**UNIVERSITATEA DIN BUCUREȘTI**



**FACULTATEA**

**DE**

**MATEMATICĂ ȘI INFORMATICĂ**

**SPECIALIZAREA INFORMATICA**

**Lucrare de licență**

**Aplicație Web pentru căutarea destinațiilor de vacanță destinată grupurilor de turiști**

**Absolvent**

**RADU DENIS FLORIAN**

**Coordonator ştiinţific**

**IONESCU RADU**

**București, februarie 2021**

**Rezumat**

Aplicația web pentru căutarea destinațiilor de vacanță destinată grupurilor de turiști este o aplicație ce are ca obiectiv formarea si sugerarea celor mai potrivite destinații turistice pe care o persoană sau un grup de persoane le poate avea.

Ideea în sine a plecat de la situațiile reale pe care le au oamenii în momentele în care își doresc sa planifice o excursie sau o călătorie, dar intervin factori precum bugetul alocat, clima dorită, țara aleasă, s.a.m.d. Realizarea unor sugestii pentru dorințele unei singure persoane este relativ ușor, aceasta este stăpână pe modul în care își structureaza interesele și nevoile în ceea ce privește locația pentru care optează, dar problema adevărată intervine în momentul în care un grup de persoane cu interese diferite apar și acestea nu reușesc să cadă de comun acord pe o destinație anume, cauzele putând fi multiple și diversificate. În situația acesta, aplicația realizează pe baza unui algoritm, filtrarea intereselor fiecarei persoane si caută cele mai apropiate rezultate de alegerile fiecăruia per total.

Situațiile speciale în care niciun rezultat nu a fost găsit pentru un grup, algoritmul va reanaliza filtrele si va extinde aria de căutare, avand posibilitatea sa intervină cu sugestii ce pot ieși din aria de interes pe care persoanele din grup le-au sugerat, făcând posibilă propunerea unor destinații pe care aceștia nu le-au luat în vedere, dar care prezintă anumite criterii de selecție per individ și care pot fi incluse in rezultatul final ca opțiuni secundare.

În paralel, persoanele unui grup au posibilitatea de a comunica prin intermediul unei mesagerii, iar fiecare conversație va fi disponibilă si diferită pentru fiecare grup în parte, dand posibilitatea unei planificări mai ușoare si avantajul de a putea actualiza lista de interese pentru una sau mai multe persoane.

Fiecare persoană dispune de o pagină principală care la un anumit interval de timp va fi actualizată și vor fi prezentate multiple recomandări de destinații turistice, obținute pe baza setărilor de interese generale pe care aceasta le-a specificat.

Un sistem de notificări permite utilizatorilor să fie înștiințați cu privire la noi invitații în grupuri, excluderea lor de către un administrator dintr-un grup si noi cereri de prietenie de la alți utilizatori.

Aplicația este adaptată pentru toate dispozitivele si este fiabilă pentru orice browser, iar tehnologiile utilizate sunt actualizate si compatibile cu versiunea limbajului pe care ruleaza, atat pe partea de front-end, cat si pe cea de back-end.

**Summary**

The web application built for searching holiday destinations for groups of tourists is an application that aims to form and suggest the most suitable tourist destinations a person or a group of people can have.

The idea itself has been based on the real situations people have when they want to plan a trip or a trip, but factors such as the budget allocation, the desired climate, the chosen country, and so on. Making suggestions for the wishes of a single person is relatively easy, it has ownership of how it structures its interests and needs in terms of the location of its choice, but the real problem arises when a group of persons with different interests emerge, and they fail to agree on a particular destination, the causes can be multiple and diverse. In this situation, the application performs on the basis of an algorithm, filtering the interests of each individual and seeking the closest results to each individual's choices in total.

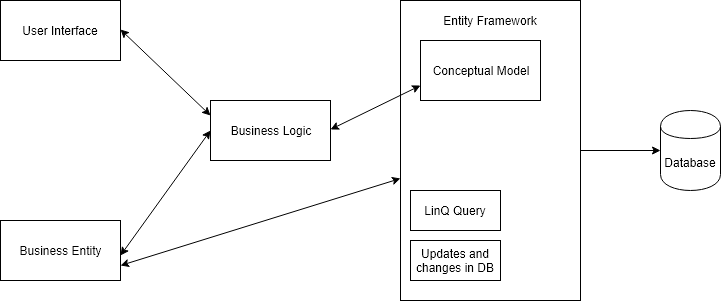
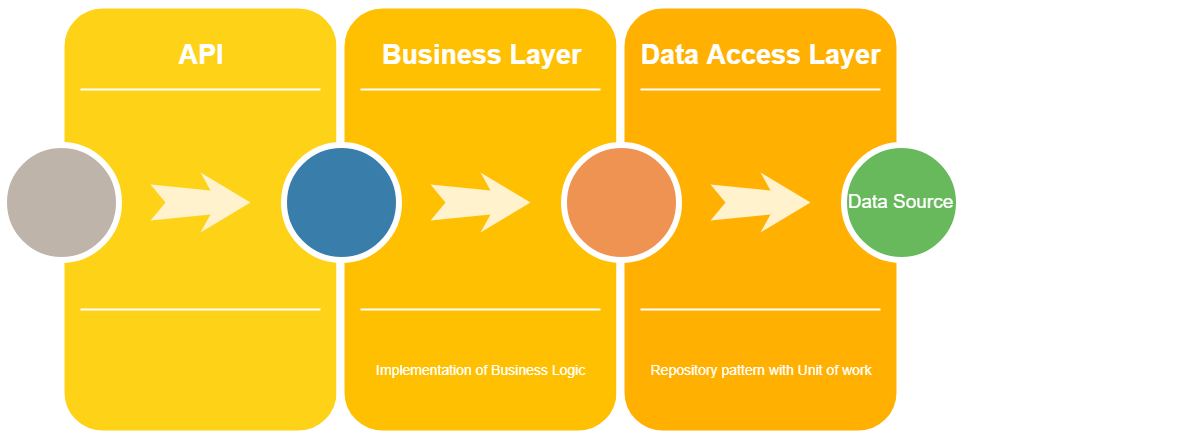
Special situations where no results have been found for a group, the algorithm will re-analyze the filters and extend the search area, with the possibility of intervening with suggestions that may leave the area of interest suggested by the people in the group, making it possible to propose destinations which they have not considered but which have certain selection criteria per individual and which can be included in the final result as secondary options.

In parallel, people in a group have the opportunity to communicate via a messaging, and each conversation will be available and different for each group, allowing easier planning and the advantage of being able to update the list of interests for one or more people.

Each person has a home page that will be updated at a certain time frame and multiple tourist destination recommendations will be presented based on the general interest settings it has specified.

A notification system allows users to be notified of new invitations in groups, their exclusion by an administrator from a group, and new friend requests from other users.

The application is tailored for all devices and is reliable for any browser, and the technologies used are up-to-date and compatible with the version of the language on which it is running, both front-end and back-end.

1. Introducere
   1. Tipul lucrării și subdomeniul specific în care se încadrează tema  
      Lucrarea este o aplicație web, structurată pe cele două părți importante ale programării, partea de front-end și partea de back-end, astfel încât construcția acesteia să fie încadrată tiparului general de aplicație, cel putin cand ne referim la arhitectura oricărei aplicații sau oricărui site web.   
      Decizia de a realiza o aplicație web a pornit în urma perioadei de stagiu într-o companie românească, la care am descoperit demersurile și necesitățiile principale pentru realizarea unei astfel de aplicații. În plus, pe perioada stagiaturii, am înțeles care pot fi diferențele între cele 2 părți ale unei aplicații și cum pot fi dezvoltate astfel încât sa fie păstrate convențiile stabilite la nivel global pentru acestea.  
        
      Dorința de a pune bazele unei astfel de aplicații a fost accentuată si mai tare la următorul loc de muncă, la care am descoperit cu adevărat ce înseamnă o aplicație web și de la care am „furat” idei si cunoștiințe, oferindu-mi toate resursele necesare pentru a construi din toate punctele de vedere aplicația ce urmează a fi prezentată. Aici am învățat cum creez un proiect, cum îl configurez, cum îl structurez și cu ce ustensile și framework-uri pot sa lucrez pentru a obține o aplicație web cât de cât profesionist realizată.  
      Idea pe baza căreia am început dezvoltarea a venit dintr-o necesitate generală pe care majoritatea oamenilor o manifestă și care vine sub forma unui mic asistent în momentul în care suntem indeciși, avem dubii sau poate chiar nu știm care este varianta favorabilă pentru ce ne dorim in momentul acela.  
        
