

UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IAȘI

FACULTATEA DE INFORMATICĂ



LUCRARE DE LICENȚĂ

Memory Saver

propusă de

Marian Gurau

Sesiunea: *Februarie, 2018*

Coordonator științific

Lect. Drd. Florin Olariu

UNIVERSITATEA “ALEXANDRU IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

Memory Saver

Marian Gurau

Sesiunea: *Februarie, 2018*

Coordonator științific

Lect. Drd. Florin Olariu

DECLARAȚIE PRIVIND ORIGINALITATE ȘI RESPECTAREA

DREPTURILOR DE AUTOR

Prin prezenta declar că Lucrarea de licență cu titlul „*Memory Saver*” este scrisă de mine și nu a mai fost prezentată niciodată la o altă facultate sau instituție de învățământ superior din țară sau din străinătate. De asemenea, declar că toate sursele utilizate, inclusive cele preluate de pe Internet, sunt indicate în lucrare, cu respectarea regulilor de evitare a plagiatului:

- Toate fragmentele de text reproduse exact, chiar și în traducere proprie din altă limbă, sunt scrise între ghilimele și dețin referința precisă a sursei;
- Reformularea în cuvinte proprii a textelor scrise de către alți autori deține referința precisă;
- Codul sursă, imaginile etc. preluate din proiecte *open-source* sau alte surse sunt utilizate cu respectarea drepturilor de autor și dețin referințe precise;
- Rezumarea ideilor altor autori precizează referința precisă la textul original.

Iași, *data*

Absolvent *Marian Gurau*

(semnătura în original)

Contents

| | |
|---------------------------------|----|
| 1. Introducere | 6 |
| 1.1 Motivația..... | 6 |
| 1.2 Descrierea Aplicației..... | 6 |
| 1.3 Contribuții..... | 7 |
| 2 Tehnologii Folosite | 9 |
| 2.1 IDE..... | 9 |
| 2.2 Team Foundation Server..... | 9 |
| 2.3 Framework | 9 |
| 2.4 Baze de date | 10 |
| 2.5 Angular 5 | 11 |
| 2.6 VisualStudio Code | 11 |
| 2.7 ORM | 12 |
| 2.8 NuGet | 12 |
| 2.9 Arhitectura..... | 12 |
| 3 Arhitectura Aplicației..... | 14 |
| 3.1 Aplicația Backend..... | 14 |
| 3.1.1 Domain..... | 15 |
| 3.1.2 Data | 15 |
| 3.1.3 Infrastructure..... | 16 |
| 3.1.4 Presentation..... | 17 |
| 3.2 Aplicatia Frontend | 17 |
| 3.2.1 Modulul app | 17 |
| 3.2.2 Modulul shared..... | 18 |
| 3.2.3 Modulul chest..... | 18 |

| | |
|---------------------------------|----|
| 4 Detalii de implementare | 19 |
| 4.1 Aplicatia Backend..... | 19 |
| 4.1.1 Domeniul..... | 19 |
| 4.1.2 Data | 25 |
| 4.1.3 Infrastructura..... | 28 |
| 4.1.4 Presentation..... | 31 |
| 4.2 Aplicatia Frontend | 33 |
| 4.2.1 App | 34 |
| 4.2.2 Shared | 39 |
| 4.2.3 Chest | 42 |
| 5. Îmbunătățiri..... | 46 |
| 6. Concluzii..... | 48 |
| 7. Bibliografie..... | 49 |

1. Introducere

1.1 Motivația

Într-o lume în care principala formă de socializare sunt rețelele sociale iar toate cunoștințele tale sunt împărțite pe numeroasele platforme de socializare, sarcina de ați manageria toate conturile cât și conținutul care se află pe acestea poate deveni o sarcina care consumă foarte mult timp, timp care ar putea fi folosit în alte scopuri.

De asemenea, de multe ori uităm ce lucruri au fost sau nu postate, cât și unde au fost acestea publicate pentru ca toți să le vadă, fiind astfel pierdute cu trecerea timpului deoarece popularitatea unei anumite rețele scade iar persoanele relevante pentru noi se mută pe o altă platformă mai nouă, care oferă lucruri noi, amintirile noastre devenind astfel pierdute.

Acestea fiind spuse, scopul aplicației Memory Saver este să îi ofere unui utilizator posibilitatea de a-și aduna într-un singur loc toate amintirile sale, sub orice formă ar fi acestea cât și ai pune la îndemână un loc din care poate manageria ce și unde este postat.

1.2 Descrierea aplicației

Memory Saver se bazează pe API-urile puse la dispoziție de către rețelele de socializare pentru a interacționa cu acestea, fiind o soluție scalabilă, deoarece atunci când o nouă platformă apare, va fi adăugat un nou API care se va ocupa de interacțiunea dintre cele două aplicații.

Una dintre funcționalitățile aplicației va fi faptul că un utilizator își va putea salva fișierele de pe diversele platforme cât și creare de containere în care pot fi încărcate fișiere locale cât și cele luate de pe rețelele sociale, dar și posibilitatea de customizare ale acestor containere, cu nume, descriere și thumbnail alese de către utilizator.

O altă funcționalitate a aplicației este dată de faptul că un utilizator poate alege oricând să posteze un fișier pe oricare platforma care suportă acel tip de fișier, împreună cu o descriere aleasă de el. De asemenea își poate modifica sau șterge postări în măsură în care API-ul platformei permite lucrul acesta.

De asemenea un utilizator poate seta ca un anumit container să fie public sau privat, iar containerele publice vor putea fi observate doar de către alți utilizatori, în măsură în care aceștia au permisiunea de a o face. Permisuniunea de a vedea containerele publice ale unui utilizator se va baza pe un sistem de prietenie, putând astfel vedea containerele publice doar dacă ești prieten cu persoană respectivă.

1.3 Contribuții

Prin acesta aplicație tind să pun la dispoziția utilizatorilor o modalitate prin care aceștia își pot manageria conturile de pe platformele de socializare salvând astfel timp, dar în același timp și o modalitate de ași păstra într-un loc toate amintirile și fișierele importante, indiferent de formatul acestora, lucru care a fost până acum limitat.

Îmbinând un framework robust suportat de cei de la Microsoft (.NET Core) pe parte de backend dar cu un framework de WEB cu o comunitate imensă în spate și dezvoltat de o altă companie majoră în spațiul IT, Google, pe partea de frontend, care ne oferă atât scalabilitate cât și beneficiile date de SPA¹-uri, precum performanță crescută, în momentul navigării încărcându-se doar o parte din întreagă pagină. De asemenea, de multe ori într-o SPA, sunt transmise doar date pe rețea, nu o întreagă pagină. Un alt avantaj al SPA-urilor îl reprezintă partea de deployment în producție, acestea fiind foarte ușor instalate pe orice tip de server, chiar și separate de parte de backend, deoarece comunicarea cu acesta se face prin apeluri API, având nevoie doar de adresa la care se află acesta.

¹ SPA: Singel Page Aplication

Prin această aplicație doresc să pun la dispoziția utilizatorilor o unealtă cu care aceștia să își poată manageria conturile de pe platformele sociale. Prin utilizarea aplicației puse la dispoziția lor, aceștia vor avea următoarele beneficii :

- Timp salvat – putând să facă aceeași operațiune pe mai multe platforme o dată
- Modalitatea de a-și alege conținutul prezentat pe fiecare platforma în parte
- Salvarea fișierelor într-un loc central, fără a mai fi nevoie că utilizatorul să își caute toate conturile pentru acel fișier pierdut.

2 Tehnologii folosite

2.1 IDE

Ca IDE am folosit Visual Studio 2017 Community deoarece este foarte flexibil dar oferă și support pentru toate tehnologiile pe care le voi folosi, cât și pentru utilități third-party care vin în ajutorul procesului de dezvoltare al unei aplicații.

2.2 Team Foundation Server

Pentru repository-ul de cod am ales să folosesc soluția oferită de cei de la Microsoft și anume TFS.

Această platform este disponibilă atât online, rulând pe serverele celor de la Microsoft cât și local putând fi instalată pe propriile servere. Funcționalitatea este vândută pe baza de subscripții lunare, prețul variind în funcție de cât de multe opțiuni în baza de funcționalități ai ales. În cazul meu am ales să folosesc o subscripție online gratuită.

Din multitudinea de funcționalități puse la dispoziție de TFS cele mai relevante pentru această lucrare au fost:

- Accesul la un Backlog
- Posibilitatea de a crea o echipă
- Subversionarea codului

2.3 Framework

Ca framework am ales să folosesc .NET Core 2.0, acesta fiind o soluție nouă și foarte modulară, dar și una care permite deployment-ul aplicațiilor oriunde, indiferent de sistemul de operare sau platform pe care dorești să pui soluția. De asemenea, poți folosi librării third-party, ceea ce face dezvoltarea de aplicații scalabile și sigure mult mai ușor și fezabil.



Figura 1 - Structura .NET Core framework²

2.4 Baze de date

Pentru managementul bazelor de date am ales să folosesc SQL Server 2016 Express Edition, acesta punând la dispoziția dezvoltatorilor o modalitate de management al bazelor de date relaționale, dar oferă și securitate avansată și performanță ridicată.

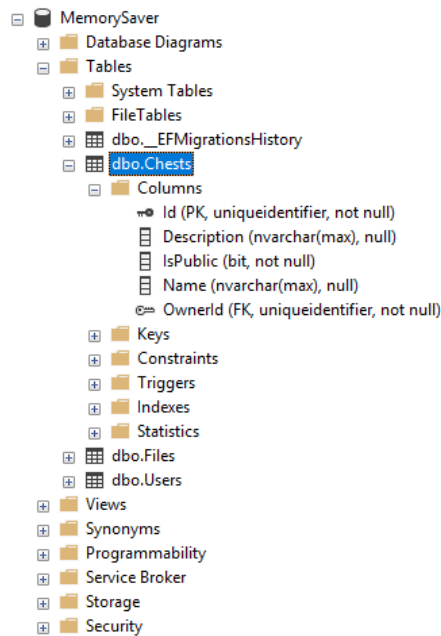


Figura 2 - Meniul SQL Server Manager³

² Sursa: <https://www.slideshare.net/aminmesbahi/net-core-20-aspnet-core-20-entity-framework-core-20-overview>

³ Sursa: SQL Server Management Sistem

2.5 Angular 5

Pentru partea de front-end am ales Angular 5, framework-ul oferind dezvoltare cross-platform cât și performanță ridicată, fiind bazat pe ultimele versiuni de ECMA script, astfel putându-se folosi ultimele tehnologii împreună cu HTML5 și CSS pentru a obține o aplicație robustă.

