cât și neautentificat, pentru a beneficia de anumite opțiuni, acesta va fi nevoie să se înregistreze. Pentru a se înregistra, clientul va trebui să introducă o serie de informații, precum: prenumele, numele de familie, adresa de email, data de naștere și parolă, urmând să reintroducă parola încă o dată pentru confirmare.

Dacă datele introduse de utilizator sunt eronate, acestuia îi vor fi afisate mesaje de eroare sugestive, oferindu-i sansa de a corecta erorile.

De asemenea, dacă acesta deține deja un cont, are opțiunea să navigheze direct către pagina de autentificare.

## Create account

First name \*

First name is required.

Last name \*

Last name is required.

Email \*

Email is required.

Birthday \*

Birthday is required.

Password \*

..

Password should have at least 8 characters

Repeat password

..

Passwords must match

Create your account

Already have an account? [Sign-in](#)

The screenshot shows a 'Create account' form with six input fields. The first three fields (First name, Last name, Email) have red borders and error messages below them: 'First name is required.', 'Last name is required.', and 'Email is required.'. The fourth field (Birthday) has a red border and an error message: 'Birthday is required.' The fifth field (Password) has a red border and an error message: 'Password should have at least 8 characters'. The sixth field (Repeat password) also has a red border and an error message: 'Passwords must match'. Below the form is a blue button labeled 'Create your account' and a link 'Already have an account? [Sign-in](#)'.

Figure 4.5: Pagina de înregistrare

În această pagină se află, de asemenea, un formular reactiv, format din 6 câmpuri, toate fiind obligatorii. La fel ca celealte formulare, și acesta folosește validatori pentru a evita ca utilizatorul să trimită date eronate precum câmpuri necompletate sau o parolă prea slabă.

```
this.signupForm = this.formBuilder.group({
  firstName: [this.user.firstName, [Validators.required]],
  lastName: ['', [Validators.required]],
  email: [
    '',
    [
      Validators.required,
      Validators.pattern('[A-Za-z0-9._%-]+@[A-Za-z0-9._%-]+\.\.[a-z]{2,3}'),
    ],
  ],
  password: ['', [Validators.required, Validators.minLength(8)]],
  repeatPassword: [''],
  birthday: [this.user.birthday, [Validators.required]],
});
```

Figure 4.6: Formular reactiv

Formularul reactiv de mai sus este cel folosit în pagina de înregistrare. Fiecare proprietate este un form-control, care conține valoarea implicită a acestuia și o serie de validatori, precum:

- **Prenumele și numele de familie sunt obligatorii.** Aceste trebuie să contină cel puțin un caracter.
- **Adresa de email este obligatorie**, dar spre deosebire de nume și prenume, aceasta are un validator în plus care verifică că adresa să fie în formatul corect.
  - **Exemplu email valid:** teodor@gmail.com
  - **Exemplu email invalid:** teodor@gmail
- Parola este obligatorie și trebuie să contină cel puțin 8 caractere.
- Campul de confirmare parola nu are validator, acesta validându-se odată ce apasăm butonul de submit.
- **Data nașterii** conține, de asemenea, un singur validator care face campul să fie obligatoriu.

Formularul nu poate fi trimis dacă conține erori. Odată ce înregistrarea este efectuată cu succes, utilizatorul va fi, ulterior, autentificat și redirecționat către pagină principală.

#### 4.2.4 Bara de instrumente a aplicației

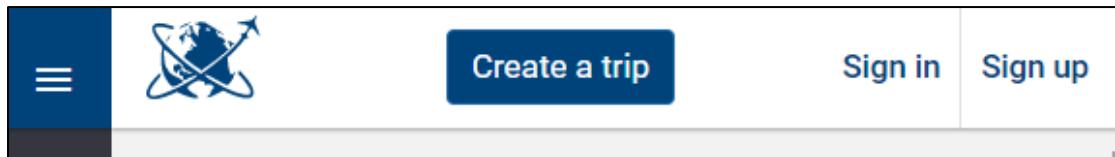


Figure 4.7: Bară de instrumente pentru utilizatori neautentificați

Bara de instrumente este vizibilă în permanent, indiferent de pagina pe care navigăm. Aceasta conține:

- Simbolul aplicației, care poate fi utilizat pentru navigarea pe pagina principală.
- Un buton pentru crearea unei excursii – dacă utilizatorul este autentificat, va fi redirectionat către o pagină unde va putea să creeze o excursie și să o previzualizeze, altfel va fi redirectionat către pagina de autentificare.
- Butoanele pentru navigarea către paginile de autentificare și înregistrare – acestea sunt vizibile doar dacă utilizatorul nu este autentificat.



Figure 4.8: Bară de instrumente pentru utilizatori autentificați

Pentru utilizatori autentificați, se poate observa că bara de instrumente conține mai multe opțiuni, acestea fiind:

- Simbolul aplicației și butonul pentru crearea excursiei, care au fost explicate mai devreme.
- Un buton care va deschide un meniu cu două tab-uri, unul ce conține lista de prieteni, și unul care conține cererile de prietenie. De fiecare dată când utilizator trimite o cerere de prietenie, utilizatorul către care se face cererea va primi o

notificare cu acest lucru instantaneu și va putea accepta sau respinge cererea.

- Butonul care va deschide lista de conversații, de unde vom putea alege una sau putem naviga către pagina de chat.
- Butonul care va deschide lista de notificări. Un utilizator primește notificări atunci când cineva se alătură unei excursii organizate de el, o părăsește, anulează o excursie la care era participant sau când face o cerere de participare la excursie.

