

## Cuprins

Cuprins .....	1
1. Introducere.....	3
1.1 Tema lucrării .....	3
1.2 Tehnologii şi framework-uri folosite.....	4
1.3 Versiuni ulterioare .....	5
2. Descriere tehnologii BackEnd .....	6
2.1 ASP.NET WebApi .....	6
2.1.1 Ce este? .....	6
2.1.2 Caracteristicile API-ului ASP.NET Web .....	7
2.1.3 Serializare .....	7
2.1.4 Controler API .....	8
2.1.5 Configurări.....	9
2.1.6 Parametrizare .....	9
2.1.7 Tipuri de returnare .....	10
2.1.8 REST.....	11
2.2 C# .....	13
2.3 Unity Container .....	14
2.4 Entity Framework .....	16
2.4.1 Caracteristici .....	16
2.4.2 Modele de programare.....	17
2.5 LINQ .....	18
2.5.1 Ce este? .....	18
2.5.2 Folosire LINQ API .....	20
2.5.3 Query Syntax .....	20
2.5.4 Method Syntax.....	21
2.6 SQL SERVER .....	21
2.6.1 Ce este? .....	21
2.6.2 Componente cheie si servicii .....	21
2.6.3 Instanţe SQL Server .....	22
2.7 * HELP PAGE .....	23
3. Descriere tehnologii FrontEnd.....	26
3.1 HTML.....	26
3.2 Stiluri CSS .....	31
3.2.1 Ce este CSS? .....	31
3.2.2 Unde se definesc stilurile? .....	32
3.2.3 Bootstrap .....	33
3.4 Javascript.....	35
3.4.1 Ce este javascript? .....	35
3.4.2 ECMAScript .....	37
3.4.3 Sintaxa .....	38
3.4.4 Variabile.....	38
3.4.5 Tipuri de date.....	39
3.5 Typescript .....	40
3.6 ReactJS .....	41
3.6.1 Ce este? .....	41
3.6.2 JSX.....	42
3.6.3 VirtualDOM.....	42

3.6.4 Componente .....	43
3.6.5 Metode utile din viaţa unei componente.....	44
4. Detalii proiect.....	46
4.1 BackEnd.....	46
4.1.1 Structura .....	46
4.1.1.1 Repositories.....	48
4.1.1.2 Services.....	49
4.1.1.3 WebSite.....	49
4.1.2 Repositories .....	50
4.1.3 Servicii .....	53
4.1.4 WebSite .....	59
4.2 FrontEnd .....	64
4.2.1 Structura .....	64
4.2.2 Modele .....	64
4.2.3 Rutare .....	65
4.2.4 Componente .....	68
4.2.5 Stiluri .....	78
4.3 Screenshots .....	79
5. Concluzii.....	81
Bibliografie .....	82
ANEXE.....	83

### Cuprins figuri

Fig. 1 Reprezentare Web Api .....	6
Fig. 2 – Prezentare generala a controlerului API .....	8
Fig. 3 Configurare Unity Container .....	15
Fig. 4 Unity – Dependency Injection .....	15
Fig. 5 Entity Framework.....	18
Fig. 6 Utilizare LINQ .....	19
Fig. 7 Help Page – lista .....	24
Fig. 8 Help page – detaliere request .....	25
Fig. 9 Model cod HTML .....	26
Fig. 10 Tabel in HTML.....	27
Fig. 11 Formular HTML .....	29
Fig. 12 Stil definit in HEAD .....	32
Fig. 13 Coloanele in Bootstrap in functie de rezolutia dispozitivului .....	34
Fig. 14 Coloane Bootstrap.....	34
Fig. 15 Meniu in Bootstrap.....	35
Fig. 16 VirtualDOM.....	42
Fig. 17 Lifecycle methods – react component .....	43
Fig. 18 Repositories .....	47
Fig. 19 Servicii.....	48
Fig. 20 WebSite.....	49

## 1. Introducere

Un serviciu web este un serviciu pus la dispoziţie utilizatorilor pe Internet. Multitudinea de protocoale şi standarde disponibile începând de la sfârşitul secolului trecut în sfera Internetului au dat posibilitatea comunicării între aplicaţii pe sisteme aflate la distanţe mari, cu acces la Internet. Astfel, există sisteme ce oferă servicii de informare şi procesare a informaţiilor care în general sunt independente de platforma hardware; accesul la acestea se face prin servicii web.

Exemple clasice de servicii web de informare sunt aflarea cursului de bursă momentan al unei acţiuni anume sau aflarea condiţiilor climaterice într-un anumit punct de pe glob. Serviciile de prelucrare de informaţii pornesc de la cele mai banale servicii, cum ar fi execuţia de operaţii aritmetice asupra unor numere, şi până la servicii complexe cum ar fi serviciile de autentificare.

În ultima vreme ideea de serviciu web a luat o amploare în rândul siturilor web care oferă din ce în ce mai multe protocoale pentru trimiterea diverselor date către diferite tipuri şi categorii de utilizatori. [1]

### 1.1 Tema lucrării

Această lucrare se bazează pe crearea unui API independent, dezvoltat cu ajutorul **serviciilor web** folosind arhitectura REST. Atât timp cât partea de server a acestei aplicaţii e complet independentă, pentru partea de FrontEnd se poate opta la orice, ținând cont de necesități (ex. Dacă se dorește aplicație mobilă, desktop, web). Aplicația de frontend va consuma API-ul creat.

Pentru curenta aplicație s-a folosit librăria **ReactJS**, creată de echipa de dezvoltare Facebook. ReactJS permite să se creeze rapid un frontend scalabil și ușor de utilizat pentru aplicațiile web. Este una dintre librăriile open-source cele mai populare în rândul programatorilor dar și al mediului de afaceri, mulțumită avantajelor sale în dezvoltarea aplicațiilor web. [2]

Aplicația prezentată în această lucrare are ca scop fluidizarea operațiilor necesare în cazul unei companii ce administrează asociații de bloc.

În momentul de față, principalele funcționalități ale acestei aplicații sunt:

- Design responsive - compatibilitatea cu orice dispozitiv (desktop, smartphone, tabletă)
- Administrator:
  - o Adăugare/editare/ștergere provideri, facturi, useri, imobile etc
  - o Generare rapoarte lunare -> lista de plată per imobil
  - o descărcare liste în format .csv
- User:
  - o Vizualizare liste plăți lunare și descărcarea lor în format .csv
  - o trimitere de consum apă
  - o modificare parolă
- exportare informații tabelare în .csv (de pe fiecare pagină)

- 
- filtrare avansată în tabele

Un administrator poate să adauge o multitudine de date pentru diferite asociaţii de locatari. Pentru datele adăugate de user (consum apă), administratorul poate să le modifice. Pentru diferite acţiuni, în momentul unei erori, administratorul va fi notificat printr-o alertă (în pop-up). Nu sunt acoperite toate erorile. (unele se observă doar analizând răspunsul request-ului – cod, mesaj).

Pentru generarea listă de plăţi, administratorul are de urmat anumiţi paşi clari. (cel puţin o factură de apă, cel puţin o factură de curent, să existe apartamente definite pe clădire). Poate adăuga şi “cheltuieli” custom, ce vor fi folosite în generarea listei. Odată generată lista, ea va fi salvată ca string în DB. Oriunde există mai multe date adăugate în aceeaşi lună, sistemul le va ordona descrescător după data de adăugare şi va folosi prima instanţă (ex: facturi, citiri).

Un user poate să îşi modifice parola cât şi să adauge consum. În momentul în care adaugă consum, nu îi e oferită posibilitatea de a alege data, ea fiind adăugată automat (ziua curentă). Dacă un utilizator nu îşi adaugă consum de apa într-o lună, la generare îi va fi adăugat acelaşi consum ca luna precedentă.

Un user poate vedea şi listele de plăţi generate, cât şi să le descarce în format .csv. Userul e foarte limitat, ceea ce îl obligă să valideze toate modificările cu administratorul (acesta având grijă să nu fie încălcate regulile legale).

## 1.2 Tehnologii şi framework-uri folosite

Pentru a implementa cu succes această platformă s-au folosit următoarele tehnologii/framework-uri/librării:

- BackEnd:
  - ASP.NET **WebApi**
    - (folosind şi generarea automată a documentaţiei - HelpPage)
  - C#
  - Unity Container (dependency injection)
  - Entity Framework
  - LINQ
  - Sql Server 2019
- FronEnd:
  - **ReactJS**
  - HTML5
  - CSS3 (Bootstrap 4)
  - Javascript + TypeScript
  - Librării:
    - react-bootstrap-table
    - react-datepicker
    - create-react-app
    - react-router
    - downloadjs

---

### **1.3 Versiuni ulterioare**

În următoarea versiune a acestei platforme vor fi implementate funcţionalităţi ca:

- plata cu cardul (comunicarea cu un alt API)
- adăugarea de penalizări (algoritm de calculare)
- user: vizualizare raport detaliat per lună (cont asociaţie, penalizări, consumabile, facturi)
- management pierderi apă
- adăugarea mai multor apartamente per user
- alertarea user-ului în mai multe cazuri

## 2. Descriere tehnologii BackEnd

### 2.1 ASP.NET WebApi

#### 2.1.1 Ce este?

Înainte de a înţelege ce este API-ul Web, să vedem ce este o API (Interfaţă de programare a aplicaţiilor). În programarea computerului, o interfaţă de programare a aplicaţiilor (API) este un set de definiţii de subrutină, protocoale şi instrumente pentru construirea de software şi aplicaţii.

Pentru a spune în termeni simpli, API este un fel de interfaţă care are un set de funcţii care permit programatorilor să acceseze caracteristici specifice sau date ale unei aplicaţii, sistem de operare sau alte servicii.

API-ul Web, după cum sugerează şi numele, este o API de pe web la care poate fi accesat folosind protocolul HTTP. Este un concept şi nu o tehnologie. Se poate construi API-ul Web folosind diferite tehnologii, cum ar fi Java, .NET etc

ASP.NET Web API este un cadru extensibil pentru construirea de servicii bazate pe HTTP, care poate fi accesat în aplicaţii diferite pe platforme diferite, cum ar fi Web, Windows, mobil etc. Funcţionează mai mult sau mai puţin la fel ca aplicaţia web ASP.NET MVC, cu excepţia că trimite date ca răspuns în loc de fişiere html. Este ca un serviciu Web sau un serviciu WCF, dar excepţia este că acceptă doar protocolul HTTP.

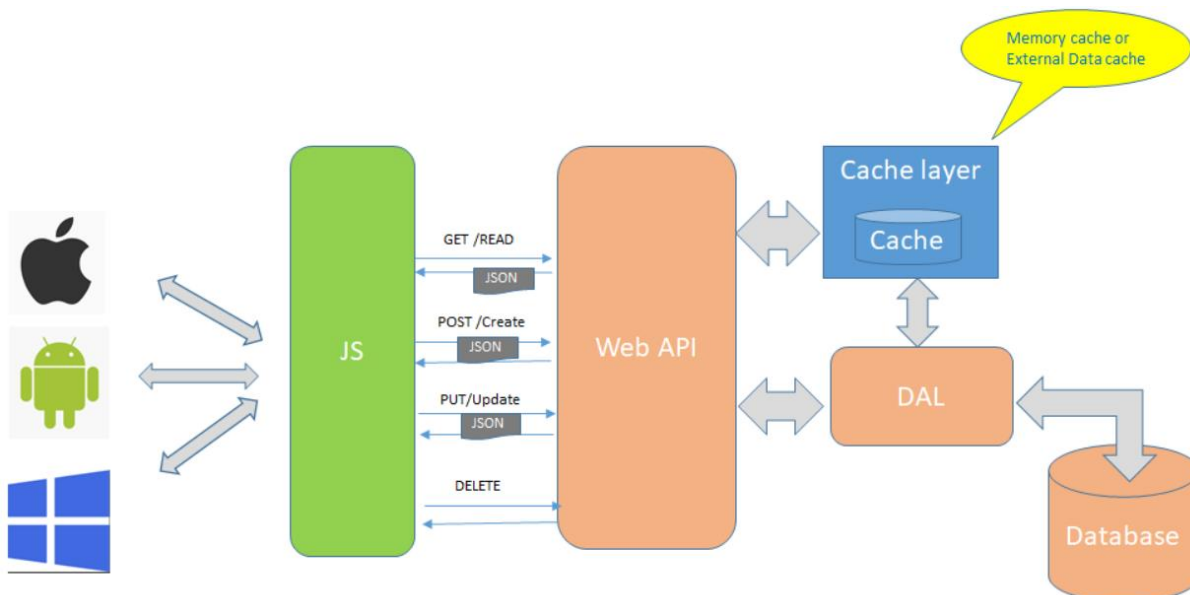


Fig. 1 Reprezentare Web Api

### 2.1.2 Caracteristicile API-ului ASP.NET Web

1. ASP.NET Web API este o platformă ideală pentru construirea serviciilor RESTful.
2. ASP.NET Web API este construită pe ASP.NET şi acceptă request/response pipeline
3. ASP.NET Web API mapează verbele HTTP cu numele metodei.
4. ASP.NET Web API acceptă diferite formate de date de răspuns. Suport încorporat pentru formatul JSON, XML, BSON.
5. ASP.NET Web API poate fi găzduit în IIS sau în alt server web care acceptă .NET 4.0+.
6. Cadrul de API Web ASP.NET include HttpClient nou pentru a comunica cu serverul API Web. HttpClient poate fi utilizat în serverul ASP.MVC, aplicaţia Windows Form, aplicaţia Console sau alte aplicaţii. [3]

### 2.1.3 Serializare

ASP.NET a fost proiectat pentru experienţe web moderne. Endpoint-urile serializează automat clasele dvs. pentru a putea realiza corect un răspuns în format JSON. Nu este necesară configurarea specială. Desigur, serializarea poate fi personalizată pentru puncte finale care au cerinţe unice.

Exemplu controller WebApi:

```
[ApiController]
public class PeopleController : ControllerBase
{
    [HttpGet("people/all")]
    public ActionResult<IEnumerable<Person>> GetAll()
    {
        return new []
        {
            new Person { Name = "Ana" },
            new Person { Name = "Felipe" },
            new Person { Name = "Emillia" }
        };
    }
}

public class Person
{
    public string Name { get; set; }
}
```

Comandă rulare api:

Curl <https://localhost:2334/people/all>

Răspuns în format JSON:

```
[{"name": "Ana"}, {"name": "Felipe"}, {"name": "Emilia"}]
```

## 2.1.4 Controler API

Controlerul API Web este o clasă care poate fi creată sub folderul **Controllers** sau orice alt folder din folderul rădăcină al proiectului. Numele unei clase de controler trebuie să se termine cu „Controller” şi trebuie să fie derivat din clasa **ApiController** (System.Web.Http). Toate metodele publice ale controlerului se numesc metode de acţiune.

Pe baza adresei URL a cererii primite şi a verbului HTTP (GET / POST / PUT / PATCH / DELETE), API-ul Web decide care este controlorul API şi metoda de acţiune care trebuie executată, de exemplu, metoda Get() va gestiona cererea HTTP GET, metoda Post() se va ocupa Solicitare POST HTTP, metoda Put() se va ocupa de cererea HTTP PUT şi metoda Delete() va gestiona cererea HTTP DELETE pentru API-ul Web de mai sus.

Pe baza adresei URL a cererii primite şi a verbului HTTP (GET / POST / PUT / PATCH / DELETE), API-ul Web decide care este controlorul API şi metoda de acţiune care trebuie executată, de exemplu, metoda Get () va gestiona cererea HTTP GET, metoda Post () se va ocupa Solicitare POST HTTP, metoda Put () se va ocupa de cererea HTTP PUT şi metoda Delete () va gestiona cererea HTTP DELETE pentru API-ul Web de mai sus.

```
public class ValuesController : ApiController
{
    // GET api/values
    public IEnumerable<string> Get()
    {
        return new string[] { "value1", "value2" };
    }

    // GET api/values/5
    public string Get(int id)
    {
        return "value";
    }

    // POST api/values
    public void Post([FromBody]string value)
    {
    }

    // PUT api/values/5
    public void Put(int id, [FromBody]string value)
    {
    }

    // DELETE api/values/5
    public void Delete(int id)
    {
    }
}
```

Diagram illustrating the mapping of HTTP requests to API controller methods:

- Web API controller Base class**: ApiController
- Handles Http GET request**: `Get()` (http://localhost:1234/api/values)
- Handles Http GET request with query string**: `Get(int id)` (http://localhost:1234/api/values?id=1)
- Handles Http POST request**: `Post([FromBody]string value)` (http://localhost:1234/api/values)
- Handles Http Put request**: `Put(int id, [FromBody]string value)` (http://localhost:1234/api/values?id=1)
- Handles Http DELETE request**: `Delete(int id)` (http://localhost:1234/api/values?id=1)

Fig. 2 – Prezentare generala a controlerului API



### 2.1.5 Configurări

API-ul Web acceptă să fie configurat din cod. Nu poate fi configurat în fişierul web.config. Se poate configura API-ul Web pentru a personaliza comportamentul infrastructurii şi componentele de găzduire a API-ului Web, precum rutele, formatorii, filtrele, DependencyResolver, MessageHandlers, ParameterBindingRules, proprietăţi, servicii etc.