      Tema aleasă este incadrată turismului, combinat cu social-media, deoarece putem spune că ideea aplicației se bazează pe rezolvări și răspunsuri ce se încadrează turismului, dar oferind un mediu prietenos de socializare între utilizatori.
   2. Prezentarea generală, scopul și motivația alegerii temei  
      Aplicația web pentru căutarea destinațiilor de vacanță adresată grupurilor de turiști, este o aplicație bazată pe nevoia utilizatorilor și dorințelor lor, astfel încât acestora să li se pună la dispoziție un număr mare de opțiuni pentru a alege cât mai ușor o destinație turistică.   
      Idea de aplicație pentru generarea unor destinațîi turistice a pornit de la nevoia comună pe care o au oamenii atunci când doresc să calatorească sau să petreacă o vacanță în anumite locații pe care fie nu le-au vizitat vreodată, fie despre care nu au auzit până în acel moment, dar care au o șansă destul de mare să fie pe placul acestora.   
        
      Cu ce vine totuși diferit această aplicație față de altele care sunt deja cunoscute și existente pe piață de mult timp? Ei bine, aplicația această spre deosebire de aplicațiile existente pentru generarea unor sugestii bazate pe interesele și opțiunile alese de utilizatori, vine în plus cu un mecanism prin care le este oferită posibilitatea utilizatorilor să creeze grupuri prin care să poată invită alți oameni să participe la generarea unor destinațîi turistice prin selectarea unor interese și opțiuni dorite. Mecanismul vine în ajutorul familiilor, grupurilor de prieteni, excursiilor școlare, taberelor și multor altor adunări de cel puțîn 2 oameni în care există o posibilitate foarte mare că cel puțîn o persoană să aibă interese diferite și să își dorească vizitarea altor atracții turistice, dar totuși să se poată ajunge la un consens între membrii grupurilor pentru alegerea celor mai favorabile destinații turistice încât să ii poată mulțumi pe aproape toți participanții.   
        
      Pe lângă mecanismul principal în jurul căruia fost construită ideea acestei aplicații, mai există și opțiunea de a creea un set de destinațîi turistice pentru o singură persoană, dar asupra căruia nu vor putea fi adăugate și alte persoane pentru modelare. De asemenea, pe pagină principala vor există mereu destinații turistice sugerate automat de către server pentru utilizatori, în funcție de setările pe care aceștia le vor face în pagină de administrare a contului.  
        
      Fiecare grup dispune de o pagină modală de administrare a grupului, prin care administratorii desemnați vor putea invită alți utilizatori să se alăture grupului și vor putea să dea afară utilizatori nedoriți, iar oricare membru al grupului va avea posibilitatea de a resetă lista de interese aleasă și de a părăși grupul.  
      Un lucru foarte important fără de care am considerat că nu poate există un grup pe această aplicație, a fost un chat pe care l-am alocat fiecărui grup în parte, făcând posibilă o comunicare deschisă între membri, să poată discuta cât mai ușor despre destinațiile și interesele dorite și alese.   
        
      Pentru a putea gestiona invitațiile primite, a fi înștiințați de grupurile din care au fost dați afară și cererile de prietenie primite, utilizatorilor le este pus la dispoziție un sistem de notificări, care este actualizat de fiecare dată când aceștia reîncarcă orice pagină de pe aplicație.  
        
      În cazul în care utilizatorii doresc să revadă atât grupurile cât și sugestiile generate din acestea, sau drumețiile pe cont propriu, fiecare dispune de o pagină în care sunt memorate toate acestea și sunt accesibile pentru a fi vizitate. Totuși, în cazul în care un utilizator nu mai face parte dintr-un grup sau a abandonat crearea drumeției pe cont propriu, acesta nu va mai putea accesa paginile acestora, ci doar vor fi înștiințați că au făcut parte din ele.   
        
      Înainte de a putea utiliza aplicația, utilizatorii vor fi nevoiți să își creeze un cont pe baza căruia vor fi stocate toate interesele acestora.
   3. Contribuția proprie in realizarea lucrării  
      Realizarea lucrării a fost făcută pe cont propriu, astfel toate apoartele asupra arhitecturii, modul de organizare a structurii, separarea codului în multiple fișiere, decizia legată de limbajul folosit pentru partea de server, cât și framework-urile aferente, limbajul pentru front-end cât și librăriile utilizate, au fost alese bazându-mă pe cunoștiintele pe care le dețineam până la momentul începerii acestui proiect.   
        
      Ulterior proiectul a suferit modificări majore, de la structura gândită inițial, până la regândirea interfeței pe partea de client. Cel mai semnificativ lucru a fost implementarea unor design pattern-uri pentru modelarea proiectului într-o manieră eficientă și ușor de modificat sau alterat ulterior.
   4. Structura lucrării  
      În momentul începerii aplicației, această a fost construită pe baza a două proiecte C# și un proiect backup în care era ținută structura bazei de date și alterată cu fiecare modificare adusă acesteia.   
      Proiectul principal reprezenta serverul aplicației și totodată partea de client, fiind un serviciu web bazat pe conexiuni REST API și construit folosindu-mă de modelul arhitectural „Model-View-Controller” pentru a permite apelarea requesturilor HTTP de către client către partea de server. Framework-ul principal pentru dezvoltarea serviciului web a fost .NET Core 2.2, asupra căruia au fost setate diferite configurări pentru cuplarea corectă cu partea de client. De asemenea, pentru partea de front-end, am utilizat framework-ul AngularJs împreună cu Javascript pur pentru funcționalitate, dar împreună cu framework-ul C# Răzor pentru a putea folosi diferite metode ale serverului în interiorul interfeței.   
        
      Acest proiect a fost secționat pe mai multe foldere precum Controllers, Models, Views, AppViews, AppStyle, App, Images, Scripts, Workers acestea fiind asociate următoarelor lucruri:  
                  - Controllers -> folderul principal unde se află toate fișierele în care se regăsesc clasele intermediare dintre client și server, acestea oferind prin diferite metode accesul către server prin requesturi HTTP.  
                  - Models -> aici se regăsesc toate entitățile folosite că intermediari pentru preluarea unor date ce vin în requesturi către server sau utilizate că răspuns pentru metode pentru a trimite obiectul json copie al entității către partea de client.  
                  - Views -> reprezintă zona principala unde se regăsesc fișierele Răzor cu extensia CSHTML, ele fiind la baza de fapt fișiere HTML, dar care permit utilizarea sintaxelor C# în acestea. Acest folder este obligatoriu să existe, deoarece framework-ul .NET Core împreună cu arhitectură MVC, asociază toate fișierele controller din folderul Controllers cu view-urile aferente din acest folder, folosind convenția de nume în care folderul are numele controllerului, iar fișierul CSHTML are denumirea index.  
                  - AppViews -> în acest folder există toate fișierele HTML create pentru interfață aplicației, secționate în funcție de zonele de care aparțîn. Aceste fisirele sunt utilizate în interiorul unui fișier global CSHTML care reprezintă părintele aplicației la nivel de interfață, astfel aplicația se încadrează categoriei de o site cu o singură pagină, deoarece orice trecere către alt pagină, de fapt reprezintă alt html ce înlocuiește cel curent fără a apela un nou DNS request.  
                  - AppStyle -> folderul în care stilizarile css ale paginilor html se regăsesc, secționate pentru a putea fi diferențiate în funcție de nevoie.  
                  - App -> reprezintă folderul vital pentru funcționarea corectă a interfeței, deoarece aici se află toate fișierele javascript cu angularjs, fișiere ce au rolul de controllere, request-uri către server, fișiere ajutătoare ce conțîn metode reutilizabile, directive și fișiere de constante. În cazul în care un fișier html nu ar avea asociat un fișier controller js, nu ar avea nicio funcționalitate și ar ffi un fișier static.  
                  - Images -> așa cum spune și numele folderului, acesta conține imagini utilizate pe interfață. Acestea au fost salvate local, deoarece erau criticale, iar pierderea lor ar fi stricat interfață, dar există și imagini ce sunt încărcate din mediul online, acestea fiind suplimentare și fără să fie nevoie de un backup.  
                  - Scripts -> folder în care există librării și framework-uri externe precum cel de angularjs, bootstrap pentru css, librării adiționale pentru angularjs sau javascript, evitând astfel legături către librării din mediul online în cazul în care ar există o problema de conexiune. De asemenea, sunt utilizate și librării externe, dar care sunt suplimentare și nu pot cauza o discrepanța.  
                  - Workers -> toate funcționalitatea serverului, mai precis metodele și variabilele claselor ce oferă funcționalitatea serverului, erau păstrate în acest folder, dar nu orice funcționalitate, ci doar cea în care erau folosite metodele principale ce se vor regăsi în proiectul ce urmează să fie prezentat mai jos.  
        
      Proiectul secundar poartă denumirea de DataLayer, zona unde sunt executate toate comenzile principale către baza de date, cât și query-uri mai complexe, dar toate având că scop interacțiunea cu baza de date, precum metode de înserare, ștergere, modificare și retragere a datelor din baza de date. Pe lângă aceste funcționalități, tot în acest proiect se regăseau toate interfețele, modelele și clasele utilizate în proiectul principal.  
      Specificat și anterior, aplicația folosește o baza de date gratuită de la Microsoft, mai precis SQL Server, instalată pe calculatorul local. Pentru a putea funcționa aplicația, această verifică la runtime dacă conexiunea către baza de date poate fi realizată conform unui text ce conține conexiunea către sql server.  
        
