A picture containing graphical user interface

Description automatically generatedA picture containing text

Description automatically generated

**UNIVERSITATEA DIN BUCUREȘTI**

**FACULTATEA**

**DE**

**MATEMATICĂ ȘI INFORMATICĂ**

**SPECIALIZAREA**

**Calculatoare și Tehnologia Informației**

**Lucrare de licență**

**Absolvent**

**Dima Ana-Maria**

**Coordonator științific**

**Lect.dr. Anca Dobrovăț**

**București, iulie 2022**

A picture containing graphical user interface

Description automatically generatedA picture containing text

Description automatically generated**UNIVERSITATEA DIN BUCUREȘTI**

**FACULTATEA**

**DE**

**MATEMATICĂ ȘI INFORMATICĂ**

**SPECIALIZAREA**

**Calculatoare și Tehnologia Informației**

**Drinkționar**

**Absolvent**

**Dima Ana-Maria**

**Coordonator științific**

**Lect.dr. Anca Dobrovăț**

**București, iulie 2022**

# Cuprins

[Cuprins 3](#_Toc102654919)

[I. Introducere 5](#_Toc102654920)

[I.1. Context 5](#_Toc102654921)

[I.2. Motivație 5](#_Toc102654922)

[II. Tehnologii utilizate în dezvoltarea aplicației 7](#_Toc102654923)

[II.1. Baza de date 7](#_Toc102654924)

[II.1.1. Generalități despre Microsoft SQL Server 7](#_Toc102654925)

[II.1.2. Cum funcționează Microsoft SQL Server? 7](#_Toc102654926)

[II.1.3. Proprietățile ACID 8](#_Toc102654927)

[II.2. Modelarea conceptuală a aplicației 9](#_Toc102654928)

[II.2.1. Diagrama Entitate-Relație 9](#_Toc102654929)

[II.2.2. Diagrama Conceptuală 10](#_Toc102654930)

[II.2.3. Descrierea entităților, atributelor, cheilor și a tipurilor de date folosite 10](#_Toc102654931)

[II.2.4 Descrierea cardinalităților stabilite între entitățile bazei de date 14](#_Toc102654932)

[II.2.5. Descrierea constrângerilor de integritate 15](#_Toc102654933)

[II.2.6. Schemele relaționale 17](#_Toc102654934)

[II.2.7. Crearea tabelelor și a relațiilor dintre acestea în SQL Server 17](#_Toc102654935)

[II.3. Cum funcționează o aplicație de tip web? 18](#_Toc102654936)

[II.3.1. Cererile web 18](#_Toc102654937)

[II.4. Backend-ul aplicației 20](#_Toc102654938)

[II.4.1. ASP .Net Core 20](#_Toc102654939)

[II.4.2. Modelul arhitectural Model-View-Controller (MVC) 20](#_Toc102654940)

[II.4.3.Metodologia Code-First folosind Entity Framework 22](#_Toc102654941)

[II.4.4. Repository Pattern 22](#_Toc102654942)

[II.4.5. Controlere și API-uri 23](#_Toc102654943)

[II.4.6. DTO-uri 24](#_Toc102654944)

[II.5. Mecanismul de autentificare in aplicație pe partea de server 24](#_Toc102654945)

[II.5.1. JWT Token 24](#_Toc102654946)

[II.5.2. Algoritmul de criptare folosit - HmacSha256Signature 25](#_Toc102654947)

[II.5.3. Autentificarea propiu-zisă 26](#_Toc102654948)

[II.6. Algoritmi utilizați 28](#_Toc102654949)

[II.6.1. Algoritm de generare rețetă pe baza unei liste de ingrediente 28](#_Toc102654950)

[II.6.2. Algoritm de sugerare a unei rețete bazat pe preferințele anterioare 28](#_Toc102654951)

[II.7. Frontend-ul aplicației 29](#_Toc102654952)

[II.7.1. Introducere 29](#_Toc102654953)

[II.7.2. Framework-ul Ionic 30](#_Toc102654954)

[III. Bibliografie 32](#_Toc102654955)

# Introducere

## I.1. Context

Ultimii ani au adus schimbări majore în stilul de viață al oamenilor. Odată cu declararea oficială a pandemiei de COVID-19 viața a suferit modificări pe toate planurile. De la felul în care muncim, învățăm, călătorim sau socializăm, nimic nu mai seamănă cu ceea ce înainte consideram a fi viața normală.

Mersul la munca sau la scoală unde ne întâlneam cu prietenii și colegii a devenit o serie nesfârșită de întâlniri de tip audio/video, care deși ne-au diminuat timpul pe care îl petrecem în trafic, ne-au îndepărtat simțitor de ceilalți oameni. Serile de ieșit in oraș, in cluburi sau pub-uri, împreună cu o mulțime de prieteni sau de oameni necunoscuți s-au transformat în seri monotone petrecute alături de familie sau într-un cerc forte restrâns.

Ca urmare a creșterii alarmante a îmbolnăvirilor datorate infecției cu Coronavirus atât restaurantele cat si barurile au fost închise. Astfel a apărut ideea dezvoltării aplicației *Drinkționar.*

Aceasta aplicație își propune să aducă la tine acasă o colecție bogată și variată de rețete de cocktail-uri si băuturi (atât alcoolice cat și non-alcoolice) cu o notorietate mai scăzută. Aplicația va pune la dispoziția utilizatorului o bază de date cu foarte multe rețete, însoțită de o metoda intuitiva de alegere a ingredientelor/băuturilor care vor sta la baza sugerării unor cocktail-uri adaptate gusturilor și preferințelor fiecărui om. În plus, prin intermediul opțiunii de apreciere a unei rețete, aplicația va colecta date statistice, pentru a oferi cercetătorilor din domeniu un set de date actual ce va reliefa comportamentul în materie de consum al băuturilor pe teritoriul țării noastre.

## I.2. Motivație

Deși consumul excesiv de alcool nu este recomandat de către niciun specialist, cantități moderate de alcool, consumate într-un mediu controlat, s-au dovedit a avea beneficii foarte importante asupra sănătății.

Un consum moderat de alcool, ceea ce reprezintă nu mai mult de o băutură (un pahar de vin/cocktail, o halba de bere sau 50 ml de tărie) pentru femei si doua băuturi pentru bărbați pe zi, poate aduce beneficii foarte importante organismului uman. Astfel, printre beneficiile aduse de consumul moderat de alcool se numără: protejarea inimii de atacuri de cord și îmbunătățirea funcționării acesteia prin creșterea nivelului de HDL, cunoscut și sub denumirea de colesterol bun; creșterea dorinței de a face exerciții fizice și îmbunătățirea stării de spirit; prevenirea apariției pietrelor la rinichi; creșterea fertilității și îmbunătățirea vieții sexuale; păstrarea unui nivel optim al zahărului în sânge; precum și o mai bună stimulare a activității cognitive.[[[1]](#endnote-1)]

Pentru a putea să ne bucurăm de toate beneficiile pe care le-am menționat mai sus este nevoie, pe lângă limitarea cantității consumate, să acordăm o atenție sporită calității băuturilor pe care le consumam. De multe ori, barurile în care alegem să ieșim pentru a savura o băutură gustoasă fac rabat de la calitatea ingredientelor de bază în vederea maximizării profitului, lucru ce este cu siguranță în detrimentul consumatorului. De aceea, pentru a avea un control cât mai bun asupra a ceea ce consumam, citirea cu atenție a ingredientelor ce sunt listate pe eticheta produsului, urmată de achiziționarea celor ce corespund unui standard înalt de calitate, reprezintă elemente foarte importante.

Astfel, Drinkționar vine în sprijinul utilizatorului/consumatorului, oferindu-i oportunitatea de a se bucura de o gamă variată de băuturi, pe care, de regulă, doar un barman cu experiență le-ar putea cunoaște, în confortul casei sale, având un control absolut asupra cantităților și mai ales a calității produselor pe care le consumă.

# Tehnologii utilizate în dezvoltarea aplicației

## II.1. Baza de date

### II.1.1. Generalități despre Microsoft SQL Server

Pentru a stoca datele necesare funcționării optime a aplicației am ales sa utilizez un sistem de management al bazelor de date relaționale *Microsoft SQL(Structured Query Language) Server.*

Microsoft SQL Server este una dintre tehnologiile de baze de date, ce este lider pe piață, fiind construita pe baza SQL. SQL este un limbaj de programare standardizat și consacrat folosit pentru construirea, gestionarea și interogarea bazelor de date și a conținutului acestora.

Codul inițial SQL Server a fost dezvoltat în anii 1980 de compania ce atunci purta denumire Sybase Inc., deținută acum de SAP. Inițial, tehnologia se adresa sistemelor Unix și platformelor de minicalculatoare, însă, în scurt timp, Microsoft și Ashton-Tate Corp. (principalul furnizor de baze de date de la acel moment pentru calculatoare) s-au unit și au lansat în 1989 prima versiune de SQL Server concepută pentru sistemul de operare OS/2.[[[2]](#endnote-2)]

### II.1.2. Cum funcționează Microsoft SQL Server?

Principiul de bază al funcționării acestui sistem este reprezentat de stocarea datelor într-un mod cât mai eficient, astfel încât sa nu fie nevoie să existe aceleași date de mai multe ori, evitându-se problema redundanței.

Pe baza unei structuri tabelare se realizează asocierea unor date din mai multe tabele, utilizând anumite relații bine definite.

La baza SQL Server stau două componente principale:[[[3]](#endnote-3)]

1. Motorul bazei de date (Database Engine):

Acesta reprezintă componenta de bază a serverului, fiind la rândul sau alcătuit din:

1. Motor relațional, care mai este cunoscut și sub numele de procesor de interogări. Acesta determină cea mai bună modalitate de a executa o interogare.
2. Motorul de stocare, care este responsabil cu stocarea și prelucrarea datelor.