De asemenea, framework-ul este bazat pe componente și module, oferindu-ți posibilitatea de ați modulariza aplicația oricât de mult dorești fără prea multe bătăi de cap, rezultând astfel și o mai bună reutilizare a codului.

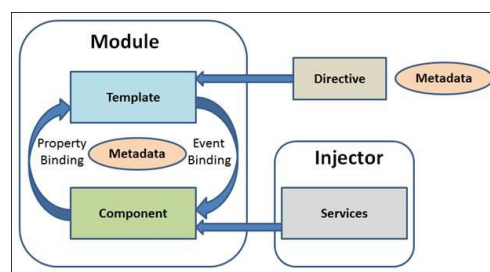


Figura 3 - Structura componentelor Angular⁴

2.6 VisualStudio Code

Pentru a face development pentru partea de front end am ales să folosesc IDE-ul lightweight oferit de cei de la Microsoft deoarece oferă o performanță crescută când vine vorba dezvoltarea de aplicații front end.

⁴ Sursa: <http://www.c-sharpcorner.com/article/basic-architecture-of-angular-2-applications/>

2.7 ORM

Pentru a relaționa întrebaza de date și aplicație am ales Entity Framework Core, acesta fiind compatibil cu .NET Core 2.0 dar și fiind o opțiune robustă și în continuă expansiune, care beneficiază de suport.



Figura 4 - Logo-ul EF Core⁵

2.8 NuGet

NuGet este o platform Microsoft prin care se face distribuirea publică de librării prin pachete DLL. Librăriile NuGet folosesc un format specific de împachetare și sunt administrate de Visual Studio prin intermediul utilitarului NuGet Package Manager. Orice utilizator își poate crea pachete NuGet, publice sau private iar prin intermediul Package Manager le poate instala în orice proiecte are nevoie de ele.

2.9 Arhitectura

Că arhitectură am ales să folosim Onion, această având ca scop rezolvarea unor probleme ale arhitecturilor tradiționale precum **coupling** și **separation of concerns**.

⁵ Sursa: <https://ardalis.com/encapsulated-collections-in-entity-framework-core>

Această are un centru care conține obiectele domeniului tău, apoi în jurul acestuia straturi care aduc noi funcționalități, de aici și numele acestei arhitecturi, iar fiecare strat depinzând doar de straturile inferioare acestuia.

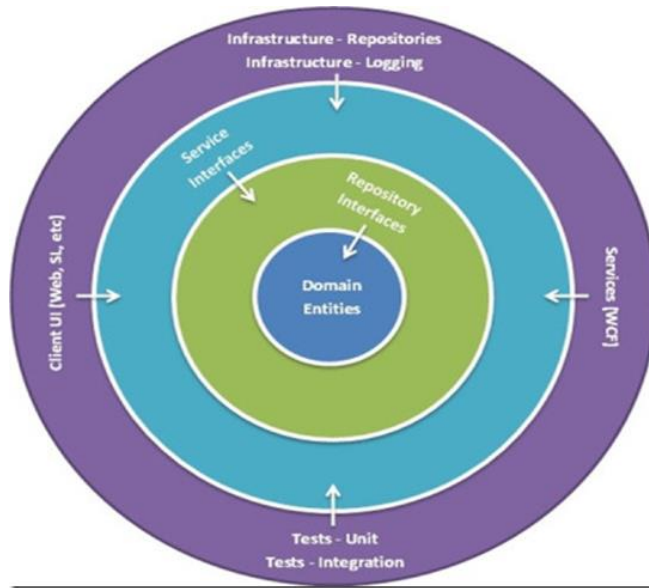


Figura 5 - Structura arhitecturii ONION⁶

⁶ Sursa: <http://blog.thedigitalgroup.com/chetanv/2015/07/06/understanding-onion-architecture/>

3 Arhitectura aplicației

Am ales că aplicația să fie structurată pe două module mari: backend și frontend sau server și client. Această abordare ajută cu separation of concerns. La rândul lor, cele două module sunt împărțite în submodule, fiecare ocupându-se cu o parte funcțională a aplicației. Un alt motiv pentru care am ales această abordare este faptul că oferă flexibilitate când vine vorba de upgrade-uri viitoare, fiind necesară schimbarea unei singure bucăți din aplicație, nu rescrierea totală. Pentru a evita confuzia, ne vom referi la cele două module că fiind aplicații.

3.1 Aplicația backend

În dezvoltarea părții de backend a aplicației am ales să folosesc ca pattern architectural Onion. Acest patern se bazează pe structurarea aplicației pe nivele multiple, fiecare nivel depinzând doar de nivelele inferioare acestuia și având un rol bine definit pe care trebuie să îl îndeplinească.

Am ajuns astfel ca modulul de backend să fie împărțit în următoarele nivele:

- Domain
- Data
- Infrastructure
- Presentation

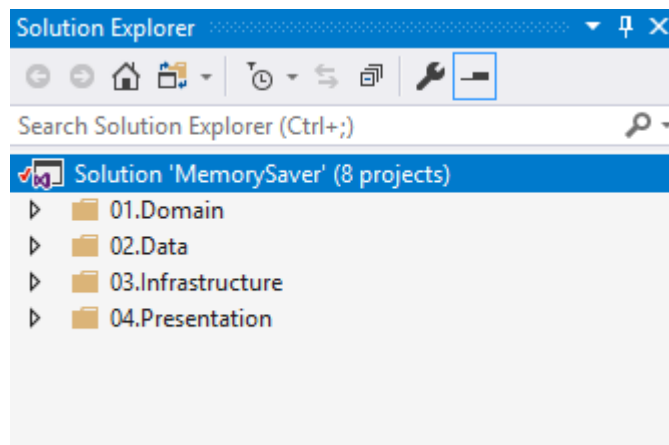


Figura 6 - Nivelele așa cum sunt prezente în aplicație

3.1.1 Domain

Domeniul are ca scop stocarea entităților aplicației cât și a interfețelor acestora însă este limitat doar la atât. Nu există logică sau acces de resurse în acest nivel. La rândul lui, acest nivel este împărțit și el în trei părți:

- Entități – se ocupă cu declararea entităților
- Interfețe – organizează și tochează interfețele folosite în aplicație
- Contracte de servicii – se ocupă cu stocarea interfețelor folosite de servicii

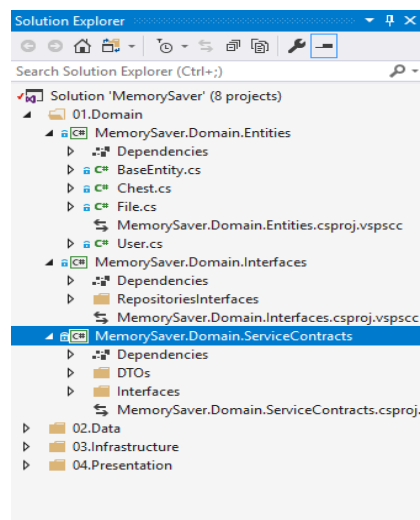


Figura 7 - Structura nivelului Domain

3.1.2 Data

Acesta este nivelul care se ocupă cu accesul la baza de date și de asemenea conține și migrațiile survenite bazei de date denumite migrări. Accesul la baza de date se face prin intermediul Entity Framework și a repository-urilor, acestea implementând interfețele definite în domeniu. În acest nivel se accesează, crează, modifică și șterge date, nu trebuie să conțină logică, pentru asta există un alt nivel.

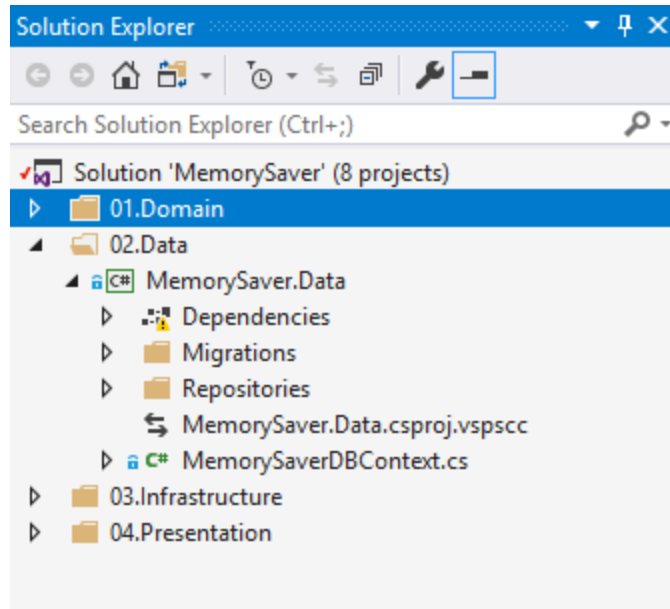


Figura 8 - Structura nivelului Data

3.1.3 Infrastructure

În acest nivel este încapsulată toată logică din spatele aplicației. Acest nivel se ocupă atât de furnizarea corectă a datelor și interacțiunea dintre componente dar și de configurarea diverselor librării third-party cum ar fi AutoFac și AutoMapper. O altă funcționalitate a nivelului Infrastructure este configurarea tokenului JWT (Jason Web Token), acesta fiind folosit la autorizarea cererilor făcute de către client.