      Pe parcursul modelării aplicației, această a suferit schimbări majore, printre care și reorganizarea proiectelor, astfel că proiectul de DataLayer a fost împărțit în 3 proiecte sub denumirile Domain, DataLayer, Helpers. Asemenea proiectului principal i-au fost aduse schimbări ce au rezultat în separarea de proiectul principal a functionalitatilor din folderul Workers ce au fost duse într-un nou proiect denumit BusinessLogic. Formarea acestei separări a venit de-a lungul procesului meu de învățare la compania la care am lucrat pe parcursul dezvoltării acestei aplicații, unde am înțeles nevoia unei mai bune și eficiente organizări, astfel am aplicat design pattern-ul ce poartă denumirea de “Repository Pattern”. Acest pattern presupune consolidarea următoarei arhitecturi:  
                  - Crearea proiectului Domain în care vor fi ținute toate modelele/entitățile utilizate pentru maparea datelor din sau spre baza de date, temporare pentru diferite prelucrări sau diferite clase în care sunt ținute constante sau enum-uri.  
                  - Crearea proiectului Helpers în care vor fi ținute toate clasele ce conțîn metode ajutătoare, mai precis metode și clase statice sau diferite constante ce vor fi refolosite fie în metodele ajutătoare, fie în restul aplicației.  
                  - Păstrarea proiectului DataLayer în care rămân doar interfețele și clasele cu metodele primare ce accesează baza de date pentru modelarea tabelelor sau preluarea datelor din ele.  
                  - Crearea proiectului BusinessLogic ce conține doar logică de prelucrare a datelor venite prin controller, utilizând toate ustensilele puse la dispoziție de proiectele Domain, Helpers și DataLayer.  
        
      Folosind pattern-ul menționat, permite aplicației să fie scalabila și poată fi create și alte servicii web precum proiectul principal, care la rândul lor să poată utiliza clasele, metodele și variabilele din proiectele ajutătoare.  
        
        
        
        
      Pentru conexiunea către baza de date, am ales să utilizez framework-ul “Entity Framework” care mi-a fost recomandat de ceilalți colegi de la locul de muncă, dar pe care îl cunoșteam la rândul meu și știam exact cum trebuie implementat. Avantajele acestui framework este simplitatea sintaxei și posibilitățile oferite pentru prelucrarea datelor, folosindu-mă de metode predefinite în acest framework ce utilizau în spate query-uri optimizate către SQL Server. Pe lângă Entity Framework, am fost nevoit să utilizez și librăria de .ADO NET, o librărie ce permite crearea unei conexiuni directe cu baza de date și interogarea acesteia prin query-uri de SQL create manual. Motivul principal pentru care am ales să adaug și această modalitate, a fost imposibilitatea de a lucra simultan cu două conexiuni de entity framework pornite în paralel, motiv pentru care această rezultă mereu într-o excepție aruncată.  
        
      Ulterior, proiectul a migrat de la versiunea .NET Core 2.2 către .NET Core 3.1, deoarece această venea cu îmbunătățiri și solutionari la diferite probleme ale framework-ului. Odată cu această modificare, au apărut și modificări la nivelul configurarii făcute în proiectul principal, care au forțat și schimbarea anumitor parametrii ale unor metode din fișierele controller existente.
   5. Repere istorice și rezultate cunoscute  
      În trecut oamenii planificau excursiile în funcție de locațiile de care auziseră sau văzuseră în poze, sau chiar în funcție de zonele pe care deja le vizitaseră. Rareori se întâmplă că cineva să meargă într-o locație fără să știe un minim de detalii despre această, astfel multe atracții turistice ale pământului rămâneau nevizitate. Evoluția tehnologiei a început să acopere aceste lipsuri și să creeze noi aplicații web sau disponibile pentru orice tip de dispozitiv, că oameni să se poată ajută de ele pentru a fi informați despre zone turistice existente, în care puteau să vizualizeze poze cu peisaje, să descopere tradițiile, climatul, modul de transport și multe alte repere turistice legate de locația aleasă. Aplicații Web precum Booking, Trivago și multe altele, au pornit acest concept prin care expun utilizatorilor diferite locații și oferă informații referitoare la condițiile și beneficiile pe care le vor avea turiștii.   
        
      Deși la nivel de turist sunt prezentate diferite recomandări, personal nu am descoperit o aplicație web care să prezinte recomandări pentru grupuri de oameni, care să își pună problema dacă aceștia au individual recomandări diferite și vor totuși să obțînă sugestii care să fie plăcute pentru toate taberele. Lipsa unui astfel de mecanism limitează foarte mult grupurile de oameni, mai ales în perioada sărbătorilor în care site-urile de traveling au un trafic imens pentru că familii și grupuri de prieteni doresc să plece în diferite locații, desigur aceștia având cel mai probabil interese diferite.  
        
      Aplicația această acoperă lipsul menționat, dar vine și ea cu lipsuri ce sunt acoperite de celelalte aplicații menționate, precum lipsa unor cazări și rezervarea acestora, achiziționarea unor modalități de transport către acele locații și numărul limitat de atracții turistice menționate, deoarece capacitatea de date stocată este limitată la posibilitatea personală de a găși și adaugă referințe turistice, în comparație cu aplicațiile celelalte care dispun de un grup de oameni ce completează aceste informații.
2. Tehnologii
   1. Tehnologii Back-End  
      Alegerea tehnologiilor pentru partea de back-end a aplicatiei a fost relativ intuitiva, deoarece majoritatea dintre acestea imi erau familiare la momentul inceperii proiectului, avand experienta precedenta cu ele la locul de munca. Desi le cunosteam si stiam la ce sunt utile, provocarea a venit in momentul in care a trebuit sa le integrez, sa le configurez si sa le utilizez cat mai eficient. Pentru realizarea aplicatiei, am decis sa folosesc IDE-ul Visual Studio Community 2017, ulterior migrand la Visual Studio Community 2019, fiind cel mai recomandat pentru dezvoltarea aplicatiilor web in C# in conditiile in care acesta era si este in continuare gratuit, destinat pentru studenti si proiecte personale.  
      Serverul este creierul unei aplicatii, iar acesta trebuie sa fie cat mai bine organizat si sa se foloseasca de cat mai multe lucruri benefice pentru a obtine o performanta cat mai buna, evitarea a cator mai multe exceptii si formarea unei scalabilitati eficiente pentru marirea numarului de functionalitati si diminuarea greselilor de buna practica pentru limbajul C#.  
        
      Asa cum am mentionat si anterior, serverul este bazat pe limbajul C#, iar framework-ul principal pentru a fi posibila crearea unui server web, este .NET Core. Prima versiune pe care am abordat-o si utila la momentul acela, a fost versiunea 2.2, deoarece era ultima disponibila si aparuta, dar si compatibila cu toate framework-urile utilizate, si desigur cea mai potrivita din punct de vedere al optimizarilor aduse framework-ului si rezolvarilor de erori pe care le avusese. Desi existau alte alternative precum framework-ul .NET Standard 4.5, acesta nu era gratuit si open-source pentru a fi accesibil codul oricarui dezvoltator, facand limitat accesul spre imbunatatiri si idei ce ar fi putut crea din acesta un framework cautat si usor de utilizat. In plus, comunitatea persoanelor ce utilizau .NET Standard era destul de restransa in comparatie cu cea pentru .NET Core, astfel solutiile si parerile referitoare la acesta erau putine si de multe ori, fara un raspuns la anumite probleme.  
      Pentru a putea compila solutia pentru partea de back-end, au fost instalate fisierele SDK si Runtime ce apartineau de .NET Core si aferente versiunii utilizate. Fisierul SDK este utilizat pentru a avea acces la librarii si ustensile oferite de framework-ul .NET Core pentru a putea realiza aplicatia sau cel putin partea de back-end si conexiunea catre partea de front-end, dar in special permite compilarea solutiilor. Fisierul Runtime este utilizat pentru a permite rularea aplicatiilor in urma compilariilor facute, fara de care nu ar putea fi accesibile spre utilizare.  
        
