Cuprins

[Introducere 2](#_Toc484353520)

[Motivaţie 2](#_Toc484353521)

[Context 2](#_Toc484353522)

[Cerinţe funcţionale 2](#_Toc484353523)

[Abordare tehnică 3](#_Toc484353524)

[Baza de date 3](#_Toc484353525)

[Backend 3](#_Toc484353526)

[Frontend 3](#_Toc484353527)

[Framework-uri 3](#_Toc484353528)

[Entity Framework Core 3](#_Toc484353529)

[Bootstrap 3](#_Toc484353530)

[Selenium 4](#_Toc484353531)

[Contribuţii 4](#_Toc484353532)

[Proiectare 5](#_Toc484353533)

[Arhitectura soluției 5](#_Toc484353534)

[Capitolul 1: Dezvoltarea aplicaţiei client 5](#_Toc484353535)

[1.1. Arhitectura generală a aplicației 5](#_Toc484353536)

[1.2. Angular Views 6](#_Toc484353537)

[1.3. Angular Controllers 6](#_Toc484353538)

[1.4. Angular Services 7](#_Toc484353539)

[Capitolul 2: Dezvoltarea aplicaţiei server 7](#_Toc484353540)

[2.1. Arhitectura generală a aplicaţiei 7](#_Toc484353541)

[Modelarea datelor 7](#_Toc484353542)

[Protocoale de comunicare client – server 7](#_Toc484353543)

[Interfața cu utilizatorul 7](#_Toc484353544)

[Implementare 7](#_Toc484353545)

[Manual de utilizare 7](#_Toc484353546)

[Referințe 7](#_Toc484353547)

# Introducere

## Motivaţie

În această lucrare voi propune o soluţie pentru dezvoltarea unei aplicaţii web, atât pentru client cât şi pentru server, care să ofere informaţii utilizatorilor despre cărţi pe care doresc să le împrumute şi nu le deţin sau nu le pot procura de la o bibliotecă sau librărie.

De asemenea, consider că ar putea fi considerată şi o aplicaţie socială deoarece utilizatorii vor avea posibilitatea de a cunoaşte persoane noi şi de a comunica cu acestea prin intermediul chat-ului din interiorul aplicaţiei.

## Context

Dat fiind faptul că, în momentul în care cineva doreşte să împrumute o carte are nevoie de permis la o bibliotecă, permis care nu se oferă în orice condiţii, spre exemplu, anumite biblioteci sunt doar universitare; oferă cărţi spre împrumut doar studenţilor/profesorilor sau anumite cărţi le poţi citi doar la sala de lectură. La o căutare pe google se poate observa că există destul de puţine asemenea aplicaţii, cea mai cunoscută fiind Bookster (o bibliotecă pentru companii din România).

Conform ultimelor statistici oficiale[[1]](#footnote-1) din anul 2011 interesul pentru lectură este destul de scăzut. De asemenea, datorită numărului continuu în creştere de utilizatori în mediul online[[2]](#footnote-2) consider că o asemenea aplicaţie online ar putea fi benefică.

## Cerinţe funcţionale

Pentru a avea un impact pozitiv dar şi pentru eventualitatea intrării pe piaţă a aplicaţiei, ea va trebui să respecte următoarele cerinţe funcţionale:

* Utilizatorul va avea posibilitatea de a-şi crea un cont nou
* Utilizatorul va putea folosi ca şi metode de autentificare Google, Facebook sau contul creat la inregistrare
* Utilizatorul va putea să caute cartea dorită în funcţie de anumite opţiuni: titlu, numele autorului, anul publicării cărţii
* Utilizatorul va putea să-şi editeze informaţiile personale
* Utilizatorul va putea să-şi managerieze cărţile pe care le deţine
* Utilizatorul va fi putea fi contactat prin email sau prin intermediul chat-ului
* Utilizatorul va putea să vadă ce utilizatori mai sunt prezenţi pe site şi cărţile pe care aceştia le deţin
* Utilizatorul va avea posibilitatea de a se loga/deloga

## Abordare tehnică

Baza de date

Pentru baza de date am ales soluţia oferită de cei de la Microsoft, şi anume, Microsoft SQL Server. Este o bază de date relaţională, care are capacitatea de stocare şi expunere a datelor pe baza unor cereri primite de la solicitant şi poate fi rulată pe acelaşi computer sau pe un computer din reţea.

### Backend

Pentru partea de backend voi folosit C#, un limbaj de programare functional, generic, orientat obiect, imperativ, declarativ. De asemenea, această componentă va face legătura între baza de date şi frontend. Tot aici se vor transmite notificările şi mesajele de la un utilizator către un altul.