Pentru început, motorul de interogare solicită date de la motorul de stocare. După primirea unui răspuns, acesta se ocupa de procesarea acestuia, cât și de managementul firelor de execuție, a sarcinilor și a buffer-ului.

1. SQLOS

SQLOS reprezintă sistemul de operare din spatele SQL Server. Acesta oferă multe funcționalități precum: managementul memoriei și I/O, gestionarea excepțiilor și sincronizarea tranzacțiilor.

Totodată, acest model relațional aderă la proprietățile ACID, pentru a garanta că tranzacțiile din baza de date au loc într-un mod fiabil.

### II.1.3. Proprietățile ACID

ACID reprezintă un acronim de la Atomicitate, Consecvență, Izolare, Durabilitate.[[[4]](#endnote-4)]

* *Atomicitatea* face referire la felul în care o tranzacție (o singura operație logică care constă din unul sau mai mulți pași) se încheie: fie aceasta a reușit în întregime, fie a eșuat în întregime. Daca doar o parte din tranzacție prezintă erori și eșuează, întreaga tranzacție va fi considerată ca fiind eșuată. Acest principiu este unul de tipul totul sau nimic.
* *Consecvența* asigură validitatea tuturor datelor în conformitate cu regulile definite, constrângerile, declanșatoarele, sau operațiile de tip cascadă ce au fost aplicate.
* *Izolarea* asigură faptul ca fiecare tranzacție se va efectua independent, nefiind afectată de nicio altă tranzacție.
* *Durabilitatea* asigură faptul că orice modificare va fi stocată permanent, chiar dacă imediat după finalizarea acesteia în sistem a apărut o blocare.

Aceste principii, explicate anterior, sunt respectate de SQL Server pentru a se asigura că datele nu se corup, chiar dacă în sistem apar diverse erori.

## II.2. Modelarea conceptuală a aplicației

### II.2.1. Diagrama Entitate-Relație

Datele sunt elemente cu care ne întâlnim la fiecare pas, sub diverse forme. Însă pentru ca acestea să fie relevante, ușor de gestionat și interpretat este necesar să acordam o buna atenție felului în care acestea sunt structurate și abstractizate.

În trecut, înaintea apariției mediilor de stocare electronice, procesarea, culegerea și stocarea datelor se realiza manual, pe hârtie. Acest proces era unul foarte greoi și nesecurizat, deoarece nu se putea asigura o confidențialitate a datelor. Ulterior, odată cu evoluția tehnologiei și creșterea volumului de date a fost necesară dezvoltarea unor modalități fiabile, mult mai flexibile de manipulare a datelor. Astfel, a apărut conceptul de bază de date.[[5]](#endnote-5)

O bază de date reprezintă modelarea, într-un mod abstract, dar totuși relevant, a unui univers. Acest lucru se realizează prin folosirea unor colecții de date ce formează obiecte, cunoscute și sub denumirea de entități.

Pentru a reliefa cat mai corect și complet modul în care entitățile ce alcătuiesc structura bazei de date a acestei aplicații a fost necesară construirea unei diagrame Entitate-Relație (Figura 1.1.).

Diagram

Description automatically generated

Figura 1.1. Diagrama Entitate-Relație a modelului; model pentru reprezentarea sistemului din lumea reală.

### II.2.2. Diagrama Conceptuală

Diagrama conceptuală este utilizata pentru a furniza majoritatea informațiilor necesare implementării, construindu-se cu ajutorul ei o reprezentare la nivel înalt de abstractizare.

Astfel, a fost construita diagrama conceptuală de mai jos (Figura 2.2.) pentru a furniza, sumarizat și modularizat toate elementele constituente ale modelului.

Graphical user interface, application, Word

Description automatically generated

Figura 2.2. Diagrama conceptuală a modelului. Are rost implementare tabela de roluri?

### II.2.3. Descrierea entităților, atributelor, cheilor și a tipurilor de date folosite

Entitatile si atributele utilizate în modelarea acestei baze de date sunt:

*Categorii\_Ingrediente*: În acest tabel sunt stocate informații cu privire la categoria/tipul din care face parte un ingredient folosit într-o rețetă. Exemple de intrări în acest tabel: siropuri, lichior, răcoritoare, vin, bere, băuturi spirtuase.

Atributele acestei entități sunt:

* # id\_categorie\_ingredient (guid)- cheie primară, cod unic cu ajutorul căruia sunt identificate tipurile de ingrediente;
* nume\_categorie\_ingredient (varchar)- numele categoriei;
* descriere\_categorie\_ingredient (varchar)- eventuale explicații sau exemple de produse ce fac parte din acea categorie.
* poza\_categorie\_ingredient (varchar)- calea catre locul in care este salvată poza

*SubCategorii\_Ingrediente*: În acest tabel sunt stocate informații cu privire la subcategoria din care face parte un ingredient folosit într-o rețetă. Exemple de intrări în acest tabel: lichior din fructe, bere blondă, rom.

Atributele acestei entități sunt:

* # id\_subcategorie\_ingredient (guid)- cheie primară, cod unic cu ajutorul căruia sunt identificate subcategoriile din care fac parte ingrediente;
* \*id\_categorie\_ingredient (guid)- cheie străină pentru realizarea legăturii cu tabelul Categorii\_Ingrediente;
* nume\_subcategorie\_ingredient (varchar)- numele subcategoriei;
* descriere\_subcategorie ingredient (varchar)- eventuale explicații sau exemple de produse ce fac parte din acea subcategorie.
* poza\_subcategorie\_ingredient (varchar)- calea către locul în care este salvată poza

*Ingrediente*: În acest tabel sunt stocate informații cu privire la ingredientele utilizate în cadrul unei rețete. Exemple de intrări în acest tabel: lichior de capșuni, lămaie, sirop de soc.

Atributele acestei entități sunt:

* # id\_ingredient (guid)- cheie primară, cod unic cu ajutorul căruia sunt identificate ingredientele;
* \*id\_subcategorie\_ingredient (guid)- cheie străina pentru realizarea legăturii cu tabelul SubCategorii\_Ingrediente;
* nume \_ingredient (varchar).

*Unități*: În acest tabel sunt stocate informații cu privire la unitătile de masura ce sunt utilizate pentru a ,,cântări” ingredientele necesare unei rețete. Exemple de intrări în acest tabel: grame, linguri, mililitri.

Atributele acestei entități sunt:

* # id\_unitate(guid)- cheie primară, cod unic cu ajutorul căruia sunt identificate tipurile de unități de măsura
* nume\_ unitate(varchar).

*Categorii\_Rețete*: În acest tabel sunt stocate informații cu privire la categoriile în care se încadreaza o băutură/rețetă. Exemple de intrări în acest tabel: cocktail, shot, shake, cafea.

Atributele acestei entități sunt:

* # id\_categorie\_rețetă (guid)- cheie primară, cod unic cu ajutorul căruia sunt identificate categoriile din care fac parte anumite rețete;
* nume\_categorie\_rețetă (varchar).

*Tipuri\_Rețete*: În acest tabel sunt stocate informații cu privire la tipul din care face parte o rețetă. Exemple de intrări în acest tabel: alcoolic, non-alcoolic, alcool opțional.

Atributele acestei entități sunt:

* # id\_tip\_rețetă (guid)- cheie primară, cod unic cu ajutorul căruia sunt identificate tipurile de rețete;
* nume\_tip\_rețetă (varchar).

*Pahare*: În acest tabel sunt stocate informații cu privire la tipul de pahar ce este recomandat pentru a fi utilizat într-o anumită rețetă. Exemple de intrări în acest tabel: halbă, pahar cu picior, pahar înalt.

Atributele acestei entități sunt:

* # id\_pahar (guid)- cheie primară, cod unic cu ajutorul căruia sunt identificate tipurile de pahare;
* nume\_pahar (varchar).

*Utilizatori*: În acest tabel sunt stocate informații cu privire utilizatorii aplicației.

Atributele acestei entități sunt:

* # id\_utilizator (guid)- cheie primară, cod unic cu ajutorul căruia sunt identificați utilizatorii;
* nume\_utilizator (varchar);
* prenume\_utilizator (varchar);
* username (varchar);
* email (varchar);
* parola (varchar);
* moment\_creare (timestamp);
* rol (varchar);
* activ (boolean);
* poza\_utilizator (varchar)- calea catre locul in care este salvata poza;
* sex (char);
* vârstă (int).

*Aprecieri*: Acest tabel este unul de legătură, ce explicitează relația de ,,mulți-la-mulți” dintre Utilizatori și Rețete. În acest tabel sunt stocate informații cu privire la interacțiunea utilizatorilor cu rețetele din aplicație.

Atributele acestei entități sunt:

* # id\_apreciere (guid)- cheie primară, cod unic cu ajutorul căruia sunt identificate înregistrările aprecierilor/interacțiunilor;
* \*id\_rețetă (guid)- cheie străină pentru realizarea legăturii cu tabelul Rețetă;
* \*id\_utilizator (guid)- cheie străină pentru realizarea legăturii cu tabelul Utilizatori;
* moment\_apreciere (timestamp);
* scor\_apreciere (float) – se oferă o nota cuprinsă între 1 și 5.

*Rețete*: În acest tabel sunt stocate informații cu privire la rețetele propriu-zise de băuturi.. Exemple de intrări în acest tabel: Gin Tonic, Cuba Libre.