Proprietate	Descriere
DependencyResolver	Obţine sau setează DependencyInjection
Filtre	Obţine sau setează filtrele.
formatters	Obţine sau setează formatatoarele de tip media.
IncludeErrorDetailPolicy	Obţine sau setează o valoare care indică dacă detaliile de eroare trebuie incluse în mesajele de eroare.
MessageHandlers	Obţine sau setează manipulatorii de mesaje.
ParameterBindingRules	Obţine colecţia de reguli pentru legarea parametrilor.
Proprietăţi	Obţine proprietăţile asociate cu această instanţă API Web.
Rutele	Obţine colecţia de rute configurate pentru API-ul Web.
Servicii	Obţine serviciile API API Web.

```
public static class WebApiConfig
{
    public static void Register(HttpConfiguration config)
    {
        config.MapHttpAttributeRoutes();

        config.Routes.MapHttpRoute(
            name: "DefaultApi",
            routeTemplate: "api/{controller}/{id}",
            defaults: new { id = RouteParameter.Optional }
        );

        // configure additional webapi settings here..
    }
}
```

### 2.1.6 Parametrizare

Metodele de acţiune în controlerul API Web pot avea unul sau mai mulţi parametri de diferite tipuri. Poate fi de tip primitiv sau de tip complex. API-ul Web leagă parametrii metodei de acţiune fie cu şirul de interogare URL, fie cu corpul de solicitare, în

funcţie de tipul parametrului. În mod implicit, dacă tipul parametrului este de tip primitiv .NET, cum ar fi int, bool, double, string, GUID, DateTime, zecimal sau orice alt tip care poate fi convertit din tip string, atunci setează valoarea unui parametru din şirul de interogare, şi dacă tipul de parametru este de tip complex, atunci API-ul Web încearcă să obţină valoarea din corpul solicitării în mod implicit.

Reguli:

HTTP Method	Query String	Request Body
GET	Primitiv, Complex	NA
POST	Primitiv	Complex
PUT	Primitiv	Complex
PATCH	Primitiv	Complex
DELETE	Primitiv, Complex	NA

### [FromUri] / [FromBody]

Aţi văzut că în mod implicit API-ul Web primeşte valoarea unui parametru primitiv din şirul de interogare şi parametrul de tip complex din corpul de solicitare. Dar ce se întâmplă dacă vrem să schimbăm acest comportament implicit?

Utilizam atributul [FromUri] pentru a forţa API-ul Web pentru a obţine valoarea tipului complex din şirul de interogare şi atributul [FromBody] pentru a obţine valoarea de tip primitiv din corpul de solicitare, opus regulilor implicite.

```
public class StudentController : ApiController
{
    public Student Get([FromUri] Student stud)
    {
    }
}
```

### 2.1.7 Tipuri de returnare

Metodele de acţiune a API-ului Web pot returna următoarele tipuri:

- Void
- Primitive sau complexe (valoare, referinta)
- HttpResponseMessage
- IHttpActionResult

Ex. HttpResponseMessage:

```
public HttpResponseMessage Get(int id)
{
    Student stud = GetStudentFromDB(id);

    if (stud == null) {
        return Request.CreateResponse(HttpStatusCode.NotFound, id);
    }

    return Request.CreateResponse(HttpStatusCode.OK, stud);
}
```

În metoda de acţiune de mai sus, dacă nu există nici un student cu id specificat în DB, atunci acesta va returna codul de stare HTTP 404 Not Found, în caz contrar va returna 200 de stare OK cu datele studentului. [4]

### 2.1.8 REST

RESTful Web Services sunt servicii web bazate pe metodele HTTP şi conceptul de REST. De obicei următoarele patru metode HTTP sunt folosite în definirea serviciilor RESTful:

- POST: upload-ul unei noi resurse (creare sau modificare). Execuţii repetate pot avea efecte distincte.
- PUT: crearea unei noi resurse. Execuţii repetate vor avea acelaşi efect ca şi o singură execuţie IDEMPOTENT.
- GET: citirea unei resurse fără a modifica resursa. Operaţia nu trebuie să fie folosită la creare de resurse.
- DELETE: ştergerea unei resurse. Execuţii repetate vor avea acelaşi efect ca şi o singură execuţie IDEMPOTENT.

REST a fost descris în mod formal de Roy J. Fielding în teza sa de doctorat. El pleacă de la un sistem care nu are delimitări clare între componente, şi aplică incremental cinci constrângeri obligatorii şi una opţională asupra elementelor care compun arhitectura:

- *client-server*: separarea aspectelor interfeţei utilizator de stocarea datelor
- *fără stare*: păstrarea detaliilor despre sesiune strict la client, eliberând astfel serverul de povara managementului sesiunilor, pentru a aduce scalabilitate, siguranţă, şi vizibilitate; dezavantajul este că fiecare cerere va trebui să conţină suficiente informaţii încât să poată fi procesată corect
- *cache*: datele care compun un răspuns trebuie să fie etichetate ca şi cache-abile sau non-cache-abile
- *interfaţă uniformă între componente*, aşa cum e definită de următoarele constrângeri secundare: identificarea resurselor, manipularea resurselor prin intermediul reprezentărilor acestora, mesaje auto-descriptive; şi hypermedia ca motor al stării aplicaţiei (aka HATEOAS)

- *sistem stratificat*: componenta poate să "vadă" şi să interacţioneze doar cu straturile din imediata sa apropiere; spre exemplu, clienţii nu pot presupune că interacţionează direct cu sursa de date, deoarece pot comunica cu un nivel de cache
- *[opţional] cod-la-cerere*: funcţionalitatea clientului poate fi extinsă prin descărcarea şi execuţia de cod extern; aceasta înseamnă că nu este necesar să se pornească execuţia în client atunci când tot codul este disponibil, deoarece el poate fi obţinut ulterior la cerere; imaginaţi-vă cum este adăugată funcţionalitate prin injectarea de cod Javascript în browser

Reprezentarea este o parte a stării resursei care este transferată între client şi server. Se referă de obicei la starea curentă a resursei, dar poate indica şi starea dorită, ne putem gândi la acest lucru ca la un dry-run atunci când se face cererea.

Deşi nu constituie o constrângere în sine, mecanismele de comunicare oferite de HTTP sunt alegerea celor mai mulţi dezvoltatori care implementează REST. Se va folosi HTTP în acest proiect, iar verbele sale vor ajuta la definirea operaţiilor care se efectuează asupra resurselor. În cele ce urmează se va folosi paradigma puternică a calendarului pentru a ilustra uşor diferenţa dintre resursă şi reprezentarea acesteia, prin studierea reprezentării .ics în prima fază:

```
GET /calendar/123sample
Host example.dev
Accept: text/calendar
```

ar putea returna ceva similar cu:

```
BEGIN:VCALENDAR
VERSION:2.0
PRODID:-//Tekkie Consulting//123sample//EN
CALSCALE:GREGORIAN
BEGIN:VTIMEZONE
TZID:Europe/Bucharest
END:VTIMEZONE
BEGIN:VEVENT
UID:123456789@example.dev
DTSTART;TZID=Z:20150311T123456
DTEND;TZID=Z:20150311T125959
END:VEVENT
END:VCALENDAR
```

Exemplu dezvoltare CRUD pe un API REST:

Pentru entitatea User, va fi UserController.

Metode:

- **GET /users/** - va returna lista completa cu useri
- **GET /users/{id}** – va returna un user pe baza de id
- **POST /users/** - in body adaugam continutul unui user, iar prin aceasta metoda vom introduce un user in DB
- **PUT /users/{id}** - in body adaugam continutul unui user, iar prin aceasta metoda vom modifica un user in DB
- **DELETE /users/{id}** – vom sterge un user pe baza id-ului

## 2.2 C#

Numele limbajului C# a fost inspirat din notaţia # (diez) din muzică, care indică faptul că nota muzicală urmată de # este mai înaltă cu un semiton. Este o similitudine cu numele limbajului C++, unde ++ reprezintă atât incrementarea unei variabile cu valoarea 1, dar şi faptul că C++ este mai mult decât limbajul C.

Limbajul C# a fost dezvoltat în cadrul Microsoft. Principalii creatori ai limbajului sunt Anders Hejlsberg, Scott Wiltamuth şi Peter Golde. Prima implementare C# larg distribuită a fost lansată de către Microsoft ca parte a iniţiativei .NET în iulie 2000. Din acel moment, se poate vorbi despre o evoluţie spectaculoasă. Mii de programatori de C, C++ şi Java, au migrat cu uşurinţă spre C#, graţie asemănării acestor limbaje, dar mai ales calităţilor noului limbaj. La acest moment, C# şi-a câştigat şi atrage în continuare numeroşi adepţi, devenind unul dintre cele mai utilizate limbaje din lume.

Creatorii C# au intenţionat să înzestreze limbajul cu mai multe facilităţi. Succesul de care se bucură în prezent, confirmă calităţile sale:

- Este un limbaj de programare simplu, modern, de utilitate generală, cu productivitate mare în programare.
- Este un limbaj orientat pe obiecte.
- Permite dezvoltarea de aplicaţii industriale robuste, durabile.
- Oferă suport complet pentru dezvoltarea de componente software, foarte necesare de pildă în medii distribuite. De altfel, se poate caracteriza C# ca fiind nu numai orientat obiect, ci şi orientat spre componente.

La aceste caracteristici generale se adaugă şi alte trăsături, cum este de pildă suportul pentru internaţionalizare, adică posibilitatea de a scrie aplicaţii care pot fi adaptate cu uşurinţă pentru a fi utilizate în diferite regiuni ale lumii unde se vorbesc limbi diferite, fără să fie necesare pentru aceasta schimbări în arhitectura software.

În strânsă legătură cu **Arhitectura .NET (.NET Framework)** pe care funcţionează, C# gestionează în mod automat memoria utilizată. Eliberarea memoriei ocupate (garbage collection) de către obiectele care nu mai sunt necesare aplicaţiei, este o facilitate importantă a limbajului. Programatorii nu mai trebuie să decidă singuri, aşa cum o fac de pildă în C++, care este locul şi momentul în care obiectele trebuie distruse. În C# se scriu de asemenea aplicaţii pentru sisteme complexe care funcţionează sub o mare varietate de sisteme de operare, cât şi pentru sisteme dedicate (embedded systems). Acestea din urmă se întind pe o arie largă, de la dispozitive portabile cum ar fi ceasuri digitale, telefoane mobile, MP3 playere, până la dispozitive staţionare ca semafoare de trafic, sau controlere pentru automatizarea producţiei.

---

Din punct de vedere sintactic C# derivă din limbajul C++, dar include şi influenţe din alte limbaje, mai ales Java. [6]

## 2.3 Unity Container

Unity container este un container IoC open source pentru aplicaţii .NET acceptate de Microsoft. Este un recipient IoC uşor şi extensibil.

Caracteristici ale containerului Unity:

- Înregistrare simplificată de mapare a tipurilor pentru tipul de interfaţă sau tipul de bază.
- Acceptă înregistrarea unei instanţe existente.
- Suportă înregistrarea pe bază de cod, precum şi înregistrarea în timp de proiectare.
- Injectează automat tipul înregistrat la timpul de execuţie printr-un constructor, o proprietate sau o metodă.
- Acceptă rezoluţia amânată.
- Acceptă containere cuibare.
- Eliminarea automată a instanţelor bazate pe managerii de viaţă; managerii de viaţă includ ierarhici, pe rezolvare, controlaţi extern, pe cerere şi pe thread.
- Suportă capacitatea de locaţie a serviciului; acest lucru permite clienţilor să stocheze sau să memoreze în cache containerul.
- Acceptă interceptarea tipului şi interceptarea instanţelor.
- Uşor de extins.

S-a folosit Unity Container pentru configurarea dependintelor între clase şi interfeţe. [7]

Dependency inversion spune că entităţile ce depind unele de altele ar trebui să interacţioneze printr-o abstractizare, nu direct cu o implementare concretă.

**Exemplu:** Dacă avem un strat de acces la date şi un strat de afaceri, atunci acestea nu ar trebui să depindă direct unele de altele, acestea ar trebui să depindă de o interfaţă sau de un abstract pentru crearea obiectelor.

**Avantaje:**

1. Folosirea abstractizării permite diversele componente să fie dezvoltate şi schimbate independent unele de altele.
2. Şi uşor pentru testarea componentelor.

**Microsoft Unity Framework** ajută la injectarea dependenţe externe în componentele software. Pentru a o utiliza într-un proiect, trebuie doar să se adăuge o referinţă pentru DLL-urile Unity Container la proiect. [8]

Există următoarele trei tipuri de dependenţe:

1. **Constructor** – folosit in aplicaţia prezentată
2. **Setter** (Proprietăţi)
3. **Metodă**.

```
1 reference | biancabm, 1 day ago | 2 authors, 4 changes
public static void Register(HttpConfiguration config)
{
    config.EnableCors();

    // Web API configuration and services
    var container = new UnityContainer();
    RegisterRepositories(container);
    RegisterServices(container);

    config.DependencyResolver = new UnityResolver(container);
}
```

Fig. 3 Configurare Unity Container

```
1 reference | biancabm, 3 hours ago | 1 author, 4 changes
private static void RegisterRepositories(UnityContainer container)
{
    container.RegisterType<IUserRepository, UserRepository>();
    container.RegisterType<IProviderBillRepository, ProviderBillRepository>();
    container.RegisterType<IProviderRepository, ProviderRepository>();
    container.RegisterType<IApartmentRepository, ApartmentRepository>();
    container.RegisterType<IWaterConsumptionRepository, WaterConsumptionRepository>();
    container.RegisterType<IMansionRepository, MansionRepository>();
}

1 reference | biancabm, 3 hours ago | 1 author, 4 changes
private static void RegisterServices(UnityContainer container)
{
    container.RegisterType<IUserService, UserService>();
    container.RegisterType<IProviderBillService, ProviderBillService>();
    container.RegisterType<IProviderService, ProviderService>();
    container.RegisterType<IApartmentService, ApartmentService>();
    container.RegisterType<IWaterConsumptionService, WaterConsumptionService>();
    container.RegisterType<IMansionService, MansionService>();
}
```

Fig. 4 Unity – Dependency Injection

## 2.4 Entity Framework

### 2.4.1 Caracteristici

Entity Framework este un set de tehnologii în ADO.NET ce suportă dezvoltarea de aplicaţii software cu baze de date, aplicaţii orientate pe obiecte. Comenzile din ADO.NET lucrează cu scalari (date la nivel de coloană dintr-o tabelă) în timp ce ADO.NET Entity Framework lucrează cu obiecte (din baza de date se returnează obiecte).

Arhitectura ADO.NET Entity Framework consta din urmatoarele:

- Provideri specifici pentru sursa de date (**Data source**) ce abstractizeaza interfetele ADO.NET pentru conectare la baza de date cand programam folosind schema conceptuala (model conceptual).
- **Provider specific** bazei de date ce translateaza comenzile Entity SQL in cereri native SQL (comenzi SQL din limbajul de definire a bazei de date, limbajul de cereri).
- Parser EDM (**Entity Data Model**) si mapare vizualizari prin tratarea specificatiilor SDL (Storage Data Language –model de memorare) al modelului de date, stabilirea asociatiilor dintre modelul relational (baza de date) si modelul conceptual. Din schema relationala se creaza vizualizari ale datelor ce corespund modelului conceptual. Informatii din mai multe tabele sunt agregate intr-o entitate. Actualizarea bazei de date (apel metoda **SaveChanges()**) are ca efect construirea comenzilor SQL specifice fiecărei tabele ce apare in acea entitate.
- **Servicii pentru metadata** ce gestioneaza metadata entitatilor, relatiilor si maparilor.
- **Tranzactii** – pentru a suporta posibilitatile tranzactionale ale bazei de date.
- **Utilitare pentru proiectare** – Mapping Designer – incluse in mediul de dezvoltare.
- **API pentru nivelul conceptual** – runtime ce expune modelul de programare pentru a scrie cod folosind nivelul conceptual. Se folosesc obiecte de tip **Connection**, **Command** asemanator cu ADO.NET si se returneaza rezultate de tip **EntityResultSets** sau **EntitySets**.
- **Componente deconectate** – realizeaza un cache local pentru dataset si multimile entitati.
- **Nivel de programare** – ce expune EDM ca o constructie programabila si poate fi folosita in limbajele de programare ceea ce inseamna generare automata de cod pentru clasele CLR ce expun aceleasi proprietati ca o entitate, permitand instantierea entitatilor ca obiecte .NET sau expun entitatile ca servicii web.



## 2.4.2 Modele de programare

- Modelul de programare **centrat pe baza de date**, presupune ca baza de date este creata si apoi se genereaza modelul logic ce contine tipurile folosite in logica aplicatiei. Acest lucru se face folosind mediul de dezvoltare VStudio. Se genereaza clasele POCO (EF versiune mai mare ca 4 si VS 2012/VS 2013) si fisierele necesare pentru nivelul conceptual, nivelul de mapare si nivelul de memorare.
- Aplicatia **centrata pe model** poate fi dezvoltata alegand una din variantele:
  - Code first.
  - Model design first.

In cazul **Code First**, dezvoltatorul scrie toate clasele (POCO) modelului si clasa derivata din **DbContext** cu toate entitatile necesare si apoi cu ajutorul mediului de dezvoltare se creaza si genereaza baza de date, tabelele din baza de date si informatiile aditionale necesare pentru EF.

In cazul **Model Design First**, mediul de dezvoltare permite dezvoltarea unei diagrame a modelului aplicatiei si pe baza acesteia se va crea si genera baza de date, tabelele din baza de date si informatiile aditionale necesare pentru EF.

ADO.NET clasic presupune obiecte **DataReader/ DataAdapter** pentru a citi informatii din baza de date si obiecte **Command** pentru a executa insert, update, delete, etc. **DataAdapter** din ADO.NET este folosit pentru **DataSet** si nu are echivalent in Entity Framework. In EF randurile si coloanele din tabele sunt returnate ca obiecte si nu se foloseste in mod direct **Command**, se translateaza datele din forma tabelara in obiecte.

EF foloseste un model numit **Entity Data Model(EDM)**, dezvoltat din **Entity Relationship Modeling(ERM)**. Conceptele principale introduse de EDM sunt:

- **Entity**: entitatile sunt instante ale tipului Entity (de exemplu Customer, Order). Acestea reprezinta structura unei inregistrari identificata printr-o cheie. Entitatile sunt grupate in multimi de Entity (Entity-Sets).
- **Relationship**: relatiile asociaza entitatile si sunt instante ale tipurilor Relationship (de exemplu Orderpostat de Customer). Relatiile sunt grupate in multimi de relatii – Relationship-Sets.