### Frontend

La nivel de frontend se vor folosi AngularJS, framework de Javascript (limbaj de nivel înalt şi interpretat), care foloseşte desing pattern-ul MVC. De asemenea, AngularJS oferă posibilitatea de a crea elemente HTML proprii, altele decât cele specifice limbajului. Structura de bază a acestui framework este bazată pe servicii, controllere şi directive.

De asemenea, pentru a face partea vizuală cât mai plăcută pentru utilizatorul final se va folosi CSS, un standard pentru formatarea elementelor unui document HTML. Stilurile se pot atașa elementelor HTML prin intermediul unor fișiere externe sau în cadrul documentului prin elementul <style> și/sau atributul style. Cu alte cuvinte, CSS este canditatul ideal pentru partea de web design.

Pentru crearea efectivă a paginilor se va folosi HTML. Este un [limbaj de marcare](https://ro.wikipedia.org/wiki/Limbaj_de_marcare) utilizat pentru crearea paginilor web ce pot fi afișate într-un [browser](https://ro.wikipedia.org/wiki/Browser) (sau navigator). Scopul HTML este mai degrabă prezentarea informațiilor – paragrafe, fonturi, tabele ș.a.m.d. – decât descrierea semanticii documentului. Specificațiile HTML sunt dictate de [World Wide Web Consortium](https://ro.wikipedia.org/wiki/World_Wide_Web_Consortium) (W3C).

### Framework-uri

#### Entity Framework Core

Este o tehnologie folosită pentru a accesa o bază de date, o versiune independentă de platforma de dezvoltare, extensibilă şi mai uşoară comparativ cu Entity Framework. EF Core este un mapper de obiecte relaţionale (ORM, mai pe scurt), care le oferă dezvoltatorilor de .NET posibilitatea de a lucra cu o bază de date folosind obiecte .NET. Totodată, elimină nevoia de a scrie cod pentru a accesa baza de date, cod pe care în mod normal un programator oricum ar fi nevoit să-l scrie. EF Core are şi suport pentru mai multe motoare de baze de date.

#### Bootstrap

Este un framework pentru web pe partea de frontend gratuit şi open-source, care ajută la realizarea design-ului pentru site-uri sau aplicaţii web. Conţine anumite şabloane de design - HTML şi CSS - pentru elementele vizibile în interfaţa utilizatorului dar şi opţional, extensii Javascript. Spre deosebire de multe alte framework-uri web, se concentrează doar pe dezvoltarea frontend-ului.

#### Selenium

Este un framework de testare software portabil folosit pentru aplicații de tip web. Oferă o unealtă pentru înregistrare/playback pentru teste fără a învăța un limbaj de testare specific. De asemenea, oferă posibilitatea de a scrie teste în diverse limbaje de programare cum ar fi: C#, Java, PHP, Python, Ruby ș.a. . Testele pot rula pe browserele modern. Dezvoltarea lor se poate realiza atât pe Windows cât și pe Linux sau OS X. Un alt avantaj este că este open-source și poate fi descărcat gratuit.

# Contribuţii

Lucrarea este structurată în patru capitole mari, fiecare din ele tratând, pe rând, procesul de dezvoltare din cele două perspective principale(client şi server). Se va oferi motivaţia pentru anumite decizii luate pe parcursul proceseului de dezvoltare dar şi detalii relativ la implementare sau eventuale diferenţe care există comparând elemente similare. De asemenea, se vor oferi detalii despre asigurarea calităţii produsului şi a funcţionalităţii pe termen scurt şi lung. Cele patru capitole sunt enumerate mai jos:

* Capitolul 1: Dezvoltarea aplicației client
* Capitolul 2: Dezvoltarea aplicației server
* Capitolul 3: Baza de date
* Capitolul 4: Testarea funcţionalităţilor şi asigurarea calităţii

În cadrul dezvoltării aplicaţiei client s-au avut în vedere aspecte precum:

* Să fie user-friendly şi intuitivă
* Să fie rapidă
* Să fie scalabilă în funcţie de dispozitivul folosit
* Utilizarea de design-pattern-uri pentru separarea responsabilităţilor

Pe partea de server s-a avut în vedere ca aplicaţia să facă faţă unui volum consistent de utilizatori, să fie sigură și să păstreze confidențialitatea informațiilor despre utilizatori sau cărți:

* Folosirea autentificării și autorizării
* Cererile procesate asincron sau sincron
* Utilizarea principiilor SOLID
* Utilizarea de design-pattern-uri pentru separarea responsabilităţilor

În ceea ce priveşte baza de date s-a dorit a fi o bază de date de dimensiuni reduse dar robustă:

* Legăturile între tabele folosind chei străine
* Utilizarea de chei primare compuse din alte chei străine

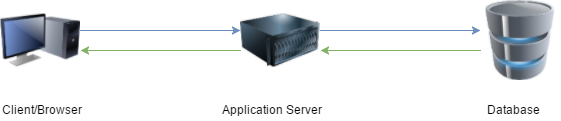
Testarea funţionalităţii şi asigurarea calităţii are ca scopuri esenţiale asigurarea faptului că aplicaţia respectă cerinţele funcţionale şi posibilitatea de dezvoltare, îmbunătăţire, aspecte ce pot fi aduse pe viitor:

* Crearea de unit teste pentru fiecare modul şi layer în parte
* Crearea de teste de integrare pentru verificarea funcţionalităţii modulelor împreună
* Crearea de teste automate pentru verificarea funcţionalităţii din punctul de vedere al utilizatorului

# Proiectare

## Arhitectura soluției

Arhitectura generală a soluţiei(aplicaţiei) nu este diferită faţă de cea a unei aplicaţii web obişnuite. Ea cuprinde trei părţi mari şi importante: clientul, serverul şi baza de date. Comunicarea generală se face după cum urmează: clientul trimite o cerere către server, acesta din urmă o evaluează. Dacă este cazul, se trimite o cerere către baza de date şi se aşteaptă un răspuns, altfel serverul va evalua cererea primită de la client şi va pregăti un răspuns. Indiferent de scenariu, după pregătirea răspunsului de către server, acesta va trimite înapoi către client acest răspuns, iar pe partea clientului vor apărea modificările necesare. Următoarea diagramă ilustrează arhitectura generală a aplicaţiei:

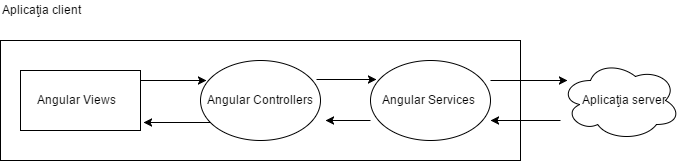


### Capitolul 1: Dezvoltarea aplicaţiei client

Aplicaţia client are ca scop facilitarea procesului de comunicare, de obţinere şi oferire de informaţii între utilizatori prin intermediul unei conexiuni la Internet. Aplicaţia a fost dezvoltată cu ajutorul editorului pus la dispoziţie de cei de la Microsoft – Visual Studio 2017 Enterprise. Pe parcursul dezvoltării s-a avut în vedere utilizarea tehnicilor necesare pentru obţinerea unei aplicaţii de tip SPA[[3]](#footnote-3) astfel încât utilizatorul să obţină rezultate mai rapide afişate în pagină iar aplicaţia să fie utilizabilă în condiţii reale, pe multiple dispozitive. Tehnicile folosite vor fi descrise, pe larg, în subcapitolele ce vor urma.

#### Arhitectura generală a aplicației

Arhitectura aplicaţiei client este puţin diferită de cea a unei aplicaţii client de tip web obişnuite. Astfel, cererile utilizatorului vor fi preluate din partea de view (fie că este vorba de butoane, câmpuri de input, hyperlink-uri) şi vor fi transmise către controllerul Angular specific (fiecare view, în parte, are asignat câte un controller care se ocupă de administrarea primului menţionat). De aici, controllerul Angular preia cererea şi dacă este necesar o trimite mai departe către un serviciu Angular, urmând apoi să aştepte un răspuns. Acesta, în cele din urmă, va fi delegatul care va trimite cererea către aplicaţia server. Serviciul va aştepta un răspuns de la server după care va întoarce rezultatele necesare către controller, iar acesta, la rândul său, va transmite modificările necesare către view. Această aplicaţie trebuie să fie capabilă să ofere utilizatorului o experienţă plăcută de utilizare, să fie uşor de folosit şi să notifice utilizatorul în legătură cu mesaje noi. Diagrama aplicaţiei client poate fi văzută mai jos, după cum urmează:



#### Angular Views

Utilizatorul va avea contact direct cu partea de Angular Views a aplicaţiei client. Deşi ele sunt menţionate la plural, component principal este reprezentată de un view general, similar unei pagini de home. Diferenţa esenţială faţă de aplicaţiile web clasice vine acum în discuţie. Deoarece Angular permite dezvoltarea de aplicaţii de tip SPA3, asta înseamnă că nu vom încărca de fiecare data o întreagă pagină web ci doar o anumită porţiune, porţiune care este definite sub formă de view. De aceea avem mai multe view-uri în aplicaţie, însă scheletul este acelaşi, având astfel posibilitatea de a modifica doar anumite porţiuni ale view-ului principal şi renunţând astfel la clasica reîncărcare a tuturor elementelor de pe o pagină HTML standard.