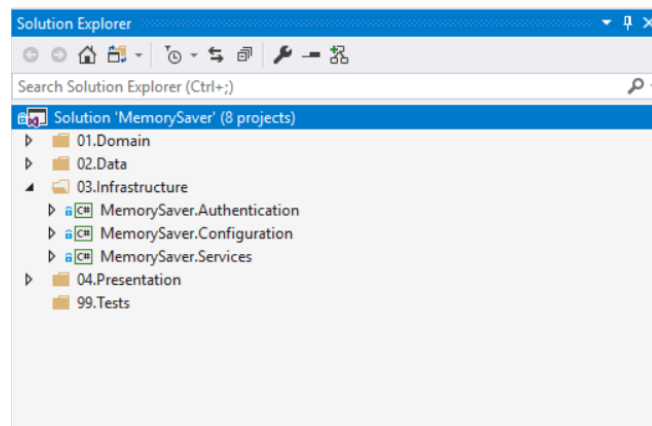


Figura 9 - Structura nivelului Infrastructure

3.1.4 Presentation

Presentation este nivelul în care se află toate end-point-urile aplicației. Este rolul acestui nivel să facă legătură între Frontend și restul aplicației, răspunzând la apelurile venite din Frontend și apelând la rândul lui pipeline-urile necesare pentru a furniza datele cerute.

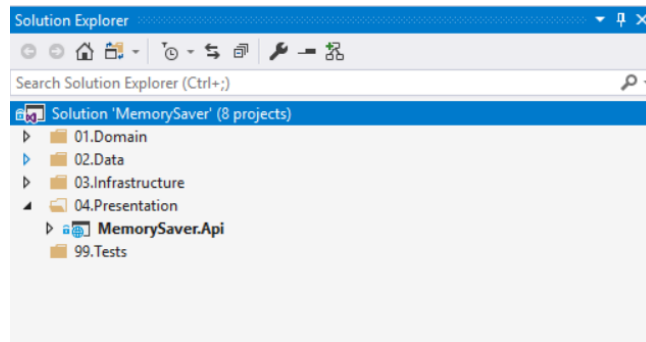


Figura 10 - Nivelul Presentation

3.2 Aplicația frontend

Aplicația Frontend este partea care este văzută de utilizator. Acesta este modulul cu care end-user-ul interacționează și prin care acesta face requesturile de care are nevoie. La rândul ei, aplicația Frontend este împărțit în 3 module mari, acestea fiind :

- app
- shared
- chest

3.2.1 Modulul app

Modulul app este rădăcina aplicației frontend. Acesta încapsulează componentele de baza cum ar fi componentă Login și cea Register, dar și celelalte module, și anume shared și

chest. De asemenea, are că responsabilitate unirea și sincronizarea celorlalte module, și facilitarea comunicării între ele și componentele de baza.

3.2.2 Modulul shared

Modulul shared are că rol, după cum sugerează și numele acestuia, încapsularea tuturor componentelor, serviciilor, modelelor folosite în aplicația de frontend. Aici sunt stocate toate părțile aplicației frontend care nu sunt specifice unui singur modul.

3.2.3 Modulul chest

În modulul chest al aplicație Frontend este locul în care un utilizator își va petrece majoritatea timpului, acesta având responsabilitatea de a manageria toate chesturile unui utilizator cât și fișierele care se află în fiecare.

4 Detalii de implementare

În continuare vom trece prin fiecare parte a celor două aplicații și voi prezenta modul în care aplicația funcționează și anumite abordări pe care le-am ales și de ce am făcut aceste alegeri în detrimentul altora.

4.1 Aplicația backend

4.1.1 Domeniul

După cum am precizat anterior, acest nivel are ca responsabilitate declararea entităților cu care aplicația lucrează cât și a contractelor prin care această face conexiunea între diversele părți ale aplicației.

Prin urmare am despărțit acest nivel în 3 categorii, fiecare având ca rol stocarea și declararea de entități sau interfețe.

a) Domain.Entities – declara cele 3 entități care vor fi folosite în întreaga aplicație. Aceste 3 entități extind o entitate de baza, lucru care ne permite declararea unui repository generic. Pornind de la entitatea de baza, acestea sunt, precum urmează :

- BaseEntity : Entitatea extinsă de orice altă entitate din aplicație. Are ca singură proprietate un id.

```
5 references | Gurau Marian, 119 days ago | 1 author, 1 change
public class BaseEntity
{
    8 references | Gurau Marian, 119 days ago | 1 author, 1 change | 0 exceptions
    public Guid Id { get; set; }
}
```

Figura 11 - Definirea entității de baza

- User : Entitatea care stocheaza informatiile unui user

```
13 references | Gurau Marian, 87 days ago | 1 author, 3 changes
public class User : BaseEntity
{
    6 references | Gurau Marian, 119 days ago | 1 author, 1 change | 0 exceptions
    public string Email { get; set; }

    5 references | Gurau Marian, 87 days ago | 1 author, 1 change | 0 exceptions
    public string FirstName { get; set; }

    5 references | Gurau Marian, 87 days ago | 1 author, 1 change | 0 exceptions
    public string LastName { get; set; }

    3 references | Gurau Marian, 119 days ago | 1 author, 1 change | 0 exceptions
    public string Password { get; set; }

    3 references | Gurau Marian, 114 days ago | 1 author, 1 change | 0 exceptions
    public IEnumerable<Chest> OwnedChests { get; set; }
}
```

Figura 12 - Definirea entității User

- Chest : Stocheaza si organizeaza informatiile necesare lucrului cu chest-uri

```
14 references | Gurau Marian, 3 days ago | 1 author, 4 changes
public class Chest : BaseEntity
{
    2 references | Gurau Marian, 119 days ago | 1 author, 1 change | 0 exceptions
    public Guid OwnerId { get; set; }

    2 references | Gurau Marian, 119 days ago | 1 author, 1 change | 0 exceptions
    public string Name { get; set; }

    2 references | Gurau Marian, 119 days ago | 1 author, 1 change | 0 exceptions
    public string Description { get; set; }

    2 references | Gurau Marian, 119 days ago | 1 author, 1 change | 0 exceptions
    public bool IsPublic { get; set; }

    1 reference | Gurau Marian, 72 days ago | 1 author, 1 change | 0 exceptions
    public User Owner { get; set; }

    3 references | Gurau Marian, 3 days ago | 1 author, 1 change | 0 exceptions
    public IEnumerable<File> FilesInChest { get; set; }
}
```

Figura 13 - Definirea entității chest

- File : Stochează informații despre un fișier. Proprietățile FacebookId și VimeoId sunt cele care stochează id-urile asignate de către cele 2 rețele sociale integrate în aplicație.

```
8 references | Gurau Marian, 3 days ago | 1 author, 2 changes
public class File : BaseEntity
{
    2 references | Gurau Marian, 3 days ago | 1 author, 1 change | 0 exceptions
    public string Description { get; set; }

    1 reference | Gurau Marian, 3 days ago | 1 author, 1 change | 0 exceptions
    public string FileName { get; set; }

    3 references | Gurau Marian, 3 days ago | 1 author, 1 change | 0 exceptions
    public string Type { get; set; }

    2 references | Gurau Marian, 3 days ago | 1 author, 1 change | 0 exceptions
    public string FacebookId { get; set; }

    0 references | Gurau Marian, 3 days ago | 1 author, 1 change | 0 exceptions
    public string VimeoId { get; set; }

    3 references | Gurau Marian, 72 days ago | 1 author, 1 change | 0 exceptions
    public Guid ChestId { get; set; }

    1 reference | Gurau Marian, 72 days ago | 1 author, 1 change | 0 exceptions
    public Chest Chest { get; set; }
}
```

Figura 14 - Definirea entității File

b) Domain.Interfaces – Aici sunt declarate interfețele repository-urilor, toate fiind extensii ale unei interfețe de baza, numită sugestiv IGenericRepository :

- IGenericRepository : Interfață pentru repository-ul de baza

```
5 references | Gurau Marian, 87 days ago | 1 author, 2 changes
public interface IGenericRepository<T>
    where T : BaseEntity
{
    5 references | Gurau Marian, 119 days ago | 1 author, 1 change | 0 exceptions
    void Add(T entity);

    2 references | Gurau Marian, 119 days ago | 1 author, 1 change | 0 exceptions
    T GetById(Guid id);

    1 reference | Gurau Marian, 119 days ago | 1 author, 1 change | 0 exceptions
    void Update(T entity);

    1 reference | Gurau Marian, 119 days ago | 1 author, 1 change | 0 exceptions
    void Delete(Guid id);

    5 references | Gurau Marian, 87 days ago | 1 author, 2 changes | 0 exceptions
    bool SaveChanges();
}
```

Figura 15 - Interfața repository-ului de baza

- IUserRepository : Repository-ul care se ocupă cu managementul user-ilor

```
4 references | Gurau Marian, 72 days ago | 1 author, 3 changes
public interface IUserRepository : IGenericRepository<User>
{
    3 references | Gurau Marian, 114 days ago | 1 author, 1 change | 0 exceptions
    User GetByEmail(string userEmail);

    2 references | Gurau Marian, 72 days ago | 1 author, 1 change | 0 exceptions
    IEnumerable<Chest> GetUserChests(Guid userId);
}
```

Figura 16 - Repository-ul pentru user

- IChestRepository : Managementul cuferelor

```
4 references | Gurau Marian, 3 days ago | 1 author, 2 changes
public interface IChestRepository : IGenericRepository<Chest>
{
    2 references | Gurau Marian, 3 days ago | 1 author, 1 change | 0 exceptions
    Chest GetByIdIncludeAll(Guid chestId);
}
```

Figura 17 - Repository-ul pentru cufere

- IFileRepository : Lucru cu fișiere

```
4 references
public interface IFileRepository : IGenericRepository<File>
{
}
```

Figura 18 - Repository-ul pentru fișiere

c) Domain.ServiceContracts – Locul în care sunt stocate interfetele serviciilor cât și DTO-urile folosite de acestea. La rândul lor, DTO-urile sunt despărțite în două categorii, și anume Request și Response, rolul fiecărei categorii fiind evident.