Atributele acestei entități sunt:

* # id\_rețetă (guid)- cheie primară, cod unic cu ajutorul căruia sunt identificate rețetele;
* nume\_ rețetă (varchar).
* \*id\_categorierețetă (guid)- cheie străină pentru realizarea legăturii cu tabelul Categorii\_Rețete;
* \*id\_tip\_rețetă (guid)- cheie străină pentru realizarea legăturii cu tabelul Tipuri\_Rețete;
* \*id\_pahar (guid)- cheie străină pentru realizarea legăturii cu tabelul Pahare;
* nume\_rețetă (varchar);
* descriere\_rețetă (varchar);
* poza\_rețetă (varchar);
* instrucțiuni\_rețetă (varchar);
* scor\_rețetă (float).

*Retetă\_Ingrediente*: Acest tabel este unul de legătură, ce explicitează relatia de ,,mulți-la-mulți” dintre Rețete și Ingrediente. În acest tabel sunt stocate informații cu privire la asociarea ingredientelor cu rețetele. Practic, pentru fiecare ingredient dintr-o rețetă se va face o nouă înregistrare în acest tabel.

Atributele acestei entități sunt:

* # id\_rețetă\_ingredient (guid)- cheie primară, cod unic cu ajutorul căruia sunt identificate intrarile în acest tabel;
* \*id\_rețetă (guid)- cheie străină pentru realizarea legăturii cu tabelul Rețetă;
* \*id\_ingredient (guid)- cheie străină pentru realizarea legăturii cu tabelul Ingredient;
* \*id\_unitate (guid)- cheie străină pentru realizarea legăturii cu tabelul Unitate;
* cantitate\_ingredient (float).

### II.2.4 Descrierea cardinalităților stabilite între entitățile bazei de date

1. Categorii\_Rețete - Rețete

* O rețeta aparține unei singure categorii de rețete.
* O categorie de rețete conține una sau mai multe rețete.

1. Tipuri\_Rețete - Rețete

* O rețeta aparține unui singur tip de rețete.
* Un tip de rețete conține una sau mai multe rețete.

1. Pahare - Rețete

* O rețetă folosește cel mult un pahar.
* Un pahar este folosit în una sau mai multe rețete.

1. Utilizatori – Rețete

* O rețetă este apreciată de zero sau mai mulți utilizatori.
* Un utilizator apreciază zero sau mai multe rețete.

1. Categorii\_Ingrediente – SubCategorii\_Ingrediente

* O subcategorie de ingrediente aparține unei singure categorii de ingrediente.
* O categorie de ingrediente conține una sau mai multe subcategorii de ingrediente.

1. SubCategorii\_Ingrediente - Ingrediente

* Un ingredient aparține unei singure subcategorii de ingrediente
* O subcategorie conține unul sau mai multe ingrediente.

1. Ingrediente -Rețete

* O rețetă conține unul sau mai multe ingrediente.
* Un ingredient face parte din una sau mai multe rețete.

1. Rețete - Unități

* O unitate de măsură este utilizată în una sau mai multe rețete.
* O rețetă utilizează maximum o unitate de măsură per ingredient.

### II.2.5. Descrierea constrângerilor de integritate

1. Chei primare:

* Categorii\_Ingrediente: id\_categorie\_ingredient;
* SubCategorii\_Ingrediente: id\_subcategorii\_ingredient;
* Ingrediente: id\_ingredient;
* Unitați: id\_unitate;
* Rețete\_Ingrediente: id\_rețetă\_ingredient;
* Utilizatori: id\_utilizator;
* Aprecieri: id\_apreciere;
* Rețete: id\_rețetă;
* Categorii\_Rețete: id\_categorie\_rețetă;
* Tipuri\_Rețete: id\_tip\_rețetă;
* Pahare: id\_pahar.

1. Constrangere de unicitate (Unique):

* Categorii\_Ingrediente: nume\_categorie\_ingredient;
* SubCategorii\_Ingrediente: nume\_subcategorii\_ingredient;
* Ingrediente: nume\_ingredient;
* Unitați: nume\_unitate;
* Rețete\_Ingrediente: id\_rețetă+ id\_ingredient;
* Utilizatori: nume\_utilizator, email;
* Aprecieri: id\_rețetă+id\_urilizator;
* Rețete: nume \_rețetă;
* Categorii\_Rețete: nume\_categorie\_rețetă;
* Tipuri\_Rețete: nume\_tip\_rețetă;
* Pahare: nume\_pahar.

1. Atribute ce nu au voie să fie null (Not NULL):

* Categorii\_Ingrediente: nume\_categorie\_ingredient;
* SubCategorii\_Ingrediente: nume\_subcategorii\_ingredient;
* Ingrediente: nume\_ingredient;
* Unități: nume\_unitate;
* Rețete\_Ingrediente: id\_rețetă, id\_ingredient;
* Utilizatori: nume\_utilizator, email, nume, prenume, parolă, vârsta, sex;
* Aprecieri: id\_rețetă, id\_urilizator, scor\_apreciere;
* Rețete: nume \_rețetă;
* Categorii\_Rețete: nume\_categorie\_rețetă;
* Tipuri\_Rețete: nume\_tip\_rețetă;
* Pahare: nume\_pahar.

1. Cheie externă/străină:

* Categoriii\_Ingrediente – SubCategorii\_Ingrediente: cheia externă dintre cele două entități este id\_categorie;
* SubCategorii\_Ingrediente – Ingrediente: cheia externa dintre cele două entități este id\_ingredient;
* Ingrediente-Rețetă\_Ingrediente: cheia externa dintre cele două entități este id\_ingredient;
* Rețetă\_Ingrediente - Unități : cheia externa dintre cele două entități este id\_unitate;
* Rețetă\_Ingrediente - Rețete: cheia externa dintre cele două entități este id\_ rețetă;
* Categorii\_ Rețete - Rețete: cheia externa dintre cele două entități este id\_ categorie\_rețetă;
* Tipuri\_ Rețete - Rețete: cheia externa dintre cele două entități este id\_tip\_rețetă;
* Pahare - Rețete: cheia externa dintre cele două entități este id\_pahar;
* Rețete - Aprecieri: cheia externa dintre cele două entități este id\_rețetă;
* Aprecieri - Utilizatori : cheia externa dintre cele două entități este id\_utilizator.

1. Constrângeri de validare (check constrains)

* Utilizatori: vârstă >= 18.

### II.2.6. Schemele relaționale

Pentru a evita păstrarea unor date incomplete, ce nu mai au însemnătate ca urmare a unor ștergeri din alte tabele cu care erau într-o relație de asociere a fost necesară implementarea unor proceduri ce au efect asupra mai multor tabele.

Astfel, am aplicat constrângerile de tip *on delete cascade* pentru cheia externă dintre tabele Rețete și Rețetă\_Ingrediente, adică pentru atributul id\_rețetă. Această constrângere asigură ștergerea asocierilor dintre toate ingredientele constituente ale unei rețete ce se dorește a fi ștearsă.

### II.2.7. Crearea tabelelor și a relațiilor dintre acestea în SQL Server

Pentru construirea modelului relațional prezentat anterior a fost folosita metoda [Code-First](#_II.3.4.Metodologia_Code-_First), cu ajutorul căreia a fost generat scheletul bazei de date așa cum a fost prezentat în acest capitol.

II.2.8. Popularea bazei de date

Pentru popularea bazei de date, am folosit dataset-ul ,,Cocktail Ingredients”[[[6]](#endnote-6)] de pe platforma Kaggle pe care l-am prelucrat cu ajutorul funcțiilor puse la dispoziție de Microsoft Excel. După ce datele au fost împărțite conform tabelelor anterior prezentate, am folosit o aplicație online Flatfile csvjson[[[7]](#endnote-7)] cu ajutorulcăreia am transformat fișierele CSV în obiecte de tip JSON. Cele din urmă menționate au fost utilizate împreuna cu API-urile realizate pe partea de backend pentru a popula baza de date.

## II.3. Cum funcționează o aplicație de tip web?

### II.3.1. Cererile web

Orice aplicație web are cel puțin două componente: clientul și serverul. Prin client se înțelege de cele mai multe ori un browser, dar acest termen poate face referire și la o aplicație executabilă sau la un dispozitiv de tip terminal, în timp ce serverul găzduiește informația și la nevoie este cel care oferă ceea ce clientul a solicitat, în limita resurselor de care acesta dispune. Serverul și clientul sunt de regulă pe dispozitive diferite, însă există și posibilitatea folosirii unui server local, astfel clientul și serverul pot coexista pe același calculator/dispozitiv.

Cele două entități (serverul si clientul) ce alcătuiesc aplicația comunică și schimbă date prin ceea ce se numesc cereri web. Pentru a putea comunica într-o manieră ce sa fie înțeleasă de ambii participanți, au fost dezvoltate anumite protocoale specifice.

Graphical user interface, diagram, application

Description automatically generated

Figura 3.1. Reprezentarea schematică a comunicării client-server [[[8]](#endnote-8)]

Un protocol reprezintă o metoda acceptată și înțeleasă de toți participanții, în vederea stabilirii unor sesiuni de comunicare prin intermediul cărora are loc schimbul de informații. Printre cele mai cunoscute protocoale folosite pentru comunicarea web se numără: Hypertext Transfer Protocol (HTTP), Secure Hypertext Transfer Protocol(HTTPS), File Transfer Protocol(FTP) și Network News Transfer Protocol(NNTP). Aceste protocoale sunt dublate de cel de rețea numit Transmission Control Protocol/Internet Protocol (TCP/IP), protocolul standard de comunicare globală cu ajutorul căruia se stabilesc regulile fundamentale pe baza cărora cele două dispozitive schimbă informații.

Pentru a se stabilii o sesiune, se utilizează protocolul de DNS. Acesta poate fi înțeles cel mai simplu ca o carte mare de telefon în care fiecărui site web ii corespunde o adresă IP. Odata descoperită adresa IP la care se găsește site-ul se încearcă inițierea unei sesiuni.