EDM suporta diverse constructii ce extind aceste concepte de entitate si relatie:

- **Inheritance**: tipurile entitate pot fi definite astfel incat sa fie derivate din alte tipuri. Mostenirea in acest caz este una structurala, adica nu se mosteneste comportarea ci numai structura tipului entitate de baza.; in plus la aceasta mostenire a structurii, o instanta a tipului entitate derivat satisface relatia "is a".
- **Tipuri complexe**: EDM suporta definirea tipurilor complexe si folosirea acestora ca membri ai tipurilor entitate. De exempluse poate defini tipul

(clasa) Address ce are proprietatile Street, City si Telephone si apoi sa folosim o proprietate de tip Address in tipul entitate Customer.

ERM defineste o schema a entitatilor si a relatiilor dintre acestea. Entitatile definesc schema unui obiect, dar nu si comportarea acestuia. Entitatea este asemanatoare cu schema unei tabele din baza de date numai ca aceasta descrie schema obiectelor problemei de rezolvat -pe scurt modelul.

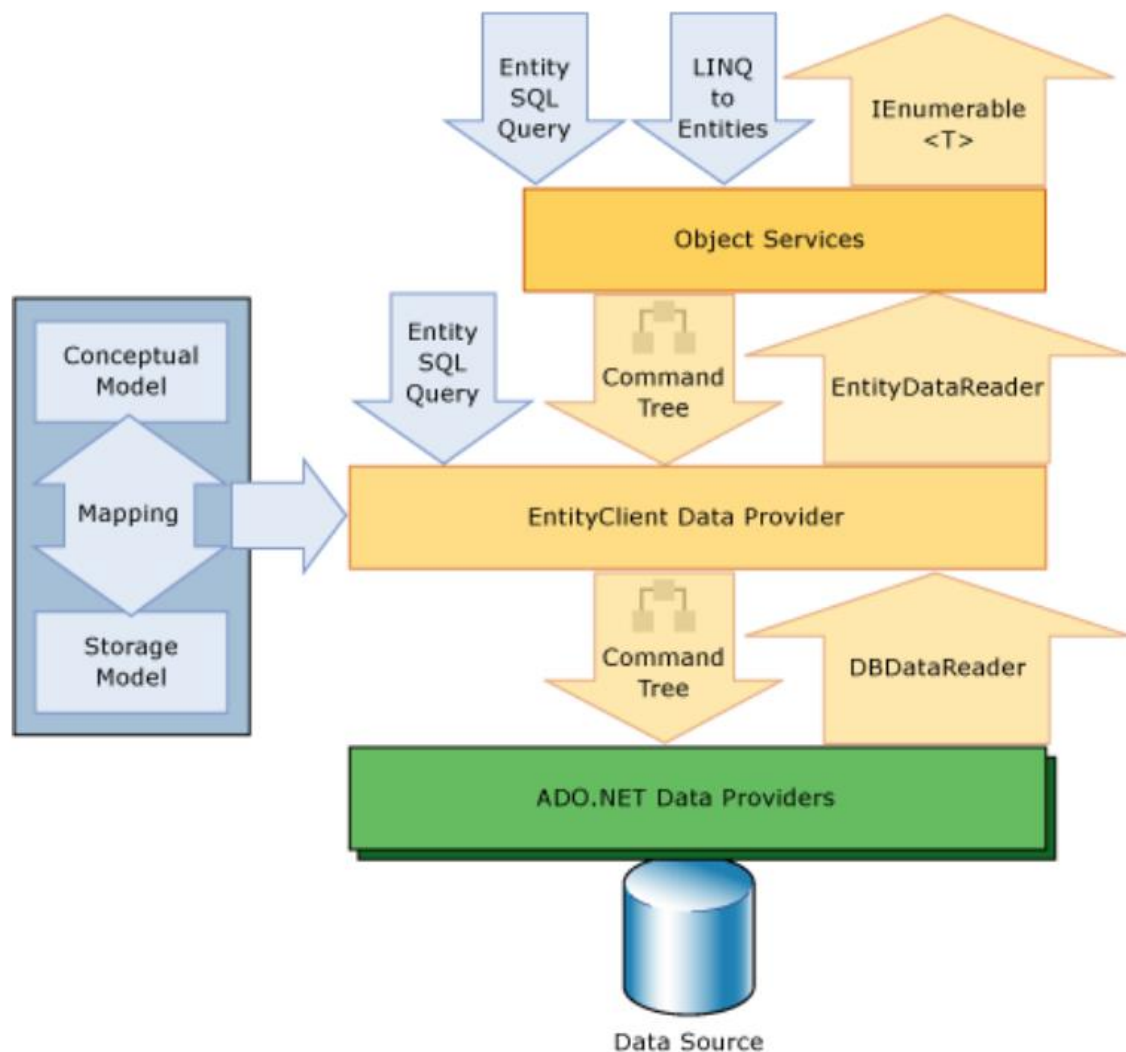


Fig. 5 Entity Framework

## 2.5 LINQ

### 2.5.1 Ce este?

LINQ (Language Integrated Query) este sintaxa de interogare în C # şi VB.NET pentru a prelua date din diferite surse şi formate. Este integrat în C # sau VB, eliminând astfel nepotrivirea dintre limbajele de programare şi bazele de date, precum şi oferind o singură interfaţă de interogare pentru diferite tipuri de surse de date.

De exemplu, SQL este un limbaj de interogare structurat utilizat pentru salvarea şi preluarea datelor dintr-o bază de date. În acelaşi mod, LINQ este o sintaxă de interogare structurată construită în C # şi VB.NET pentru a prelua date din diferite tipuri de surse de date, cum ar fi colecţiile, ADO.Net DataSet, XML Docs, serviciul web şi MS SQL Server şi alte baze de date. [9]

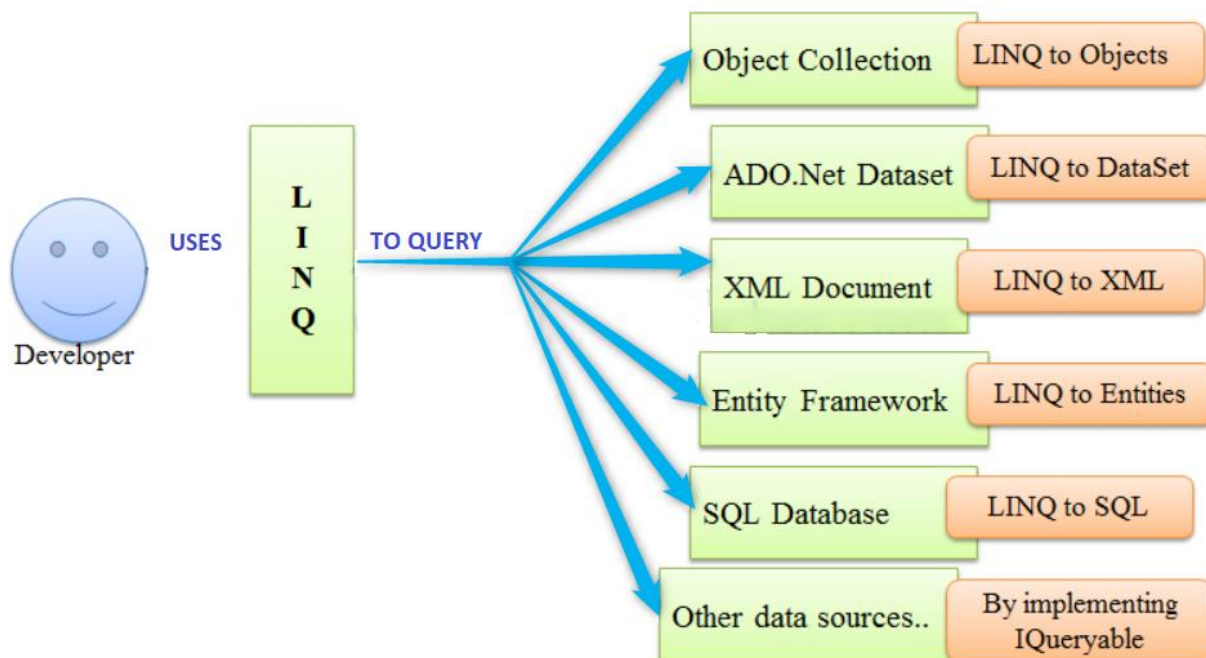


Fig. 6 Utilizare LINQ

Interogările LINQ returnează rezultatele ca obiecte. Vă permite să utilizaţi o abordare orientată pe obiect pe setul de rezultate şi să nu vă faceţi griji cu privire la transformarea formatelor diferite de rezultate în obiecte.

#### Exemplu: interogare LINQ pentru Array

```
// Data source
string[] names = {"Bill", "Steve", "James", "Mohan" };

// LINQ Query
var myLinqQuery = from name in names
                  where name.Contains('a')
                  select name;

// Query execution
foreach(var name in myLinqQuery)
    Console.Write(name + " ");
```

Nu se obţine rezultatul unei interogări LINQ până nu se execută. Interogarea LINQ poate fi executată în mai multe moduri, aici s-a folosit *foreach* pentru a executa interogarea stocată în *myLinqQuery*. *foreach* executa interogarea de pe sursa de date şi pentru a obţine rezultatul şi apoi peste setul reiterează rezultate.

Astfel, fiecare interogare LINQ trebuie să interogheze la un fel de surse de date, dacă poate fi tablou, colecţii, XML sau alte baze de date. După scrierea interogării LINQ, aceasta trebuie să fie executată pentru a obţine rezultatul.

## 2.5.2 Folosire LINQ API

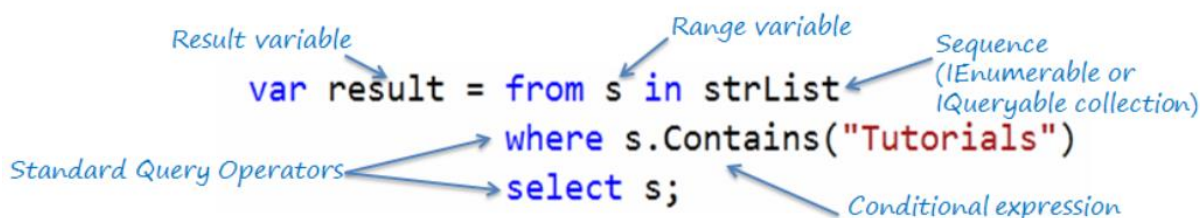
- Utilizaţi spaţiul de nume **System.Linq** pentru a utiliza LINQ.
- API LINQ include două clase statice principale **Enumerable & Queryable**.
- Clasa statică Enumerable include metode de extensie pentru clase care implementează interfaţa IEnumerable <T>.
- IEnumerable <T> tip de colecţii sunt colecţii în memorie precum List, Dicţionar, SortedList, Queue, HashSet, LinkedList.
- Clasa de interogare statică include metode de extensie pentru clase care implementează interfaţa IQueryable <T>.
- Furnizorul de interogare de la distanţă implementează, de exemplu, Linq-la-SQL, LINQ-la-Amazon etc.

## 2.5.3 Query Syntax

Sintaxa de interogare este similară cu SQL (Structured Query Language) pentru baza de date. Este definit în codul C # sau VB.

```
// string collection
IList<string> stringList = new List<string>() {
    "C# Tutorials",
    "VB.NET Tutorials",
    "Learn C++",
    "MVC Tutorials" ,
    "Java"
};
```

```
// LINQ Query Syntax
var result = from s in stringList
              where s.Contains("Tutorials")
              select s;
```



## 2.5.4 Method Syntax

Această sintaxă utilizează metode de extensie incluse în enumerable sau Queryable clasa statică, similar cu modul în care ar apela metoda de extindere a oricărei clase.

```
// string collection
IList<string> stringList = new List<string>() {
    "C# Tutorials",
    "VB.NET Tutorials",
    "Learn C++",
    "MVC Tutorials" ,
    "Java"
};

// LINQ Query Syntax
var result = stringList.Where(s => s.Contains("Tutorials"));
```

```
var result = strList.Where(s => s.Contains("Tutorials"));
```

Extension method      Lambda expression

## 2.6 SQL SERVER

### 2.6.1 Ce este?

SQL Server este un sistem relațional de gestionare a bazelor de date (RDBMS) dezvoltat de Microsoft. Este conceput și dezvoltat în primul rând pentru a concura cu baza de date **MySQL** și **Oracle**.

SQL Server acceptă ANSI SQL, care este limbajul standard SQL (Structured Query Language). Cu toate acestea, SQL Server vine cu propria implementare a limbajului SQL, T-SQL (Transact-SQL).

**T-SQL** este un limbaj de proprietate Microsoft cunoscut sub numele de Transact-SQL. Oferă capacități suplimentare de declarare a variabilei, tratarea excepțiilor, procedura stocată etc.

**SQL Server Management Studio (SSMS)** este instrumentul principal de interfață pentru SQL Server și acceptă atât medii pe 32 de biți cât și pe 64 de biți.

### 2.6.2 Componente cheie si servicii

**Motorul de baze de date:** această componentă gestionează stocarea, procesarea rapidă a tranzacțiilor și securizarea datelor.

**SQL Server:** Acest serviciu porneşte, opreşte, întrerupe şi continuă o instanţă a Microsoft SQL Server. Numele executabil este sqlservr.exe.

**SQL Server Agent:** Îndeplineşte rolul de Scheduler Task. Poate fi declanşat de orice eveniment sau în funcţie de cerere. Numele executabil este sqlagent.exe.

**Browser SQL Server:** Acesta ascultă solicitarea primită şi se conectează la instanţa dorită a serverului SQL. Numele executabil este sqlbrowser.exe.

**Căutare cu text complet SQL Server:** Aceasta permite utilizatorului să ruleze interogări de text complet împotriva datelor de caracter din tabelele SQL. Numele executabil este fdlauncher.exe.

**SQL Server VSS Writer:** Aceasta permite copierea de rezervă şi restaurarea fişierelor de date atunci când serverul SQL nu rulează. Numele executabil este sqlwriter.exe.

**Servicii de analiză SQL Server (SSAS):** oferă funcţii de analiză a datelor, extragere de date şi învăţare automată. Serverul SQL este integrat cu limbajul R şi Python pentru analiză avansată. Numele executabil este msmdsrv.exe.

**Servicii de raportare SQL Server (SSRS):** oferă funcţii de raportare şi capacităţi de luare a deciziilor. Include integrarea cu Hadoop. Numele executabil este ReportingServicesService.exe

**Servicii de integrare SQL Server (SSIS):** oferite funcţii de extragere-transformare şi încărcare a diferitelor tipuri de date de la o sursă la alta. Poate fi vizualizată ca transformând informaţia brută în informaţii utile. Numele executabil este MsDtsSrvr.exe

### 2.6.3 Instanţe SQL Server

SQL Server permite să ruleze mai multe servicii simultan, fiecare serviciu având conectări separate, porturi, baze de date etc. Acestea sunt împărţite în două:

- Instanţe primare
- Instanţe numite.

Există două modalităţi prin care se poate accesa instanţa primară. În primul rând, se poate folosi numele serverului. În al doilea rând, se poate folosi adresa sa IP. Instanţele numite sunt accesate prin adăugarea unui nume de instanţă şi a unei versiuni anterioare.

De exemplu, pentru conectarea la o instanţă numită *xyx* pe serverul local, ar trebui să se utilizeze *127.0.0.1 \ xyz*. Din SQL Server 2005 şi versiuni ulterioare, se poate să ruleze simultan până la 50 de instanţe pe un server. [10]

De reţinut că, deşi se poate să fie mai multe instanţe pe acelaşi server, doar una dintre ele trebuie să fie instanţa implicită, în timp ce restul trebuie să fie numite

instanţe. Se pot rula toate instanţele concomitent şi fiecare instanţă rulează independent de celelalte instanţe.

Următoarele sunt avantajele instanţelor SQL Server:

### **1. Pentru instalarea diferitelor versiuni pe un singur aparat**

Puteţi avea diferite versiuni ale SQL Server pe o singură maşină. Fiecare instalaţie funcţionează independent de celelalte instalaţii.

### **2. Pentru reducerea costurilor**

Instanţele ajută la reducerea costurilor de operare SQL Server, în special în achiziţionarea licenţei SQL Server. Se poate obţine servicii diferite din diferite instanţe, deci nu este necesară achiziţionarea unei licenţe pentru toate serviciile.

### **3. Pentru întreţinerea mediilor de dezvoltare, producţie şi testare separat**

Acesta este principalul beneficiu de a avea multe instanţe SQL Server pe o singură maşină. Puteţi utiliza diferite cazuri pentru dezvoltare, producţie şi testare.

### **4. Pentru reducerea problemelor de bază de date temporare**

Când sunt toate serviciile care rulează pe o singură instanţă SQL Server, există şanse mari să fie probleme cu problemele, în special problemele care continuă să se repete. Atunci când aceste servicii sunt rulate în diferite cazuri, se poate evita astfel de probleme.

### **5. Pentru separarea privilegiilor de securitate**

Când diferite servicii rulează pe diferite instanţe SQL Server, se poate să se concentreze pe securizarea instanţei care rulează cel mai sensibil serviciu.

### **6. Pentru menţinerea unui server de aşteptare**

O instanţă SQL Server poate eşua, ceea ce duce la o întrerupere a serviciilor. Acest lucru explică importanţa ca un server de aşteptare să fie introdus în cazul în care serverul curent nu reuşeşte. Acest lucru poate fi realizat cu uşurinţă folosind instanţe SQL Server.

## **2.7 \* HELP PAGE**

Pachetul HelpPage joacă un rol foarte important în viaţa unui dezvoltator. Este util să creaţi o pagină de ajutor, deoarece fiecare dezvoltator ar trebui să ştie să apeleze API-ul Web şi să ştie care este comportamentul metodei şi să cunoască câteva indicii despre metodă sau apel.

API-ul Web are implicit o pagină de ajutor. Dacă dorim să extindem documentul paginii de ajutor de la înţelegerea utilizatorului, să vedem procedura şi să creăm o pagină de ajutor pas cu pas

# ASP.NET Web API Help Page

## Introduction

Provide a general description of your APIs here.