- IUserService: serviciul care se ocupă cu logică din spatele unui user

```
8 references | Gurau Marian, 3 days ago | 1 author, 4 changes
public interface IUserService
{
    2 references | Gurau Marian, 87 days ago | 1 author, 2 changes | 0 exceptions
    bool CreateUser(CreateUserRequestDTO newUser);

    2 references | Gurau Marian, 114 days ago | 1 author, 1 change | 0 exceptions
    LoginUserResponseDTO Login(LoginUserRequestDTO loginUser);

    2 references | Gurau Marian, 72 days ago | 1 author, 1 change | 0 exceptions
    IEnumerable<Chest> GetUserChests(Guid userId);

    2 references | Gurau Marian, 3 days ago | 1 author, 1 change | 0 exceptions
    LoginUserResponseDTO VerifySocialUser(LoginSocialUserRequestDTO socialUser);
}
```

Figura 19 - Contractul serviciului user

- IChestService: Are grija de logica legata de cufere

```
4 references | Gurau Marian, 3 days ago | 1 author, 2 changes
public interface IChestService
{
    2 references | Gurau Marian, 72 days ago | 1 author, 1 change | 0 exceptions
    bool CreateChest(CreateChestRequestDTO chestModel);

    2 references | Gurau Marian, 3 days ago | 1 author, 1 change | 0 exceptions
    GetChestResponseDTO GetChest(Guid chestId);

    1 reference | 0 changes | 0 authors, 0 changes | 0 exceptions
    bool EditChest(EditChestRequestDTO newChestInfo);

    1 reference | 0 changes | 0 authors, 0 changes | 0 exceptions
    bool DeleteChest(Guid chestId);
}
```

Figura 20 - Serviciul pentru cufere

- IFileService: serviciu responsabil cu logica fisierelor.

```
6 references
public interface IFileService
{
    1 reference | 0 exceptions
    bool UploadFile(byte[] file,
        string fileName,
        string description,
        string facebookId,
        string vimeoId,
        Guid chestId);

    2 references | 0 exceptions
    GetFileResponseDTO GetFile(Guid fileId);

    0 references | 0 exceptions
    bool DeleteFile(Guid fileId);
}
```

Figura 21 - Serviciul responsabil cu fişierele

- Request DTOs: sunt obiectele care încapsulează informațiile necesare unui request

```

4 references | Gurau Marian, 114 days ago | 1 author, 1 change
public class CreateUserRequestDTO
{
    1 reference | Gurau Marian, 114 days ago | 1 author, 1 change | 0 exceptions
    public string Email { get; set; }

    1 reference | Gurau Marian, 114 days ago | 1 author, 1 change | 0 exceptions
    public string FirstName { get; set; }

    1 reference | Gurau Marian, 114 days ago | 1 author, 1 change | 0 exceptions
    public string LastName { get; set; }

    1 reference | Gurau Marian, 114 days ago | 1 author, 1 change | 0 exceptions
    public string Password { get; set; }
}

```

Figura 22 - DTO pentru crearea de cufere

```

2 references
public class EditChestRequestDTO
{
    0 references | 0 exceptions
    public Guid Id { get; set; }

    0 references | 0 exceptions
    public string Name { get; set; }

    0 references | 0 exceptions
    public string Description { get; set; }

    0 references | 0 exceptions
    public bool IsPublic { get; set; }
}

```

Figure 23 - DTO pentru editare de cufere

- Response DTOs: obiectele care încapsulează informațiile cerute de un utilizator despre anumite entități ale aplicației

| | |
|---|---|
| <pre> 4 references Gurau Marian, 3 days ago 1 author, 1 change public class GetChestResponseDTO { 1 reference Gurau Marian, 3 days ago 1 author, 1 change 0 exceptions public string Name { get; set; } 1 reference Gurau Marian, 3 days ago 1 author, 1 change 0 exceptions public string Description { get; set; } 1 reference Gurau Marian, 3 days ago 1 author, 1 change 0 exceptions public bool IsPublic { get; set; } 1 reference Gurau Marian, 3 days ago 1 author, 1 change 0 exceptions public IEnumerable<File> FilesInChest { get; set; } } </pre> | <pre> 5 references Gurau Marian, 3 days ago 1 author, 1 change public class GetFileResponseDTO { 1 reference Gurau Marian, 3 days ago 1 author, 1 change 0 exceptions public byte[] FileBase64 { get; set; } 1 reference Gurau Marian, 3 days ago 1 author, 1 change 0 exceptions public string Description { get; set; } 1 reference Gurau Marian, 3 days ago 1 author, 1 change 0 exceptions public string FileType { get; set; } 1 reference Gurau Marian, 3 days ago 1 author, 1 change 0 exceptions public string FacebookId { get; set; } 0 references 0 changes 0 authors, 0 changes 0 exceptions public string VimeoId { get; set; } } </pre> |
|---|---|

Figura 24 – DTO-uri de raspuns


```

12 references | Gurau Marian, 72 days ago | 1 author, 2 changes
public class LoginUserResponseDTO
{
    4 references | Gurau Marian, 114 days ago | 1 author, 1 change | 0 exceptions
    public Guid Id { get; set; }

    4 references | Gurau Marian, 114 days ago | 1 author, 1 change | 0 exceptions
    public string FirstName { get; set; }

    4 references | Gurau Marian, 114 days ago | 1 author, 1 change | 0 exceptions
    public string LastName { get; set; }

    4 references | Gurau Marian, 72 days ago | 1 author, 1 change | 0 exceptions
    public string Email { get; set; }
}

```

4.1.2 Data

Nivelul Data este locul în care aplicația comunica cu baza de date. Alegând o abordare code first, schemă bazei de date este dată de un context care este construit pe baza entităților aplicației noastre, astfel încât fiecare dintre entități va avea un tabel, iar fiecare proprietate a entităților va fi asociată cu o coloană din tabel.

```

2 references | Gurau Marian, 119 days ago | 1 author, 1 change | 0 exceptions
public DbSet<User> Users { get; set; }

1 reference | Gurau Marian, 119 days ago | 1 author, 1 change | 0 exceptions
public DbSet<Chest> Chests { get; set; }

0 references | Gurau Marian, 3 days ago | 1 author, 1 change | 0 exceptions
public DbSet<File> Files { get; set; }

```

Figura 25 - Setarea tabelelor în EF Core

Un alt avantaj pe care îl prezintă această abordare sunt atributele de navigare, specificând în context în ce fel sunt entitățile înrudite, îi vom putea spune librăriei Entity Framework dacă vrem sau nu, la o anumită acțiune asupra resursei să ne aducă împreună cu entitatea și celelalte entități înrudite. În cazul în care o entitate este înrudită cu mai multe alte entități, putem alege care dintre acestea să fie aduse împreună cu entitatea cerută de noi.

```

0 references | Gurau Marian, 3 days ago | 1 author, 2 changes | 0 exceptions
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<User>()
        .HasMany(p => p.OwnedChests)
        .WithOne(t => t.Owner)
        .HasForeignKey(k => k.OwnerId)
        .IsRequired(true);

    modelBuilder.Entity<Chest>()
        .HasMany(p => p.FilesInChest)
        .WithOne(f => f.Chest)
        .HasForeignKey(k => k.ChestId)
        .IsRequired(true);
}

```

Figura 26 - Setarea relațiilor în EF Core

De asemenea, acest nivel este locul în care interfețele repository-urilor își au implementările concrete. La fel că în nivelul Domain, toate repository-urile extind repository-ul de baza, și anume GenericRepository. Faptul că toate entitățile extind BaseEntity face că acest lucru să se poată realiza fără prea mari bătăi de cap.

```

5 references | Gurau Marian, 87 days ago | 1 author, 3 changes
public class GenericRepository<T> : IGenericRepository<T>
    where T : BaseEntity
{
    private readonly MemorySaverDbContext context;
    private readonly DbSet<T> entityDbSet;

    3 references | Gurau Marian, 114 days ago | 1 author, 2 changes | 0 exceptions
    public GenericRepository(MemorySaverDbContext context)
    {
        this.context = context;
        entityDbSet = this.context.Set<T>();
    }

    5 references | Gurau Marian, 119 days ago | 1 author, 1 change | 0 exceptions
    public void Add(T entity)
    {
        entityDbSet.Add(entity);
    }

    1 reference | Gurau Marian, 119 days ago | 1 author, 1 change | 0 exceptions
    public void Delete(Guid id)
    {
        T entity = entityDbSet.FirstOrDefault(e => e.Id == id);

        if (entity != null)
        {
            entityDbSet.Remove(entity);
        }
    }
}

```

Figura 27 - Setarea repository-ului de bază

După cum putem observă, repository-ul generic nu face decât să atașeze tipul de entitate pe care îl vom folosi și contextul bazei de date în care vom face acțiunile necesare, iar restul procesului este realizat de Entity Framework.

Încă o responsabilitate cu care se ocupă acest nivel este managementul migrărilor către baza de date. După cum spuneam mai devreme, schemă bazei de date este dată de codul fiecărei entități și de relațiile care sunt declarate în context. Dacă în unul din cele două locuri apar modificări, înseamnă că și baza de date la rândul ei trebuie să se modifice.

Pentru a face această, Entity Framework pune la dispoziția developerilor mecanismul de migrare care atunci când este apelat, în urmă analizării codului și stării bazei de date, generează schimbările bazei de date corespunzătoare cu noile schimbări apărute în cod.

Migrările sunt realizate cu ajutorul ferestrei package manager console care este pusă la dispoziția utilizatorului de către VisualStudio.

Din moment ce suntem la nivelul responsabil cu accesul informațiilor în baza de date, este un moment oportun să aducem la cunoștință și schemă bazei de date pe care o folosim. După cum spuneam mai sus, această se bazează pe entitățile declarate în cod, iar librăria Entity Framework este cea care o generează. Acestea fiind spuse, vom obține o schemă destul de simplă, formată din trei tabele care au între ele relații one-to-many. Fiecare user putând avea unul sau mai multe cufare, și fiecare cufăr putând avea unul sau mai mult fișiere.