Pentru a se iniția o comunicare între client si server, clientul are sarcina de trimite o cere de inițializare pentru începerea unei sesiuni, în timp ce serverul doar așteaptă. Inițializarea sesiunii se realizează prin accesarea uni link, ceea ce declanșează transmiterea unei serii de biți pe care serverul o înțelege. In cazul în care cererea este acceptată, serverul răspunde cu o altă serie de biți pentru a confirma că sesiunea a fost deschisa.

După ce sesiunea a fost stabilită serverul trimite către client o copie a ceea ce el găzduiește, browser-ul fiind cel care se ocupă de recepționarea pachetelor de date, asamblarea acestor si afișarea lor într-o manieră lizibilă de către oameni.

## II.4. Backend-ul aplicației

### II.4.1. ASP .Net Core

Pentru construirea părții de backend a aplicației și în special a ceea ce reprezintă conceptul de server al acestei aplicații (server-side) am ales folosirea framework-ului ASP .Net Core. Aceasta este o platformă open-source dezvoltată de Microsoft ce a fost gândită pentru a construi, în mare parte, aplicații si servicii web. Acesta suportă trei șabloane arhitecturale: Web Forms, Web Pages și Model-View-Controller. Aplicația ce face obiectul acestei lucrări a fost realizată utilizând cea din urmă arhitectură, MVC.

### II.4.2. Modelul arhitectural Model-View-Controller (MVC)

Conceptul de Model-View-Controller (MVC) aduce multe îmbunătățiri în comparație cu predecesorul sau ASP .NET Web Form. Acesta reprezintă un model/șablon arhitectural ce oferă utilizatorilor săi o abordare abstractizată si foarte bine partiționată, în vederea facilitării dezvoltării și mentenanței unei aplicații web. Implementarea unui astfel de șablon conduce la o separare foarte bine delimitată a interfeței de logica operațională a aplicației și conferă codului un grad mare de reutilizare si posibilitatea de a fi dezvoltat cu ușurință în paralel de mai mulți programatori.

Principala caracteristica a acestui model arhitectural este divizarea unei aplicații mari în trei părți importante: Model, View și Controller. Fiecare dintre cele trei componente este gândită pentru a îndeplinii sarcini diferite, ce împreuna construiesc aplicația și definitivează funcționalitățile acesteia.

Astfel, modelul (Model) este componenta ce conține toata logica legată de datele cu care aplicația lucrează. Altfel spus, cu ajutorul modelelor se structurează entitățile si atributele constituente bazei de date. Prin urmare, pentru fiecare entitate prezentate în capitolul [*II.1. Baza de date*](#_II.1._Baza_de)a fost definit un model reprezentat de o clasă, însoțit de atributele aferente ce reprezintă membrii clasei. Un exemplu elocvent pentru ilustrarea acestui concept este clasa utilizator. Un obiect de tip utilizator va avea asociate toate datele relevante precum: nume, prenume, email, parolă, acest obiect având posibilitatea de a actualiza datele existente sau de a le utiliza în vederea îndeplinirii unor acțiuni/servicii.

Componenta View face referire la interfata ce se construiește pentru un anumit obiect. Astfel, pentru modelul utilizatorului, menționat anterior, aceasta componentă va conține componentele UI cu care acesta interacționează.

Controllerul reprezintă ,,creierul” aplicației, el fiind cel care se ocupa de toata logica acesteia, intermediind tranzacțiile ce au loc intre model si view. De exemplu, în cazul utilizatorului, controller-ul său se va ocupa de preluarea date din formularul de înregistrare (componenta view) le va prelucra și valida, iar mai apoi si le va trimite către model pentru a fi înregistrate în baza de date.

Diagram

Description automatically generated

Figura 2.2. Arhitectura MVC. Cum funcționează MVC ?[[[9]](#endnote-9)]

Tot procesul prin care cele trei componente interacționează este unul foarte simplu si bine definit. Astfel, odată ce browser-ul trimite o solicitare către aplicație, aceasta ajunge la controller. De aici, controllerul se ocupa cu interacțiunea cu modelele definite pentru a primi si a trimite apoi mai departe, către view, datele. View-ul se ocupa doar de afișarea informațiilor, prin intermediul unor pagini HTML dinamice, ce combina cod static de HTML cu conținut adus de controller din baza de date prin intermediul modelului. Odată ce acțiunile utilizatorului asupra componentei vizibile a aplicației (view) s-au încheiat, view-ul retrimite răspunsul către controller, care analizează și interpretează datele primite, pentru ca mai apoi tranzacția sa se finalizeze cu inserarea acestora în baza de date, prin intermediul modelului.

### II.4.3.Metodologia Code-First folosind Entity Framework

Odată cu apariția lui Entity Framework, dezvoltatorii au posibilitatea de a accesa si manipula datele folosind obiecte ce sunt asociate claselor ce intra in componența aplicației, lucru ce conduce la un grad mai mare de abstractizare al mediului de dezvoltare.

Ca o consecință foarte importantă a introducerii acestui framework, a apărut metodologia de dezvoltare a aplicațiilor in maniera Code-First, pe care am ales sa o folosesc si în dezvoltarea acestei aplicații.

Astfel, aceasta abordare a presupus creare in prima etapa a claselor asociate entităților bazei de date. Am definit entitățile, atributele si relațiile dintre acestea, prin scrierea de cod in limbajul C#. Pentru a coordona întreaga activitate a EF a fost necesară creare unei clase principale, numită Context ce derivă din clasa Microsoft.EntityFrameworkCore.DbContext și cu ajutorul căreia se stabilesc entitățile și relațiile dintre acestea. (este necesaa introducerea unei anexe ?). Pentru ca API-ul sa știe unde trebuie sa se creeze noua bază de date, în fișierul de configurații al proiectului *appsettings.json*  trebuie adăugat șirul de caractere ce definește conexiunea:

"ConnectionStrings": {

"DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=licenta3;Trusted\_Connection=True;MultipleActiveResultSets=true"

}

Odată făcute aceste configurări, am realizat migrarea bazei de date către serverul local de SQL.

### II.4.4. Repository Pattern

Repository pattern este o metodă eficientă de a implementa accesul la date. Acesta permite extragerea unuia sau mai multor înregistrări din baza de date, pe care mai apoi se poate acționa precum asupra unei colecții de obiecte. Atât datele, cât și metodele aferente acestora sunt încapsulate, formând un tot unitar si oferind codului o structură orientată pe obiect, fapt ce permite programatorului sa prelucreze cu ușurință datele înainte de a le folosi.

Printre avantajele folosirii acestei metode se numără creșterea capacitații de testare și facilitarea schimbărilor datelor, fără a fi nevoie de modificarea directă a API-ului. Entity Framework implementează de asemenea acesta structura pentru a ține evidența si a salva modificările ce apar în compoziția modelelor implementate.

Având in vedere ca multe dintre operațiile ce se execută asupra claselor definite sunt similare, implementarea acelorași funcții pentru fiecare model ar conduce la apariția unui cod repetitiv. Pentru a rezolva aceasta probleme se apelează la construirea unui repository generic ce ulterior va fi moștenit de toate modelele spre a fi implementat și, eventual completat, cu alte metode specifice fiecaruia.

Prin urmare, am ales utilizarea acestei metode de dezvoltare si organizare a codului pentru proiectul in cauză, întrucât este dovedită a fi una foarte eficientă.

Astfel, pentru început, am definit o interfața pentru repository-ul generic (*IGenericRepository*) ce conține metodele de bază (operațiile de *CRUD*) ce pot fi efectuate asupra unui model de date. Printre aceste metode se numără metodele de: *Create*, *Update*, *Delete*, *Find* și *Save.* Această interfață a fost implementată ulterior in clasa GenericRepository, unde funcțiilor definite anterior le-a fost asociat și un comportament.

Mai apoi, pentru fiecare clasă/model definit am implementat atât o interfața a repository-ului, cât și o clasa propriu-zisă ce moștenesc entitățile generice menționate anterior. Pe lângă metodele moștenite din clasa generica, fiecare repository este completat cu alte funcții specifice clasei sale precum: *GetByNume*, *GetByCategory* sau *GetBySubcategory*.

### II.4.5. Controlere și API-uri

Așa cum a fost prezentat și în unul dintre capitolele anterioare ,,[Modelul arhitectural Model-View-Controller (MVC)](#_II.4.2._Modelul_arhitectural)”, controlerul este responsabil pentru gestionarea modului în care utilizatorul interacționează cu o aplicație de tip MVC.

Acestea conțin logica cu ajutorul căreia se controlează fluxul de date, determinând astfel ce răspuns se trimite utilizatorului ca urmare a solicitării pe care acesta a făcut-o prin intermediul browser-ului sau aplicației. În esență, controlerul reprezintă tot o clasă, ce utilizează metodele implementate in partea de repository pentru a efectua [cereri web](#_II.3.1._Cererile_web) de tipul get, post și nu numai și astfel contribuie la construirea API-urilor ce vor interacționa cu partea de frontend a aplicației.

API reprezintă un acronim pentru Application Programming Interface (Interfața de programare a aplicației). Un API este un intermediar software ce permite ca două aplicații sa comunice între ele. În cazul de față, cele două aplicații pentru care trebuie facilitată o cale de comunicare sunt cele două părți ale aplicației: backend-ul, realizat in C# ASP .Net Core și frontend-ul, realizat in Angular cu ajutorul framework-ului Ionic.

În esență, API-ul este cel care se ocupa de preluarea cererii pe care un utilizator o efectuează către server (aici backend-ul ce comunică cu baza de date). Odată primită cererea, serverul se ocupa de gestionarea acesteia și trimite înapoi, tot prin intermediul API-ului răspunsul așteptat de către utilizator.