## Mansions

---

### API

---

[GET api/Mansions](#)

---

[GET api/Mansions/{id}](#)

---

[POST api/Mansions](#)

---

[DELETE api/Mansions/{id}](#)

---

[GET api/Mansions/Get](#)

---

[GET api/Mansions/Get/{id}](#)

---

[POST api/Mansions/Post](#)

---

[DELETE api/Mansions/Delete/{id}](#)

---

**Fig. 7 Help Page – lista**



### Body Parameters

[ApartmentViewModel](#)

Name	Description	Type
ApartmentId		integer
Surface		integer
Number		integer
Floor		integer

### Request Formats

application/json, text/json

Sample:

```
{
  "apartmentId": 1,
  "surface": 1,
  "number": 2,
  "floor": 3
}
```

**Fig. 8 Help page – detaliere request**

### 3. Descriere tehnologii FrontEnd

#### 3.1 HTML

HTML reprezintă scheletul oricărei pagini Web, el descriind formatul primar în care documentele sunt vizualizate şi distribuite pe Internet. HTML nu este un limbaj de programare, deci nu veţi lucra aici cu variabile, expresii, tipuri de date, structuri de control.

HTML este un limbaj descriptiv, prin care sunt descrise elementele structurale ale paginii Web: titluri, liste, tabele, paragrafe, legături cu alte pagini, precum şi aspectul pe care îl are pagina din punct de vedere grafic. Fiind un limbaj de marcare, el utilizează etichete (marcaje) ce dau indicaţii browsere-lor cu privire la ierarhizarea şi afişarea informaţiilor.

**Un document HTML este structurat astfel:**

- zona head (antet) cu etichetele `<head> </head>`
- zona body (corp) cu etichetele `<body> </body>`
- sau `<frameset> </frameset>`

Codul HTML prezentat mai jos utilizează următoarele marcaje :

`<p>` -pentru definirea unui paragraph  
`<h...(1,2->6)>` - pentru headere  
`<hr>` -pentru trasarea unei linii orizontale  
`<font>` -pentru formatarea fontului  
`` -pentru inserarea unei imagini  
`<i>` -pentru definirea unui stil înclinat  
`<a href="...">` - folosit pentru hiperlinkuri  
`<br/>` - linie noua

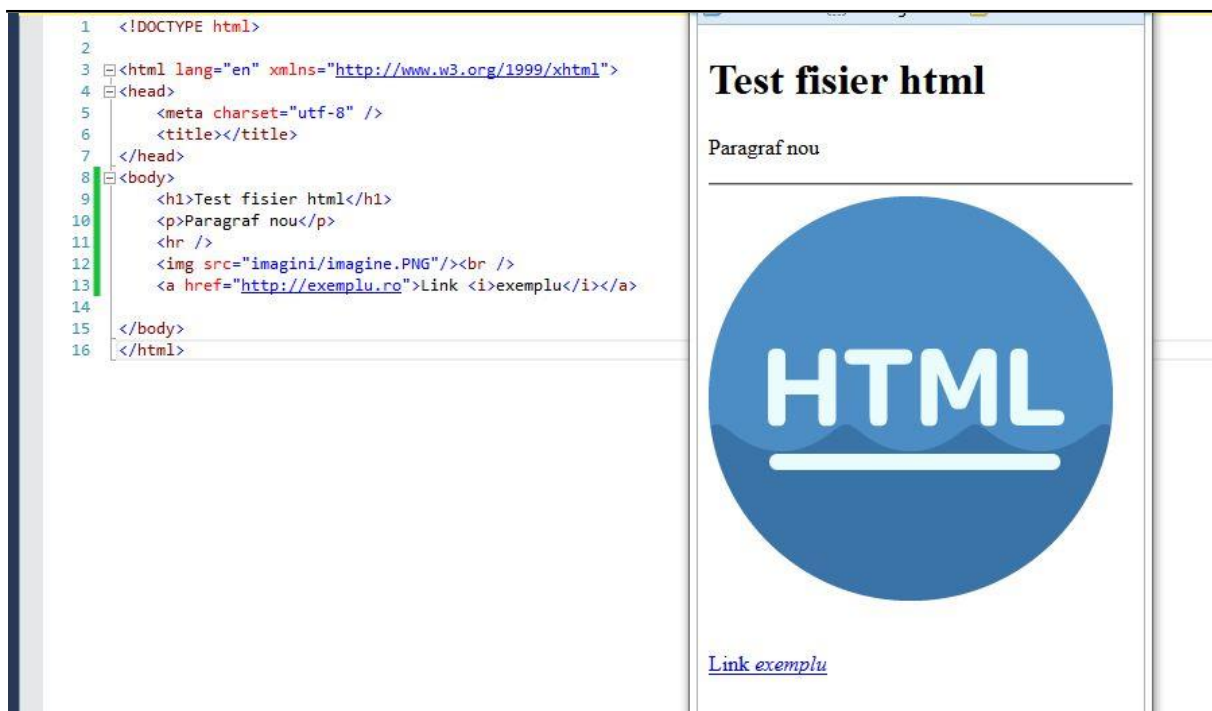


Fig. 9 Model cod HTML

Element important în HTML este şi **tabelul**. Tabelele permit construirea unei reţele dreptunghiulare de domenii, fiecare domeniu având propriile opţiuni de formatare: culoarea fondului, culoarea textului, alinierea textului etc. Prezentarea datelor sub formă de tabele oferă importante avantaje: claritate, sistematizare, posibilităţi de comparare.

Marcarea unui tabel se efectuează printr-un tag de introducere a tabelului şi de definire a atributelor globale. Acesta include şi definiţiile pentru liniile şi celulele tabelului.

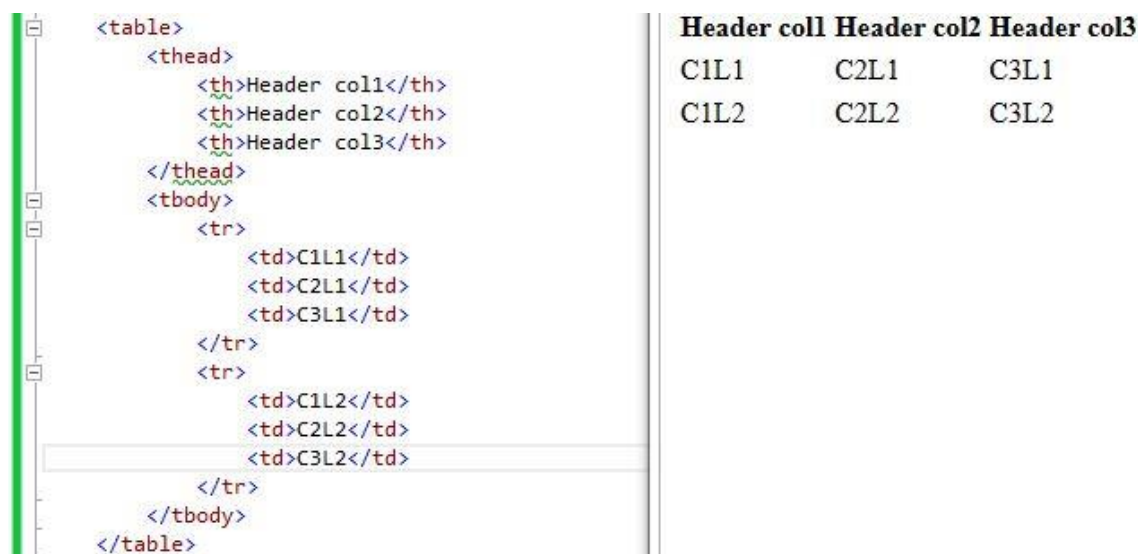
Sintaxa generală pentru declararea unui tabel este:

```
<table>
<caption>...</caption>
<TR><TH><TH> ... </TR>
<TR><TD><TD> ... </TR>
...
<TR><TD><TD> ... </TR>
</table>
```

Unde etichetele:

- <table></table> - delimitează tabelul
- <tr></tr> - delimitează o linie a tabelului
- <td></td> - delimitează o celulă de date a tabelului
- <th></th> - delimitează o celulă a primei linii din tabel (a capului de tabel)
- <thead></thead> - delimitează zona pentru headerul tabelului
- <tbody></tbody> - delimitează zona pentru continutul tabelului

Exemplu tabel:



```
<table>
  <thead>
    <th>Header col1</th>
    <th>Header col2</th>
    <th>Header col3</th>
  </thead>
  <tbody>
    <tr>
      <td>C1L1</td>
      <td>C2L1</td>
      <td>C3L1</td>
    </tr>
    <tr>
      <td>C1L2</td>
      <td>C2L2</td>
      <td>C3L2</td>
    </tr>
  </tbody>
</table>
```

Header col1	Header col2	Header col3
C1L1	C2L1	C3L1
C1L2	C2L2	C3L2

Fig. 10 Tabel in HTML

Formularul este şi el un element important oferit de HTML. Un formular este un ansamblu de zone active alcătuit din casete combinate, câmpuri de editare, butoane radio, butoane de comandă etc. Formularele asigură construirea unor pagini Web care permit utilizatorilor să introducă informaţii şi să le transmită serverului. O sesiune cu o pagină Web ce conţine un formular cuprinde două etape:

- utilizatorul completează formularul şi trimite serverului (prin apăsarea butonului de expediere) datele înscrise în formular
- o aplicaţie dedicată de pe server (un script) analizează informaţiile transmise şi, în funcţie de configuraţia scriptului, fie stochează datele într-o bază de date, fie le transmite la o adresă de mail indicată de dumneavoastră. Dacă este necesar, serverul poate expedia şi un mesaj de răspuns utilizatorului.

Un formular este definit într-un bloc delimitat de etichetele **<form>** **</form>**. În interiorul blocului sunt incluse:

- elementele formularului, în care vizitatorul urmează să introducă informaţii,
- un buton de expediere, la apăsarea căruia, datele sunt transmise către server,
- opţional, un buton de anulare, prin care utilizatorul poate anula datele înscrise în formular.

Cele mai importante atribute ale etichetei **<form>** sunt:

- **action**: comunică browserului unde să trimită datele introduse în formular. În general valoarea atributului action este adresa URL a scriptului aflat pe serverul care primeşte datele formularului:
  - o `<form action="http://www.yahoo.com/cgi-bin/fisier.cgi">`
- **method**:
  - o **get** (valoarea implicită) - datele din formular sunt adăugate la adresa URL precizată de atributul action (nu sunt permise cantităţi mari de date)

- **post** - folosită cel mai des. În acest caz datele sunt expediate separat. Sunt permise cantităţi mari de date.

Majoritatea elementelor unui formular sunt definite cu ajutorul etichetei **<input>**. Aceasta este utilizată împreună cu următoarele atribute:

Atribut	Valoare	Element introdus	Semnificaţie
type	text	casetă de text	permite introducerea unui şir de caractere pe un singur rând
	radio	buton radio	permite alegerea, la un moment dat, a unei singure variante din mai multe posibile
	checkbox	căsuţă de validare	permite selectarea sau deselectarea unei opţiuni
	button	buton de comandă	permite declanşarea unei operaţii atunci când utilizatorul execută click sau dublu-click pe suprafaţa acestuia
	submit	buton de transmitere	este butonul a cărui activare declanşează operaţiunea de trimitere a datelor către server
	reset	buton de resetare	este butonul a cărui activare readuce controalele din formular la valorile lor iniţiale
	Image	image	permite înlocuirea unui buton submit cu o imagine specificată
	password	casetă de text specială	este similară controlului text, diferenţele constând în faptul că datele introduse de utilizator vor fi afişate printr-un caracter Țmască" (ex: "***") pentru a oferi un anumit grad de confidenţialitate. Este folosit de obicei la introducerea unor parole.
	Hidden	câmp ascuns	permite introducerea în formular a unui camp ascuns
name	permite ataşarea unui nume fiecărui element al formularului		
value	permite atribuirea unei valori iniţiale unui element al formularului		
checked	are rolul de a preseta o anumită opţiune, pe care însă utilizatorul o poate schimba, dacă doreşte		
size	setează numărul de caractere al căsuţei de text afişate		
maxlength	setează numărul maxim de caractere al căsuţei de text afişate		

Cu ajutorul etichetei **<textarea> </textarea>** puteţi insera în pagină o casetă de text multilinie care permite vizitatorului să introducă un text mai lung, care se poate întinde pe mai multe linii.

**<textarea name="adresa" rows=2 cols=30></textarea>**

În HTML5 formularul are si alte tipuri cum ar fi de datetime, range etc.

Exemplu: Fig. 11 Formular HTML conţine elemente de mai multe tipuri încadrate într-un formular unic. Pentru alinierea elementelor utilizate pentru informaţiile personale am utilizat un tabel.

```
8 <form action="mailto: text@test.com" method="post">
9   Text: <input type="text" /><br/>
10   Radio: <br/>
11   <input type="radio" value="Raspuns1" />Raspuns1<br/>
12   <input type="radio" value="Raspuns2" />Raspuns2<br/>
13   Parola: <input type="password" /><br/>
14   <input type="button" value="Inapoi" />
15   <input type="submit" value="Trimite" />
16 </form>
```

Text:

Radio:

☒ Raspuns1

☒ Raspuns2

Parola:

Fig. 11 Formular HTML

## Ce este nou in HTML5?

In HTML 5 au fost adaugate tag-uri noi pentru a usura introducerea fisierelor multimedia in pagina. Intre ele enumeram: fisiere audio, video, grafica, documente interactive etc.

Alte elemente noi in HTML 5 sunt tag-urile <header>, <nav> <figure> si <footer>, fiecare marcand o zona concreta.

Am adaugat ma jos o lista cu noile tag-uri introduse odata cu HTML 5

- <article> marcheaza un articol
- <aside> marcheaza un continut aparte fata de continutul paginii, dar care are legatura cu el.
- <audio> marcheaza introducerea de continut audio
- <canvas> marcheaza introducerea de continut grafic
- <command> marcheaza un buton de comanda
- <datalist> marcheaza un meniu drop-down
- <details> marcheaza detaliile unui element
- <dialog> marcheaza un dialog, o conversatie
- <embed> marcheaza continut interactiv extern sau introducerea unui plugin
- <figure> marcheaza un grup de elemente care au legatura unul cu celalalt si care pot fi considerate in pagina, continut de sine statator.
- <footer> marcheaza sectiunea footer a pagini
- <header> marcheaza sectiunea header a pagini
- <hgroup> marcheaza marcheaza o sectiune a pagini
- <keygen> marcheaza un cod generat automat intr-un formular
- <mark> marcheaza text evidentiat
- <meter> marcheaza valoarea unei unitati de masura cunoscute
- <nav> marcheaza o bara de navigare cu linkuri
- <output> marcheaza diferite tipuri de rezultate ale unui script oarecare.
- <progress> marcheaza o bara de progres fie ea grafica sau numerica
- <rp> defineste continut care va fi afisat in cazul in care browser-ul nu supotra tag-ul ruby

---

<rt>	defineste o regula sau o explicatie pentru tagul ruby
<ruby>	folosit impreuna cu caracterele asiatice
<section>	marcheaza o sectiune oarecare (header, footer, bara de navigare, capitole sau orice alta sectiune)
<source>	marcheaza sursa fisierului multimedia
<time>	marcheaza ora / data
<video>	marcheaza introducerea unui video

Pe de altă parte tag-urile scoase din HTML 5 sunt dupa cum urmează:

acronim  
applet  
basefont  
big  
center  
dir  
font  
frame  
frameset  
noframes  
s  
strake  
tt  
u  
xmp

Deşi multe dintre tag-urile enumerate mai sus nu fac altceva decat sa incapsuleze conţinut, sunt totuşi câteva dintre ele, precum canvas si altele, care sunt destul de interesante. Voi reveni in scurt timp cu explicaţii si exemple pentru fiecare tag html 5 care merită prezentat. [11]

## 3.2 Stiluri CSS

### 3.2.1 Ce este CSS?

CSS este acronimul pentru Cascading Style Sheets. Este un limbaj (**style language**) care defineşte "**layout-ul**" pentru documentele HTML. CSS acoperă culori, font-uri, margini (borders), linii, înălţime, lăţime, imagini de fundal, poziţii avansate şi multe alte optiuni.[6]

HTML este de multe ori folosit necorespunzător pentru a crea layoutul site-urilor de internet. CSS oferă mai multe opţiuni, este mai exact şi sofisticat. În plus, este suportat de toate browserele actuale.[3]

Sintaxa CSS este alcătuită din trei părţi: un selector, o proprietate şi o valoare, în următorul format:

selector { proprietate: valoare}

Selectorul este reprezentat de elementul (tag-ul) căruia se doreşte să i se aplice un anumit stil, proprietatea este atributul care se doreşte a fi schimbat şi fiecare proprietate poate lua o anumită valoare. Proprietatea şi valoarea sunt separate de doua puncte (:) şi sunt încadrate de acolade {}. De exemplu:

```
body {color: blue}
```

Dacă se doreşte modificarea multor proprietăţi a aceluiaşi element, trebuie să se separe fiecare proprietate cu punct şi virgulă (;). Exemplul de mai jos arată cum trebuie definit un paragraf aliniat pe centrul paginii iar textului i se aplică culoarea roşie[4]:

```
p {  
    text-align: center;  
    color: red  
}
```

De asemenea, ca şi în HTML, pentru a putea introduce şi explicaţii ce pot fi utile ulterior, în CSS pot fi inserate comentarii ce vor fi ignorate de către browser. Comentariile încep cu (/\*) şi se încheie cu (\*/), ca în exemplul următor:

```
/* Acesta este un comentariu */
```

### 3.2.2 Unde se definesc stilurile?

Stilurile pentru o pagină pot fi definite **în partea de Head**(Fig. 12 Stil definit in HEAD) a documentului html, pot fi definite într-un **fişier css extern**, ce poate fi chemat tot din partea de head a paginii, sau **poate fi aplicat un stil diferit în partea Body** a fişierului html, la fiecare tag html în parte ().[5]

```
1  <!DOCTYPE html>  
2  
3  <html lang="en" xmlns="http://www.w3.org/1999/xhtml">  
4  <head>  
5      <title>Stil definit in head</title>  
6      <style type="text/css">  
7          p {  
8              font-family: Arial;  
9              font-size: 14px;  
10             color: #003300;  
11         }  
12     </style>  
13 </head>  
14 <body>  
15  
16     Paragraf la care se va aplica stilul definit mai sus!  
17  
18 </body>  
19 </html>
```

Fig. 12 Stil definit in HEAD



În partea **body** a fişierului html, se poate adăuga un stil diferit la fiecare tag html(Fig. 5), ba chiar şi suprascrie un stil definit în partea **head**.

```
<body>  
  <p style="font-family:Arial; color:#478600;font-size:14px;">  
    Paragraf care a fost stilizat in interiorul tagului p!  
  </p>  
</body>
```

În acest exemplu s-a folosit Css inline pentru a adăuga un stil. Cu aceasta metodă se poate adăuga un stil cu css la orice tag html, şi se poate suprascrie stilul definit în partea head. Pentru a adăuga un stil diferit faţă de restul documentului, s-a folosit atributul style pentru a redefini, a adăuga stilul respectiv. Parametri ca şi în cazul stilurilor definite în head, sunt separate prin (;) punct şi virgulă.