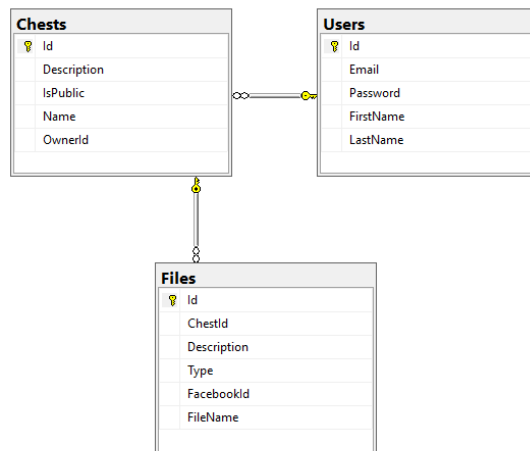


Figura 28 - Structura bazei de date⁷

4.1.3 Infrastructure

Nivelul infrastructurii este în mare parte, cel mai important. În interiorul acestui nivel se află toată logica aplicației, împreună cu configurările ce țin de librăriile third-party pe care le folosesc în aplicație, dar și configurările pentru token-ul JWT.

- a) MemorySaver.Authentication – este locul în care configurările token-ului JWT sunt realizate, iar comportamentul acestuia este de asemenea definit.
- b) MemorySaver.Configuration – în interiorul acestei părți din aplicație sunt configurate librăriile third-party precum Autofac și Automapper.

- Librăria Autofac este bazată pe înregistrarea unor reguli în care specificăm ce instanță a unei interfețe dorim să folosim atunci când această este folosită în alte părți ale aplicației. Pentru a face asta, este nevoie de unul sau mai multe fișiere de configurare. În cazul aplicației MemorySaver, avem 2 seturi de reguli, și anume RepositoriesModule care

⁷ Sursa: SQL Server Management Sistem

are ca scop rezolvarea cerințelor în cazul repository-urilor, și ServicesModule care la rândul lui se ocupă cu dependency injection pentru servicii.

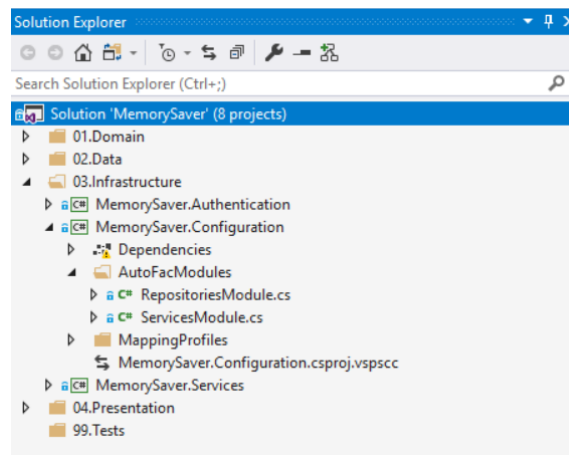


Figura 29 - Modulele Autofac

```
1 reference | Gurau Marian, 3 days ago | 1 author, 3 changes
public class RepositoriesModule : Module
{
    1 reference | Gurau Marian, 3 days ago | 1 author, 3 changes | 0 exceptions
    protected override void Load(ContainerBuilder builder)
    {
        builder.RegisterGeneric(typeof(GenericRepository<>)).As(typeof(IGenericRepository<>));
        builder.RegisterType<UserRepository>().As<IUserRepository>();
        builder.RegisterType<ChestRepository>().As<IChestRepository>();
        builder.RegisterType<FileRepository>().As<IFileRepository>();
    }
}
```

Figura 30 - Cum se realizeaza Dependency Injection prin autofac

- Librăria AutoMapper are la baza mapping profiles în care se realizează relațiile dintre obiectele între care se dorește mapparea de informații. Atât timp cât numele și tipurile proprietăților sunt identice, librăria AutoMapper știe să facă singură mapparea, însă dacă dorim să facem mapparea între două proprietăți cu nume diferite sau tipuri diferite, va trebui să ii specificăm care proprietăți trebuie mapate astfel și în ce formă. Am ales această librărie deoarece elimina necesitatea de a scrie mapparile de mână și în același timp ne ajută să ținem toate conversiile dintre obiectele cu care aplicația funcționează într-un singur loc, ajutând astfel la abilitatea de a menține aplicația.

```

1 reference | Gurau Marian, 87 days ago | 1 author, 2 changes
public class UserMappingProfile : Profile
{
    0 references | Gurau Marian, 87 days ago | 1 author, 2 changes | 0 exceptions
    public UserMappingProfile()
    {
        CreateMap<CreateUserRequestDTO, User>();
        CreateMap<User, LoginUserResponseDTO>();
    }
}

```

Figura 31 - Inregistrarea de mapari intre entitatea User si DTO-urile acestea

- c) MemorySaver.Services – aici se întâmplă toată magia din spatele aplicației, este locul în care toată logică este menținută. Acesta este nivelul în care datele și request-urile ajung pentru a fi prelucrate și locul în care toată aplicația se îmbină. Rolul acestui nivel este poate cel mai important, fără el neputându-se face legătură între restul nivelelor din aplicație într-o manieră cât mai în rând cu principiile OOP.

```

2 references | Gurau Marian, 3 days ago | 1 author, 5 changes
public class UserService : IUserService
{
    private readonly IUserRepository userRepository;
    private readonly IMapper mapper;

    0 references | Gurau Marian, 114 days ago | 1 author, 2 changes | 0 exceptions
    public UserService(IUserRepository userRepository, IMapper mapper)
    {
        Ensure.That(userRepository).NotNull();
        Ensure.That(mapper).NotNull();

        this.mapper = mapper;
        this.userRepository = userRepository;
    }

    2 references | Gurau Marian, 87 days ago | 1 author, 3 changes | 0 exceptions
    public bool CreateUser(CreateUserRequestDTO newUser)
    {
        User userToBeCreated = new User();
        mapper.Map<CreateUserRequestDTO, User>(newUser, userToBeCreated);
        userRepository.Add(userToBeCreated);
        return userRepository.SaveChanges();
    }
}

```

Figura 32 - Implementarea serviciului user

După cum putem vedea în exemplul anterior, toate nivelele inferioare, inclusive părți din nivelul actual sunt folosite în servicii, astfel realizându-se comunicarea între toate nivelele aplicației și formând un tot unitar, reprezentând miezul aplicației.

4.1.4 Presentation

Nivelul presentation este reprezentat de API – ul aplicației noastre. Acesta face posibilă comunicare aplicației Backend cu exteriorul, orice altă formă de comunicare fiind inexistentă. În dezvoltarea API-ului aplicației am urmat principiile CRUD și REST, obținând astfel un API restful, asta însemnând că toate requesturile către server sunt stateless, ori de câte ori un anumit request este efectuat, rezultatul va fi întotdeauna același.

În contextul API-ului, entitățile cu care aplicația lucrează nu sunt altceva decât niște resurse. Acestea fiind spuse, am separat accesul la fiecare dintre ele prin propriul sau controller, încercând pe cât de mult posibil să urmez principiile CRUD.

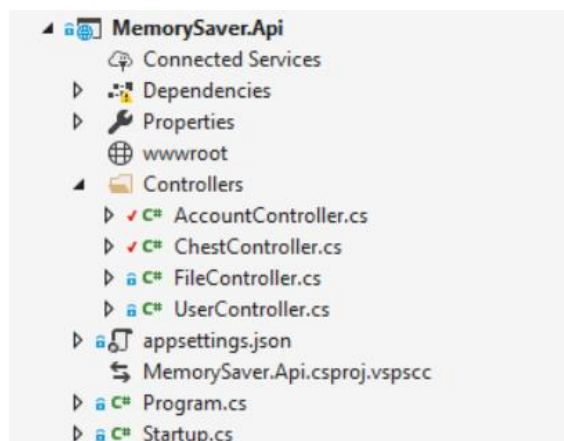


Figura 33 - Structura proiectului API

De asemenea, fiind punctul de început al aplicației, nivelul presentation împreună cu API-ul aferent sunt responsabile cu înregistrarea serviciilor și variabilelor folosite în toată aplicația. Din fericire, framework-ul ne pune la dispoziție mai multe mecanisme pentru a face lucrurile acestea, iar în cazul meu, am optat să folosesc appsettings.json pentru

înregistrarea variabilelor, cât și clasa Startup.cs, pentru a înregistra serviciile și librăriile third-party folosite.

```
"ConnectionStrings": {  
  "MemorySaver": "Data Source=.\SQLExpress;Initial Catalog=MemorySaver;"  
},  
  
"Whitelist": "http://localhost:4200",  
  
"Tokens": {  
  "Issuer": "MemorySaver",  
  "Key": "98c4d130-af60-416b-b172-35699b492a41"  
}
```

Figura 34 - Declararea variabilelor globale

```
0 references | Gurau Marian, 73 days ago | 1 author, 4 changes | 0 exceptions  
public void Configure(IApplicationBuilder app, IHostingEnvironment env)  
{  
    if (env.IsDevelopment())  
    {  
        app.UseDeveloperExceptionPage();  
    }  
  
    app.UseCors("SiteCorsPolicy");  
  
    app.UseAuthentication();  
  
    app.UseStaticFiles();  
  
    AddAuth(app);  
  
    app.UseMvc();  
}
```

Figura 35 - Înregistrarea serviciilor folosite de aplicație

După cum putem observă, AddAuth este o funcție pe care o apelăm în momemtul configurarii aplicației. Acesta se ocupă cu configurarea parametrilor de validare ai tokenului JWT dar și cu configurarea entry-pointurilor pentru aplicație atunci când un utilizator dorește să se autentifice. După cum urmează să și vedem, variabilele folosite la configurarea tokenului JWT sunt și ele de asemenea declarate de către appsettings.json.


```

var tokenValidationParameters = new TokenValidationParameters
{
    // The signing key must match!
    ValidateIssuerSigningKey = true,
    IssuerSigningKey = signingKey,

    // Validate the JWT Issuer (iss) claim
    ValidateIssuer = true,
    ValidIssuer = Configuration["Tokens:Issuer"],

    // Validate the JWT Audience (aud) claim
    ValidateAudience = true,
    ValidAudience = Configuration["Tokens:Audience"],

    // Validate the token expiry
    ValidateLifetime = true,

    // If you want to allow a certain amount of clock drift, set that here:
    ClockSkew = TimeSpan.Zero,
};