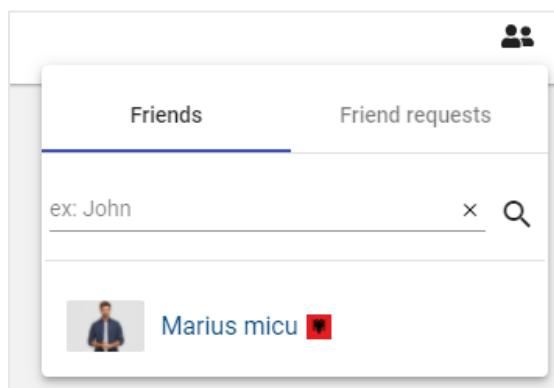


Figure 4.9: Panoul cu prieteni

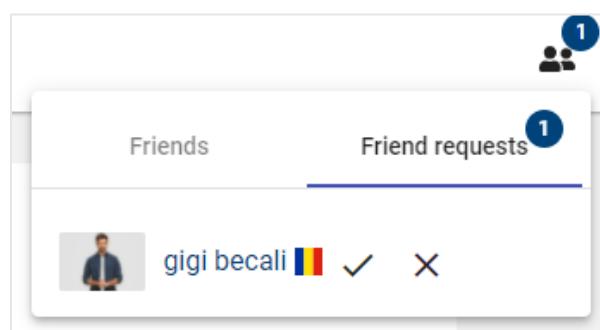


Figure 4.10: Panoul cu cereri de prietenie

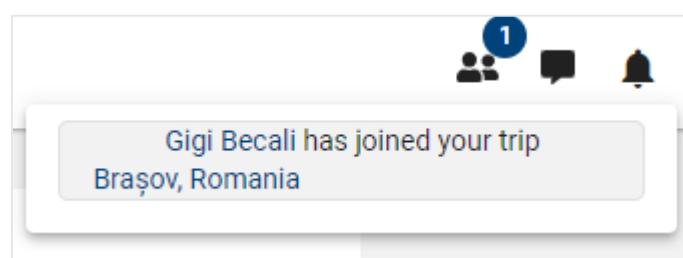


Figure 4.11: Panoul cu notificări

În figura 4.10 se poate observa o cerere de prietenie. Aceasta are două butoane, primul este pentru a accepta cererea de prietenie, iar al doilea pentru a o respinge.

De asemenea, pe bara de instrumente, în partea dreaptă se află un element ce conține poza de profil a utilizatorului și prenumele acestuia. Dacă apăsăm pe acest element, se va deschide un meniu de unde putem să navigăm către pagina cu detaliile contului sau să ne deconectăm.

#### 4.2.5 Pagina de profil

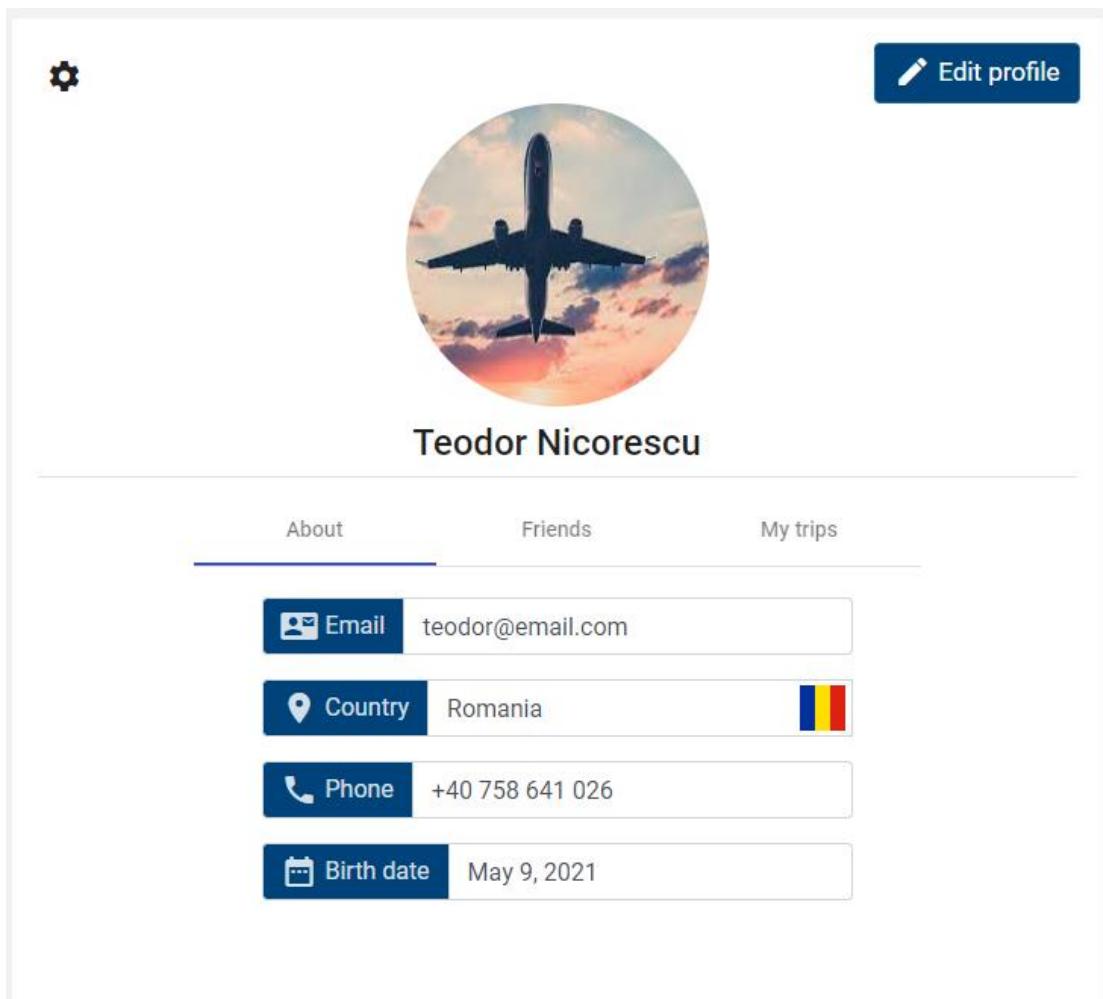


Figure 4.12: Pagina cu detaliiile utilizatorului

În partea de sus a acestei pagini se află două butoane și poza de profil.

Butonul din partea stânga va deschide un meniu de unde putem selecta opțiunea de schimbare a parolei care va deschide un formular unde vom fi nevoiți să introducem parola veche și o parolă nouă pentru a o putea schimba. La fel ca cele anterioare, acesta este tot un formular reactiv, în care informațiile trebuie introduse corect sau vor apărea mesaje de eroare și nu vom putea continua până nu le fixăm.

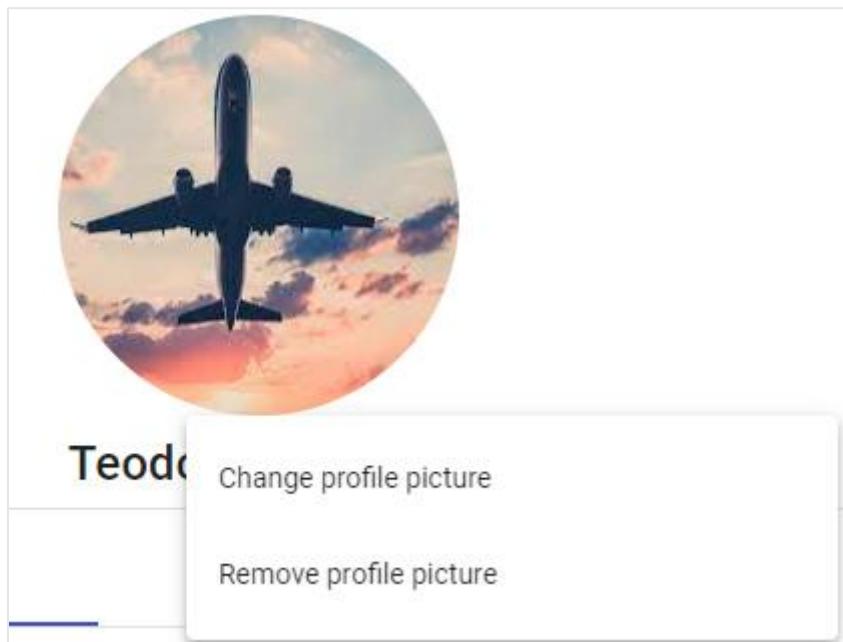


Figure 4.13: Opțiuni poză de profil

Dacă utilizatorul va apăsa un click pe imaginea de profil, acestuia i se va deschide un meniu cu două opțiuni.