**Forma: <tag-html style="...." >**

Pentru a ţine toate stilurile într-un fişier extern, tot ce trebuie respectat, este să se definească stilurile într-un fişier cu extensia css. exemplu: stiluri.css ce poate fi invocat, chemat din partea de head a documentului[5].

Exemplu:

```
<head>  
  <link rel="stylesheet" href="stiluri.css" />  
</head>
```

### 3.2.3 Bootstrap

Bootstrap este un framework pentru UI alcătuit din stiluri şi fişiere .js. Este în momentul de faţă cel mai utilizat framework pentru dezvoltarea interfeţelor web devenind foarte rapid standardul în crearea template-urilor pentru principalele sisteme CMS cum sunt WordPress şi Joomla.

Se poate spune că Bootstrap este un instrument utilizat pentru a gestiona cât mai bine faza iniţială a unui proiect deoarece se poate conta pe o serie de componente care pot fi reutilizate şi personalizate oferindu-ne o bază solidă de pornire a proiectelor pentru a nu fi nevoie să se începă de la zero. **Cea mai importantă trăsătură** a acestui framework este aceea ca permite realizarea de site-uri web responsive, care se adaptează la orice rezoluţie de dispozitiv: desktop, tablete si telefoane mobile.[8]

Bootstrap foloseşte elemente de HTML şi CSS deoarece e nevoie în paginile unde este folosit să se codeze folosind doctype HTML5 si CSS3 ca stil.

```
<!DOCTYPE html>  
<html lang="en">  
...  
</html>
```

Pentru a asigura o redare corectă folosind zoom-ul dispozitivului, e nevoie de adăugarea următorului tag:

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

Bootstrap vine cu o structură clară de realizare a unei pagini folosind elemente de tip container, linie etc pentru a realiza un grid(layout-ul) al paginii ce se autoalineaază în funcţie de rezoluţia dispozitivului bazat pe linii şi coloane.[9]

Principalele reguli de funcţionalitate sunt:

- liniile (.row) trebuie introduse în .container (cu lăţime fixă) sau .container-fluid (lăţime întreagă a ecranului)
- grupurile de coloane sunt introduse în linii
- clasele predefinite ca .row şi .col-xs-4 sunt valabile pentru o formatare rapidă a layoutului
- layoutul e împărţit în 12 coloane. De exemplu pentru a avea 3 coloane se foloseşte clasa .col-xs-4 ( $12/4=3$ ), dacă se doreşte pe 2 coloane atunci se foloseşte clasa .col-xs-6
- coloanele sunt realizate pentru mai multe tipuri de dispozitive. De exemplu un dispozitiv cu rezoluţie mare va folosi clase ca col-lg-\*, un dispozitiv cu rezoluţie medie va folosi clase ca col-md-\* ş.a.m.d (exemplu Fig. 13 Coloanele în Bootstrap în funcţie de rezoluţia dispozitivului)[12]

	Extra small devices Phones (<768px)	Small devices Tablets (≥768px)	Medium devices Desktops (≥992px)	Large devices Desktops (≥1200px)
<b>Grid behavior</b>	Horizontal at all times	Collapsed to start, horizontal above breakpoints		
<b>Container width</b>	None (auto)	750px	970px	1170px
<b>Class prefix</b>	.col-xs-	.col-sm-	.col-md-	.col-lg-
<b># of columns</b>	12			
<b>Column width</b>	Auto	~62px	~81px	~97px
<b>Gutter width</b>	30px (15px on each side of a column)			
<b>Nestable</b>	Yes			
<b>Offsets</b>	Yes			
<b>Column ordering</b>	Yes			

**Fig. 13** Coloanele în Bootstrap în funcţie de rezoluţia dispozitivului

Un element poate avea mai multe clase pentru diferite dispozitive. De exemplu dacă pentru dispozitivele cu rezoluţie mare layoutul se vrea a fi împărţit în 4 coloane iar pentru dispozitivele cu rezoluţie medie se vrea să fie doar pe două coloane iar pentru cele cu rezoluţie minimă se vrea doar pe o coloană se pot introduce astfel clasele:

```
<div class="col-lg-3 col-md-6 col-xs-12">Aliniere</div>
```

O bună prezentare a ce a fost explicat mai sus(Fig. 14 Coloane Bootstrap):

.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1
.col-md-8									.col-md-4			
.col-md-4				.col-md-4					.col-md-4			
.col-md-6						.col-md-6						

Fig. 14 Coloane Bootstrap

Stilurile oferite de bootstrap tratează designul pentru: tabele, meniuri, paragrafe, headere etc cât şi pentru formulare, unde, ajutat de fişierele de javascript oferă şi o funcţionalitate aparte pentru meniuri(când sunt pe ecrane mici meniul poate dispărea şi să fie accesat doar de la un buton exemplu Fig. 15 Meniu in Bootstrap15).

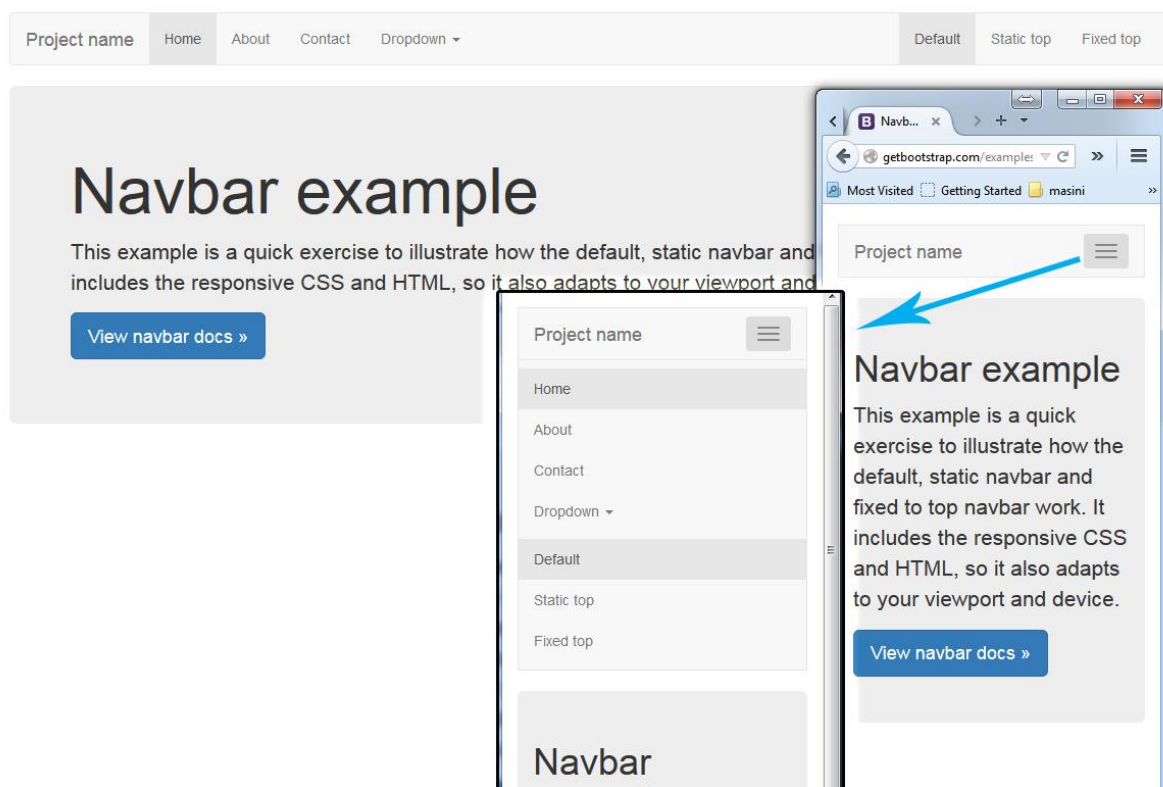


Fig. 15 Meniu in Bootstrap

## 3.4 Javascript

### 3.4.1 Ce este javascript?

JavaScript a fost dezvoltat prima dată de către firma Netscape, cu numele de Live Script, un limbaj de script care extindea capacităţile HTML, oferă o alternativă parţială la utilizarea unui număr mare de scripturi CGI pentru prelucrarea informaţiilor din formulare şi care adaugă dinamism în paginile web. După lansarea limbajului Java, Netscape a început să lucreze cu firma Sun, cu scopul de a crea un limbaj de

script cu o sintaxă şi semantică asemanatoare cu a limbajului Java, şi din motive de marketing numele noului limbaj de script a fost schimbat în "JavaScript".

JavaScript a apărut din nevoia ca logica şi inteligenţa să fie şi pe partea de client, nu doar pe partea de server. Dacă toată logica este pe partea de server, întreaga prelucrare este făcută la server, chiar şi pentru lucruri simple, asa cum este validarea datelor. Astfel, JavaScript îl înzestrează pe client şi face ca relaţia să fie un adevarat sistem client-server.

Limbajul HTML oferă autorilor de pagini Web o anumită flexibilitate, dar statică. Documentele HTML nu pot interacţiona cu utilizatorul în alt mod mai dinamic, decât pune la dispoziţia acestuia legături la alte resurse (URL-uri). Crearea de CGI-uri (Common Graphics Interface) - [programe care ruleaza pe serverul Web şi care accepta informatii primite din pagina de web şi returneaza cod HTML] - a dus la îmbogăţirea posibilităţilor de lucru. Astfel, un pas important spre interactivizare a fost realizat JavaScript, care permite inserarea în paginile web a script-urilor care se executa în cadrul paginii web, mai exact în cadrul browser-ului utilizatorului, usurand astfel şi traficul dintre server şi client. De exemplu, într-o pagina pentru colectarea de date de la utilizator se pot adauga scripturi JavaScript pentru a valida corectitudinea introducerii şi apoi pentru a trimite serverului doar date corecte spre procesare.

Versiunile ulterioare de JavaScript şi diversele diferente specifice platformelor de operare au început să dea destule probleme programatorilor web şi astfel Netscape, Microsoft şi alţi distribuitori au fost de acord să predea limbajul unei organizaţii internaţionale de standardizare - ECMA ; aceasta a finalizat o specificaţie de limbaj, cunoscuta ca **ECMAScript**, recunoscuta de toţi distribuitorii. Deşi standardul ECMA este util, atât Netscape cât şi Microsoft au propriile lor implementări ale limbajului şi continua să extindă limbajul dincolo de standardul de baza. Pe lângă Jscript, Microsoft a introdus şi un concurent pentru JavaScript, numit VBScript, realizat pentru a usura patrunderea pe web a programatorilor VB. VBScript este un subset al limbajului Visual Basic. Cu toate acestea JavaScript a devenit cunoscut ca limbajul de scripting standard pentru web. În general se consideră că există zece aspecte fundamentale ale limbajului JavaScript pe care orice programator în acest limbaj ar trebui să le cunoască :

1. **JavaScript poate fi introdus în HTML** - De obicei codul JavaScript este gazduit în documentele HTML şi executat în interiorul lor. Majoritatea obiectelor JavaScript au etichete HTML pe care le reprezintă, astfel încât programul este inclus pe partea de client a limbajului. JavaScript foloseşte HTML pentru a intra în cadrul de lucru al aplicaţiilor pentru web.
2. **JavaScript este dependent de mediu** - JavaScript este un limbaj de scriptare; software-ul care rulează de fapt programul este browser-ul web (Firefox, Opera, Netscape Navigator, Internet Explorer, Safari, etc.) Este important să luăm în considerare această dependenţă de browser atunci când utilizăm aplicaţii JavaScript.
3. **JavaScript este un limbaj în totalitate interpretat** - codul scriptului va fi interpretat de browser înainte de a fi executat. JavaScript nu necesită compilări sau preprocesări, ci rămâne parte integrantă a documentului HTML. Dezavantajul acestui limbaj este că rularea durează ceva mai mult deoarece comenzile JavaScript vor fi citite de navigatorul Web şi procesate atunci când

user-ul apelează la acele funcţii (prin completare de formulare, apăsare de butoane, etc). Avantajul principal este faptul că se poate mult mai uşor să se actualizeze codul sursă.

4. **JavaScript este un limbaj flexibil** - în aceasta privinţă limbajul diferă radical de C++ sau Java. În JavaScript se poate declara o variabilă de un anumit tip, sau se poate lucra cu o variabilă deşi nu-i se cunoaşte tipul specificat înainte de rulare.
5. **JavaScript este bazat pe obiecte** - JavaScript nu este un limbaj de programare orientat obiect, ca Java, ci mai corect, este "bazat pe obiecte"; modelul de obiect JavaScript este bazat pe instanţă şi nu pe moştenire.
6. **JavaScript este condus de evenimente** - mare parte a codului JavaScript răspunde la evenimente generate de utilizator sau de sistem. Obiectele HTML, cum ar fi butoanele, sunt îmbunătăţite pentru a accepta handler de evenimente.
7. **JavaScript nu este Java** - Cele doua limbaje au fost create de companii diferite, motivul denumirii asemănătoare este legat doar de marketing.
8. **JavaScript este multifuncţional** - limbajul poate fi folosit într-o multitudine de contexte pentru a rezolva diferite probleme: grafice, matematice, şi altele.
9. **JavaScript evoluează** - limbajul evoluează, fapt pozitiv care însă poate genera şi probleme, programatorii trebuind să verifice permanent ce versiune să folosească pentru ca aplicaţiile să poată fi disponibile unui număr cât mai mare de utilizatori de browsere diferite.
10. **JavaScript acoperă contexte diverse** - programarea cu acest limbaj este îndreptată mai ales către partea de client, dar se poate folosi JavaScript şi pentru partea de Server. JavaScript este limbajul nativ pentru unele instrumente de dezvoltare web, ca Borland IntraBuilder sau Macromedia Dreamweaver. [13]

### 3.4.2 ECMAScript

- este un standard care defineşte funcţionalitatea şi sintaxa limbajului JS
- specificaţiile tehnice ale limbajului
- nu este adresat dezvoltatorilor
- prima versiune - ECMA-262 (1997)
- ECMAScript 5.1 (2011) – se găseşte în majoritatea browserelor
- ECMAScript 2015 (ES6) - se găseşte în unele browsere recente (ex. Chrome, Firefox...)
- ECMAScript 2015+ (ES Next) – se poate folosi cu un compilator (ex. babeljs)

Datorită faptului că JS este un limbaj interpretat, este nevoie de un interpretor care să ruleze codul JS. Acest interpretor de obicei este creat de vendorii de browsere, ceea ce-l face să difere de la un browser la altul (găsiţi aici câteva exemple). Aceste diferenţe pot face viaţa dezvoltatorilor foarte dificilă, astfel, ECMA International ne vine în ajutor dezvoltând un standard denumit şi ECMAScript.

### 3.4.3 Sintaxa

Pentru a rula cod de JS este nevoie de un interpretor cum ar fi browserul.

Se poate ajunge în consola browserului prin CTRL + Shift + J, de unde se poate rula cod de JS.

```
function greetMe(yourName) {  
    return 'Hello ' + yourName  
}
```

```
greetMe('World')
```

- sintaxa JS este inspirată în mare parte din Java, dar are şi influenţe din Awk, Perl şi Python.
- JS este **case-sensitive**
- JS suportă Unicode (UTF-16)
- HTML este **case-insensitive**
- se folosesc elementele din HTML fie în **lower case** fie în **camel case**
- clasele din HTML sunt **case-sensitive** (atât în CSS cât şi-n JS)
- spaţiile, taburile şi liniile noi sunt considerate un singur spaţiu (whitespace)
- afirmaţiile din JS pot fi finisate fie prin punct şi virgulă (;) fie prin linie nouă (enter)
- în general, (;) este considerată o practică bună în JS

```
// JS este case-sensitive  
var userName = 'Victor';  
console.log(username); // eroare: username nu este definit  
console.log(userName); // log: Victor  
  
// linia nouă nu mereu finisează afirmaţia  
var a = 10  
+ 1  
+ 2;  
// a = 13
```

### 3.4.4 Variabile

Se poate declara o variabilă cu ajutorul:

var - declară o variabilă (function scope)

let - declară o variabilă locală (block scope)

const - declară o constantă locală (block scope)

Numele variabilei trebuie să respecte următoarele reguli:

- să înceapă cu o literă, (\_), sau (\$)
- următoarele caractere pot fi şi cifre (0 - 9)

```
// corect  
var userName, PetsClass, user102, $money, _myVar;
```

```
// incorect  
var 102user
```

## DIFERENŢE

	<b>var</b>	<b>let</b>	<b>const</b>
durata de viaţă	function scope	block scope	block scope
redeclarare	da	nu	nu
schimbarea valorii	da	da	nu
inițializarea cu o valoare	opțional	opțional	obligatoriu
specificat de	ECMAScript 1	ECMAScript 2015	ECMAScript 2015

### 3.4.5 Tipuri de date

1. *Boolean*: true şi false
2. *null*: null - obiect fără referinţă
3. *undefined*: undefined - fără valoare
4. *Număr*: 102, 3.14159, Infinity
5. *String* (*şir de caractere*): Hello, Bună,
6. *Symbol*: o variabilă unică şi imutabilă
7. *Obiecte* [assets/print.pdf](#)

Conversia tipurilor de date:

Valoarea	String	Număr	Boolean	Obiect
undefined	"undefined"	NaN	false	Eroare
null	"null"	0	false	Eroare
string	~	valoarea numerică sau NaN	true	String
string gol	~	0	false	String
0	"0"	~	false	Number
NaN	"NaN"	~	false	Number
Infinity	"Infinity"	~	true	Number
-Infinity	"-Infinity"	~	true	Number
orice alt număr	valoarea numărului în formă de string	~	true	Number
true	"true"	1	~	Boolean
false	"false"	0	~	Boolean
obiect	toString()	valueOf(), toString() sau NaN	true	~

- ~ - valoarea rămâne la fel
- Number - obiect de tip Number
- String - obiect de tip String
- Boolean - obiect de tip Boolean
- toString() - se apelează metoda toString() cu obiectul dat
- valueOf() - se apelează metoda valueOf() cu obiectul dat [14]

### 3.5 Typescript

TypeScript este un superset de JavaScript dezvoltat de Microsoft. Are toate caracteristicile Javascript. Utilizează compilatorul TypeScript pentru a converti fişierul TypeScript (ts) în fişierul JavaScript (js). TypeScript este mai uşor de integrat în proiectele JavaScript. TypeScript oferă, de asemenea, verificarea tipului static. Acesta permite programatorului să verifice şi să atribui variabile şi tipuri de funcţii. Această caracteristică face codul mai uşor de citit şi de prevenire a erorilor. TypeScript are tipuri de date cum ar fi String, Number, Boolean, Null, Array, Enum, Tuple şi Generics.