### II.4.6. DTO-uri

DTO este un acronim de la Data Transfer Object (Obiect cu ajutorul căruia se transfera date). Acest obiect definește modul în care se trimit datele prin rețea și contribuie la eliminarea situațiilor in care API-ul expune direct clientului entitățile bazei de date, adăugând astfel un strat suplimentar de securitate. In plus, prin folosirea DTO-urilor este facilitată optimizarea cantității de date ce sunt trimise prin intermediul API-urilor și, totodată, este oferit contextul necesar pentru a modifica sau prelucra datele, înainte ca acestea să ajungă la utilizator, sau chiar de a elimina anumite proprietăți ce nu sunt relevante din obiectul transmis.

Obiectele de tip DTO pot fi asemănate cu niște containere, fiind utilizate exclusiv pentru a transfera date între diversele ,,straturi” ale aplicației. In cadrul acestor obiecte nu sunt înglobate funții, metode, sau orice alt tip de acțiune ce are legătură cu partea de logică a aplicației.

## II.5. Mecanismul de autentificare in aplicație pe partea de server

### II.5.1. JWT Token

JWT(Json Web Token) este un standard ce poate fi folosit de orice pentru a realiza un transfer de date securizat între oricare doi utilizatori, chiar daca este vorba de relația server-client, oferind asigurări cu privire la integritatea datelor transmise.[[[10]](#endnote-10)]

Timeline

Description automatically generated

Figura 2.3. Structura unui token JWT [[[11]](#endnote-11)]

Un token de tip JWT este format din trei componente: antetul, date ce trebuie transmise și nu în ultimul rând o semnătură. In antet se găsesc date referitoare la algoritmul folosit precum cel utilizat în aceasta aplicație [HmacSha256](#_II.5.2._Algoritmul_de) și informații despre tipul de token. În a doua parte a token-ului, sunt trimise datele necesare autentificării precum numele de utilizator și parola sau alte informații adiționale, denumite generic *claims* (i.e. cereri). Ultima parte a token-ului este reprezentată de o combinație dintre antet, date și parola secretă, toate fiind criptate.

Din punct de vedere conceptual, mecanismul de autentificare folosind JWT poate fi descris destul de ușor, prin intermediul unor pași foarte bine definiți. Astfel, odată ce utilizatorul introduce credențialele sale pentru autentificarea în aplicație, solicitarea este trimisă către server. Serverul generează token-ul JWT pe care îl întoarce ca răspuns către client. Clientul retrimite o copie a token-ului pentru a avea loc validarea. Odată efectuată validarea de catre server, acesta trimite un răspuns clientului, comunicându-i daca tranzacția a avut loc cu succes sau nu. În caz afirmativ autentificarea poate avea loc, iar în caz contrar este arunată o excepție.

### II.5.2. Algoritmul de criptare folosit - HmacSha256Signature

Hmac este un acronim de la ,,hash-based message authentication code” ceea ce reprezintă un algoritm de hashing criptografic pe baza căruia se genereaza un cod de autentificare al mesajelor (MAC). Acest algoritm realizează tabela Hash a unui mesaj, utilizând o cheie secretă, rezultatul sau fiind considerat semnătura acelui mesaj. Procesul HMAC presupune amestecarea cheii secrete cu datele mesajului, urmat de indexarea rezultatului cu funcția hash, apoi amestecă din nou acea valoare hash cu cheia secretă și aplică a doua oară funcția hash, codul hash rezultat având lungimea de 256 biți. Odată criptat mesajul, acesta este trimis destinatarului, care, dacă este în posesia cheii secrete poate verifica dacă semnătura sa se potrivește cu cea trimisă împreună cu mesajul. Pentru a se face această validare expeditorul, trimite împreună cu mesajul si hash-ul aferent acestuia, iar receptorul recalculează independent hash-ul pe care, mai apoi, îl compară cu cel primit inițial. Prin urmare, dacă valorile hash originale coincid cu cele calculate la recepție înseamnă ca poate avea loc autentificarea.[[12]](#endnote-12)

Funcția hash utilizată în algoritmul de criptare este SHA-256, care este una dintre cele mai sigure funcții de hashing existente. Aceasta funcție este un urmaș al lui SHA-2 (Secure Hash Algorithm 2), care a fost creata de Agenția Națională de Securitate in 2001[[[13]](#endnote-13)]. Acest algoritm de criptare este utilizat în unele dintre cele mai populare protocoale de autentificare și criptare precum: SSL, TLS, IPsec, SSH și PGP, sau în cadrul unor sisteme de operare precum Unix și Linux pentru construirea hashing-ului securizat al parolelor. Totodată, criptomonedele precum Bitcoin folosesc SHA-256 pentru verificarea tranzacțiilor.

Deoarece este un algoritm folosit în foarte multe medii si contexte importante, detaliile exacte referitoare la modul in care acesta funcționează sunt clasificate. Printre datele publice despre acest algoritm de criptare se numără faptul ca este construit cu o structură Merkle-Damgård derivată dintr-o funcție de compresie unidirecțională, creată cu structura Davies-Meyer dintr-un cifru bloc specializat, care operează pe grupuri de biți cu lungime fixă.

### II.5.3. Autentificarea propiu-zisă

Din punct de vedere practic, pentru implementarea procedeului descris anterior în această aplicație am procedat după cum urmează:

Pentru a putea implementa mecanismul de autentificare al unui utilizator, a fost necesară implementarea unei clase specifice acestuia. Membrii acestei clase sunt reprezentați de datele necesare identificării și autorizării unui utilizator precum: nume, prenume, nume de utilizator, e-mail, hash-ul aferent parolei criptate, precum și rolul pe care utilizatorul îl are (obișnuit sau administrator). Cu ajutorul atributului de rol se realizează limitarea accesului pe anumite end-point-uri ale aplicației.

Ulterior am declarat o interfață și un serviciu ce se ocupa de aceste procese. Gestionarea acestor procese a fost posibilă prin implementarea unor [DTO](#_II.4.6._DTO_-uri)-uri specifice, cu ajutorul cărora se trimit solicitări și se primesc răspunsuri de la server. Astfel, am creat, pe de o parte, un DTO pentru cererea pe care un utilizator o trimite către server, numit *UserRequestDTO*, în cadrul cărui am definit atributele ce vor trebui completate de către utilizator pentru autentificare în aplicație, precum numele de utilizator, parola, email. Pentru a trimite un răspuns la aceasta solicitare, serverul va utiliza tot un obiect de tip DTO, denumit *UserResponsDTO* prin intermediul căruia va furniza date aferente contului create precum: id-ul utilizatorul, numele de utilizator și token-ul generat pentru autentificare.

În cadrul funcției de autentificare, odată găsit în baza de date utilizatorul ce dorește să se autentifice, am verificat cu ajutorul metodei *BCryptNet.Verify*, din pachetul *BCryptNet.Core,* dacă hash-ul din baza de date corespunde cu cel introdus de utilizator. Dacă această validare a avut succes, se generează un token de tip JWT care va fi memorat în aplicație și îi va permite utilizatorului să efectueze orice acțiune specifică unui utilizator autentificat.

Pentru a genera token-ul necesar autentificării, am creat o clasa ajutătoare în cadrul cărei am implementat două metode. Prima metodă, este una ce primește ca parametru id-ul utilizatorului și generează token-ul necesar autentificării. În timp ce, cea de a doua metodă efectuează validarea token-ului pe care îl primește ca parametru, întorcând id-ul utilizatorului.

Pe lângă clasa utilitară declarata anterior, am definit încă o clasă, denumită sugestiv *AppSettings,* în cadrul căreia am definit un cod secret, pe baza căruia are loc generarea token-ului. Această clasa a fost ulterior injectată prin intermediul Dependency Injection în clasa utilitară aferentă generării de token-uri și codificată utilizând metoda *Encoding.ASCII.GetBytes*.

In funcția de generare a token-ului am utilizat un obiect de tipul *JWTSecurityTokenHandler* si unul de tipul *SecurityTokenDescriptor* pentru a putea defini modul în care acest token se va genera. În cadrul celui din urmă obiect menționat am definit un alt obiect de tip *ClaimIdentity* cu ajutorul căruia pot extrage, din token, id-ul utilizatorului și pot fixa perioada de timp după care acest token va expira, fiind nevoie astfel de o reautentificare a utilizatorului. Un alt atribut al acestui obiect este reprezentat de setarea tipului de algoritm folosit pentru criptare. În cazul aplicației in cauza am ales sa folosesc o criptare cu cheie simetrica utilizând algoritmul [*HmacSha256Signature*](#_II.5.2._Algoritmul_de), descris anterior.

Funcția de validare primește ca parametru tokenul, de tip șir de caractere. Pentru a putea sa valideze acest token, aceasta folosește un obiect de tipul *TokenValidationParameters* ce definește atributele necesare. Dacă tokenul nu este null și acesta trece de validările definite anterior se extrage din claim-ul definit in jwtToken id-ul utilizatorului se a solicitat autentificarea.

În serviciul de autentificare am creat o instanță a clasei utilitare *JWTUtils* pentru a putea apela metoda de generare a tokenului bazata pe id-ul utilizatorului, primind ca răspuns un obiect de tip DTO ce conține token-ul generat.

Apelarea propiu-zisă a acestui serviciu are loc in controler-ul utilizatorului. Pe langa apelarea serviciului de autentificare, în cadrul acestui controler, am definit și metodele de creare/ înregistrare a unui utilizator, precum si o metodă de tip get ce intoarce toți utilizatorii aplicatiei. Pentru ca ultima metodă menționată să fie accesibilă doar unui utilizator autentificat cu rol de administrator, aceasta metoda este însoțită de atributul *[Authorize (Roles=”Admin”)].*

## II.6. Algoritmi utilizați

### II.6.1. Algoritm de generare rețetă pe baza unei liste de ingrediente

### II.6.2. Algoritm de sugerare a unei rețete bazat pe preferințele anterioare

## II.7. Frontend-ul aplicației

### II.7.1. Introducere

La fel ca orice alta aplicație de tip web/mobile, pe lângă nucleul aplicației (componenta de backend a acesteia) care se ocupă de logică si funcționalități, este nevoie și de o componenta UI, care sa intermedieze interacțiunile utilizatorilor cu aplicația în sine.