- Prima opțiune este de a schimba poza de profil, și utilizatorului i se va deschide un selector de fișiere de unde va putea selecta doar imagini. Dimensiunea unei imagini trebuie să fie de maxim 200x200px, altfel se va deschide o fereastră de dialog în care utilizatorul va trebui să selecteze doar o parte din imaginea respectivă.
- A doua opțiune este de a șterge poza de profil. Această opțiune este destul de simplă, adică, odată apăsat, va șterge poza de profil a utilizatorului și i se va adauga una implicită.

Ultimul buton din partea de sus a paginii este cel de editare profil. Acesta va deschide o fereastră de dialog de unde utilizatorul va putea alege ce câmp dorește să editeze.

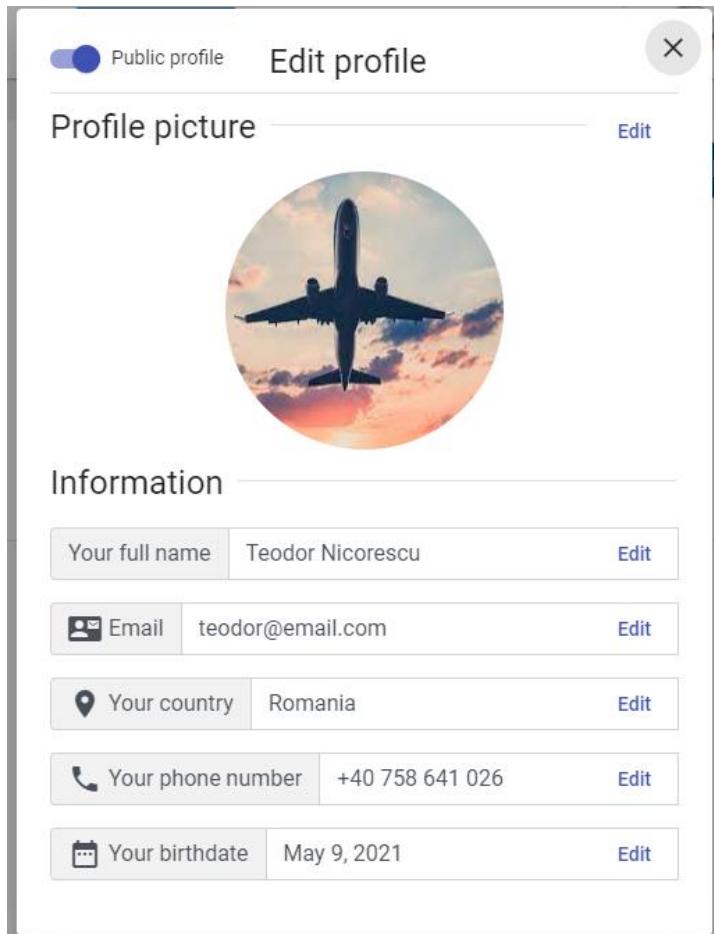


Figure 4.14: Fereastra de editare profil

După cum se poate observa în imaginea de mai sus, utilizatorul poate alege să editeze orice câmp, cu ajutorul butonului de editare, care va deschide o fereastră de dialog ce conține valoarea curentă a acelui câmp într-o formă editabilă.

În partea de sus a acestei pagini se află un buton de comutare. Dacă acesta este activ, atunci profilul va fi vizibil și pentru alți utilizatori (mai puțin numărul de telefon care este vizibil doar pentru cei din lista de prieteni). În caz contrar, utilizatorilor care vor dori să vizualizeze acest profil, li se va afișa mesajul "Acest profil este privat".

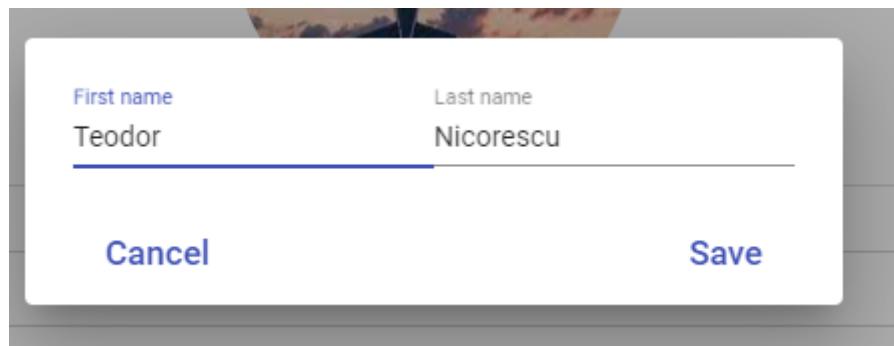


Figure 4.15: Fereastră de editare câmp

Indiferent de câmpul care urmează să fie editat, în fereastra respectivă se află un formular reactiv validat.

Utilizatorul va putea să salveze modificarea doar dacă formularul nu conține erori, sau are, de asemenea, opțiunea să anuleze modificările. Odată salvate modificările, profilul se actualizează instantaneu.

În partea de jos a paginii din figura 4.12 se află 3 tab-uri. Primul tab, care este selectat implicit, conține informațiile utilizatorului, precum adresa de email, țara, numărul de telefon și data nașterii.

În al doilea tab se află, din nou, o lista cu prietenii acestuia, de unde putem naviga pe profilul oricărui.