Principalul avantaj al TypeScript este că permite crearea de obiecte bazate pe clasă. Programatorii din C ++, Java sunt familiarizați cu concepte precum clase, obiecte, moștenire. Atunci când încearcă să programeze folosind JavaScript, poate fi dificil să se aplice acele concepte în scenariul JavaScript. Pentru a crea o clasă în JavaScript, un programator ar trebui să creeze o funcție. Pentru moștenire, trebuie să folosească, prototipuri. Cu toate acestea, TypeScript este bazat pe clasă, astfel încât este capabil să suporte moștenirea, încapsularea și modificatorul ca limbaj de programare orientat pe obiect. [15]

JavaScript vs. TypeScript	
JavaScript este un limbaj bazat pe interpreți pentru a adăuga interactivitate la o pagină Web.	TypeScript este un superset de Javascript care se compilează în JavaScript simplu.
Categorie de limbi	
Javascript este un limbaj de scripting.	TypeScript este un limbaj de programare orientat pe obiect.
Compilare	
Javascript nu are nevoie de un compilator. Rulează pe browserul web.	TypeScript necesită un compilator TypeScript pentru a converti într-un fișier JavaScript.
Obiect-orientate caracteristici	
JavaScript nu este pur orientat spre obiect. Este bazat pe prototipuri. Nu are interfețe.	TypeScript este un limbaj de programare orientat pe obiecte și este bazat pe clasă. Poate utiliza clase, moștenire, interfețe și modificatori.
Metodă de executare	
JavaScript rulează pe partea clientului.	TypeScript rulează atât pe partea clientului, cât și pe partea de server.
Analiza statică	
Javascript nu are control de tip static.	Tipul TypeScript are o verificare de tip static.
modularitate	
Javascript nu permite modulele de sprijin.	Typescript poate importa fișiere și module.

## 3.6 ReactJS

### 3.6.1 Ce este?

ReactJS este o librărie javascript creată de Facebook și folosită pentru realizarea interfeței utilizatorului (UI). ReactJS se bazează pe crearea UI-ului folosind componente. Fiecare componentă se poate dezvolta separat (putând avea o stare specifică) fiind reutilizabilă în mai multe situații. React este utilizat pentru a construi aplicații cu o singură pagină (SPA).

### 3.6.2 JSX

React cuprinde faptul că logica de redare este în mod inerent cuplată cu alte logici ale interfeţei interioare: modul în care sunt gestionate evenimentele, modul în care starea se schimbă în timp şi modul în care datele sunt pregătite pentru afişare.

În loc să separe artificial tehnologiile, punând marcajul şi logica în fişiere separate, React separă preocupările cu unităţile slab cuplate numite „componente” care conţin ambele.

React nu necesită utilizarea JSX, dar majoritatea oamenilor consideră că este util ca ajutor vizual atunci când lucrează cu IU în codul JavaScript. De asemenea, permite React să afişeze mesaje de eroare şi avertizare mai utile.

JSX este o extensie de sintaxă asemănătoare XML la ECMAScript fără semantică definită. NU este destinat a fi implementat de motoare sau browsere. NU este o propunere de încorporare a JSX în specimenul ECMAScript în sine. Este destinat să fie folosit de diverşi preprocesoare (transpilatoare) pentru a transforma aceste jetoane în ECMAScript standard. [16]

```
function getGreeting(user) {  
  if (user) {  
    return <h1>Hello, {formatName(user)}!</h1>;  
  }  
  return <h1>Hello, Stranger.</h1>;  
}
```

### 3.6.3 VirtualDOM

VirtualDOM (VDOM) este un concept de programare în care o reprezentare ideală sau „virtuală” a unei interfeţe de utilizator este păstrată în memorie şi sincronizată cu DOM-ul „real” de o bibliotecă precum ReactDOM. Acest proces se numeşte „reconciliation”.

Această abordare permite API-ul declarativ al React: Spuneţi React în ce stare doriţi să fie UI şi să vă asiguraţi că DOM se potriveşte cu acea stare. Aceasta rezumă manipularea atributelor, gestionarea evenimentelor şi actualizarea manuală a DOM pe care altfel ar trebui să o utilizaţi pentru a crea aplicaţia.

Deoarece „virtualDOM” este mai mult un model decât o tehnologie specifică, oamenii spun uneori că înseamnă lucruri diferite. În React World, termenul „DOM virtual” este de obicei asociat cu elementele React, deoarece sunt obiectele reprezentând interfaţa cu utilizatorul. React, totuşi, foloseşte şi obiecte interne numite „fibre” pentru a deţine informaţii suplimentare despre arborele de componente. De asemenea, pot fi considerate o parte a implementării „DOM virtual” în React.

Pentru fiecare nod din DOM exista un echivalent în VirtualDOM.

“Reconciliation” – este procesul prin care React modifica DOM-ul real. Cand starea unei componente se schimba, React calculeaza daca e necesara o actualizare a DOM-ului. Acest lucru se face folosind VirtualDOM. La fiecare schimbare a starii unei componente, se creaza un nou VirtualDOM si se compara cu cel precedent. Rezultatul comparatiei va fi un set de modificari ce vor fi aplicate, o data, impreuna, pe DOM.

## Virtual DOM

JavaScript tree of React elements and components

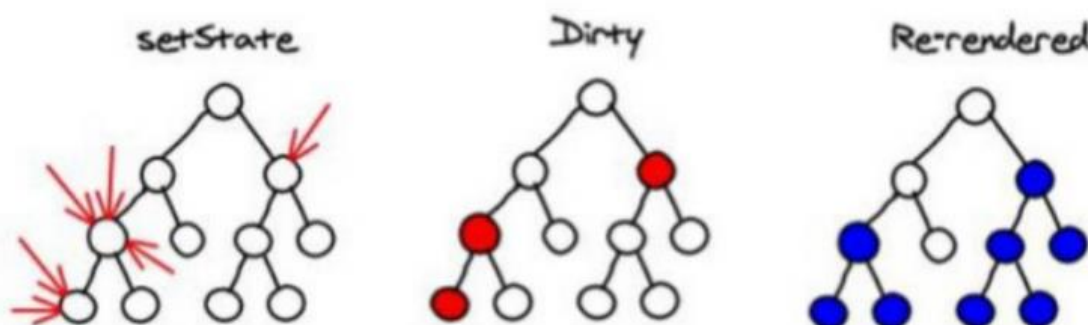


Fig. 16 VirtualDOM

### 3.6.4 Componente

- permit granularizarea interfeței UI in elemente independente, reutilizabile, izolate
- conceptual, componentele sunt functii Javascript ce primesc o listă de parametri si returnează un element de React
- pot fi definite fie folosind “function” sau “class” din javascript

Pentru a randa in DOM pentru prima oara, aplicatia noastra (SPA), e nevoie sa folosim metoda `render()` din biblioteca ReactDOM

```
ReactDOM.render(<ComponentaMea />, document.getElementById('root'));
```

Exemplu componentă:

```
class Welcome extends React.Component {  
  render() {  
    return <h1>Hello, {this.props.name}</h1>;  
  }  
}
```

### 3.6.5 Metode utile din viaţa unei componente

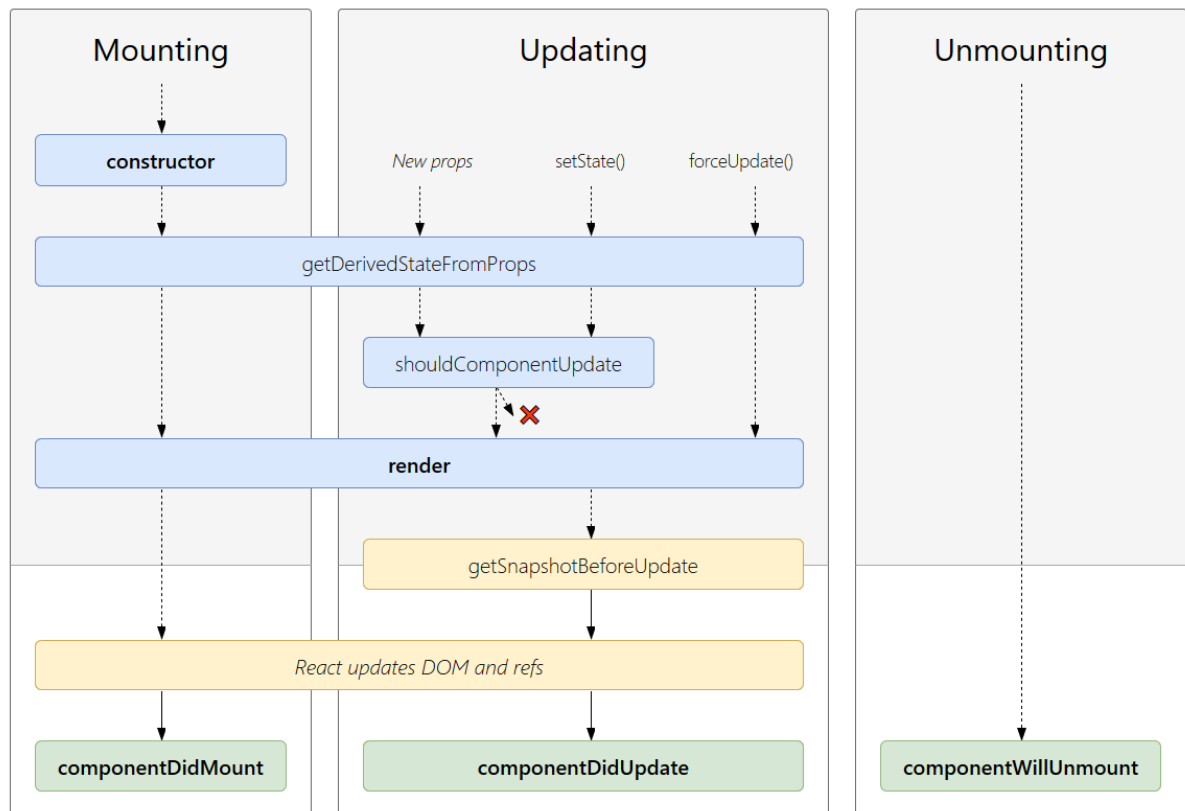


Fig. 17 Lifecycle methods – react component

#### Render()

- Singura metodă obligatoriu de implementat într-o clasă
- Când e invocată, ea va examina ce a primit ca proprietăți (`this.props`) cât și ce are în starea locală (`this.state`) și va randa codul HTML
- Trebuie să fie o funcție pură ce nu modifică starea componenteii
- Nu va fi invocată dacă metoda `"shouldComponentUpdate"` va returna un răspuns fals

```
export class Profiles extends React.Component {
  render() {
    return <MyProfile firstName={"Morar"} />
  }
}

export class MyProfile extends React.Component {
  render() {
    return(
      <div>
        My name is: {this.props.firstName}
      </div>
    )
  }
}
```

## Constructor()

- Este invocat înaintea montării componentei
- În constructor se inițializează starea cât și se conectează metodele folosite pentru evenimente

```
constructor(props) {  
  super(props);  
  // Don't call this.setState() here!  
  this.state = { counter: 0 };  
  this.handleClick = this.handleClick.bind(this);  
}
```

## componentDidMount()

- e invocată imediat după montarea componentei
- e un loc potrivit pentru a aduce date de pe server cât și pentru a modifica starea componentei

```
export class MyProfile extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = { firstName: "Morar" }  
  }  
  
  componentDidMount() {  
    //can get data from server  
    this.setState({firstName: "Albert"});  
  }  
  
  render() {  
    return(  
      <div>  
        My name is: {this.state.firstName}  
      </div>  
    )  
  }  
}
```

## shouldComponentUpdate(nextProps, nextState)

- folosită pentru a spune react-ului când o componentă poate sau nu poate fi rerandată
- e invocată înainte de a reranda o componentă
- dacă nu este implementată, ea va returna true

### **componentDidUpdate(prevProps, prevState)**

- este invocata imediat dupa ce o componenta este rerandata
- e locul potrivit unde se pot actualiza datele pe baza unui request pe server
- e locul potrivit pentru a actualiza starea componentei
- aceasta metoda nu va fi invocata daca “shouldComponentUpdate” va returna un raspuns fals
- e important sa fie folosita conditionand/comparand starea initiala cu starea actuala a componentei sau a proprietatilor primite. Daca acest lucru nu va fi realizat, aceasta metoda se va apela la infinit producand blocaje [17]

```
componentDidUpdate(prevProps) {  
  // Typical usage (don't forget to compare props):  
  if (this.props.userID !== prevProps.userID) {  
    this.fetchData(this.props.userID);  
  }  
}
```

## **4. Detalii proiect**

Pentru a rula proiectul, e nevoie ca soluția de .net să fie rulată cu IIS Express (sau configurată pe IIS) iar proiectul de FRONT END se pornește cu comanda “npm start” rulată în directorul “client”.

În momentul în care se dorește a se rula pentru prima dată proiectului pe FRONTEND, e nevoie să se ruleze întâi comanda de “npm install” (în directorul “client”) pentru a instala toate librăriile utilizate în proiect, apoi comanda “npm start”. Aceste comenzi vor porni proiectul în modul “development”. Pentru a genera proiectul pentru producție e nevoie de a se rula comanda “npm build” în directorul “client”.

### **4.1 BackEnd**

#### **4.1.1 Structura**

Pentru realizarea acestui proiect s-au folosit următoarele programe:

- Visual Studio Community 2019 for Web
- SQL Server Studio Management 2019 (crearea si moderarea bazei de date)

Pentru generarea bazei de date s-a folosit metoda de **CODE FIRST** implementată cu ajutorul Entity Framework-ului. (s-au creat clasele pentru entități, dupa care s-a generat scripturi ce vor creea/șterge/modifica tabele in DB).

Aplicația a fost creată folosind o singură soluție ce include 3 proiecte:

<b>Proiect</b>	<b>Template Proiect VS</b>	<b>Scop</b>
Repositories	Class Library	- modelele(entitatiile) tabelor din baza de date - clase per entitate ce se ocupa de comunicarea cu baza de date
Services	Class Library	- clase ce contin logica de analiza si generare de rezultate (folosind date venite din Repositories)
WebSite	WebApi	- helpere, viewModele si controller ce defines structura unui API

#### 4.1.1.1 Repositories

Contracts – interfeţe

Entities – entităţile ce definesc structura bazei de date

Enums – enum-uri folosite în entităţi

Migrations – scripturi generate de Entity Framework

Repositories – clasele concrete ce implementează interfeţele

Fiecare repositories se ocupă doar de o entitate. Nu există conexiuni între repository-uri.