Deși cele doua componente: Frontend și Backend reprezintă doua concepe radical diferite in dezvoltarea web acestea trebuie sa comunice și să interacționeze eficient una cu cealaltă, pentru a putea construi o aplicație eficientă si ușor de utilizat.

Frontend-ul reprezintă acea parte a site-ului web sau a aplicației de mobil cu care utilizatorii interacționează în mod direct. Aceasta include tot ceea ce utilizatorii vad atunci când accesează aplicația. De la imagini, text, culori și până la grafice, tabele, meniuri sau animații, toate aceste elemente fac parte din interfața de utilizare, adică din componenta de fronend a aplicației.

Principalele obiective ce trebuie urmărite pe parcursul dezvoltării frontendu-ului sunt receptivitatea si performanța aplicației. Conținutul aplicației trebuie sa poată fi încărcat și afișat într-un mod cat mai eficient din punct de vedere computațional, dar si într-un mod atractiv și intuitiv, pentru a facilita interacțiunea utilizatorului cu aplicația.

În general, pentru dezvoltarea acestei componente a aplicației sunt utilizate câteva limbaje de programare de bază din care au fost derivate, de-a lungul timpului mai multe framework-uri.

La baza dezvoltării interfeței cu utilizatorul stau tehnologiile precum: HTML, CSS și JavaScript. Acestea sunt folosite atât pentru definirea elementelor statice ale site-ului (imagini, text), cât și pentru stilizarea sau implementarea diverselor funcționalități care țin de navigarea prin aplicație.

Tehnologiile menționate anterior sunt folosite pentru dezvoltarea diverselor framework-uri. Printre cele mai utilizate framework-uri se numără: Angular și React care folosesc la baza JavaScript, Danjo care utilizeaza Python și Flutter care se bazează pe o tehnologie numită Dart.

### II.7.2. Compararea aplicațiilor web, hibrid și native

Pentru dezvoltarea interfeței cu care vor interacționa utilizatorii, am ales sa folosesc framework-ul **Ionic**. Cu ajutorul acestui framework se pot dezvolta aplicații de tip hibrid, de aceea am considerat utilă ințelegerea diferentelor dintre principalele concepte ce sunt abordate in realizarea aplicațiilor.

Conceptul unei aplicații hibrid este unul simplu, dar eficient. Întreaga aplicație este scrisă folosind tehnologii web rezultatul fiind un proiect pur web. Acest proiect web este prezentat utilizatorului într-un web view. Acest lucru înseamnă că folosind uneltele oferite de producător, proiectul web este compilat și transformat în tehnologiile de bază, pentru a asigura compatibilitate și eficiența maximă, iar mai apoi este împachetat într-o aplicație în care se creează un server web izolat și local. În aceeași aplicație este creat și prezentat un web view (nucleul unui browser obișnuit, un browser fără controale sau posibilitate de control direct) care reprezintă adevărata „aplicație”, și anume proiectul web menționat anterior.

În figura de mai jos se poate observa comparația unei aplicații de tip hibrid cu o aplicație nativă și una pur web.

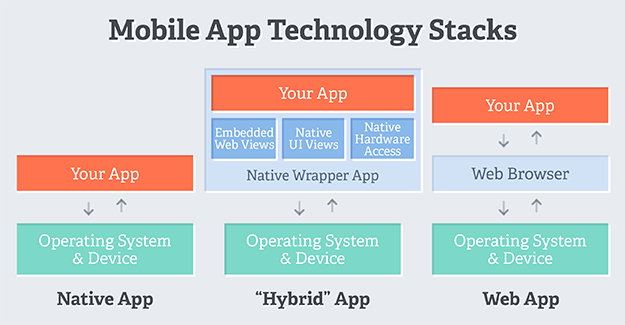


Figura 2.4. Compararea mai multor tipuri de aplicații[[[14]](#endnote-14)]

Din punctul meu de vedere, printre principalele avantaje ale utilizării acestui framework se numără viteza de dezvoltare a aplicației, întrucât se poate scrie o singură dată cod, iar mai apoi aplicația poate fi folosita în mai multe medii (web, Android, IOS), oferirea de suport pentru un număr forte mare de componente ce sunt utilizate, de regulă, în aplicațiile de mobil native, animații interesante si un design simplu, frumos și modern.

Pe lângă avantajele anterior menționate, în poza de mai jos sunt prezentate în detaliu atât punctele forte, cât și cele mai slabe ale fiecărui tip de aplicație.

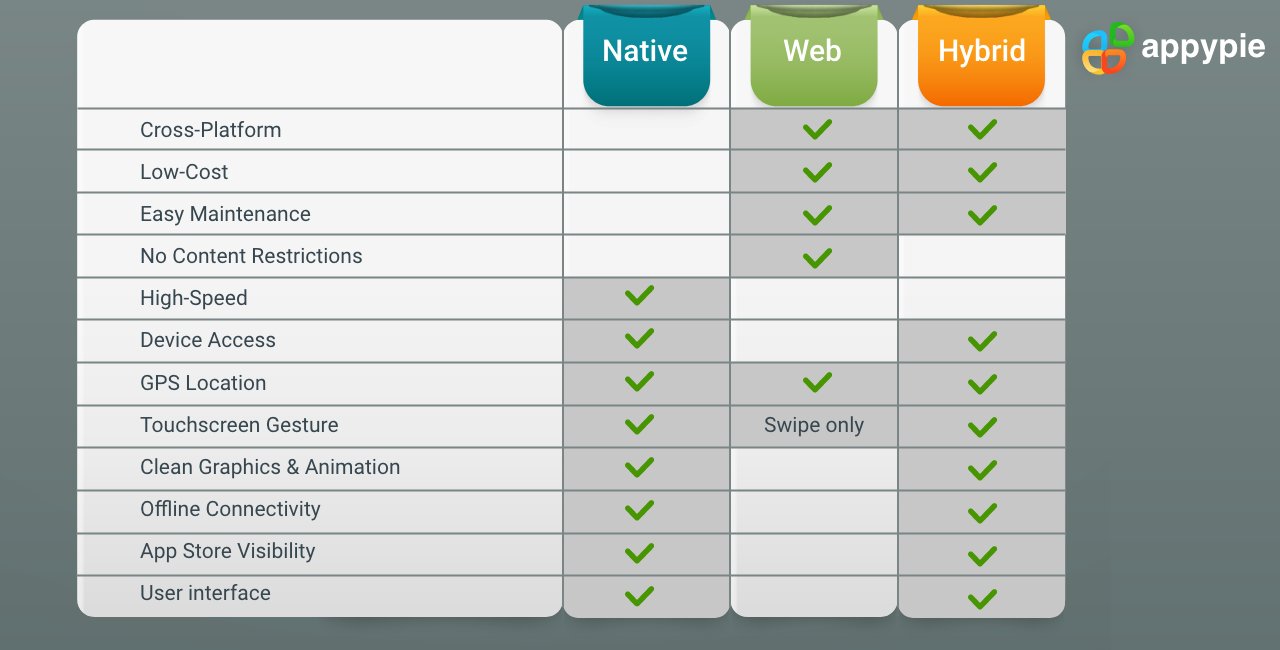


Figura 2.5. Compararea mai multor tipuri de aplicații (Avantaje și dezavantaje)[[[15]](#endnote-15)]

### II.7.3. Framework-ul Ionic

Acesta este un framework de dezvoltare conceput pentru construirea de aplicatiii de mobil hibride ce au la bază HTML5. Aceste aplicatii hibride reprezinta site-uri web de dimensiuni mici care rulează intr-un shell al browserului in cadrul unei aplicatii care are acces la nivelul platformei native. [ [[16]](#endnote-16)] Într-o maniera asemănătoare tehnologiei Java, acesta promovează motto-ul „Scrie o data, rulează peste tot”.

În comparație cu un framework responsive, Ionic aduce elemente de interfață mobila care se pot obține cu un SDK (Software development kit, i.e. un set de instrumente furnizat de producătorul unei platforme, sistem de operare sau limbaj de programare [[[17]](#endnote-17)]) nativ pe IOS sau Android, dar care până la apariția lui (prima versiune- 2013 [[[18]](#endnote-18)]) nu existau cu adevărat.

Deoarece Ionic este un framework HTML5, acesta are nevoie de un wrapper nativ precum Cordova sau PhoneGap pentru a rula precum o aplicație nativă.

Întrucât Ionic are ca fundație o pagina web ce rulează într-un shell nativ de aplicație mobilă, se pot folosi oricare dintre tehnologiile de bază utilizate în dezvoltarea site-urilor web consacrate precum: HTML, CSS sau JavaScript, precum si AngularJS. Singur diferență fundamentală între un site web si o aplicație Ionic fiind reprezentata de faptul ca link-urile sunt înlocuite de o experiența de navigare similară unei aplicații mobile de sine stătătoare.

In ceea ce privește API-urile, Ionic folosește API-uri web moderne, cum ar fi Shadow DOM. Astfel, componentele Ionic au un API stabil și nu depind de un singur furnizor de platformă.

Începând cu versiunea 4 de Ionic, acest framework a fost îmbunătățit și făcut să funcționeze ca o bibliotecă de componente web în cadrul căreia au fost integrate cele mai noi framework-uri de JavaScript precum: Angular, React sau Vue. In cadrul aplicației mele am ales să utilizez integrarea Ionic-ului cu [Angular](#_II.7.5._Framework-ul_Angular).