```

Figura 36 - Setarea validării token-ului JWT

```

1 reference | Gurau Marian, 4 days ago | 1 author, 2 changes | 0 exceptions
private void AddAuth(IApplicationBuilder app)
{
    appBuilder = app;

    secretKey = Configuration["Tokens:Key"];
    var signingKey = new SymmetricSecurityKey(Encoding.ASCII.GetBytes(secretKey));

    app.UseSimpleTokenProvider(new TokenProviderOptions
    {
        Path = "/api/login",
        Audience = Configuration["Tokens:Audience"],
        Issuer = Configuration["Tokens:Issuer"],
        SigningCredentials = new SigningCredentials(signingKey, SecurityAlgorithms.HmacSha256),
        IdentityResolver = GetIdentity,
    });

    app.UseSimpleTokenProvider(new TokenProviderOptions
    {
        Path = "/api/social-login",
        Audience = Configuration["Tokens:Audience"],
        Issuer = Configuration["Tokens:Issuer"],
        SigningCredentials = new SigningCredentials(signingKey, SecurityAlgorithms.HmacSha256),
        IdentityResolver = GetSocialIdentity,
    });
}

```

Figura 37 - Setarea punctelor de acces pentru generarea token-ului

Un avantaj al acestei abordări este faptul că toate variabilele și serviciile sunt definite și înregistrate într-un singur loc în aplicație, lucru care permite schimbarea delibrării, componenete sau valori ale variabilelor globale într-un timp mult mai scurt și într-o manieră mult mai sigură.

4.2 Aplicația frontend

Așa cum am precizat anterior, în dezvoltarea aplicației Frontend am folosit framework-ul javascript Angular 5. Acesta este axat foarte mult pe modularizarea unei

aplicații, totul bazându-se pe module și componente. O componentă este o bucată structurală cât de mică dorește utilizatorul, acesta având responsabilitatea de a-și organiza aplicația cât mai corect.

La rândul său, un modul este o colecție de componente și module, după caz, care fac parte în general din aceeași parte funcțională a unei aplicații.

Acestea fiind spuse, aplicația Frontend este despărțită pe trei module și anume :

- app : acesta este modulul de baza și rădăcina aplicației. El încapsulează toate celelalte module și componente și are grijă de sincronizarea dintre acestea.
- shared : după cum sugerează și numele acestuia, este modulul în care sunt depozitate toate componentele, modelele și serviciile folosite în întreaga aplicație, nu doar de o anumită componentă sau model.
- chest : pe departe partea aplicației cu care utilizatorul interacționează cel mai mult, acest modul este responsabil cu prezentarea funcționalității către utilizator și procesarea comenzilor primite de la acesta.

4.2.1 App

Fiind rădăcina aplicației Frontend, pe lângă faptul că încapsulează alte două module ale aplicației, această are și propriile sale componente precum componentă de LOGIN, care se ocupă cu autentificarea user-ilor și componentă REGISTER care la rândul ei se ocupă cu înregistrarea user-ilor în aplicație.

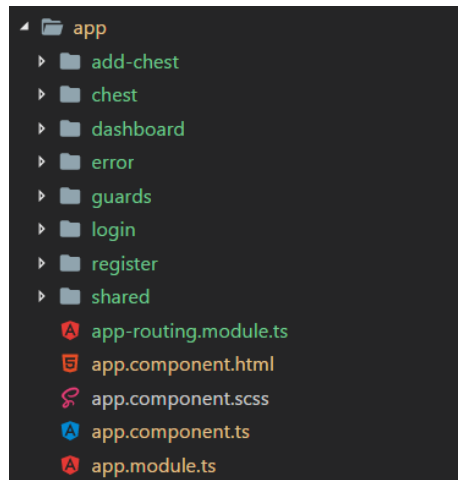


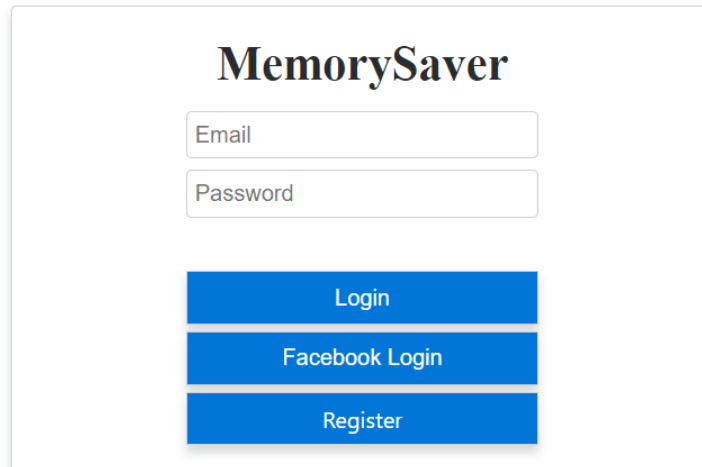
Figura 38 - Structura modulului app

Fiecare componentă are propriile sale seturi de constrângeri cum ar fi lungimea numelui, lungimea email-ului cât și a parolei ceea ce este o practică standard pentru a nu lasă un user să introducă un email de 1000 de caractere sau o parolă de 500.

```
<div class="input-container">
  <input name="email" id="email" type="text" class="input-field" placeholder="Email" maxlength="100"
    #email="ngModel" [(ngModel)]="registerModel.email"
  />
</div>
```

Figura 39 - Validări pe partea de front-end

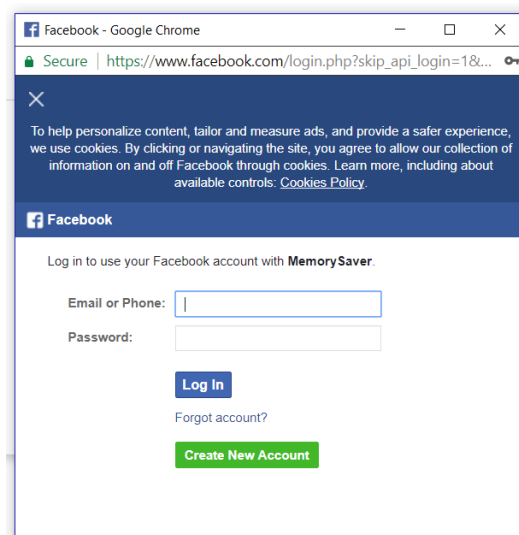
În momentul în care un utilizator dorește să se logeze, acestuia îi sunt prezentate două opțiuni, fie să se logheze prin intermediul aplicație, în măsură în care acesta are deja un cont existent, fie să se logheze prin intermediul Facebook.



The image shows a login window for an application named "MemorySaver". It features a title "MemorySaver" at the top. Below the title are two input fields: "Email" and "Password". Underneath these fields are three blue buttons stacked vertically: "Login", "Facebook Login", and "Register".

Figura 40 - Fereastra de logare prin aplicatie

În cazul în care acesta are deja un cont și decide să se logheze prin intermediu Facebook, atât timp cât la crearea contului a fost folosită aceeași adresa de email, aplicație îl va loga în acel cont. Dacă însă utilizatorul nu are cont, un cont îi va fi generat automat pe baza informațiilor primite de pe platforma de socializare.



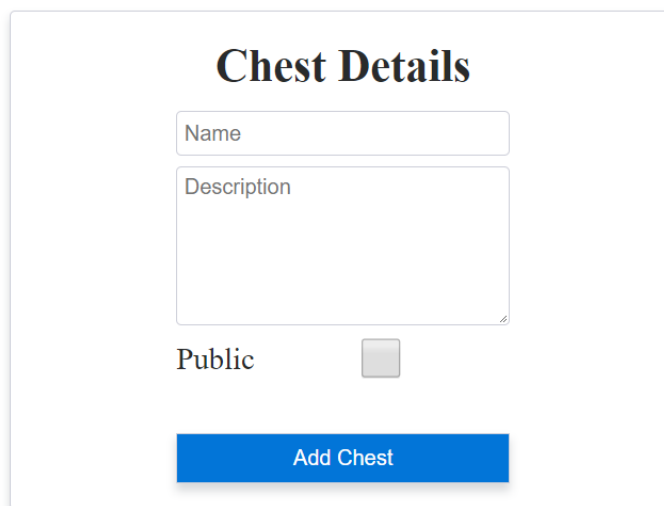
The image shows a Facebook login window in a Google Chrome browser. The address bar shows the URL "https://www.facebook.com/login.php?skip_api_login=1&...". The page has a dark blue header with the Facebook logo and the text "Log in to use your Facebook account with MemorySaver." Below this, there are two input fields: "Email or Phone:" and "Password:". Underneath these fields are two buttons: a blue "Log In" button and a green "Create New Account" button. There is also a link "Forgot account?" between the two buttons.

Figura 41 - Fereastra de logare prin intermediul Facebook

La rândul sau, există validări și pe partea de serve, acesta verificând dacă un user cu același email există deja și dacă acesta este un email valid.

O altă responsabilitate a modulului app este adăugarea de cufare și afișarea unei liste cu cuferele user-ului deja existente.

Pentru asta există două componente, Add-chest care se ocupă cu adăugarea de cufere noi, și Dashboard, care se ocupă cu afișarea unei liste complete cu cufarele unui utilizator.



The image shows a web form titled "Chest Details". It contains a text input field for "Name", a larger text area for "Description", and a checkbox labeled "Public". At the bottom of the form is a blue button with the text "Add Chest".

Figura 42 - Fereastra de adăugare a cufarelor

În cazul în care o eroare neașteptă apare, aplicația are un mecanism de backup, și anume o pagină de eroare care va afișa un mesaj de eroare generic.