Sintaxa pentru un asemenea view este similară cu cea a unei pagini HTML, cu menţiunea că unele elemente pe care le găsim într-o pagină standard devin redundante. Unele dintre acestea ar fi: <html></html>, <head></head>, <body></body>. Astfel, un view poate să înceapă foarte bine, fără nicio problemă cu elemente de tipul: <div></div>, <table></table>, <span></span> ş.a. Singura excepţie de la această abordare o reprezintă view-ul general, aşa numitul index.html. Această pagină arată în totalitate ca o pagină HTML autentică relativ la părţile de <head>, <body>, <script>. Aşadar, view-urile sunt cele pe care utilizatorul le va întâlni în aplicaţie şi va interacţiona cu aceasta prin intermediul lor.

#### Angular Controllers

A doua componentă a aplicaţiei client, extrem de importantă şi partea centrală din punctul meu de vedere, este reprezentată de controllerele Angular. Aceste componente sunt definite, de regulă, câte unul pe fiecare view. Acest fapt este realizat pentru a aplica un best practice asupra codului şi pentru respectarea principiului de separare a responsabilităţilor. Aşadar, fiecare view are asignat un controller care se ocupă de managementul view-ului respectiv.

Ceea ce face controller-ul atât de puternic este faptul că el este cel care se ocupă de modificările survenite în partea de view dar şi în partea de servicii. Astfel, el este cel care notifică view-ul în privinţa unor schimbări şi tot el este cel care trimite cereri către servicii, pe care le vom detalia în subcapitolul următor, pentru a primi datele cerute sau transmise de utilizator, prin intermediul view-ului.

De asemenea, controller-ul este puntea de legătură între view şi datele care trebuiesc transmise către aplicaţia server. Totodată, deoarece abordarea a fost aleasă în acest mod, câte un controller pe fiecare view, va duce la creşterea vitezei de execuţie şi rezolvare a cererilor, un aspect esenţial urmărit de aplicaţiile contemporane şi implicit, de aplicaţia de faţă.

#### Angular Services

Ultima componentă majoră şi esenţială a aplicaţiei client este reprezentată de serviciile Angular. Aceste servicii sunt obiecte ce pot fi înlocuite şi sunt legate împreună folosind dependency injection. Cu ajutorul lor se poate organiza şi partaja codul în aplicaţie. Serviciile se bazează pe două design pattern-uri: Lazy initialization şi Singleton.

Deoarece folosesc aceste design pattern-uri, serviciile Angular oferă diferite avantaje precum:

* Iniţializarea în momentul în care avem nevoie de anumite resurse
* Crearea unei singure instanţe de-a lungul rulării aplicaţiei
* Evitarea consumului inutil de resurse
* Creşterea performanţelor datorită consumului mai redus de resurse

De asemenea, serviciile Angular sunt cele care fac cererile către API-ul aplicaţiei, un API care oferă resursele necesare aplicaţiei client. Această componentă va fi detaliată mai pe larg în capitolul dedicate aplicaţiei server. Aşadar, serviciile sunt cele care fac legătura între aplicaţia server şi cererile primite de la utilizator – pot fi considerate, pe bună dreptate, legătura între cele două aplicaţii.

### Capitolul 2: Dezvoltarea aplicaţiei server

#### 2.1. Arhitectura generală a aplicaţiei

## Modelarea datelor

## Protocoale de comunicare client – server

## Interfața cu utilizatorul

# Implementare

# Manual de utilizare

# Referințe

1. Obiceiuri lectură - http://www.ires.com.ro/articol/172/obiceiurile-de-lectura-ale-romanilor [↑](#footnote-ref-1)
2. Navigare pe Internet - http://ec.europa.eu/eurostat/statistics-explained/index.php/Information\_society\_statistics\_-\_households\_and\_individuals/ro [↑](#footnote-ref-2)
3. SPA(Single-page application) – aplicaţie de tip web sau site web al cărui conţinut să poate fi mapat pe o singură pagină, urmărindu-se astfel simularea unei aplicaţii desktop ca şi experienţă a utilizatorului. Într-o asemenea aplicaţie, fie tot codul necesar – HTML, Javascript, CSS – este primit la o singură încărcare de pagină, fie anumite resurse sunt încărcate în mod dynamic şi adăugate în pagină, ca răspuns la acţiunile utilizatorului. [↑](#footnote-ref-3)