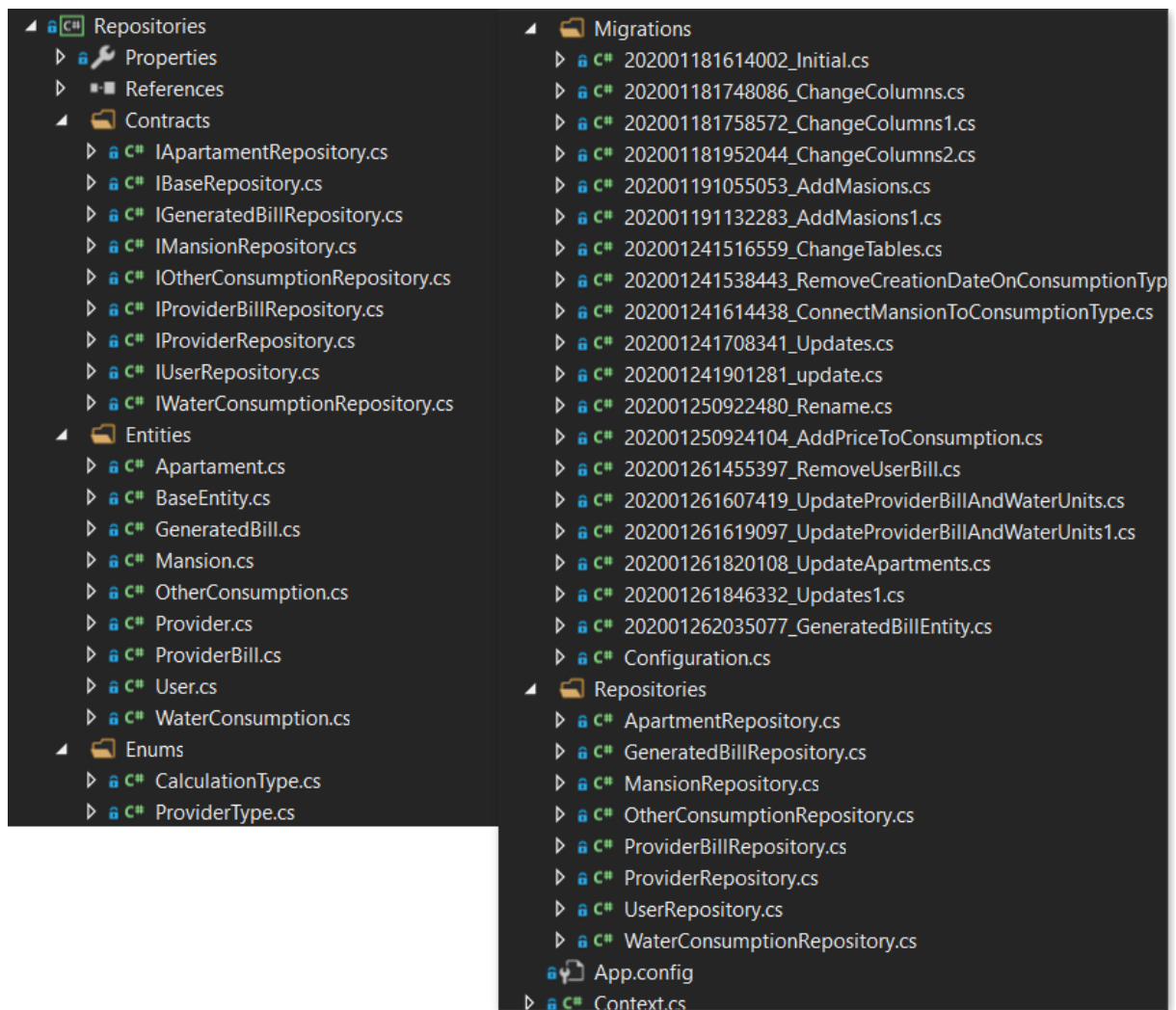


Fig. 18 Repositories



#### 4.1.1.2 Services

Contracts – interfeţe

Helpers – clase folosite ca helpere în servicii

Services – clase concrete ce implementează interfeţele

Proiectul de servicii referă proiectul de Repositories, folosind clasele entităţilor pentru prelucrarea datelor.

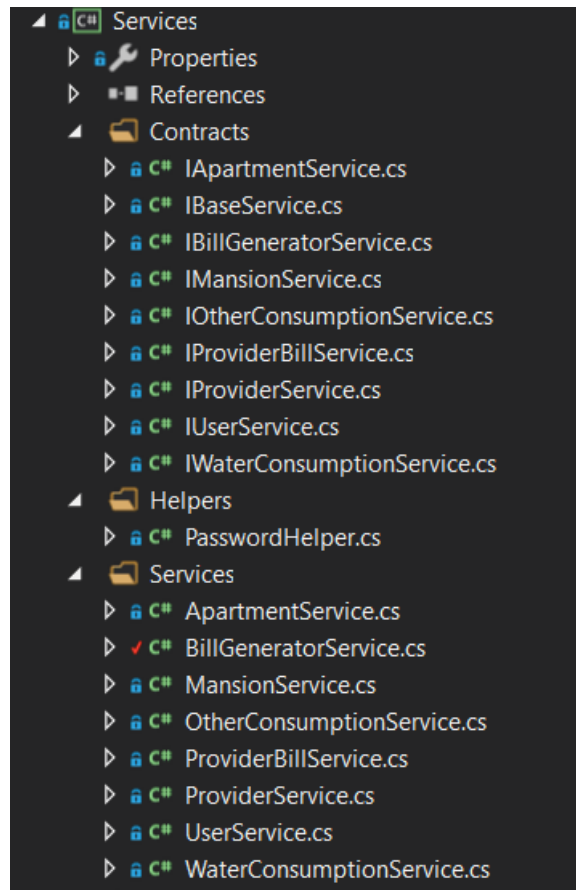


Fig. 19 Servicii

#### 4.1.1.3 WebSite

În acest proiect se definesc toate configurările pentru api. Aici se găsesc modele publice pentru API (generate ca si interfețe pentru typescript), pentru a nu folosi modelele specifice din Service/Repositories. Aceste modele utilizate în controlere sunt denumite ViewModele.

Controllers – clasele ce implementează o arhitectură REST pentru fiecare entitate

Extensions – clase în care sunt implementate metode de extensie pentru modele-viewmodele folosite pentru convertire din-spre viewModel-mdel.

Helpers – helpere folosite pentru autentificare şi autorizare

ViewModels – clasele ce definesc viewmodelele

App\_Start - configurări

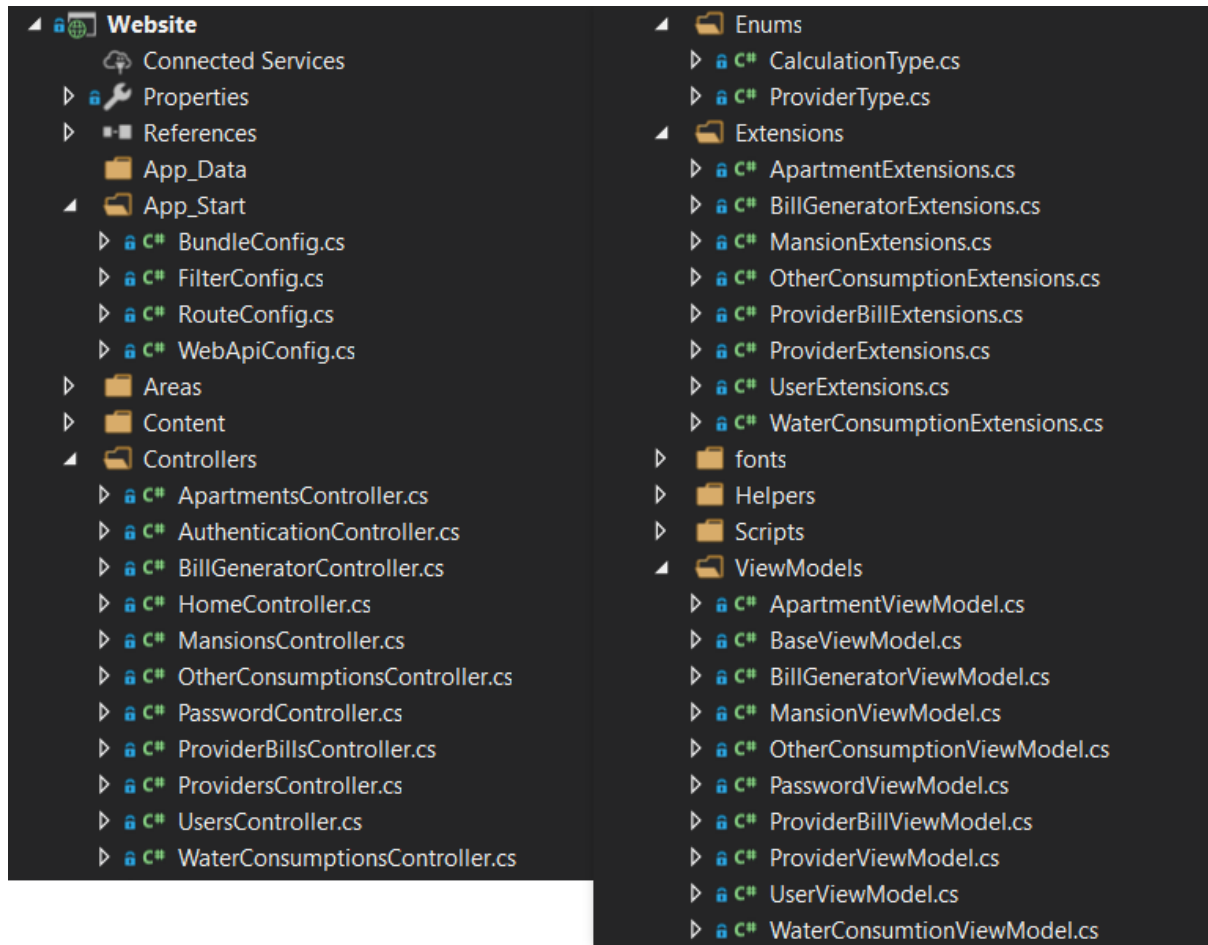


Fig. 20 WebSite

### 4.1.2 Repositories

Fiecare repository se ocupă să creeze, modifice, ştergă, citească o entitate. Ele extind interfețe, care la rândul lor extind o interfață de bază (ce conține Uniqueld).

Exemplu:

```
using Repositories.Contracts;
using Repositories.Entities;
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;

namespace Repositories.Repositories
{
    public class GeneratedBillRepository : IGeneratedBillRepository
    {
        private BuildingAssociationContext _ctx;
        private DbSet<GeneratedBill> GeneratedBills { get; set; }

        public GeneratedBillRepository(BuildingAssociationContext context)
        {
            _ctx = context;
            GeneratedBills = context.GeneratedBills;
        }
    }
}
```

```
}

public void Delete(long id)
{
    var toBeRemoved = GeneratedBills.FirstOrDefault(x => x.UniqueId == id);
    GeneratedBills.Remove(toBeRemoved);

    _ctx.SaveChanges();
}

public GeneratedBill Get(long id)
{
    return GeneratedBills.FirstOrDefault(x => x.UniqueId == id);
}

public IEnumerable<GeneratedBill> Get(IEnumerable<long> ids)
{
    return GeneratedBills.Where(x => ids.Any(id => id ==
x.UniqueId)).ToList();
}

public IEnumerable<GeneratedBill> GetAll()
{
    return GeneratedBills.ToList();
}

public GeneratedBill Insert(GeneratedBill item)
{
    var inserted = GeneratedBills.Add(item);
    _ctx.SaveChanges();

    return inserted;
}

public void Update(GeneratedBill item)
{
    var updated = GeneratedBills.FirstOrDefault(x => x.UniqueId ==
item.UniqueId);
    updated.Date = item.Date;
    updated.CSV = item.CSV;

    _ctx.SaveChanges();
}
}
```

Folosind Entity Framework, pentru fiecare entitate este un DbSet pe Context. Pe acest set de date se poate face modificări, însă când se vrea să le aplice în DB e nevoie să se cheme SaveChanges (în acel moment, toate modificările vor fi aplicate).

Contextul pentru Entity Framework e definit în clasa **BuildingAssociationContext**

```
using Repositories.Entities;
using System.Configuration;
using System.Data.Entity;

namespace Repositories
{
    public class BuildingAssociationContext : DbContext
    {
```

```
public DbSet<User> Users { get; set; }
public DbSet<Apartment> Apartments { get; set; }
public DbSet<ProviderBill> Bills { get; set; }
public DbSet<WaterConsumption> WaterConsumptions { get; set; }
public DbSet<Provider> Providers { get; set; }
public DbSet<OtherConsumption> OtherConsumptions { get; set; }
public DbSet<Mansion> Mansions { get; set; }
public DbSet<GeneratedBill> GeneratedBills { get; set; }

public BuildingAssociationContext() : base("BuildingAssociation")
{
    this.Configuration.ProxyCreationEnabled = true;
    this.Configuration.LazyLoadingEnabled = true;
    this.Configuration.AutoDetectChangesEnabled = true;
}

protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
}
}
```

Pe entităţi se pot adăuga unele atribute pentru a ajuta Entity Framework să genereze baza de date. Exemplu:

Definirea de ForeignKey – declarând acest lucru, EF va şti să genereze automat toate legăturile

```
using System;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Repositories.Entities
{
    public class Apartment : BaseEntity
    {
        [Required]
        public double Surface { get; set; }

        [Required]
        public int Number { get; set; }

        [Required]
        public int Floor { get; set; }

        [Required]
        public double IndividualQuota { get; set; }

        [Required]
        public int MembersCount { get; set; }

        [ForeignKey("User")]
        public long? UserId { get; set; }
        public virtual User User { get; set; }

        [ForeignKey("Mansion")]
        public long? MansionId { get; set; }
        public virtual Mansion Mansion { get; set; }
    }
}
```

### 4.1.3 Servicii

Servicile se ocupă cu filtrarea datelor şi prelucrarea lor. În urma acestor acţiuni, datele sunt returnate şi folosite în controller.

Ca şi în cazul repository-urilor, vorbind aici de webservicii, se dezvoltă un întreg FLOW pentru fiecare entitate în parte.

În servicii se aruncă excepţii în cazul erorilor, ele fiind prinse şi tratate în controller-e.

```
using System;
using System.Collections.Generic;
using System.Globalization;
using System.IO;
using System.Linq;
using System.Text;
using CsvHelper;
using CsvHelper.Configuration;
using Repositories.Contracts;
using Repositories.Entities;
using Repositories.Entities.Enums;
using Services.Contracts;

namespace Services.Services
{
    public class BillGeneratorService : IBillGeneratorService
    {
        private IGeneratedBillRepository _generatedBillRepository;
        private IMansionRepository _mansionRepository;
        private IWaterConsumptionRepository _waterConsumptionRepository;

        private class Item
        {
            public string Key { get; set; }
            public string Title { get; set; }
            public string Value { get; set; }
        }

        private class Line
        {
            public int ApartmentNo { get; set; }
            public IEnumerable<Item> Items { get; set; }
        }

        public BillGeneratorService(
            IMansionRepository mansionRepository,
            IWaterConsumptionRepository waterConsumptionRepository,
            IGeneratedBillRepository generatedBillRepository)
        {
            _mansionRepository = mansionRepository;
            _waterConsumptionRepository = waterConsumptionRepository;
            _generatedBillRepository = generatedBillRepository;
        }

        public void Generate(long mansionId, int month, int year)
```

```
{
    var mansion = _mansionRepository.Get(mansionId);

    var apartments = mansion.Apartments;

    if(!apartments.Any())
    {
        throw new Exception("No apartments on this mansion!");
    }

    var users = mansion.Apartments.Select(x => x.User);
    var numberOfPersonPerMansion = this.GetNumberOfPersonsPerMansion(apartments);
    var mansionBills = mansion.Bills.Where(x => x.CreationDate.Value.Month == month &&
x.CreationDate.Value.Year == year);

    CheckElectricityBill(mansionBills);
    var waterBill = this.GetWaterBill(mansionBills);

    var billsWithoutWater = mansionBills.Where(x => x.Provider.Type != ProviderType.Water);
    var totalWaterSent = this.GetTotalWaterSent(users, month, year);
    var lostWater = waterBill.Units - totalWaterSent;

    var otherConsumption = mansion.Consumptions.Where(x => x.Date.Value.Month == month &&
x.Date.Value.Year == year);

    var linesToExport = new List<Line>();

    foreach (var apartment in apartments)
    {
        var line = new Line();
        line.ApartmentNo = apartment.Number;

        var items = new List<Item>();
        items.AddRange(this.GetSplitedBills(billsWithoutWater, numberOfPersonPerMansion, apartment));
        items.Add(this.GetSplitedWaterBill(waterBill, lostWater, numberOfPersonPerMansion, apartment, month,
year));
        if(otherConsumption.Any()) items.AddRange(this.GetSplitedConsumptions(otherConsumption, apartment,
numberOfPersonPerMansion));

        line.Items = items;

        linesToExport.Add(line);
    }

    var csv = this.GenerateCsv(linesToExport.OrderBy(x => x.ApartmentNo));

    var existingCsvFromDb = _generatedBillRepository.GetAll().FirstOrDefault(x => x.Date.Year == year &&
x.Date.Month == month && x.MansionId == mansion.UniqueId);
    if (existingCsvFromDb != null)
    {
        existingCsvFromDb.CSV = csv;
        _generatedBillRepository.Update(existingCsvFromDb);
    } else
    {
        var date = new DateTime(year, month, 1);

        _generatedBillRepository.Insert(new GeneratedBill
        {
            CSV = csv,
            Date = date,