      Datorita framework-ului .NET Core, am utilizat librarii precum Microsoft.AspNetCore.Http si Microsoft.AspNetCore.Http.Abstractions pentru a putea crea sau primi request-uri HTTP, fie prin intermediul conexiunii cu partea de client, fie prin alte modalitati precum programul Postman, pentru depanarea problemelor si/sau a codului implementat. Programul Postman este extrem de utilizat in randul programatorilor, utiliatatea sa primara fiind creearea si interograrea oricarui tip de request HTTP catre un server anume pe baza unor multiple aspecte, precum IP-ul sau DNS-ul calculatorului pe care se afla serverul, port-ul asociat acestuia, protocolul utilizat pentru request (GET,POST,PUT,DELETE,UPDATE) si multe alte configurari de acces atribuite in mod normal unui request HTTP. Astfel, cu ajutorul acestui program am putut vizualiza daca serverul receptiona apelurile facute si returna mesajul dorit sau daca exista vreo problema ce ar fi putut periclita buna functionare, dar in plus ofera o alternativa in cazul in care nu exista partea de front-end sau nu este posibil accesul catre aceasta, deoarece simuleaza un request ce ar putea fi trimis de catre client catre server.   
      Pe langa cele mentionate, cele doua librarii ofera posibilitatea realizarii unor clase middleware, ce mai poarta numele de “omul in mijloc” sau “clasa intermediara”, aceasta avand scop interceptarii request-urilor inainte ca acestea sa ajunga la controllerele existente si la functionalitatea serverului, in mod normal pentru a putea fi validate si/sau prelucrate.  
        
      Un alt framework foarte important pe care l-am utilizat este Entity Framework si este un cadru de mapare obiect-relațional open source pentru ADO.NET, fara de care nu ar fi fost posibila aproape toata conexiunea si interogarea bazei de date. Acesta face permite crearea conexiunii catre baza de date in functie de un string de conexiune. Pentru recunoasterea tabelelor, framework-ul recunoaste automat maparea catre acestea in functie de o conventie de nume utilizata in cod pe care o atribuim unor seturi de obiecte ce se leaga fiecare de cate un tabel. In general pentru interograrea unei baze de date sunt folosite scripturi sau query-uri SQL, dar entity framework foloseste query-uri predefinite ce nu sunt accesibile pentru modificare, dar optimizate si reutilizabile pentru multiple tipuri de interogari. Query-urile vin sub forma unor metode c#, facand mai simpla implementarea acestora fara a stii in mod direct limbajul SQL pentru a putea intelege cum sunt prelucrate datele intr-o baza de date relationala. Astfel framework-ul este recunoscut pentru simplitatea sintaxei si usurintei pentru configurarea acestuia. In ultimul rand, framework-ul ofera o ustensila ce are ca scop crearea unor fisiere de migrare pentru crearea automata a bazei de date si a tabelelor, in functie de clasele create si mapate, reducand timpul si necesitatea mentenantei bazei de date, dar si portabilitatea acesteia pe baza unui back-up.   
        
      Cu toate acestea, personal nu am utilizat aceasta ustensila, ci am folosit o ustensila a IDE-ului prin care baza de date a fost replicata in interiorul folderului proiectului, dar pe care am updatat-o treptat folosindu-ma de aceeasi unealta. Folosind aceasta varianta de replicare si back-up, pot vizualiza codul sursa pentru generarea intregii baze de date si alterarea acesteia fara a impacta codul serverului, deoarece uneori maparile nu se schimba, dar apar mici modificari insesizabile asupra bazei de date, iar aceasta modalitate permite modificarea bidirectional.  
        
        
        
      O tehnologie care completeaza un pattern pe care l-am implementat ulterior si datorita careia am redus semnificativ utilizarea memoriei, este framework-ul Ninject, pe care l-am folosit impreuna cu pattern-ul Composition Root pentru a implementa o clasa reutilizabila si in acelasi timp singletone, utilizata pe baza depedency injection, astfel incat aceasta leaga orice implementare a claselor “worker” sau “repository”, fara a aloca mereu o noua adresa de memorie datorita instantierii continue atunci cand aveam nevoie de utilizarea unor metode din cadrul acestora. Astfel folosind acest framework, am legat orice interfata de clasa in care era implementata si am facut posibila injectarea automata a implementarii in diferite workere, prin intermediul constructorului worker-ului specific. Utilizand acest pattern, se va aloca de la lansarea proiectului in memorie spatiu pentru clasa Composition Root, impreuna cu implementarile instantiate, pe care o vom injecta mereu prin request-uri catre controllerele aferente ce o vor folosi, urmand ca in interiorul acestora sa utilizez implementarile mentionate doar in cazul in care e nevoie de acestea si fara a aloca spatiu suplimentar in memorie.   
      Acest pattern este recunoscut in cadrul multor companii si utilizat foarte mult in proiecte, deoarece ofera o buna organizare si o usoara utilizare a logicii din spatele oricarui server.  
        
      Anterior am prezentat tehnologia folosita pentru a conecta serverul la baza de date si a putea prelua, insera si altera seturi de date din aceasta, aceasta avand numele Entity Framework. In continuare vorbesc despre SQL Server, baza de date aleasa pentru stocarea informatiilor aplicatiei. Alegerea a venit ca urmare a utilizarii acesteia si pe parcursul anilor universitari pentru diferite proiecte scolare, cat si pentru proiecte personale, ca ulterior sa fie aprofundata la stagiul dintre anii II si III, si mai apoi in locurile de munca ce au mai urmat. MS SQL Server este o baza de date relationaal oferita de Microsoft, ce vine atat in varianta gratuita, cat si contra cost, si care poate fi accesata atat prin intermediul unor comenzi in terminalul sistemului de operare, dar si prin aplicatia pusa la dispozitie de catre acestia ce poarta denumirea de Sql Server Management Studio. Aplicatia ofera o interfata vizuala pentru prelucrarea bazei de date si configurarea sa oferind o usoara interactiune cu aceasta. Personal am lucrat si cu alte baze de date relationale, precum MySQL, PostgreSQL, Oracle, cat si nerelationale cum ar fi MongoDB sau ElasticSearch, dar aceasta ofera avantajul unei conexiuni mult mai usoare cu orice aplicatie ce este realizata pe baza framework-ului .NET Core. De asemenea, IDE-ul Visual Studio ofera o ustensila mentionata anterior pentru crearea locala a unui back-up pentru structura bazei de date, sau pentru migrarea sa. Din acest motiv si adaugand experienta anterioara, am decis ca ma voi folosi de ea pentru realizarea intregii aplicatii, optand pentru varianta gratuita, deoarece oferea majoritatea optiunilor pe care le detinea si varianta contra cost.  
        
      Pe langa toate tehnologiile mentionate, am mai incercat integrarea si a altor framework-uri, care ulterior s-au dovedit a fi ineficiente pentru tipul acesta de aplicatie si care nu ar fi adus un plus de performanta sau de scalabilitate.
   2. Tehnologii Front-End  
      Tehnologiile pentru partea de client au reprezentat o provocare, deoarece ma consider o persoana ce indrageste si stapaneste mai mult partea de server si logica a unui proiect, astfel realizarea interfetei a fost un impediment initial si a reprezentat un timp de gandire si research, ca intr-un final sa descopar framework-ul AngularJs pentru Javascript. Initial acesta a fost destul de greu de inteles, sintaxa si conceptele sale nu erau prietenoase cu o persoana incepatoare, dar avantajul a fost usurinta integrarii, documentatiile si solutiile propuse de comunitatea ce dezvolta in limbajul C# partea de server si la randul lor au putut realiza aplicatii web folosindu-se de AngularJs. Astfel o perioada de timp a constat in acomodarea cu acest framework, incercarea de a-l intelege si a gandi o arhitectura pentru a fi cat mai scalabila si eficienta.   
      Folosind acest framework, nevoia de a folosi cod Javascript pur a disparut, dar fundamentele acestuia au ramas si la nivelul acestui framework, deoarece sintaxa si algoritmica ce stateau la baza framework-ului erau aceleasi ca la Javascript. Pot spune ca Javascript nu imi era necunoscut, deoarece l-am utilizat inca din perioada liceului si chiar si pe perioada facultatii la anumite materii pentru realizarea proiectelor sau la laboratoarele acestora. Adaugand suplimentar, o persoana ce nu ar fi cunoscut fundamentele Javascript-ului nu ar fi avut o intelegere usoara pentru acest framework si cel mai probabil ar fi fost aproape imposibil de integrat, deoarece asa cum spuneam si anterior, conceptele propuse de AngularJs sunt destul de complexe si necesita o intelegere mai buna.  
      AngularJS spre deosebire de utilizarea propriu-zis a limbajului Javascript pur, ofera dinamicitate interfetei si permite o conexiune cu paginile HTML extrem de rapida. In plus, posibilitatea creeri unor validari si alterari ale interfetei in timp real fac ca acesta sa fie o alegere buna pentru acest tip de aplicatie web.   
        
      Interfata paginilor web a fost realizata folosind HTML si CSS, avand un impuls mai mare datorita framework-ului Bootstrap, pe care l-am utilizat pentru prima data in primul an de facultate pentru realizarea unui proiect personal si pe care ulterior am ajuns sa il refolosesc la orice alta aplicatie web pe care am dezvoltat-o sau o dezvolt si in prezent, deoarece acesta ofera o multitudine de stilizari predefinite in CSS pentru a crea o interfata receptiva si cu un aspect cat mai placut. Cu toate acestea, stilizarile oferite de Boostrap nu erau suficiente si am fost nevoit sa recurg la implementari aditionale in CSS pentru crearea unor stilizari sau alterarea celor oferite de framework.   
        