Modulul app se ocupă de asemenea și de restricționarea navigării către paginile aplicației pentru user-ii care nu sunt deja autentificați. Acest lucru este realizat prin utilizarea de Guards, un mecanism pus la dispoziție de Angular. Aceste Guards permit sau nu unui utilizator să acceseze un link în urmă unor verificări definite în interiorul Guard-ului. Momentan există doar un singur guard în aplicație, și acela fiind AuthGuard, care verifică dacă un utilizator este autentificat.

```

export class AuthGuard implements CanActivate {

  constructor(private router: Router) {}

  Complexity is 4 Everything is cool!
  public canActivate(): boolean {
    if (localStorage['userToken']) {
      return true;
    }

    this.router.navigate(['/login']);
    return false;
  }
}

```

Figura 43 - Setarea Guard-urii de verificare a unui utilizator

Pentru a putea accesa diferitele componente ale aplicației, modulul app trebuie de asemenea să definească rutele care vor fi folosite în aplicație, fără ele neputându-se accesa diversele componente și module.

Tot în interiorul declarării rutelor sunt specificate și ce Guard-urivori vor fi folosite pentru ce rute, sau ce parametri sunt în interiorul rutelor, acest lucru permițându-ne să trimitem parametri de la o componentă la altă.

```

const routes: Routes = [
  {
    path: 'login', component: LoginComponent
  },
  {
    path: 'register', component: RegisterComponent
  },
  {
    path: 'add-chest', component: AddChestComponent, canActivate: [ AuthGuard ]
  },
  {
    path: '', component: DashboardComponent, pathMatch: 'prefix', canActivate: [ AuthGuard ]
  },
  {
    path: 'chest/:chestId', component: ChestComponent, pathMatch: 'prefix', canActivate: [ AuthGuard ],
    children: [
      {
        path: '', loadChildren: './chest/chest.module#ChestModule'
      }
    ]
  },
  {
    path: 'error', component: ErrorComponent
  },
  { path: '**', redirectTo: '' }
];

```

Figura 44 - Setarea rutelor in interiorul modului app

4.2.2 Shared

Modulul Shared, după cum îi sugerează și numele, este locul în care sunt încapsulate toate componentele, modelele și serviciile care sunt folosite în întreagă aplicație și care nu sunt specifice unei singure părți a acesteia. În majoritatea timpului serviciile și modelele vor fi stocate aici, deși există unele părți ale aplicației care au servicii sau modele specifice.

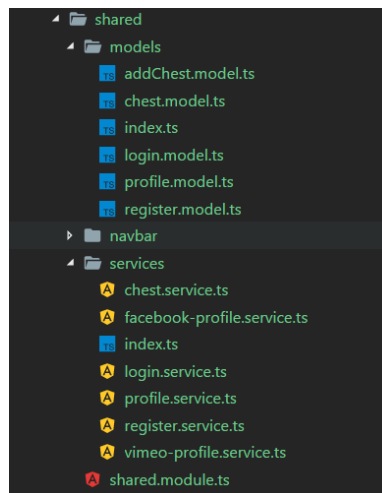


Figura 45 - Structura modulului shared

Fiind locul în care sunt păstrate serviciile aplicației, adică majoritatea logicii aplicației, îi revine astfel responsabilitatea de a manageria, prin intermediul serviciilor precizate, conexiunea cu API-urile furnizate de către rețelele de socializare.

Deocamdată am integrat 2 API-uri și anume Facebook și Vimeo, iar fiecare dintre cele două API-uri au adus cu ele un set de provocări care au avut nevoie de propriile abordări pentru a le putea depăși cu succes.

În cazul API-ului celor de la Facebook, denumit și Graph API, pentru a putea folosi utilitățile de care aveam nevoie, în urmă autentificării trebuia returnat un token. Din nefericire, nu există nici-o librărie care să pună la dispoziție această funcționalitatea. Din acest motiv, am fost forțat că în index.html, acest fișier fiind rădăcina html deasupra căreia sunt construite restul componentelor, să initializez API-ul utilizând metodă prezentată de aceștia pe site-ul lor oficial, și anume prin intermediul javascript pur.


```

<script>
var user = null;
var userAccessToken = null;
window.fbAsyncInit = function() {
  FB.init({
    appId      : '191127088286762',
    autoLogAppEvents : true,
    xfbml      : true,
    version    : 'v2.11'
  });
};

Complexity is 3 Everything is cool!
(function(d, s, id){
  var js, fjs = d.getElementsByTagName(s)[0];
  if (d.getElementById(id)) {return;}
  js = d.createElement(s); js.id = id;
  js.src = "https://connect.facebook.net/en_US/sdk.js";
  fjs.parentNode.insertBefore(js, fjs);
})(document, 'script', 'facebook-jssdk');
</script>

```

Figura 46 - Initializarea API-ului facebook

Lucrul acesta a ridicat la rândul lui un al număr de probleme, singură problema care persistă însă este faptul că trebuie să folosim o variabilă globală javascript, iar framework-ul nu știe de această deoarece este declarată prin intermediul unui script nativ.

Pe de altă parte, modul în care API-ul celor de la Vimeo funcționează, pentru a putea face orice acțiune asupra fișierelor unui utilizator este necesar un token, care este obținut în urmă unor redirecturi către o pagină de login a celor de la Vimeo, iar transferul de date, și anume token-ul de care avem nevoie pentru a putea citit sau scrie fișiere în contul Vimeo al utilizatorului este efectuată prin intermediul URL- lui de return după ce un utilizator s-a logat cu succes în aplicația acestora.



Figura 47 - Token-ul returnat de către API-ul vimeo

Pentru a putea totuși realiza conexiunea între aplicația mea și Vimeo, am ales să îi ofer utilizatorului opțiunea de a se autentifica în contul sau de Vimeo după ce se loghează în aplicație prin una dintre cele două metode, obținând astfel token-ul necesar comunicării cu Vimeo.

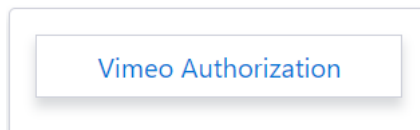


Figura 48 - Butonul de autorizare Vimeo

Însă pentru a se putea realiza comunicarea între API-uri și aplicație dar și între componentele aplicației, o modalitate de a transfera datele este necesară.

Aici intră în joc modele declarate în acest modul, care sunt folosite în toată aplicația, atunci când sunt primite răspunsuri de la API-ul propriu sau răspunsuri de la API-urile rețelelor de socializare.

Deși au un rol important în funcționarea corectă a aplicației, aceste modele nu sunt nimic mai mult decât niște clase care au ca proprietăți date care trebuie transferate între diferitele componente ale aplicației.

```
export class ProfileModel {  
  public name: string;  
  public id: string;  
  public email: string;  
  
  constructor(name: string, id: string, email: string) {  
    this.name = name;  
    this.id = id;  
    this.email = email;  
  }  
}
```

Figura 49 - Unul dintre modelele declarate de către modulul shared

4.2.3 Chest

În final ajungem la modulul chest al aplicației. După cum spuneam și mai devreme, acesta este modulul cu care utilizatorul interacționează cel mai mult și cu care acesta își manageriază cufarele și fișierele.

Fiind un modul mult mai complex decât modulul shared și datorită faptului că acesta are comportamente mult mai complexe chiar și decât modulul app, regăsim în interiorul acestuia toate părțile ce alcătuiesc un modul, plecând de la componentele specifice cum ar fi add-file, rutele pe care trebuie să le urmăm în interiorul modulului pentru a accesa aceste

componente, și ajungând la serviciile care se ocupă cu lucrul fișierelor și managementul acestora pe rețelele sociale, până la modele pe care serviciile le folosesc.

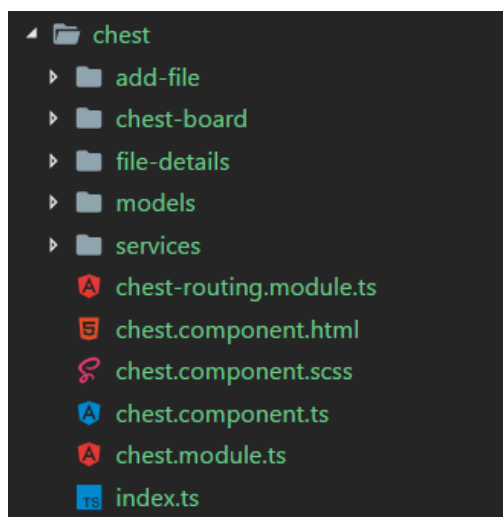


Figura 50 - Structura modulului Chest

Un prin pas pe care un utilizator îl face atunci când vine vorba de ași începe activitatea în aplicație este crearea unui cufăr, procedeu pe care l-am prezentat în modulul app, acesta conținând această funcționalitate.

După crearea unui cufăr, utilizatorul este redirecționat către pagină de prezentare a acestuia, unde utilizatorului i se pune la dispoziție, pe lângă o lista goală de fișiere, și un meniu prin care acestuia îi sunt prezentate informații despre cufăr precum numele și descrierea acestuia dar și anumite acțiuni pe care utilizatorul le poate efectua, precum adăugarea de fișiere, editarea detaliilor unui cufăr, dar și abilitatea de a șterge un cufăr.

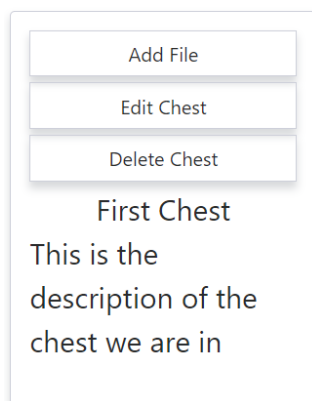


Figura 51 - Meniul din stânga de pe home-page-ul unui cufăr

În momentul în care un utilizator decide să încarce un fișier, acesta este redirecționat către o pagină care îi pune la dispoziție un set de opțiuni pentru realizarea acestor operațiuni. Pe lângă faptul că acesta poate încărca fișierul local, îl poate posta și pe diferitele rețele de socializare integrate în aplicație.