Linia de comandă a Ionic-ului vine cu un CLI ce oferă o serie de comenzi foarte utile dezvoltatorului, un server de dezvoltare încorporat, instrumente de depanare, dar și cu posibilitatea realizării unor cereri și implementări în cloud. [[[19]](#endnote-19)]

### II.7.4. Arhitectura proiectului Ionic

Arhitectura unui proiect Ionic va fi descrisă atât analitic, din punctul de vedere al componentelor software, cât și fizic, din punct de vedere al dispunerii fișierelor pe hardisk, prin următoarele ilustrații.

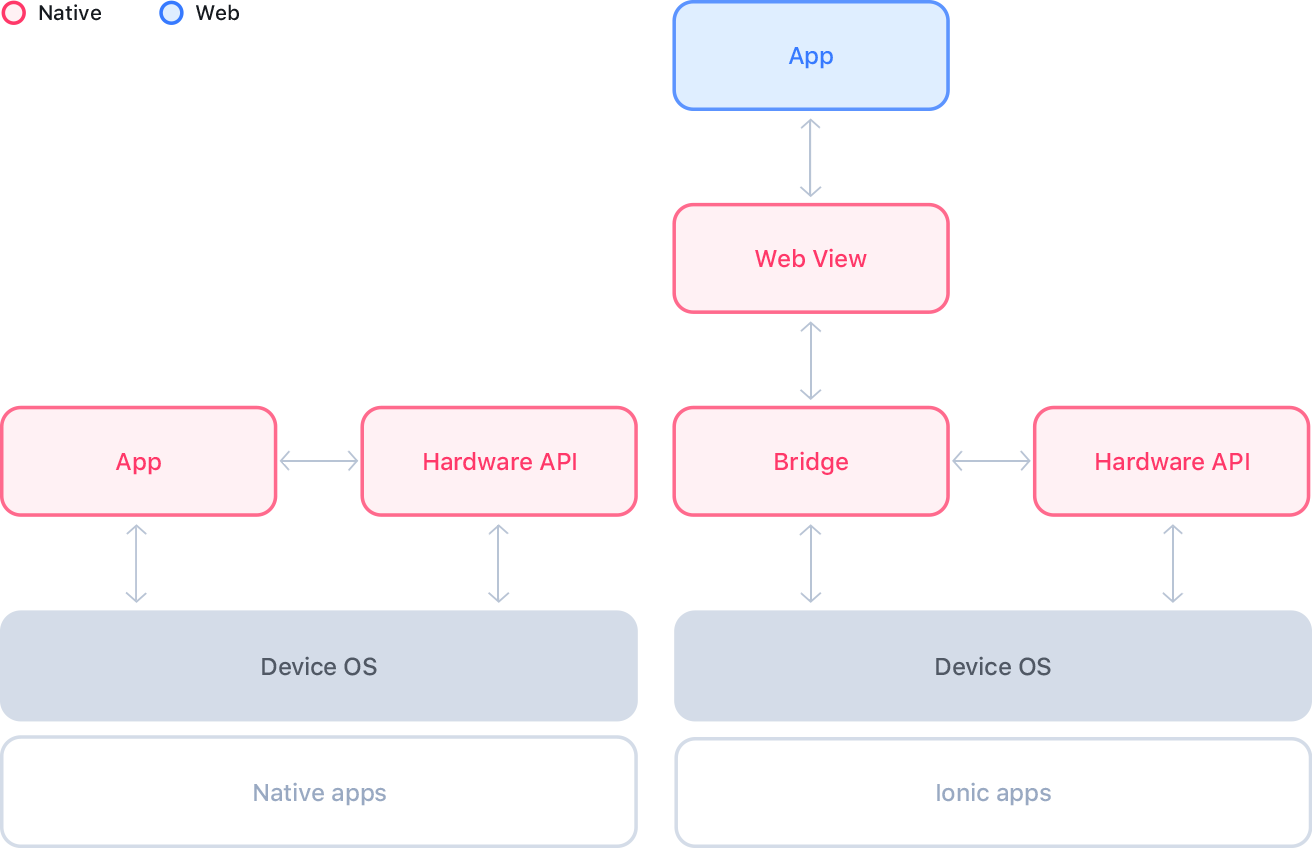


Figura 2.6. Arhitectura din punct de vedere a componentelor software [[[20]](#endnote-20)]

Graphical user interface, text, application

Description automatically generatedText

Description automatically generatedCu ajutorul imaginii de mai sus se poate observa cu ușurința ca arhitectura Ionic se pliază perfect pe tipul unei aplicații hibrid.

Figură 2.7. Arhitectura fișierelor

Fișierele config.xml si ionic.config.json sun fișierele principale care controlează întreg proiectul. Folderul „src” este locul unde proiectul se dezvolta. Folderul „www” este locul unde sursele sunt transferate și compilate. Folderul „platforms” este locul unde sunt ținute mini-proiectele ce vor servi ulterior ca „Native Wrapper” pentru aplicația hibrid. „Node\_modules” reprezintă locul unde sunt stocate pachetele NodeJS, instalate prin NPM (node package manager). Fiind strâns legat de Framework urile JS, Ionic instalat si controlat prin NodeJS, are acces la întregul univers de pachete al NPM-ului.

### II.7.5. Framework-ul Angular

Angular este o platformă de dezvoltare creată de cei de la Google, construită pe baza [TypeScript](#_II.7.6._TypeScript). Ca platformă, Angular include: un cadru bazat pe componente pentru construirea de aplicații scalabile, o colecție bogată de biblioteci care acoperă o gama variată de aplicabilități, o suită de instrumente pentru dezvoltatori si testări software.

Unul dintre cele mai importante avantaje ale tehnologiei Angular este posibilitatea de scalare a aplicației, dar si o comunitate extinsă de dezvoltatori care numără peste 1,7 milioane de dezvoltatori. [[[21]](#endnote-21)]

Scalabilitatea aplicațiilor ce utilizează Angular este dată de structura modulară, aceasta fiind constituită din mai multe componente funcționale și logice. Aceste componente reprezintă blocuri cu ajutorul cărora se construiește interfața cu utilizatorul. O componentă din Angular constă dintr-un fișier de clasă, un șablon (HTML) și un fișier eventual un fisier css/scss, folosit pentru stilizarea componentei. Componentele sunt elemente reutilizabile, prin urmare pot fi utilizate în orice altă parte a aplicației și pot fi testate independent.

Șablon HTML al unei componente este utilizat pentru a declara cum se afișează acea componentă. Pe lângă partea statică, Angular extinde HTML cu o sintaxă suplimentară ce permite să inserezi valori dinamice din componenta asociată.

Mai multe componente Angular, împreună cu o serie de directive și servicii, ce sunt legate din punct de vedere logic și desemnează, în esență ,o funcționalitate formează un modul.

Serviciile sunt clase de obiecte singleton (cu o singură instanță) într-o aplicația Angular. Prin intermediul injectării de servicii în componente se partajează a datelor între componentele aceleiași aplicații.

Pentru a putea discuta despre o altă categorie de elemente importante din cadrul oricărei aplicații Angular este nevoie sa înțelegem mai bine ce reprezintă DOM.

DOM este acronimul de la ,, The W3C Document Object Model (DOM)” și reprezintă o interfață care permite programelor si scripturilor sa acceseze și să actualizeze într-un mod dinamic conținutul, structura sau stilizarea unui document.

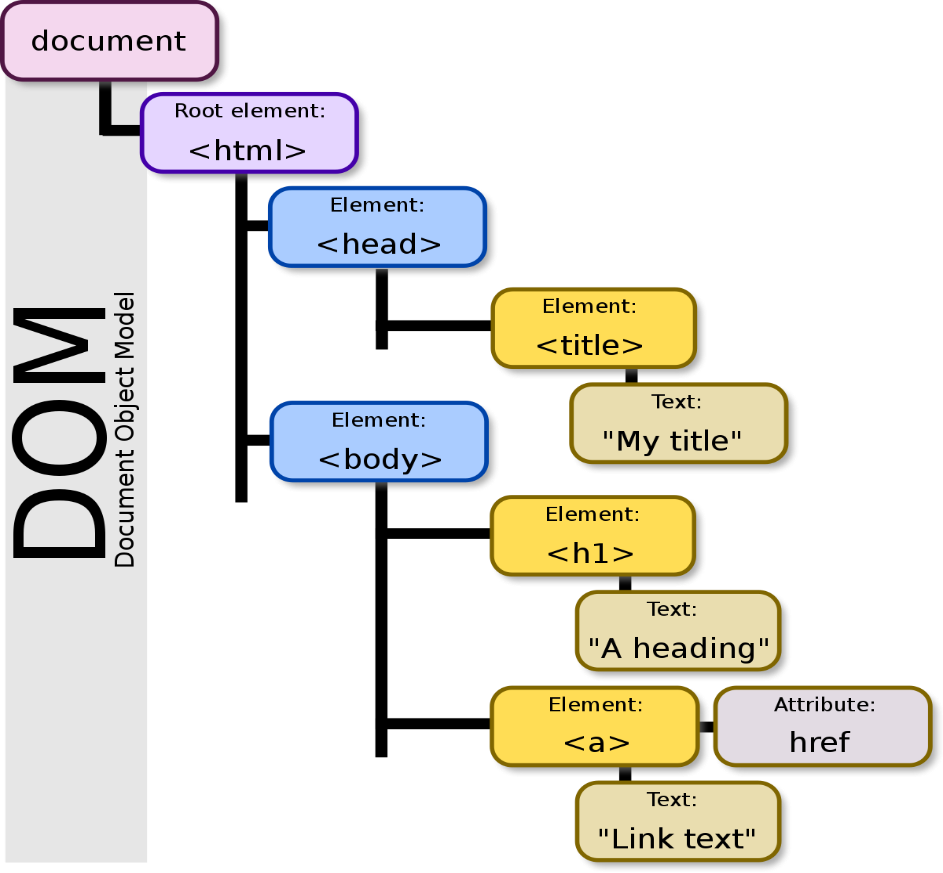


Figura 2.8. Arborescența DOM – Document Object Model [[22]](#endnote-22)

O altă categorie de element importante în orice aplicație de tip Angular sunt directivele. Directivele sunt o modalitate de a oferi instrucțiuni DOM-ului. Directivele atașează un comportament specific DOM și ne ajută să transformăm interfața de utilizare în funcție de logica de afaceri sau de interacțiunile utilizatorilor. [ [[23]](#endnote-23)] Aceste directive pot fi de mai multe feluri: directive comportamentale, structurale sau de atribute.