      Pe parcursul proiectului au aparut foarte multe fisiere ce contineau cod de AngularJs sau cod CSS, facand imposibila depanarea problemelor direct din consola pentru dezvoltatori din toate browse-rele, astfel am folosit o ustensila in Visual Studio ce compila pe baza unui fisier de configurare toate fisierele de AngularJS sau JS selectate, intr-un singur fisier principal, iar pentru CSS aceeasi procedura, doar ca fisierele de CSS trebuiau separate in mai multe fisiere pentru a nu suprapune denumiri identice de stilizari.  
      Folosind aceasta metoda minimalista, calitatea depanarii pentru diferite probleme intampinate in timpul dezvoltarii a fost sporita si a redus din timpul alocat.
3. Perspectiva utilizatorului  
   Interactiunea pusa la dispozitie utilizatorilor este una cat mai simpla, astfel optiunile pe care acestia trebuie sa le selecteze sau sa le parcurga sunt relativ la indemana lor, sau pot fi gasite printr-o cautare intr-un interval de timp foarte scurt.   
   Pentru a putea utiliza aplicatia, utilizatorii sunt obligati sa aiba un cont de utilzator. Acesta va fi automat redirectionat catre pagina de logare, in care trecand doar emailul si parola vor fi validati si in cazul in care credentialele sunt regasite in baza de date, acesta va fi trimis catre pagina principala, altfel va primi un mesaj de eroare. Pentru a crea un cont, utilizatorul are un mesaj ancora imediat sub formularul de logare, prin care este incarcata pagina de inregistrare in locul celei de logare, la care va urma procedeul de completare a tuturor campurilor cerute si dupa inregistrarea cu succes, acesta va fi trimis catre pagina de logare din nou.  
     
   Trecand de autentificare, utilizatorii vor observa pe pagina principala initial ca nu exista nimic la rubrica de sugestii bazate pe interesele lor, ci doar in rubrica de sugestii ce ar putea sa fie de interes sau nu, mai precis aleatorii. Motivul pentru care prima rubrica este goala vine din lipsa unor interese setate in pagina de interese generale, ce urmeaza sa fie prezentata mai jos. Imediat ce un utilizator va avea cel putin un interes setat, instant pagina principala va avea cel putin o sugestie pe care sa o poata prezenta in acea rubrica. Aspectul acestor sugestii este facut sub forma unui panou in care este prezentata o poza aleator aleasa de pe internet pentru a infatisa cat mai mult cu putinta orasul promovat. Sub poza acestuia se vor regasi tara si numele orasul la care se face referire, iar in dreapta acestora vor fi transpuse prin ilustratii, text sau iconite, toate atractiile turistice stocate in baza de date referitor la orasul sugerat, cat si informatii despre climatul ce poate fi intalnit si transportul de care dispun turistii doar pentru acea zona. Interfata pentru cele 2 rubrici este bazata pe acelasi aspect, diferenta este facuta de dimensiunile alocate in interiorul paginii principale, astfel rubrica secundara pentru sugestii aleatorii este mai restransa in partea dreapta. Motivul pentru care rubrica principala ocupa o mare parte din interfata acestei pagini se datoreaza unor cercetari facute in care erau explicate de ce site-uri mari ale lumii au informatia principala centrata pe ecranul utilizatorilor si extremele paginilor sunt inexistente. Pe scurt, psihologic s-a demonstrat ca oamenii acorda o atentie mai mare informatiilor primite vizual daca acestia nu sunt nevoiti sa “forteze” privirea pentru a capta informatia din locuri ce nu sunt direct intuitive pentru a fi accesate de utilizatori, precum coltul stang de jos, margina stanga a oricarei pagini si posibil multe alte zone stiute sau nestiute. Concluzionand si motivul pentru care rubrica secundara este restransa is pozitionata in extrema dreapta, e pentru a oferi o alternativa utilizatorilor, nefiind de interes principal pentru cautarea acestora, aceasta vine ca un sprijin si ramane ca reper secundar. Inainte de a continua cu urmatorul aspect al aplicatiei, mentionez ca fiecare rubrica este realizata sub forma unui depanator pentru multiple sugestii, acesta fiind automat ca la cinci secunde sa treaca la urmatoarea sugestie, iar la final sa fie reluata lista de sugestii, cu posibilitatea de a reveni sau a trece mai departe la oricare sugestie pe baza unor sageti puse in coltul drept al rubricii.  
     
   Trecand mai departe, fiecare pagina disponibila, dispune de un antet ce contine denumirea aplicatiei web si un meniu de navigare si informare. Denumirea aplicatiei pe langa informarea utilizatorilor in legatura cu ce site au accesat, mai ofera posibilitatea redirectionarii catre pagina principala printr-un click pe aceasta.   
   Meniul de navigatie vine in ajutor cu patru optiuni puse la vedere pentru utilizatori. Prima optiune poarta numele de “Parties” si apasand pe ea va fi deschis un meniu drop-down in care se vor regasi toate grupurile pentru drumetii create fie de utilizatorul ce a accesat meniul, fie in care acesta a fost invitat si a acceptat. Fiecare grup are alaturat un contor pentru numarul de persoane existente si active in grup. Apasand pe unu din grupuri, utilizatorul va fi redirectionat catre pagina grupului. Alaturat numelui optiunii, exista un contor pentru a stii in cate grupuri utilizatorul se afla, grupuri ce inca exista, deoarece grupurile sterse nu vor mai fi disponibile.   
   Trecand la a doua optiune, aceasta se numeste “Create your Trip” prin care utilizatorii pot crea grupuri. Denumirea de grupuri, drumetii, excursii sau orice alt cuvant ce poate fi asociat acestui reper, este valid si va fi reutilizat in continuare pentru a nu propune ideea de grup ca fiind principala, deoarece si numele optiunii face referirea la o drumetie, dar toate acestea impreuna dau intelesul ideii principale pe baza caruia a fost creata aplicatia web. Revenind, accesand optiunea, utilizatorul va fi redirectionat pe pagina in care va alege ce fel de drumetie urmeaza sa creeze, una pe cont propriu sau una de grup. Pentru prima varianta, crearea drumetiei este simpla si necesita doar un nume pentru cum se va numi aceasta, iar pentru cealalta varianta, va fi afisat pe langa campul desinat numelui drumetiei, si un camp pentru adaugarea utilizatorilor invitati pe baza emailului acestora, cat si un tabel in care vor aparea utilizatorii invitati, din care vor putea fi stersi in cazul in care utilizatorul se razgandeste. La final, este accesat butonul de crearea si daca validarile aferente trec, iar drumetia este creata, atunci utilizatorul va fi redirectionat pe pagina nou creata grupului.   
   Cea de-a treia optiune nu are o denumire vizibila, dar este realizata sub o forma generala pe care o intalnim pe majoritatea platformelor, si anume notificarile ce sunt reprezentate printr-o iconita avand un clopotel si un contor in dreapta acesteia pentru numarul de notificari la care nu a fost dat un raspuns sau nu au fost vizualizate. Apasand clopotelul, un meniu dropdown va fi extins in cazul in care exista cel putin o notificare, si aici vom regasi trei tipuri de notificari. Una dintre notificari este cea pentru invitatii in grupuri, la care utilizatorii au informatie despre denumirea grupului si persoana care i-a invitat, si desigur 2 optiuni de acceptare sau respingere a invitatiei. O alta notificare este cea in care utilizatorul va fi instiintat daca a fost dat afara dintr-un grup de care apartinea, de catre un administrator. Iar cea de-a treia notificare este legata de cererile de prietenie primite de la alti utlizatori, deoarece fiecare persoana are dreptul de a alege daca doreste sa faca parte din lista de contacte a altei persoane si sa aiba acces rapid catre aceasta. Notificarile vor fi actualizate atunci cand utilizatorul schimba sau reincarca aceeasi pagina.  
   Ultima optiune reprezinta si optiunea de gestiune pentru contul utilizatorului, regasita sub denumirea de “Account” care in momentul accesari, va afisa un dropdown cu multiple optiuni. Prima optiune reprezinta pagina de interese generale pe care utilizatorul le va configura astfel incat prima pagina sa aduca sugestii in rubrica principala discutata anterior, bazate pe interesele selectate aici. Pagina de interese permite utilizatorului sa selecteze dintr-o varietate de alegeri, tarile si orasele de interes, climatele favorite, modalitatile de transport preferate si atractiile turistice ce doresc a fi vizitate sau ca punct de reper. Modalitatea de a alege aceste interese, se face pe baza unei pagini modale lansate in momentul accesarii fiecareia dintre optiunile prezentate, acestea afisand multiple dropdown-uri cu date disponibile spre alegere.   
   Urmatoarea optiune poarta de numirea de “Friends” si precum spune si numele, utilizatorul va fi redirectionat catre o pagina de gestiune a persoanelor adaugate in lista de contacte. Acestea ulterior vor putea fi utilizate in momentul in care utilizatorul invita persoane la o anumita drumetie, pe baza unui dropdown special generat pentru a-i putea gasi mai usor.  
   Optiunea a treia afiseaza o pagina ce afiseaza toate drumetiile create vreodata, atat cele vechi si sterse, cat si cele existente inactive sau active, folosind un tabel in care sunt afisate si avand posibilitatea ca utilizatorul sa paraseasca grupuri sau sa stearga o anumita drumetie in cazul in care acesta va fi singurul ramas in acea drumetie. De asemenea, fiecare drumetie afisata poate fi accesata din acest tabel si utilizatorul va fi redirectionat catre aceasta printr-un click pe numele acesteia. Nu in ultimul rand, utilizatorul poate modifica denumirea drumetiilor din aceasta pagina daca detine gradul de administrator in acea drumetie.  
   Penultima optiune este destul de simpla si permite modificarea datelor de logare ale contului, precum email-ul si parola.  
   A cincea optiune si ultima este cea care deconecteaza utilizatorul si il redirectioneaza catre pagina de logare.  
     