Dacă utilizatorul decide să posteze fișierul pe una dintre rețelele sociale, aplicația va verifica dacă are toate informațiile necesare, și dacă utilizatorul este logat pe rețelele unde se dorește a se posta noul fișier. În cazul în care utilizatorul nu este logat sau aplicația nu are informațiile necesare, se fac apelurile necesare, iar utilizatorul este rugat să se logheze.

După ce aplicație are toate informațiile, fișierele sunt mai întâi postate pe rețelele de socializare și abia apoi sunt încărcate local. Am ales această abordare deoarece, după efectuarea upload-ului pe diferitele platforme, acestea furnizează o modalitate de identificare a noilor fișiere, și pentru a putea obține informații despre acestea, vom avea nevoie de aceste forme de identificare.

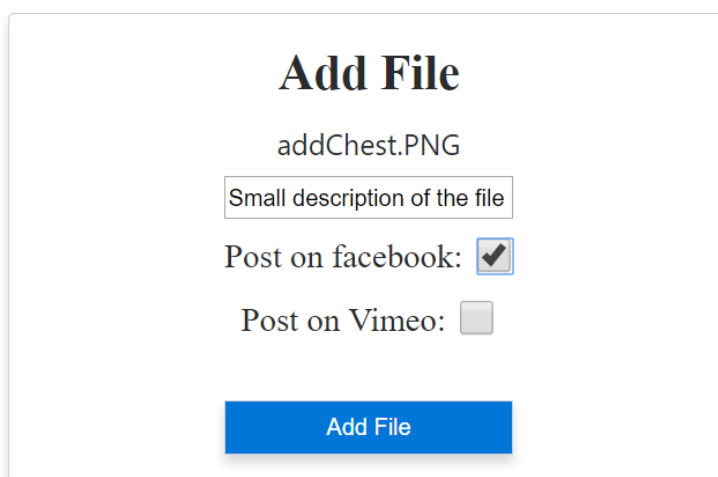
The image shows a web form titled "Add File". Below the title, the filename "addChest.PNG" is displayed. There is a text input field with the placeholder "Small description of the file". Below this, there are two checkboxes: "Post on facebook:" which is checked, and "Post on Vimeo:" which is unchecked. At the bottom of the form is a blue button labeled "Add File".

Figura 52 - Pagina de încărcare a unui fișier

De precizat este faptul că aplicație nu permite upload-ul de fișiere ce depășesc 5 MB, iar upload-ul pe diferitele rețele de socializare este limitat de către tipul fișierelor pe care acestea le permit. De asemenea, în momentul în care un fișier este postat pe o rețea de

socializare, dacă acestuia i s-a furnizat o descriere și dacă platforma permite lucrul acesta, fișierul este postat împreună cu această descriere.

După ce un fișier a fost încărcat cu succes acesta va apărea în lista de fișiere care se află pe home page-ul unui cufăr. De asemenea, în această listă sunt afișate date despre fișiere precum numele, descrierea, și tipul fișierului care este reprezentat de către o iconiță.

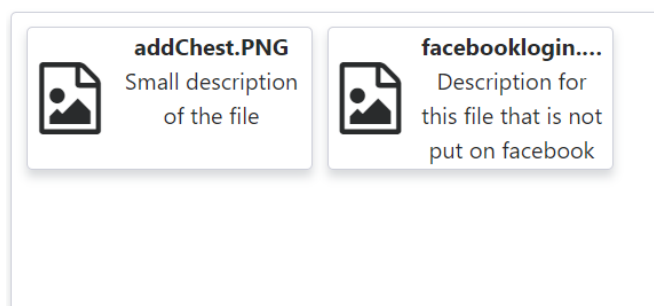


Figura 53 - Lista de fișiere aflate în cufărul curent

Dacă un utilizator dorește să obțină mai multe informații despre un fișier anume, acesta poate accesa, prin simplul click pe țile-ul fișierului din lista de elemente din cufăr, la care este redirecționat către o pagină de prezentare a acelui item, care conține toate detaliile disponibile despre acel fișier, și anume un thumbnail al sau, numele, descrierea și potențiale informații de pe rețelele de socializare, dacă acestea sunt disponibile și dacă acestea fac parte dintr-o platformă.



File Description: Some description for file

Facebook Stats: 👍 : 0 💬 : 0 ➡ : 0

Figura 54 - Pagina cu detaliile unui fișier deja salvat în aplicație, împreună cu informații după rețelele de socializare

Pe lângă faptul că un cufăr poate fi modificat, având posibilitatea să îi schimbi nume dar și descrierea, aplicația îți oferă și abilitatea de a șterge cuferele de care nu mai ai nevoie, însă această funcționalitate ar trebui să fie folosită cu atenție, deoarece odată cu ștergerea unui cufăr, sunt șterse și toate fișierele care sunt în interiorul acelui cufăr.

Acestea fiind spuse, aplicația îți oferă unui utilizator să șteargă și fișiere separate, unul câte unul. De menționat este faptul că, sub oricare formă se realizează ștergerea unui fișier, acesta nu este șters și de pe rețelele sociale pe care a fost deja postat.

5. Îmbunătățiri

Desigur, că orice lucru creat de către omenire, întotdeauna există loc de mai bine și de asemenea aplicația MemorySaver nu este scutită de la această regulă, oricât de mult am încerca să perfecționăm.

O prima îmbunătățire de la care aplicația ar avea de câștigat este integrarea a mai multor API-uri oferite de către rețelele de socializare, iar unul dintre ele ar fi cel oferit de

Instagram. Deși oferă funcționalități mult mai limitate față de API-urile deja integrate, una dintre funcționalitățile puse la dispoziție de către aplicație, și anume salvarea de fișiere de pe rețelele de socializare este în mare parte suportat de acesta.

A doua îmbunătățire la care mă pot gândi este implementarea sistemului de prietenii, în care un utilizator poate accesa cuferele unui alt utilizator pe baza prieteniei dintre aceștia. Îi poate de asemenea trimite mesaje în interiorul aplicației.

Altă arie a aplicației care ar putea fi îmbunătățită ar fi trecerea de la HTTP la HTTPS, lucru care ar asigura utilizatorul că, chiar dacă aplicația ar fi atacată de persoană rău intenționată, datele prețioase ale acestora nu ar cădea pradă acestora.

În final, ce ar însemna o îmbunătățire substanțială ar fi trecerea la arhitectură bazată pe microservicii, acestea comunicând între ele prin API-uri, fiecare dintre ele fiind o părticică a aplicației. Acest lucru ar duce la o modularizare și mai mare, oferind astfel posibilitatea de upgrade și oferind flexibilitate la modificări

6. Concluzii

După cum putem observă, ritmul alert în care lumea evoluează către un viitor cât mai conectat, în care toată lumea știe totul despre toată lumea și în care majoritatea pritenilor noștri se află în mediul online, răspândiți pe diversele platforme și rețele de socializare cere rezolvarea unor noi provocări pe omul modern.

Datorită tehnologiilor foarte noi cu care a fost dezvoltată aplicația, această se poate adapta cu ușurință noilor tehnologii, cele existente și celor ce vor apărea și poate fi extinsă cu o mare ușurință. Datorită arhitecturii foarte decuplată, aplicațiile de front-end și back-end fiind complet separate ajută de asemenea la această lucru, modificările necesare putând fi implementate doar într-una dintre cele două părți, fără a avea grijă că vom strică toată aplicația.

Acestea fiind spuse, putem spune că misiunea aplicației a fost îndeplinită cu succes, această oferind un mijloc ușor utilizatorilor de a avea grijă de conținutul fiecărei platforme pe care se află, astfel luându-le o greutate din mormatul tot mai mare, care crește în fiecare zi.

7. Bibliografie

1. <https://blogs.msdn.microsoft.com/dotnet/2017/08/14/announcing-net-core-2-0/>
2. <https://blogs.msdn.microsoft.com/dotnet/2016/09/26/introducing-net-standard/>
3. <https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework>
4. <https://docs.microsoft.com/en-us/ef/core/>
5. <http://www.dotnettricks.com/learn/webapi/what-is-web-api-and-why-to-use-it->
6. https://www.microsoft.com/nl-nl/sql-server/sql-server-2017?wt.mc_id=AID631230_SEM_s7sSGPRx&wt.srch=1&wt.mc_id=AID631230_SEM_s7sSGPRx&gclid=CjwKCAiA5OrTBRBlEiwAXXhT6O3pvQKrxFN-e2m93zya7OnI_O9c4FEXxUmDpyfEgQ0C8Uoag_wsyBoCGu4QAvD_BwE
7. <https://www.connectionstrings.com/sql-server/>
8. [https://msdn.microsoft.com/en-us/library/jj653752\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/jj653752(v=vs.110).aspx)
9. <https://www.visualstudio.com/tfs/>
10. <https://docs.microsoft.com/en-us/nuget/quickstart/install-and-use-a-package-in-visual-studio>
11. <http://www.c-sharpcorner.com/article/basic-architecture-of-angular-2-applications/>
12. <https://blog.angular.io/version-5-0-0-of-angular-now-available-37e414935ced>
13. <https://angular.io/>
14. <https://www.npmjs.com/>
15. <https://cli.angular.io/>
16. <https://www.w3schools.com/html/default.asp>
17. <https://sass-lang.com/>
18. <https://jwt.io/>
19. <http://blog.thedigitalgroup.com/chetanv/2015/07/06/understanding-onion-architecture/>
20. <https://medium.com/vandium-software/5-easy-steps-to-understanding-json-web-tokens-jwt-1164c0adfcec>

21. <https://www.w3schools.com/css/default.asp>
22. <https://www.w3schools.com/bootstrap4/default.asp>
23. https://www.w3schools.com/html/html_responsive.asp
24. <http://automapper.org/>
25. <https://autofac.org/>
26. <https://github.com/danielwertheim/Ensure.That>