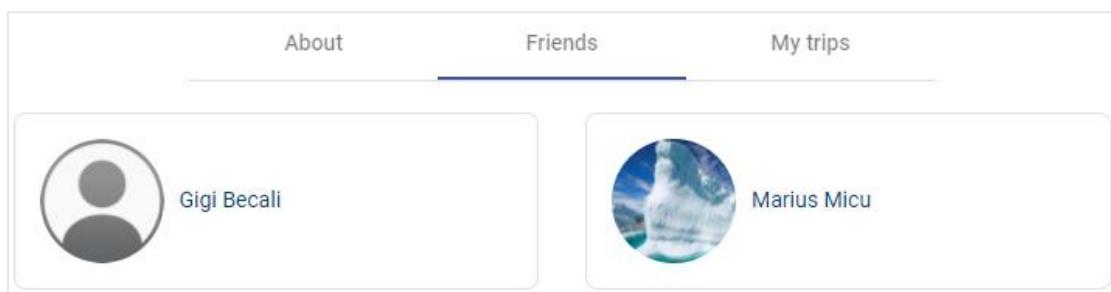


Figure 4.16 Tab-ul de prieteni de pe pagina de profil

În al treilea tab se află lista cu excursii ale utilizatorului. Excursiile organizate de acesta sunt separate de cele la care a fost doar participant.

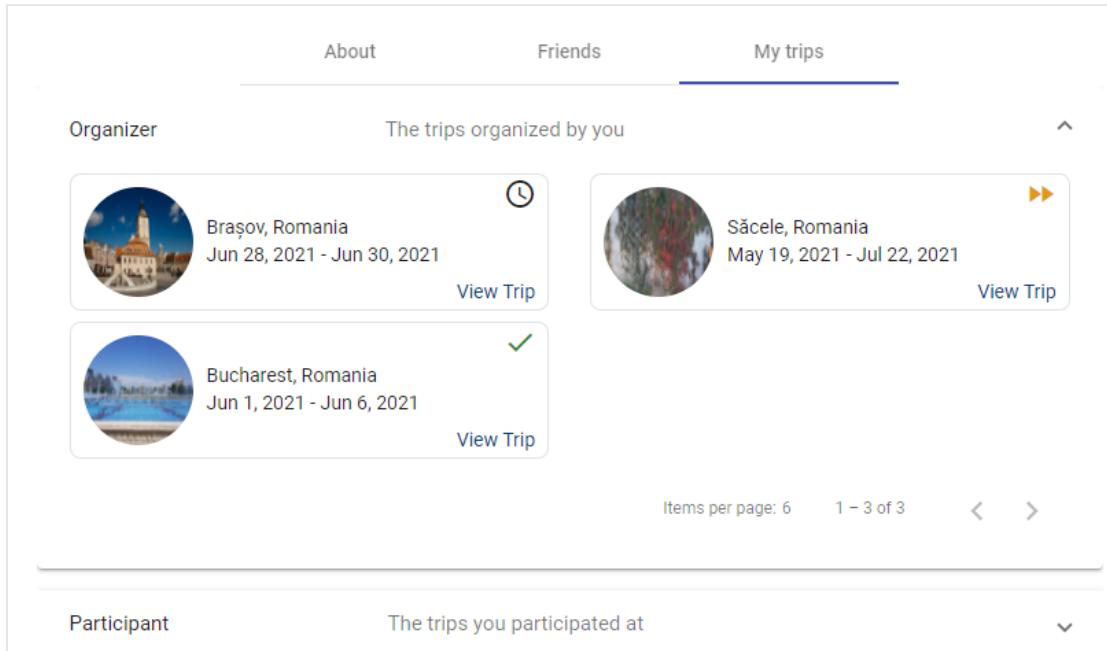


Figure 4.17: Tab-ul cu excursiile utilizatorului

Detaliile unei excursii sunt încadrate într-un chenar. Acest chenar conține imaginea în miniatură a excursiei, locația, perioada, un buton cu ajutorul căruia utilizatorul poate naviga către pagină cu excursia mult mai detaliată și un simbol care reprezintă starea acesteia. Starea unei excursii poate fi:

- În planificare
- În progres
- Încheiată
- Anulată
- Anulată de către administrator

Excursiile nu pot afișate mai mult de 6 odată, motiv pentru care în partea de jos există un paginator pe care utilizatorul va trebui să îl folosească pentru a încărca pagina următoare sau anterioară de excursii (dacă este cazul).

După cum se observă și în figura 4.17, excursiile sunt separate cu ajutorul unui panou de expansiune. În momentul în care utilizatorul deschide următorul panou, cel deschis anterior va fi închis.

Datele din orice tab sunt încărcate folosind metoda lazy-loading. Acest lucru înseamnă că datele sunt aduse din baza de date în momentul în care

utilizatorul deschide un tab, și nu sunt încărcate toate în momentul navigării pe pagină, aspect care sporește performanța aplicației.

#### 4.2.6 Organizarea unei excursii

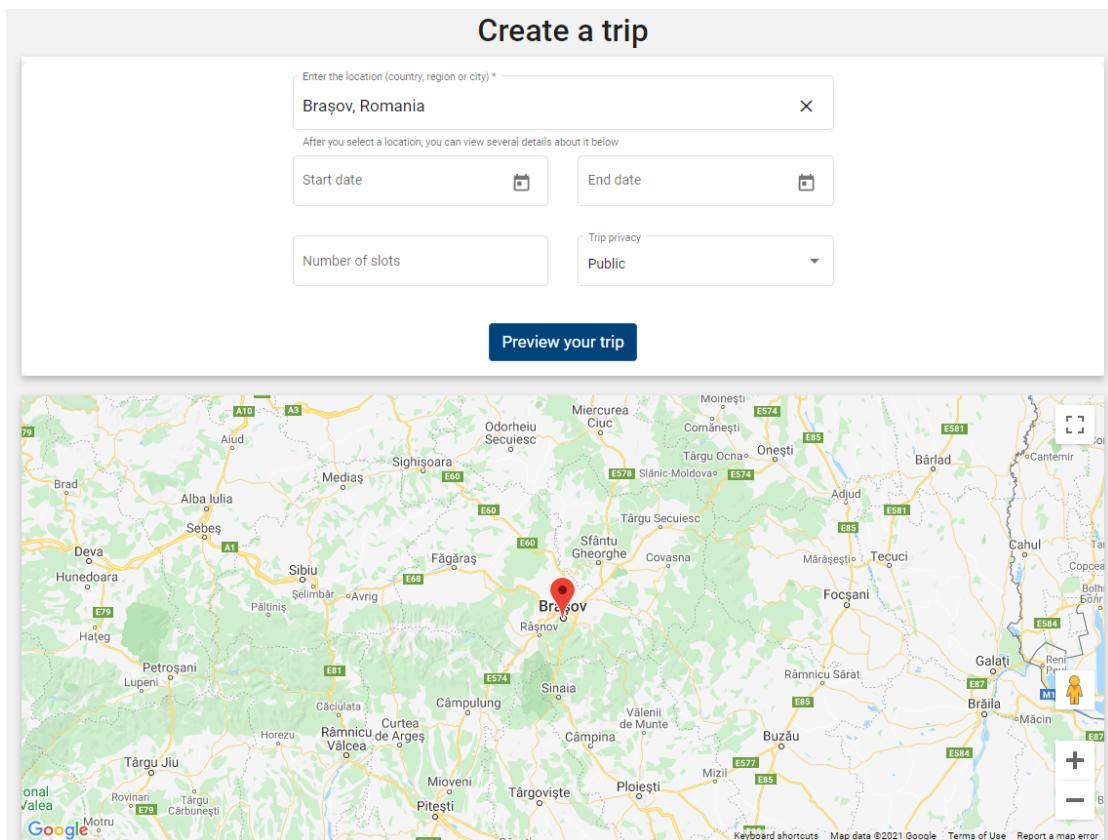


Figure 4.18: Crearea unei excursii

În pagina de creare a unei excursii se află un formular reactiv cu următoarele campuri și validări:

- **Locația** – locul unde se organizează excursia. Valoarea acestui câmp trebuie să fie neapărat selectată din sugestiile afișate cu ajutorul Google Places Autocomplete și nu poate fi lăsată necompletată.
- **Data de început și data de încheiere** a acesteia
- **Numărul de locuri disponibile** – într-o excursie trebuie să fie cel puțin 1 loc, altfel va fi afișat un mesaj de eroare și nu se va putea continua către etapa următoare.

- **Tipul excursiei** – o excursie poate fi publică, privată sau să necesite aprobarea organizatorului. Într-o excursie publică poate participa oricine printr-un simplu click, pe când la cele private pot participa doar cei invitați de către organizator.

Pe lângă un formular, în partea de jos a paginii se află și o hartă Google. În momentul în care clientul selectează o locație din câmpul formularului, harta se va centra pe locația respectivă și va crea un marcator. Utilizatorul poate mari sau micșora pentru a vizualiza împrejurimile și poate, de ce nu, să selecteze o nouă locație mai convenabilă.

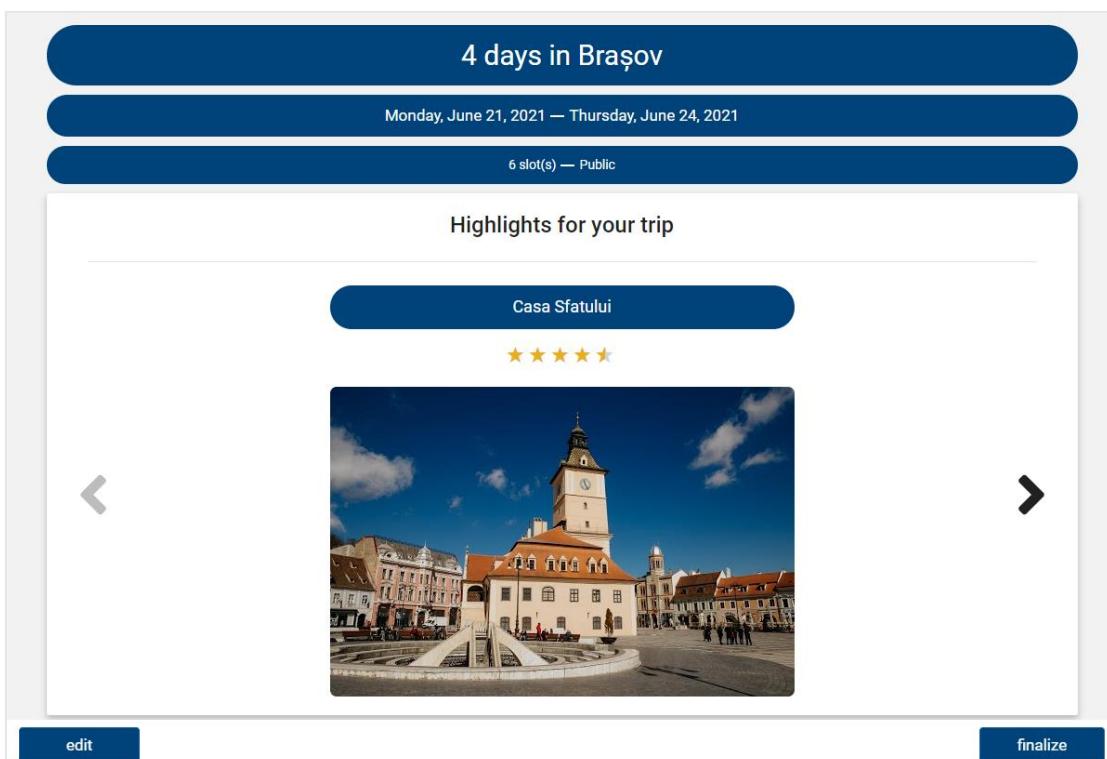


Figure 4.19: Pagina pentru previzualizarea excursiei

Odată ce am completat corect formularul din pagină de creare a excursiei, înainte de a finaliza, trebuie să navigăm către pagina de previzualizare a excursiei.

În această pagină, utilizatorul poate vizualiza detaliile ce le-a introdus anterior și câteva atracții turistice reprezentate de numele, rating-ul și câte o imagine de ansamblu.

În partea de jos a paginii se află butoanele care reprezintă urmatoarele acțiuni disponibile:

- **Butonul de edit** – cu ajutorul acestui buton putem naviga înapoi pe pagina de creare a excursiei. Formularul va rămâne completat cu datele actuale iar utilizatorul le poate modifica.
- **Butonul de finalizare** – acest buton va finaliza procesul și se va încerca crearea excursiei și adăugarea acesteia în baza de date, urmând să fie vizibilă pentru ceilalți utilizatori. Indiferent dacă procesul este încheiat cu succes sau cu o eroare, utilizatorul va fi întărit printr-un mesaj toast sugestiv și va fi redirectionat către pagina principală.