   Trecand la urmatorul aspect, mai precis paginile drumetiilor, acestea pun la indemana in prima parte a paginii rubrica de sugestii ce urmeaza a fi afisate pe baza intereselor selectate de fiecare membru al grupului. Prezentarea sugestiilor de locatii este asemanatoare celor din prima pagina a site-ului, astfel aspectul va fi similar, iar parcurgerea acestora va fi posibila sub forma unui depanator de imagini. Sugestiile vor fi actualizate atunci cand utilizatorul reincarca pagina, fie cand modifica sau acceseaza rubrica de modificat interese.   
   Pentru a putea selecta interesele dorite, o rubrica asemanatoare cu cea din pagina de setari generale pentru interese, se va regasi imediat sub rubrica de prezentat sugestiile, iar pentru a putea discuta si gestiona utilizatorii grupului, in partea dreapta rubricii de interese se va regasi un chat in care pot comunica utilizatorii, iar atasat de acesta in partea dreapta superioara vor putea accesa o iconita ce deschide un meniu drop-down pentru a gestiona utilizatorii grupului, cat si posibilitatea de a reseta interesele selectate sau a parasi grupul si drumetia.  
     
   Legand toate aceste aspecte, aplicatia ofera un aspect cat mai simplist si usor de navigat, facand experienta utilizatorului placuta si simpla.   
   Culorile alese pentru crearea interfetei au fost selectate in urma unor cercetari pentru a intelege gandirea unei persoane ce navigheaza pe site si ce l-ar putea convinge sa ramana in continuare pentru a utiliza aplicatia. In urma acestor cautari, s-a constatat faptul ca majoritatea persoanelor navigheaza pe site-uri ce au culori deschise si foarte putin proeminente, fara a deranja ochiul uman si fara a forta utilizatorul sa acorde o atentie mai mare pentru a intelege lucrurile din interfata. Initial dorinta personala a fost de a folosi o tema intunecata, deoarece temele care folosesc culoarea neagra predominant sunt considerate elegante, dar numarul utilizatorilor care opteaza pentru aceasta perspectiva este mai mic, motiv pentru care nevoia de a selecta o varietate de culori pentru a dezvolta o interfata deschisa la culoare, a fost mai mult o nevoie pentru a satisface nevoia majoritatii utilizatorilor, scopul principal fiind acela de a atrage utilizatori cat mai multi.
4. Perspectiva dezvoltatorului  
   Cand am ales sa incep dezvoltarea acestei aplicatii, mi-am propus sa ma folosesc atat de ce stiu mai bine, cat si de ce stiu mai putin. Prin urmare, decizia de a realiza partea de server a aplicatiei a venit in urma cunostintelor si experientei acumulate folosind limbajul C# impreuna cu framework-ul principal .NET Core, dandu-mi un avantaj pentru realizarea si gandirea arhitecturii astfel incat sa pot obtine o solutie scalabila, alaturi de care sa vin in completare cu diferite design pattern-uri invatate sau descoperite in timpul dezvoltarii pentru a obtine o aplicatie cat mai bine structurata si optima. Pe de alta parte, dorinta personala de a invata tehnologii noi si aplicarea acestora in proiectul ales, m-a impins catre framework-ul AngularJS pentru partea de interfata sau de client. Nu pot nega faptul ca baza de date utilizata reprezenta un punct forte cand am decis sa folosesc SQL Server din suita Microsoft, deoarece pana la momentul inceperii proiectului am utilizat acest tip de baza de date destul de putin in comparatie cu capacitatea totala pe care aceasta o punea la dispoizitie.  
   Punand cap la cap aceste tehnologii principale, dezvoltarea aplicatiei a inceput initial prin proiectarea arhitecturii solutiei realizate cu ajutorul Visual Studio Community, fiind pe tot parcursul dezvoltarii principalul IDE utilizat. Arhitectura initial a fost structurata din 3 proiecte, si anume un proiect principal in care exista toata logica de bussines, serverul propriu zis si fisierele interfetei, un proiect ce continea metodele de baza pentru operatiuni cu baza de date, cat si modelele mapate acestor operatiuni, cat si celor din logica de bussines, iar ultimul proiect reprezenta zona de backup pentru structura bazei de date. In prezent, asa cum am putut mentiona si mai sus, proiectul a suferit modificari la nivel de organizare si logica de functionare, motiv pentru care acesta s-a impartit in multiple proiecte in urma aplicarii pattern-ului numit “Repository Pattern”. Acest patern creaza un mediu favorabil scalarii proiectului pentru permiterea realizarii unor aplicatii pe alte tipuri de dispozitive, folosindu-ma de aceleasi proiecte ce vin sub forma unor librarii pentru serverul aplicatiei web.   
     
     
   Pe langa acest pattern, am venit in ajutor cu un nou pattern numit “Composition Root Pattern” descoperit in momentul in care am inceput sa invat despre ce inseamna si cum poate fi utilizata injectarea dependintelor. Injectarea dependintelor este o tehnica in care un obiect primeste alte obiecte de care depinde. Aceste alte obiecte sunt numite dependente. Practic acest pattern reduce numarul instantelor create pentru clasele utilizate, facand transferul unei singure instante necesare prin requestul HTTP ce vine in momentul accesarii API-ului asociat unui controller. Injectarea poate fi de 3 tipuri si anume, injectarea tranzitoriu in care fiecare instanta va fi diferita la orice request si orice controller accesat, injectarea cu scop in care instanta este diferita la orice request, dar aceeasi in orice controller ar fi folosita pe parcursul aceluiasi request, si injectarea singleton sau globala cum ii mai spun personal, in care instanta va fi una singura pe tot parcursul rularii serviciului ce detine si instantiata acea instanta, astfel orice request va refolosi aceeasi instanta a unui obiect.  
   Folosind pattern-ul “Composition Root”, am creat o depdendinta injectata singleton pentru clasa composition root, deoarece aceasta trebuia sa fie mereu aceeasi pe tot parcursul utilizarii serviciului. In schimb, metodele si clasele injectate folosind aceasta clasa intermediar, erau puse sub forma injectarii tranzitoriu. Initial nu exista injectare tranzitorie la nivelul acestora, ci era una cu scop, dar motivul principal pentru trecerea aceasta a fost depistarea unei probleme ce bloca reutilizarea unui obiect pe care il foloseau simultan 2 clase diferite, fapt pentru care rezulta intr-o exceptie aruncata aproximativ la cateva secunde dupa pornirea serverului.   
   Un ultim pattern care a imbunatatit calitatea codului si a obligat carearea unor functionalitati de baza repetitive pentru multiple clase, se numeste “Strategy Pattern”, prin care fiecare clasa repository ce executa operatiuni catre baza de date, fara indoiala va executa cel putin un query de inserare a unor date catre o tabela, o modificare a unuia sau a mai multor campuri unei tabele, cat si preluarea sau stergerea unor randuri din aceste tabele. Aceste query-uri de multe ori sunt aceleasi, cat si funcionalitatea implementata in codul C#, dar difera tabela pe care se executa comanda, motiv pentru care fiecare dintre aceste clase repository, sunt obligate sa implementeze metodele de baza pe care le vom regasi in fiecare dintre acestea. Astfel, viitoarele clase ce ar putea fi create, vor avea asigurata o functionalitate obligatorie prin care va acoperi cazurile cele mai frecvent utilizate.  
   Trecand mai departe, realizarea acestor metode de baza pentru apelarea unor query-uri la nivelul bazei de date, a fost facuta in proportie de 90% cel putin, cu ajutorul framework-ului Entity Framework, fiind unul dintre cele mai raspandite si utilizate la nivel global in proiecte ale companiilor de renume, cat si in proiecte personale. Entity Framework abreviat si EF, este un cadru open source de mapare relationala pe obiect (ORM) pentru ADO.NET. In spate acest framework a fost conceput utilizand un set de tehnologii ADO.NET optimizate pentru a obtine cea mai buna performanta si a fi reutilizate pentru dezvoltarea de aplicatii software orientate pe date. Pentru a pune in functiune acest framework, au fost necesari urmatorii pasi:  
    - Crearea unei clase ce mosteneste clasa “DbContext” fiind clasa principala pe care se creeaza maparea datelor din baza de date atasata conexiunii.   
    - Setarea si configurarea acesteia sa mapeze pe baza unor DbSet-uri, toate tabelele pe care dorim sa le extragem ulterior.   
    - Crearea modelelor utilizate in cod pentru a putea fi mapate catre tabelele asociate, astfel fiecare properitate sa reprezinte un camp anume din tabela.   
    - Definirea instructiunilor prin care fiecare model sa fie interpretat corect de EF pentru a intelege tipul coloanei mapate sau felul in care e dorita convertirea acesteia, cat si alte limitari sau impuneri la nivel de functionalitate.  
    - Setarea conexiunii catre baza de date folosind textul de conexiune, care poate fi generat automat sau manual creat folosind formatul propus.  
   Ulterior dupa ce toate aceste configurari au fost facute, mai ramane doar crearea unei instante pentru clasa context nou generata si accesarea modelelor mapate.  
     