Directivele comportamentale sunt utilizate în clasa principală și conține detalii despre felul in care componenta ar trebui sa fie procesata, instanțiată si utilizată.

Directivele structurale sunt folosite pentru a manipula și/sau modifica structura elementelor ce intra in componența DOM. Ele â au o sintaxa specifică, începând cu \* si fiind urmate de grupul de litere ,,ng” urmat de un cuvânt cheie pentru o instrucțiune de bază precum if, for switch. Exemple de astfel de directive sunt: \*ngIf, \*ngFor și \*ngSwitch.

Directivele de atribut sunt folosite pentru a schimba aspectul și comportamentul anumitor elemente din DOM. Exemple de astfel de directive sunt ngClass și ngStyle.[[[24]](#endnote-24)]

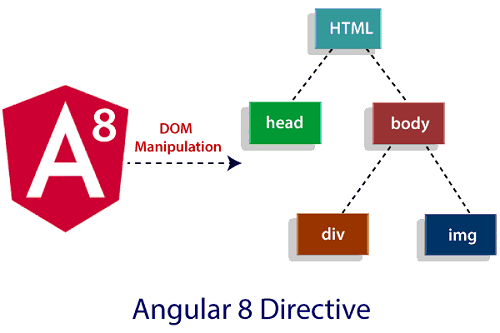


Figura 2.9. Directive Angular [[[25]](#endnote-25)]

Angular permite unei clase să folosească elemente ce au fost definite într-o altă clasă. Acest lucru este posibil prin injectarea dependințelor (,,dependency Injection”). De cele mai multe ori, acest proces se are loc prin injectarea unei clase de servicii într-o clasă asociată unei componente sau unui modul.

### II.7.6. TypeScript

TypeScript este un superset de JavaScript, ceea ce înseamnă că acesta conține toate funcționalitățile JavaScript. Prin urmare, orice program scris în JavaScript valid va rula, de asemenea, așa cum era de așteptat în TypeScript. Acesta a fost creat de Microsoft și a fost lansat în 2012, după doi ani de dezvoltare. A fost creat pentru a permite verificarea opțională a tipului static, care ar fi deosebit de utilă atunci când se dezvoltă aplicații la scară largă. TypeScript este o tehnologie open-source, iar codul este scris chiar în TypeScript.[[[26]](#endnote-26)]

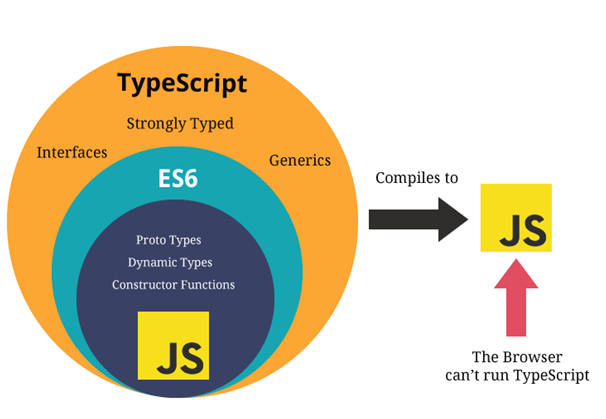


Figura 2.10. TypeScript [[[27]](#endnote-27)]

Diferența principala între JavaScript si TypeScript este faptul că cel din urmă menționat oferă mai mult control asupra codului prin adnotări de tip, interfețe și clase.

Principala caracteristică a acestui limbaj este gradul foarte crescut de ,,tipizare” , ceea ce înseamnă ca este necesară specificarea tipului de variabilă utilizat. Acesta nu rulează în browser, ci trebuie compilat cu JavaScript pentru a putea fi utilizat.

Next chapers

II.4. ML for suggesting new recepies based on your taste.

~~II.5. Frontend-ul – cateva date despre cum functioneaza Angular si componentele acestuia +cateva particularitati ale Ionic-ului.~~

# Bibliografie

1. [] Jennifer Robinson, MD: *Surprising Ways Alcohol May Be Good for You*, <https://www.webmd.com/diet/ss/slideshow-alcohol-health-benefits>, februarie 2022. [↑](#endnote-ref-1)
2. []  Adam Hughes - Site Editor, Craig Stedman- Industry Editor : *Microsoft SQL Server,*

   <https://searchdatamanagement.techtarget.com/definition/SQL-Server>, februarie 2022 [↑](#endnote-ref-2)
3. [] SQLSERVER TUTORIAL.NET: *What is SQL Server,* <https://www.sqlservertutorial.net/getting-started/what-is-sql-server/>, februarie 2022 [↑](#endnote-ref-3)
4. [] Ian, Database.Guide : *What does ACID mean in Database Systems?*, <https://database.guide/what-is-acid-in-databases/>, februarie 2022. [↑](#endnote-ref-4)
5. [] Alin Burța, *Informatică, manual pentru clasa a 12-a* , Editura ALL, 2007 [↑](#endnote-ref-5)
6. []  AIFirst*,Cocktail Ingredients*, <https://www.kaggle.com/ai-first/cocktail-ingredients>, martie 2022. [↑](#endnote-ref-6)
7. [] Flatfile csvjson, <https://csvjson.com/csv2json>, martie 2022. [↑](#endnote-ref-7)
8. [ ] MechoMotive, Working of website in real world, <https://mechomotive.com/wp-content/uploads/2021/07/Image_17_ComputerServer.png>, martie 2022. [↑](#endnote-ref-8)
9. [ ] Matthew Martin, Guru99, MVC Framework Tutorial for Beginners: What is, Architecture & Example, <https://www.guru99.com/images/1/122118_0445_MVCTutorial1.png>, martie 2022 [↑](#endnote-ref-9)
10. [] Gajendra Jangid, C-SharpCorner.com, How To Use JWT Authentication With Web API, <https://www.c-sharpcorner.com/article/how-to-use-jwt-authentication-with-web-api/>, martie 2022 [↑](#endnote-ref-10)
11. [ ] Gajendra Jangid, C-SharpCorner.com, How To Use JWT Authentication With Web API, https://csharpcorner-mindcrackerinc.netdna-ssl.com/article/how-to-use-jwt-authentication-with-web-api/Images/1.png, martie 2022 [↑](#endnote-ref-11)
12. [] Docs.mMicrosoft.com,HMACSHA256 Class, <https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.hmacsha256?view=net-6.0>, martie 2022 [↑](#endnote-ref-12)
13. [ ] N-ABLE, SHA-256 Algorithm Overview, <https://www.n-able.com/blog/sha-256-encryption>, martie 2022 [↑](#endnote-ref-13)
14. [] <http://www.adasoftware.com/software-development/mobileapp-development>, mai 2022 [↑](#endnote-ref-14)
15. [] https://images.appypie.com/wp-content/uploads/2019/08/22145854/Web-App-vs-Native-App-vs-Hybrid-App-Appy-Pie.png, mai 2022 [↑](#endnote-ref-15)
16. [ ] Welcome to Ionic – Guide, <https://ionicframework.com/docs/v1/guide/preface.html>, aprilie 2022 [↑](#endnote-ref-16)
17. [ ] What is an SDK?, <https://www.redhat.com/en/topics/cloud-native-apps/what-is-SDK>, aprilie 2022 [↑](#endnote-ref-17)
18. [ ] Ionic History, <https://www.javatpoint.com/ionic-history>, aprilie 20222 [↑](#endnote-ref-18)
19. [ ] Introduction to Ionic, https://ionicframework.com/docs , mai 2022 [↑](#endnote-ref-19)
20. [ ] <https://ionicframework.com/docs/core-concepts/webview>, mai 2022 [↑](#endnote-ref-20)
21. [ ] What is Angular?, <https://angular.io/guide/what-is-angular>, mai 2022 [↑](#endnote-ref-21)
22. [] DOM Image - https://devopedia.org/images/article/282/1930.1596042011.png , mai 2022 [↑](#endnote-ref-22)
23. [] EDUCBA: How Angular Works, by Priya Pedmarkar, <https://www.educba.com/how-angular-works/>, mai 2022 [↑](#endnote-ref-23)
24. [] Angular 8 Directives , <https://www.javatpoint.com/angular-8-directives>, mai 2022 [↑](#endnote-ref-24)
25. [] Angular 8 Directives, https://www.javatpoint.com/angular-8-directives , mai 2022 [↑](#endnote-ref-25)
26. [ ] TypeScript: What is it & when is it useful , Diaba Lease, martie 2022 [↑](#endnote-ref-26)
27. [] What is Typescript?, https://medium.com/@vishnupriya\_web/what-is-typescript-faa0890b2baf , [Vishnupriya](https://medium.com/@vishnupriya_web?source=post_page-----faa0890b2baf--------------------------------) , mai 2022 [↑](#endnote-ref-27)