#### 4.2.7 Vizualizarea excursiilor

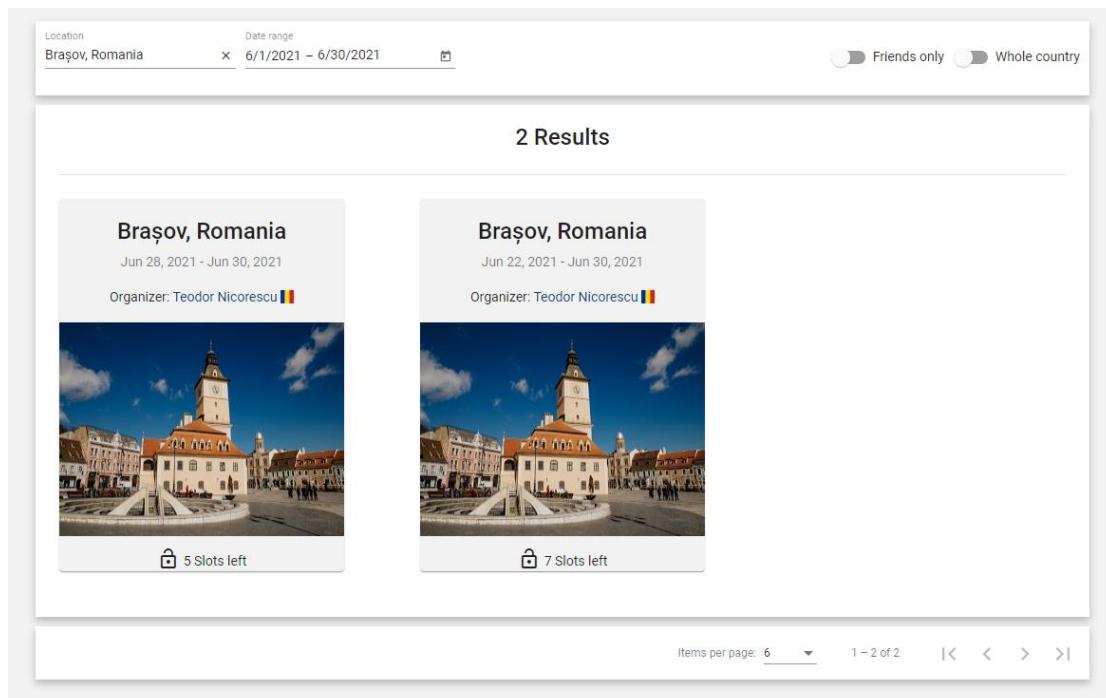


Figure 4.20: Lista cu excursii relevante

În figura 4.1 am prezentat pagina principală a aplicației și am menționat că aceasta este și pagina de căutare a excursiilor. Dacă căutarea este efectuată cu succes, utilizatorul va fi redirectionat către pagina din figura 4.20, altfel se va afișa un mesaj toast cu o eroare sugestivă.

Pagina contine:

- Filtre
- Lista de excursii
- Paginator

### 1. Filtrele

Filtrele sunt cele completate anterior, dar mai apare și un filtru adițional ce este reprezentat printr-un buton comutativ. În cazul în care utilizatorul nu este încântat de excursiile afișate din orașul pe care l-a introdus, acesta poate activa filtrul respectiv și i se vor afișa, în ordinea relevanței, și restul excursiilor din țara respectivă.

De fiecare dată când un filtru se schimbă, se încarcă, din nou, și excursiile ce respectă condițiile.

### 2. Lista de excursii

O excursie este reprezentata printr-un chenar care contine:

- Locația
- Data de început și încheiere
- Numele organizatorului cu link către profilul acestuia
- O imagine care reprezintă locația
- Un simbol ce reprezintă intimitatea excursiei ( publică sau necesită aprobare)
- Numărul de locuri rămase

Dacă nu se găsește nicio excursie care să corespundă filtrelor, utilizatorului i se va afișa un mesaj corespunzător.

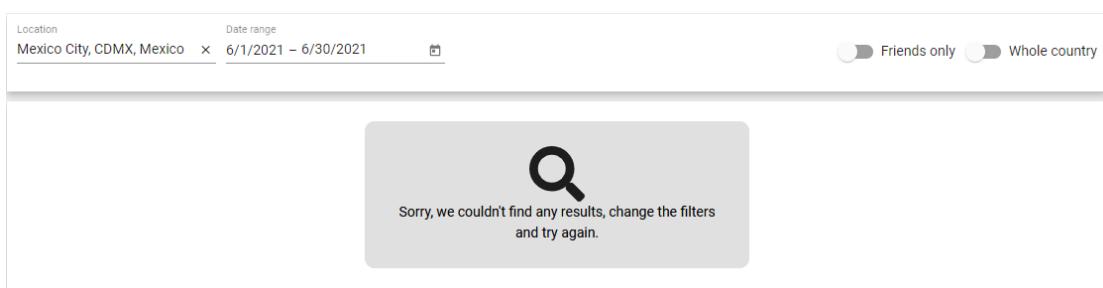


Figure 4.21: Lista goală de excursii

De asemenea, dacă o excursie are toate locurile ocupate, această nu va mai fi afișată în listă până nu se eliberează cel puțin 1 loc.

Pentru a vizualiza mai multe detalii și a putea participa la o excursie, utilizatorul va trebui să selecteze una din listă, urmând a fi redirecționat pe o altă pagină.

### 3. Paginatorul

Paginatorul oferă utilizatorului posibilitatea de a alege câte articole să se afișeze în listă și să navigheze printre acestea. Paginarea articolelor sporește atât organizarea cât și performanța aplicației.

Paginatorul conține 4 butoane care efectuează navigarea către:

1. Prima pagină de articole
2. Pagina anteroară de articole
3. Pagina urmatoare de articole
4. Ultima pagină de articole

#### 4.2.8 Vizualizarea detaliată a unei excursii

The screenshot displays a travel application interface. At the top, there are two tabs: "About trip" (which is active) and "Tourist attractions". Below the tabs, the section title "Organizer" is shown, followed by the name "Teodor Nicorescu" and a small profile icon. A large circular thumbnail image of an airplane flying over a sunset sky is centered. To the right of the organizer's information, three dark blue rounded rectangles provide key trip details: "3 days in Brașov", "Monday, June 28, 2021 — Wednesday, June 30, 2021", and "5 slot(s) left — Public". Below this, the section title "Other participants" is displayed, showing a profile picture and the name "gigi becali" next to a small Romanian flag icon. At the bottom of the screen, there are two buttons: "Pick another trip" on the left and "Leave trip" on the right.

Figure 4.22: Informații despre o excursie

Odată ce utilizatorul s-a hotărât asupra unei excursii, acesta o poate selecta printr-un simplu click, ceea ce îl va redirecționa către o pagină unde îi vor fi afișate mult mai detaliat informații despre aceasta.

Pagina conține două tab-uri:

1. Tab-ul în care sunt afișate atât informații tehnice despre excursie, cât și informații despre organizator și ceilalți participanți.
2. Tab-ul în care se pot vizualiza atracții turistice din locația respectivă.

Numele oricărui utilizator este afișat sub formă de link, de unde putem naviga către profilul public al acestora.

În partea stângă, sub numele organizatorului se află un buton cu ajutorul căruia poți trimite o cerere de prietenie, o poți anula sau poți șterge un prieten din listă.