   Avand maparea catre tabelele utilizate, mai departe aplicatia trebuie construita in jurul acestora, astfel atat proiectul cu logica de business, cat si cel cu metodele de baza, vor fi construite pentru fiecare in parte, mai precis in foldere specifice.   
   Pentru a intelege mai bine ce reprezinta fiecare tabel, mai jos voi prezenta pe scurt fiecare in parte:  
    - Principala tabela este cea a utilizatorilor denumita “Users”, in care vor exista conturile stocate in momentul inregistrarii si mai ales cele cu care se pot loga in aplicatie. Fiecare utilizator va fi recunoscut in functie de emailul ales, deoarece invitatiile ce vor fi trimise sunt bazate pe emailurile acestora.  
    - Pentru a putea crea un set de interese, aveam nevoie de o multitudine de interese dintre care utilizatorii sa poata alege, motiv pentru care a fost necesara crearea tabelei “Interests”. Alterarea acesteia se poate face doar manual accesand baza de date, deoarece valorile stocate sunt predefinite, mai precis utilizatorii nu au acces la alterarea acestora.  
    - Cu toate ca exista tabela cu interese, utilizatorii trebuie sa aiba cum sa poata alege aceste interese si sa ramana stocate undeva, motiv pentru care apare in discutie tabela “UserInterests” in care sunt stocate fie interesele generale ale utilizatorului, fie interesele selectate pentru fiecare grup sau drumetie in parte.  
    - Trecand mai departe, utilizatorii am spus mai devreme ca pot face parte din grupuri turistice pentru planificarea unor drumetii, sau chiar pe cont-propriu, prin urmare tabela “TripUsers” va reprezenta locatia unde sunt mapati utilizatorii alaturi de drumetia de care apartin, drumetie care la randul ei va fi creata si stocata in tabela “Trips”.  
    - In grupuri, utilizatorii pot discuta pentru a planifica execursia dorita, de aceea mesajele acestora vor fi stocate pentru a putea revedea ulterior discutiile avute, astfel tabela “TripMessages” va stoca fiecare mesaj trimis de utilizatori, impreuna cu drumetia la care a fost trimis mesajul respectiv.  
    - Cu toate ca putem invita utilizatori sa faca parte din anumite grupuri, acestia trebuie sa poata vedea aceste invitatii, sau daca s-a intamplat ceva anume cu grupul si a fost desfintat sau o persoana a fost data afara, de asemenea trebuie utilizatorul instiintat, de aceea tabela “Notifications” va stoca toate notificarile pe care utilizatorii le pot primii, avand posibilitatea sa interactioneze cu anumite tipuri de notificari pentru a le accepta sau respinge.  
    - Ultima tabela si care vine in ajutorul utilizatorilor, este accea de “Friends” prin care sunt memorate conexiunile dintre utilizatori, deoarece fiecare utilizator are posibilitatea de a adauga in lista de prieteni unu sau mai multi utilizatori pe baza email-ului acestora. Tabela ofera ajutor suplimentar in momentul in care administatorii de grupuri vor sa invite diferite persoane si doresc sa economiseasca timp pentru a cauta emailul persoanei dorite.  
     
   Avand structura bazei de date inconjurul careia se desfasoara toata activitatea aplicatiei, trebuie mentionata si configurarea serverului principal, cat si legatura pe care o are interfata cu acesta.  
   Atunci cand am gandit arhitectura solutiei, aveam cunoscute doua mari concepte ale dezvoltarii de aplicatii, anume conceptul de arhitectura monoltica si conceptul de arhitectura bazata pe microservicii. Arhitectura monolitica se bazeaza pe crearea unui server ce aduna toata functionalitatea si logica aplicatiei si o gestioneaza din aceeasi locatie, pe cand arhitectura bazata pe microservicii separa fiecare logica de business in multiple mini servere le-as numi personal, fiecare fiind independenta una de cealalta. Avantajul pe care il are o arhitectura monolitica fata de cea bazata pe microservicii e usurinta de accesare a functionalitatii si modificarea acesteia, cat si conexiunea directa si rapida intre resurse. Pe de alta parte, arhitectura pe microservicii ofera o gestiune mai buna a traficului pe care il produc utilizatorii cand executa anumite request-uri, dar si o usoara alterare a anumitor servicii fara a fi nevoie sa impacteze intreaga solutie, deoarece acestea pot fi facute separat.  
   Cu toate acestea, decizia mea a fost aceea de a folosi arhitectura monolitica din urmatoarele motive:  
    - traficul utilizatorilor nu ar fi prea mare, deoarece aplicatia in sine nu necesita prezenta utilizatorilor mult timp.  
    - functionalitatea este una oarecum simpla si nu necesita accesarea multor request-uri.  
    - resursele sunt intr-o continua conexiune, motiv pentru care este necesara o accesare rapida a acestora.  
     
   Serverul principal si proiectul principal in acelasi timp, este configurat sa interpreteze fisierele interfetei sub forma statica, mai precis interfata este inca de la pornirea serverului setata sa fie ascoata aceluiasi IP si aceluiasi port, facand usoara trecerea catre si dinspre client-server. Cu alte cuvinte, partea de front-end si back-end sunt hostate pe acelasi dispozitiv, fiind intr-o dependinta stransa.   
   Fiind o aplicatie web, framework-ul .NET Core permite compilarea fisierelor statice pentru interfata sub o forma prestabilita intr-un fisier de configurare, astfel toate fisierele de javascript sau angularjs vor fi compilate sub un singur fisier general, dar si fisierele de stilizare vor fi puse sub aceeasi configuratie. La compilarea acestora, ele sunt mutate automat intr-un folder specific numit “wwwroot”, in care apare tot codul compilat sub noua forma si care va fi citit pentru interpretare in momentul lansarii aplicatiei.  
     
   Realizarea interfetei nu ar fi fost posibila, fara putin ajutor din partea framework-ului Bootstrap, care ofera o gama larga de stilizari predefinite si gata de utilizat. Acesta a facut posibila crearea unor tabele, formulare si butoane utilizate in mod frecvent pe intreaga aplicatie. De asemenea, stilizari manuale au fost necesare pentru a putea obtine aspectul dorit, motiv pentru care au fost create fisiere suplimentare de CSS.   
   Fiecare pagina pentru a putea functiona corespunzator, are asociat un controller creat in AngularJs, care creeaza si request-urile ce vor prelua sau vor trimite informatii catre server.   
     
   Un lucru important de mentionat, utilizatorii pentru a ramane conectati la aplicatie chiar si dupa parasirea site-ului, sunt stocate date esentiale precum id-ul acestora si emailul, in zona de stocare numita “localstorage” oferita de fiecare browser in parte. Acest storage, mai poate fi cunoscut si sub denumirea de “cookies” care retine informatii specifice site-urilor web pentru o usoara viitoare interactiune si evitarea traficului extern catre diferite servicii sau mai ales catre server.  
     