Butonul poate fi afișat în felul următor:

-  - Dacă organizatorul nu se află în lista de prieteni. Un simplu click pe acest buton va trimite o cerere de prietenie către acesta.
-  - Dacă ai trimis deja o cerere de prietenie către organizator. Un click pe acest buton va deschide un meniu de unde vom putea anula cererea de prietenie.
-  - Dacă organizatorul se află în lista de prieteni. Odată apăsat, acest buton va deschide un meniu cu două opțiuni: Trimite un mesaj sau șterge din lista de prieteni.

În partea de jos a paginii se află două butoane:

- Butonul cu ajutorul căruia putem naviga pe pagina precedentă, de unde putem selecta o altă excursie (acesta conține textul "alege altă excursie").

- Un buton care se comportă diferit în funcție de următoarele criterii:
  1. Dacă clientul este deja participant la excursie, acesta va afișa textul "Părăsește excursia" și, odată apăsat, îl va șterge din lista de participanți.
  2. Dacă clientul nu se află pe lista de participanți iar excursia este publică, atunci acesta va afișa textul "Alătură-te excursiei". Un simplu click va adăuga utilizatorul pe lista de participanți.
  3. Dacă clientul nu se află pe lista de participanți iar participarea la excursie necesită aprobarea organizatorului, atunci pe buton se va afișa textul "Cere aprobare". Un click pe acest buton va trimite o cerere de participare către organizator, care va putea fi acceptată sau respinsă de către organizator.
  4. Dacă clientul este chiar organizatorul excursiei, atunci butonul va afișa "Anulează excursia" iar un click pe acesta va schimba starea excursiei în "Anulată".

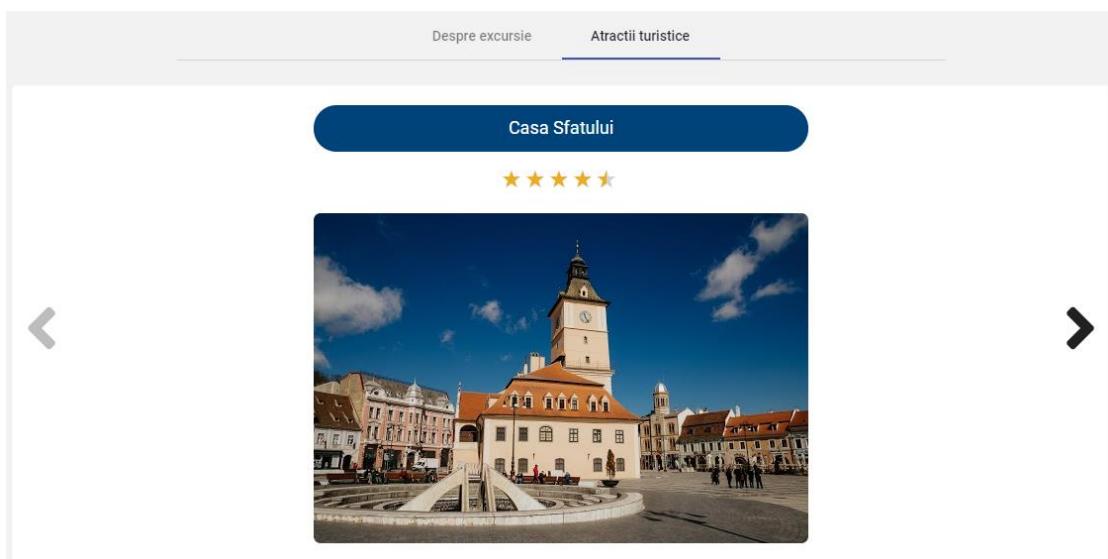


Figure 4.23: Tab-ul cu atracțiile turistice

Dacă în primul tab au fost afișate detalii despre utilizator și informații tehnice despre excursie, pe acest tab sunt afișate câteva dintre posibilele atracții

turistice din locația respectivă. Utilizatorul poate naviga printre acestea cu ajutorul săgeților din lateral. Detaliile despre o atracție turistică sunt: Numele, rating-ul și o imagine care să o reprezinte.

#### 4.2.9 Bara de navigare laterală

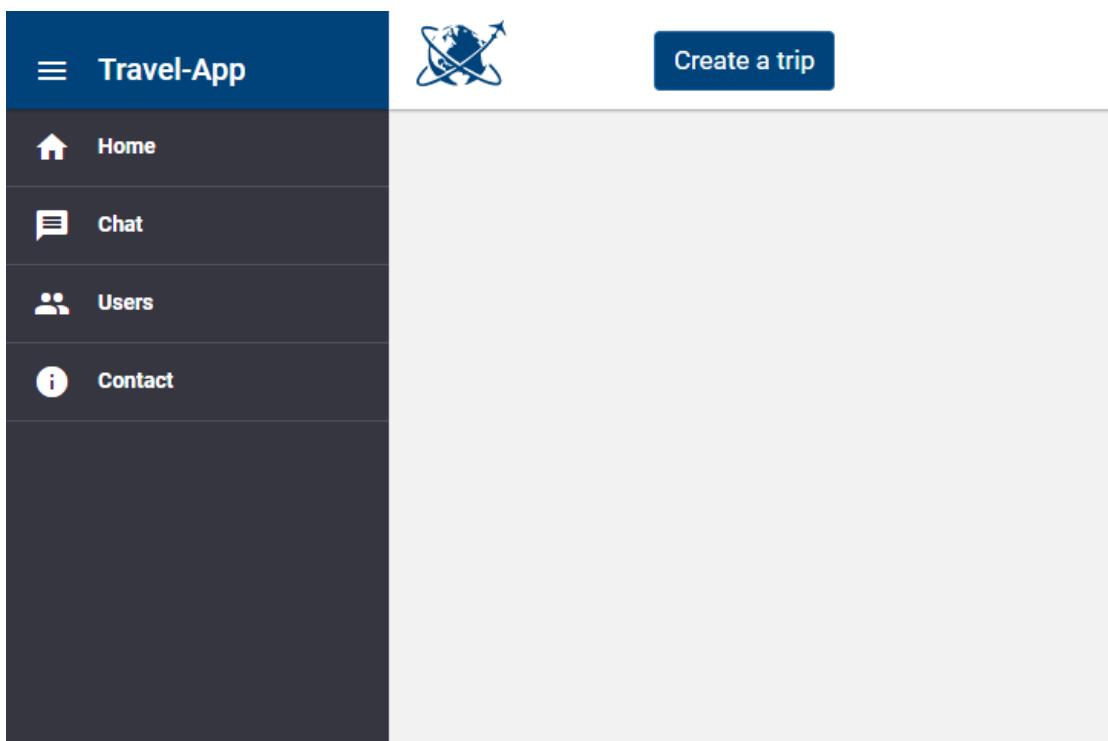


Figure 4.24: Bara de navigare laterală

Pe lângă elementele principale ce contribuie la navigarea de la o pagină la alta, aplicația dispune și de o navigare laterală care poate fi restrânsă și deschisă cu ajutorul unui buton.