   Trecand la ultimul aspect al dezvoltarii, algoritmul principal pe baza caruia sunt generate sugestiile in functie de interesele alese de utilizatori este facut sub forma unei cautari extinse, mai precis algoritmul uneste toate interesele primite si cauta prima data sugestiile ce exista doar pe baza intereselor alese. In cazul in care nu exista rezultate, algoritmul extinde cautarea pe baza conexiunilor predefinite pentru interese, in sensul in care spre exemplu o locatie este asemanatoare cu o alta locatie, iar gradul de asemanare este ridicat cu interesele selectate de utilizator, atunci rezultatul este preluat si aratat utilizatorilor. Acesta extinde cautarea cat timp exista conexiuni catre care poate inainte, asemanator cu parcurgerea printr-un graf.
5. Probleme intampinate si tratarea acestora  
   Ca in orice aplicatie, problemele nu ezita sa apara. Pe parcursul intregii dezvoltari au aparut multiple probleme, de natura tehnica mai ales ce impuneau fie o refactorizare amanuntita a codului, fie regandirea logicii sau daca norocul era de partea mea, rezolvari relativ simple ce nu necesitau un efort mai mare.   
   O problema comuna a fost tratarea exceptiilor. In momentul in care aveam obiecte ce nu erau instantiate sau nu aveau referinta catre o adresa de memorie, serverul arunca exceptii. Pentru a rezolva aceasta problema, un ajutor era oferit de Visual Studio, deoarece fiecare metoda importata de la un anumit framework, specifica si ce fel de exceptie ar putea sa arunce, astfel nevoia de tratare a exceptiilor era vizibila, dar pentru alte tipuri de exceptii intampinate precum cele in care coloanele nu erau mapate corespunzator la baza de date, conversii ce nu puteau fi efectuate, s.a.m.d., ajungeau sa fie rezolvate in timpul testarilor. Cu aceasta ocazie, intelegerea blocului de cod “try-catch” a devenit mai clara si usor de implementat.   
   Pe langa problema exceptiilor, o alta problema majora intalnita a fost cea a imposibilitatii de a avea pe acelasi request catre server, doua contexte deschise simultan catre baza de date prin Entity Framework. Problema aparea in momentul in care utilizatorul intra pe site si avea nevoie sa primeasca notificarile noi, dar in acelasi timp sa primeasca si lista de sugestii. Desi in alte cazuri aceasta nu a fost o problema, deoarece contextul era automat inchis dupa prelucrarea comenzii date, in acest caz acesta ramanea inca deschis. Solutia care a venit la indemana a fost utilizarea directa a tehnologiei ADO.NET pe baza careia este si EF bazat, astfel evitam sa folosesc obiecte ale EF-ului, ci deschideam manual o conexiune catre baza de date si executam un query. Asemanator, a fost nevoie sa refactorizez si alte metode care se suprapuneau, desi metodele erau facute sa astepte una dupa cealalta. Cu aceest prilej, am putut sa imi reamintesc si sa refolosesc conceptele invatate despre baze de date si comenzile de selectare, inserare si modificare pentru a realiza comenzile ADO.NET.   
   Pe langa problemele mentionate anterior, a existat o problema organizarii proiectelor existente. Pana a pune in practica pattern-urile prezentate anterior, proiectul nu a fost organizat pentru a putea fi scalabil si nici nu era eficient. De fiecare data cand ma foloseam de un repository sau de o clasa din logica de business, cream chiar si de doua ori in aceeasi metoda doua instante pentru acelasi obiect, lucru ce impacta atat performanta, cat si memoria alocata. Astfel implementarea pattern-ului Composition Root a redus semnificativ numarul de instante create, dar a crescut si performanta codului, iar pattern-urile “Strategy” si “Repository” au ajutat solutia prin crearea unor proiecte aditionale in care a fost mutata logica pusa haotic anterior si care ulterior a devenit usor de gestionat si modificat, iar aplicatia a devenit usor de scalat.  
   Pe partea de interfata problemele au fost in mare parte la crearea si stilizarea paginilor html, deoarece alegerea variabilelor corespunzatoare in CSS este relativ dificila stiind depedendintele parinte copil existente in html si felul in care acestea pot fi alterate de multe ori fara sa ne dam seama chiar de un parinte la care nu ne asteptam al unui tag. Astfel au existat probleme de aliniere in pagina, de dimensiuni, de font, de pozitionare si multe alte probleme comune in randul dezvoltatorilor de aplicatii web pe aceasta latura. Aceste probleme au fost rezolvate luand la rand de la cel mai “varstnic” parinte, pana la copilul tag in cauza de la care pornea problema, deoarece trebuia verificat pas cu pas daca poate cineva sa impacteze problema de stilizare. Pentru a putea face asta, browserele in mod normal pun la dispozitie o consola de inspectare pentru a parcurge sau a opri codul creat si a-l verifica. Personal dezvoltarea a fost facuta folosind Google Chrome, dar pentru teste aditionale a fost folosit si Microsoft Edge.   
   Mici alte probleme au mai aparut si in momentul configurarilor pentru comunicarea dintre client si server sau de natura tehnica ce nu puteau fi intelese in mod direct, dar acestea au fost rezolvate pe baza ajutoarelor de pe diferite site-uri, in care alte persoane au intampinat aceleasi probleme sau apropiate.
6. Surse de inspiratie  
   Aplicatia web a putut fi realizata in urma cunostintelor acumulate pe parcursul anilor de studentie, cat si in cadrul firmelor in care am fost angajat in timpul acesta. Importanta unor baze de programare si cunoasterea tehnicilor si tehnologiilor necesare, au fost factori esentiali in stadiul incipient, fara de care nu ar fi putut exista o structura a proiectului, o functionalitate, sau chiar o idee care sa poata fi implementata.  
   Datorita anilor de studentie, am reusit sa aprofundez si sa aplic notiuni importante ale programarii orientate pe obiecte precum clasele, obiectele, mostenirea, incapsularea, polimorfismul, s.a.m.d, care bineinteles stau la baza oricarei aplicatii bune din acest domeniu si fara de care acestea nu ar putea evolua. De asemenea, aprofundarea structurilor de date mi-a venit ca un ajutor inca din start, mai ales in conditiile in care aplicatia utilizeaza intr-o cantiate mare liste, dati si vectori pentru prelucrarea unor informatii multiple si stocarea acestora.   
   Cu toate acestea, cunoasterea pe larg a limbajului Javascript si a modalitatii de dezvoltare a paginilor web, a fost bine pusa la punct in cadrul cursului numit Tehnici Web, datorita caruia am putut intelege sintaxa si functiile importante ale javascript-ului, dar mai ales cum poate fi legat acesta de paginile html si css, fara de care nu ar fi fost posibila realizarea interfetei.  
   Si nu in ultimul rand, meritele pentru cunostintele de baza acumulate in ceea ce privesc limbajul C# si bazele de date relationale, sunt destinate cursurilor pentru Dezvoltarea Aplicatiilor Web si Baze de Date, alaturi de care am putut invata cum poate fi folosit limbajul C# pentru crearea unor aplicatii web, cum poate fi conectata o aplicatie la o baza de date si mai ales, cum poate fi utilizata o baza de date si sintaxa pe baza careia aceasta reactioneaza.  
   Toate acestea, plus alte informatii suplimentare invatate si studiate pe parcursul facultatii, au format baza prin care am reusit sa realizez aplicatia.  
   Totusi, pentru a putea dezvolta aplicatia la un nivel scalabil si profesionist, am apelat la cunostintele acumulate ca angajat, dar folosind si tehnologii studiate pe contpropriu. Pe perioada in care am fost intern la o firma romaneasca, am inteles cum functioneaza intr-un proiect complex o baza de date, ce inseamna sa creezi un index, o cheie primara compusa, un trigger, si multe alte informatii importante pentru aceasta. Totodata, am descoperit din ce este compusa o aplicatie, cum sunt diferentiate partile de front-end si back-end, dar mai ales cum pot sa imi organizez proiectul in functie de un tool planificator. Poate una dintre cele mai importante lucruri invatate, a fost cum sa utilizez platforma de versionare GitHub in care pot stoca proiectul si pot avea istoricul acestuia, fara de care personal am concluzionat ca nu poate exista o aplicatie la un nivel profesionist, fara o platforma de acest gen pe care sa fie dezvoltat.   
   De asemenea, o alta firma in care am mai lucrat m-a invatat ce inseamna sa lucrezi cu framework-ul pe care l-am folosit si eu, anume .NET Core si mai ales cu tehnologiile asociate acestuia. In prezent pot spune ca aceasta a devenit tehnologia principala pe baza caruia imi propun sa dezvolt aplicatii si ulterior. Acest framework permite crearea aplicatiilor sau serviciilor web intr-o maniera foarte usoara, mai ales in momentul cunoasterii sale, si care ofera acces la o multitudine de tehnologii ajutatoare pentru a obtine cea mai buna varianta din aplicatia dezvoltata.   
   Desi personal am descoperit singur AngularJs pentru front-end, am avut norocul ca ulterior sa o aprofundez si la locul de munca, unde am inteles si mai bine conctepul de controller, de modul, de directiva, fara de care nu as fi putut obtine conexiunea dorita intre cele doua parti ale aplicatiei.  
     
   Aceste cunostinte au permis realizarea aplicatiei, punand in schema notiuni si practici utilizate si la nivel de angajat intr-o firma, motiv pentru care a reprezentat un prilej foarte bun de a experimenta si a utiliza si mai mult toate cele invatate.