Bara de navigare laterală este vizibilă pe toată durata utilizării aplicației, și face posibilă navigarea către:

- Pagina principală
- Pagina de chat
- Pagina cu utilizatori, de unde îi putem adăuga în lista de prieteni
- Pagina de contact

## Capitolul 5

# Concluzii și planuri de viitor

### 5.1 Concluzii

Proiectul are ca și obiectiv dezvoltarea unei aplicații care să ofere ajutor suplimentar persoanelor atunci când vine vorba de timp liber și călătorii. Aplicația dorește îmbunătățirea comunicării dintre oameni indiferent de naționalitatea sau originea acestora, îmbunătățirea modului de a petrece timpul liber și îmbogățirea cunoștințelor despre alte locuri, culturi sau tradiții din lume. Nu în ultimul rând, aplicația ajută utilizatorii să își facă prieteni sau legături noi și să socializeze. Toate aceste obiective au fost atinse prin următoarele rezultate:

- S-a implementat o aplicație prin care se pot efectua căutări despre orice oraș al lumii, într-un mod ușor și intuitiv.
- S-a adăugat opțiunea de a crea excursii pe orice perioadă, atât publice, cât și private.
- S-a creat un sistem de căutare al excursiilor, bazat pe preferințele fiecărui.
- S-a adăugat un sistem de prieteni, oferind utilizatorilor libertatea să accepte sau să respingă o cerere de prietenie, și în același timp să poată șterge un alt utilizator din lista de prieteni.
- S-a adăugat un sistem de conversații, astfel că utilizatorii pot avea o scurtă discuție înaintea participării la o excursie, aspect ce susține atât socializarea, cât și siguranța într-o excursie.
- S-a adăugat un sistem de notificări, astfel că utilizatorii vor fi mereu la curent cu actualizările apărute.
- S-a creat o interfață cât mai prietenoasă și sugestivă, prin care se pot gestiona cu ușurință datele sistemului.

### 5.2 Planuri de viitor

Având în vedere că vorbim despre o aplicație modernă, mereu va exista loc de mai bine și de noi extensii. Câteva dintre aceste funcționalități care s-ar putea implementa sunt:

- Posibilitatea de a alege punctele turistice care vor fi urmate pe parcursul călătoriei.
- Crearea unui algoritm care să calculeze traseul zilnic pe care oamenii l-ar putea urma pentru a-și îmbunătăți experiența.
- Posibilitatea de a organiza excursii în modul business. Acest aspect pune în evidență faptul că organizatorul se va ocupa de toate detaliile unei excursii (inclusiv cazarea) și va fi, de asemenea, și un ghid turistic pentru participanți. Pentru că această extensie să fie pusă în practică, va fi nevoie de un sistem de plată.
- Un beneficiu foarte mare al aplicației ar fi integrarea acesteia pe dispozitive mobile.

## Bibliografie

- [1] <https://en.wikipedia.org/wiki/Tradition>
- [2] <https://simple.wikipedia.org/wiki/Culture>
- [3] <https://greenheart.org/blog/exchange/cultural-exchange-why-it-matters/>
- [4] <https://angular.io/guide/what-is-angular>
- [5] <https://angular.io/start>
- [6] Rutare Angular, <https://angular.io/start/start-routing>
- [7] Introducere în Formulare, <https://angular.io/start/start-forms>
- [8] Componente Angular, <https://angular.io/guide/component-overview>
- [9] Cicluri de viață ale unei Componente, <https://angular.io/guide/lifecycle-hooks>
- [10] Încapsularea view-ului, <https://angular.io/guide/view-encapsulation>
- [11] Interacțiunea componentei, <https://angular.io/guide/component-interaction>
- [12] Stilurile unei componente, <https://angular.io/guide/component-styles>
- [13] <https://angular.io/guide/inputs-outputs>
- [14] Interpolare Angular, <https://angular.io/guide/interpolation>
- [15] <https://angular.io/guide/event-binding>
- [16] Directive încorporate Angular, <https://angular.io/guide/built-in-directives>
- [17] Directive de atribut Angular, <https://angular.io/guide/attribute-directives>
- [18] Directive structurale Angular, <https://angular.io/guide/structural-directives>
- [19] Injectarea dependințelor, <https://angular.io/guide/dependency-injection>
- [20] Formulare reactive, <https://angular.io/guide/reactive-forms>
- [21] Clientul http , <https://angular.io/guide/http>

- [22] Internaționalizare Angular, <https://angular.io/guide/i18n>
- [23] Pipe-uri Angular, <https://angular.io/guide/pipes>
- [24] L. L. E. W. Alexis GoldStein, *HTML5 & CSS3 For The Real World*, SitePoint Pty. Ltd., 2011.
- [25] Eric Meyer, *CSS: The Definitive Guide, 3<sup>rd</sup> Edition*, O'Reilly, 2006
- [26] A. Troelsen, *Pro C# 5.0 And The .NET 4.5 Framework*, 2012.
- [27] P. J. Andrew Troelsen, *Pro C# 7 with .NET and .NET Core*, Apress, 2017
- [28] N. C. T. A. Sufyan bin Uzayr, *JavaScript Frameworks for Modern Web Development*, Apress, Berkeley, CA, 2019.
- [29] A. Q. Haviv, *MEAN Web Development*, Packt Publishing Ltd, 2016.
- [30] R. Gunasundaram, *Learning Angular for .NET Developers*, Packt Publishing Ltd, 2017.
- [31] D. G. P. B. S. V. Kyle Banker, *MongoDB in Action: Covers MongoDB version 3.0*, Simon and Schuster, 2016.
- [32] K. Chodorow, *MongoDB: The Definitive Guide: Powerful and Scalable Data Storage*, O'Reilly Media, Inc., 2013.
- [33] H. L. Thuan Thai, *.NET Framework Essentials*, O'Reilly Media, Inc., 2003.
- [34] "SignalR" [Online]. Available: <https://docs.microsoft.com/en-us/aspnet/signalr/overview/getting-started/introduction-to-signalr>.
- [35] DevExpress, "DevExtreme" [Online]. Available: [https://js.devexpress.com/Documentation/17\\_2/Guide/Common/Introduction\\_to\\_DevExtreme/](https://js.devexpress.com/Documentation/17_2/Guide/Common/Introduction_to_DevExtreme/).