```

---

```
MansionId = mansion.Uniqueld
    });
}
}

public GeneratedBill Get(long id)
{
    return _generatedBillRepository.Get(id);
}

public IEnumerable<GeneratedBill> Get(IEnumerable<long> ids)
{
    return _generatedBillRepository.Get(ids);
}

public void Update(GeneratedBill item)
{
    _generatedBillRepository.Update(item);
}

public GeneratedBill Insert(GeneratedBill item)
{
    return _generatedBillRepository.Insert(item);
}

public void Delete(long id)
{
    _generatedBillRepository.Delete(id);
}

public IEnumerable<GeneratedBill> GetAll()
{
    return _generatedBillRepository.GetAll();
}

private string GenerateCsv(IEnumerable<Line> linesToExport)
{
    using (var mem = new MemoryStream())
    using (var writer = new StreamWriter(mem))
    using (var csvWriter = new CsvWriter(writer, CultureInfo.InvariantCulture))
    {
        csvWriter.Configuration.Delimiter = ",";

        csvWriter.WriteField("No");

        IEnumerable<string> extractedHeaders = this.ExtractHeaders(linesToExport.FirstOrDefault());

        foreach (var header in extractedHeaders)
        {
            csvWriter.WriteField(header);
        }
        csvWriter.NextRecord();

        foreach (var line in linesToExport)
        {
            csvWriter.WriteField(line.ApartmentNo);

            foreach (var item in line.Items)
            {
                csvWriter.WriteField(item.Value);
            }
        }
    }
}
```

```
    }

    csvWriter.NextRecord();
}

writer.Flush();
var result = Encoding.UTF8.GetString(mem.ToArray());

return result;
}
}

private IEnumerable<string> ExtractHeaders(Line firstLine)
{
    var result = new List<string>();
    if(firstLine != null)
    {
        foreach(var item in firstLine.Items)
        {
            result.Add(item.Title);
        }
    }

    return result;
}

private IEnumerable<Item> GetSplitedConsumptions(IEnumerable<OtherConsumption> consumptions,
Apartment apartment, int numberOfPersonPerMansion)
{
    var items = new List<Item>();

    foreach(var item in consumptions)
    {
        items.Add(new Item
        {
            Key = item.Name.Replace(" ", "-"),
            Title = item.Name,
            Value = item.CalculationType == CalculationType.IndividualQuota
                ? (item.Price * (apartment.IndividualQuota / 100)).ToString()
                : (item.Price / (apartment.MembersCount * numberOfPersonPerMansion)).ToString()
        });
    }

    return items;
}

private Item GetSplitedWaterBill(ProviderBill waterBill, double lostWater, int numberOfPersonPerMansion,
Apartment apartment, int month, int year)
{
    double priceForSentWater = GetUserWaterConsumtionSent(apartment.User, month, year);
    double priceForLostWater = lostWater / (numberOfPersonPerMansion * apartment.MembersCount);

    double totalPriceWater = priceForLostWater + priceForSentWater;

    return new Item
    {
        Key = waterBill.Provider.Name.Replace(" ", "-"),
        Title = waterBill.Provider.Name,
        Value = Math.Round(totalPriceWater, 2).ToString()
    };
}
```



```
}

private double GetTotalWaterSent(IEnumerable<User> users, int month, int year)
{
    double totalSent = 0;
    foreach(var user in users)
    {
        double total = GetUserWaterConsumptionSent(user, month, year);
        totalSent += total;
    }

    return totalSent;
}

private double GetUserWaterConsumptionSent(User user, int month, int year)
{
    var thisMonthLastSent = user.WaterConsumptions
        .Where(x =>
            x.CreationDate.Value.Month == month
            && x.CreationDate.Value.Year == year)
        .OrderByDescending(x => x.CreationDate).FirstOrDefault();

    var precedentMonthLastSent = user.WaterConsumptions
        .Where(x =>
            x.CreationDate.Value.Month == (month == 1 ? 12 : month - 1)
            && x.CreationDate.Value.Year == year)
        .OrderByDescending(x => x.CreationDate)
        .FirstOrDefault();

    var lastSent = user.WaterConsumptions
        .OrderByDescending(x => x.CreationDate)
        .FirstOrDefault();

    if (thisMonthLastSent != null)
    {
        double total = precedentMonthLastSent != null
            ? (
                (thisMonthLastSent.BathroomUnits - precedentMonthLastSent.BathroomUnits) +
                (thisMonthLastSent.KitchenUnits - precedentMonthLastSent.KitchenUnits)
            )
            : thisMonthLastSent.KitchenUnits + thisMonthLastSent.BathroomUnits;

        return total;
    }
    else
    {
        var date = new DateTime(year, month, 28);
        _waterConsumptionRepository.Insert(new WaterConsumption
        {
            UserId = user.UniqueId,
            KitchenUnits = lastSent != null ? lastSent.KitchenUnits : 0,
            BathroomUnits = lastSent != null ? lastSent.BathroomUnits : 0,
            CreationDate = date
        });
    }

    return 0;
}
```

```
private int GetNumberOfPersonsPerMansion(IEnumerable<Apartment> apartments)
{
    int numberOfPersonPerMansion = 0;
    foreach (var apartment in apartments)
    {
        numberOfPersonPerMansion += apartment.MembersCount;
    }

    return numberOfPersonPerMansion;
}

private void CheckElectricityBill(IEnumerable<ProviderBill> bills)
{
    var bill = bills.Where(x =>
        x.Provider.Type == ProviderType.Electricity)
        .OrderByDescending(x => x.CreationDate).FirstOrDefault();

    if (bill == null)
    {
        throw new Exception("No electricity bill!");
    }
}

private ProviderBill GetWaterBill(IEnumerable<ProviderBill> bills)
{
    var bill = bills.Where(x =>
        x.Provider.Type == ProviderType.Water)
        .OrderByDescending(x => x.CreationDate).FirstOrDefault();

    if (bill == null)
    {
        throw new Exception("No water bill!");
    }

    return bill;
}

private IEnumerable<Item> GetSplitedBills(IEnumerable<ProviderBill> otherBills, int
numberOfPersonPerMansion, Apartment apartment)
{
    var items = new List<Item>();
    foreach (var bill in otherBills)
    {
        var billPrice = bill.Units * bill.Provider.UnitPrice + bill.Other;

        //we split bills using
        items.Add(new Item
        {
            Key = bill.Provider.Name.Replace(" ", "-"),
            Title = bill.Provider.Name,
            Value = Math.Round((billPrice / (numberOfPersonPerMansion * apartment.MembersCount)),
2).ToString()
        });
    }

    return items;
}
}
```

Serviciul mai sus atasat descrie şi baza acestei aplicaţii, unde se fac calcule pentru generarea listei de plată pentru fiecare bloc în parte.

#### 4.1.4 WebSite

În website se găsesc controller-ele ce extind clasa ApiController. Fiecare controller are câte o metodă pentru

- GET() – va aduce lista cu toate entităţile
- GET(id) – va aduce o entitate după ID
- POST(XViewModel viewModel) – va modifica/insera o noua entitate (entitate trimisă prin Body)
- DELETE(id) – va şterge o entitate după model

Dacă va apărea o excepție, aceasta va fi trimisă mai departe spre utilizator, pentru a fi afişată.

Pe fiecare controller se definesc limite pentru autorizare cât şi verificarea existenţei unui user autentificat.

Exemplu controller:

```
using Services.Contracts;
using System;
using System.Linq;
using System.Net.Http;
using System.Web.Http;
using System.Web.Http.Cors;
using Website.ViewModels;
using Website.Extensions;
using Website.Helpers;

namespace Website.Controllers
{
    [EnableCors(origins: "http://localhost:3000", headers: "*", methods: "*")]
    [BasicAuthentication]
    [MyAuthorize(Roles = "Admin")]
    public class ApartmentsController : ApiController
    {
        private IApartmentService _apartmentService;

        public ApartmentsController(IApartmentService service)
        {
            _apartmentService = service;
        }

        // GET api/apartments
        public HttpResponseMessage Get()
        {
            var items = _apartmentService.GetAll().Select(x => x.ToViewModel());
            return Request.CreateResponse(System.Net.HttpStatusCode.Accepted, items);
        }

        // GET api/apartments/5
        public HttpResponseMessage Get(long id)
        {
            var item = _apartmentService.Get(id).ToViewModel();
            return Request.CreateResponse(System.Net.HttpStatusCode.Accepted, item);
        }
    }
}
```

```
}

public HttpResponseMessage Post([FromBody]ApartmentViewModel item)
{
    try
    {
        var entity = item.FromViewModel();

        if (entity.UniqueId.HasValue)
        {
            _apartmentService.Update(entity);
        }
        else
        {
            _apartmentService.Insert(entity);
        }

        return Request.CreateResponse(System.Net.HttpStatusCode.Accepted);
    }
    catch (Exception e)
    {
        return
Request.CreateResponse(System.Net.HttpStatusCode.InternalServerError, e.Message);
    }
}

// DELETE api/apartments/5
public HttpResponseMessage Delete(long id)
{
    try
    {
        _apartmentService.Delete(id);
        return Request.CreateResponse(System.Net.HttpStatusCode.Accepted);
    }
    catch
    {
        return
Request.CreateResponse(System.Net.HttpStatusCode.InternalServerError);
    }
}
}
}
```

### Autentificare:

S-a creat o clasa (atribut) ce se ocupa de a verifica/autentifica un user pe baza unui token generat.

Token-ul se genereaza enodand in base64 urmatoarea structura: "username : parola".

In clasa de autentificare, se verifică ca userul să existe si se salvează pe un Thread datele (id si dacă e admin).

Aceasta clasa se va injecta ca si atribut pe fiecare controller, astfel daca se va incerca a se face un request fara o autentificare in prealabil, se va primi un raspuns de eroare.

```
using Repositories.Entities;
using Services.Contracts;
using System;
```

```
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Security.Claims;
using System.Security.Principal;
using System.Text;
using System.Threading;
using System.Web;
using System.Web.Http.Controllers;
using System.Web.Http.Filters;

namespace Website.Helpers
{
    public class BasicAuthenticationAttribute : AuthorizationFilterAttribute
    {
        private const string Realm = "My Realm";
        public override void OnAuthorization(HttpContext actionContext)
        {
            //If the Authorization header is empty or null
            //then return Unauthorized
            if (actionContext.Request.Headers.Authorization == null)
            {
                actionContext.Response = actionContext.Request
                    .CreateResponse(HttpStatusCode.Unauthorized);
                // If the request was unauthorized, add the WWW-Authenticate header
                // to the response which indicates that it require basic
                authentication if (actionContext.Response.StatusCode == HttpStatusCode.Unauthorized)
                {
                    actionContext.Response.Headers.Add("WWW-Authenticate",
                        string.Format("Basic realm=\"{0}\"", Realm));
                }
            }
            else
            {
                //Get the authentication token from the request header
                string authenticationToken = actionContext.Request.Headers
                    .Authorization.Parameter;
                //Decode the string
                string decodedAuthenticationToken = Encoding.UTF8.GetString(
                    Convert.FromBase64String(authenticationToken));
                //Convert the string into an string array
                string[] usernamePasswordArray =
                    decodedAuthenticationToken.Split(':');
                //First element of the array is the username
                string username = usernamePasswordArray[0];
                //Second element of the array is the password
                string password = usernamePasswordArray[1];

                User userByCredentials = UserValidate.GetUserDetails(username,
                    password);

                //call the login method to check the username and password
                if (userByCredentials != null)
                {
                    var identity = new GenericIdentity(username);
                    identity.AddClaim(new Claim("loggedUserId",
                        Convert.ToString(userByCredentials.UniqueId)));
                    identity.AddClaim(new Claim("isAdmin",
                        Convert.ToString(userByCredentials.Roles.Contains("Admin"))));
                }
            }
        }
    }
}
```

```
        IPrincipal principal = new GenericPrincipal(identity,
userByCredentials.Roles.Split(','));
        Thread.CurrentPrincipal = principal;
        if (HttpContext.Current != null)
        {
            HttpContext.Current.User = principal;
        }
    }
    else
    {
        actionContext.Response = actionContext.Request
            .CreateResponse(HttpStatusCode.Unauthorized);
    }
}
}
}
```

Autorizarea se face dupa roluri. Sunt doua tipuri: Admin si User

Pentru fiecare controller, sau chiar metoda a unui controller, este adaugat un atribut ce defineşte limita de autorizare.

Ex:

[MyAuthorize(Roles = "Admin")] - autorizat doar Admin-ul

### Metode de extensie (folosite pentru a converti un model in viewmodel, şi invers):

```
using Repositories.Entities;
using Repositories.Entities.Enums;
using System;
using Website.ViewModels;

namespace Website.Extensions
{
    public static class OtherConsumptionExtensions
    {
        public static OtherConsumptionViewModel ToViewModel(this OtherConsumption
item)
        {
            Enums.CalculationType calculationType;
            Enum.TryParse(item.CalculationType.ToString(), out calculationType);

            return new OtherConsumptionViewModel
            {
                CalculationType = calculationType,
                Name = item.Name,
                Id = item.UniqueId,
                Date = item.Date.Value.ToString("MM/dd/yyyy"),
                MansionId = item.Mansion.UniqueId,
                MansionName = item.Mansion.Address,
                Price = item.Price
            };
        }

        public static OtherConsumption FromViewModel(this OtherConsumptionViewModel
viewModel)
        {
            CalculationType calculationType;
            Enum.TryParse(viewModel.CalculationType.ToString(), out calculationType);
```

```
        return new OtherConsumption
        {
            UniqueId = viewModel.Id,
            Name = viewModel.Name,
            CalculationType = calculationType,
            Date = Convert.ToDateTime(viewModel.Date),
            MansionId = viewModel.MansionId,
            Price = viewModel.Price
        };
    }
}
```

### Route declarate in RouteConfig:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Http;
using System.Web.Mvc;
using System.Web.Routing;

namespace Website
{
    public class RouteConfig
    {
        public static void RegisterRoutes(RouteCollection routes)
        {
            routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

            routes.MapRoute(
                name: "Default",
                url: "{controller}/{action}/{id}",
                defaults: new { controller = "Home", action = "Index", id =
                    UrlParameter.Optional }
            );

            routes.MapHttpRoute(
                name: "ActionApi",
                routeTemplate: "api/{controller}/{action}/{id}",
                defaults: new { id = RouteParameter.Optional }
            );
        }
    }
}
```

### Dependency Injection:

Folosind dependency injection (momentan doar in constructor), toate clasele sunt folosite injectând interfețe. Pentru a cunoaște fiecare interfață unde își are implementarea, s-a folosit Unity container si s-au definit astfel:

```
private static void RegisterRepositories(UnityContainer container)
{
    container.RegisterType<IUserRepository, UserRepository>();
    container.RegisterType<IProviderBillRepository, ProviderBillRepository>();
    container.RegisterType<IProviderRepository, ProviderRepository>();
    container.RegisterType<IApartmentRepository, ApartmentRepository>();
}
```

```
        container.RegisterType<IWaterConsumptionRepository,
WaterConsumptionRepository>();
        container.RegisterType<IMansionRepository, MansionRepository>();
        container.RegisterType<IOtherConsumptionRepository,
OtherConsumptionRepository>();
        container.RegisterType<IGeneratedBillRepository,
GeneratedBillRepository>();
    }

    private static void RegisterServices(UnityContainer container)
    {
        container.RegisterType<IUserService, UserService>();
        container.RegisterType<IProviderBillService, ProviderBillService>();
        container.RegisterType<IProviderService, ProviderService>();
        container.RegisterType<IApartmentService, ApartmentService>();
        container.RegisterType<IWaterConsumptionService,
WaterConsumptionService>();
        container.RegisterType<IMansionService, MansionService>();
        container.RegisterType<IOtherConsumptionService,
OtherConsumptionService>();
        container.RegisterType<IBillGeneratorService, BillGeneratorService>();
    }
}
```

## 4.2 FrontEnd

### 4.2.1 Structura

Node-modules – folder generat de comanda “npm install” ce conține librăriile folosite/descarcate în proiect

Src – director ce conține toate componentele, interfețele și stilurile dezvoltate

Package.json – fișierul de definire a dependentelor, comenzilor de rulare

Tsconfig. – document pentru configurarea typescript-ului

### 4.2.2 Modele

În src/models se pot găsi interfețe de typescript. Ele sunt generate din .NET. În .NET, fiecare ViewModel e setat să genereze o interfață pentru typescript, în acest fel existând sincronizare continuă între backend și frontend când vine vorba de tipuri de obiecte, proprietăți.

Pentru a rula aplicația de React, s-a creat un Controller MVC în .NET ce decide un view “Index.cshtml” unde este un element cu id-ul=root. În typescript, se caută în DOM acest element și se încarcă aplicația în interiorul lui.

```
import * as React from "react";
import * as ReactDOM from "react-dom";
import 'bootstrap/dist/css/bootstrap.css';
import { BrowserRouter as Router } from 'react-router-dom';

import { configWithRouter as ConfigWithRouter } from './components/routerConfiguration';
```



```
class App extends React.Component {  
  
  render() {  
    return (  
      <Router>  
        <div>  
          <ConfigWithRouter />  
        </div>  
      </Router>  
    )  
  }  
};  
  
ReactDOM.render(<App />, document.getElementById("root"));
```

### 4.2.3 Rutare

Vorbind de o aplicație SPA (Single page application), toată dezvoltarea aplicației este într-o singură pagină. În urma request-urilor browser-ul nu se va refresh-ui. Pentru astfel de aplicații este nevoie de a crea propriul sistem de rutare, neputând a se folosi cel de pe browser. Având propriul sistem de rutare, pentru fiecare path specificat se va ști să se seteze aplicația încât să afișeze datele dorite.

În proiect s-a folosit biblioteca "react-router".

```
import * as React from 'react';  
import {Switch, Route, withRouter, RouteComponentProps } from 'react-router';  
import Navigation from '../navigation';  
import About from '../about';  
import Login from '../login';  
import Main from '../main';  
import AddBill from '../addForms/addBill';  
import './style.css';  
import BillList from '../lists/billList';  
import MansionList from '../lists/mansionList';  
import AddMansion from '../addForms/addMansion';  
import AddProvider from '../addForms/addProvider';  
import ProviderList from '../lists/providerList';  
import AddUser from '../addForms/addUser';  
import UserList from '../lists/userList';  
import OtherConsumptionList from '../lists/otherConsumptionList';  
import AddApartment from '../addForms/addApartment';  
import ApartmentList from '../lists/apartmentList';  
import AddOtherConsumption from '../addForms/addOtherConsumption';  
import AddWaterConsumption from '../addForms/addWaterConsumption';  
import WaterConsumptionList from '../lists/waterConsumptionList';  
import ChangePassword from '../addForms/changePassword';  
import GeneratedBillList from '../lists/generatedBillList';
```

```
class RouterConfiguration extends React.Component<RouteComponentProps<any>> {
  private previousRouteHash: string;

  constructor(props: RouteComponentProps<any>) {
    super(props);
    this.previousRouteHash = "";
  }

  componentDidUpdate() {
    window.scrollTo(0, 0);
    // reload page when coming back from another non training page
    if (this.previousRouteHash !== "" && this.props.location.hash === "") {
      this.previousRouteHash = this.props.location.hash;
      window.location.reload();
    }

    this.previousRouteHash = this.props.location.hash;
  }

  render() {
    const item = (props: any, component: any) => <>
      <Navigation {...props} />
      <div className="main-container">
        {component}
      </div>
    </>

    return (
      <div className="root-container">
        <Switch>
          <Route path="/addmansion/:id" exact={false} render={(props) => item(
            props, <AddMansion {...props}/>)} />
          <Route path="/addmansion/" exact={false} render={(props) => item(
            props, <AddMansion {...props}/>)} />

          <Route path="/addbill/:id" exact={false} render={(props) => item(
            props, <AddBill {...props}/>)} />
          <Route path="/addbill/" exact={false} render={(props) => item(
            props, <AddBill {...props}/>)} />

          <Route path="/addprovider/:id" exact={false} render={(props) => item(
            props, <AddProvider {...props}/>)} />
          <Route path="/addprovider/" exact={false} render={(props) => item(
            props, <AddProvider {...props}/>)} />

          <Route path="/adduser/:id" exact={false} render={(props) => item(
            props, <AddUser {...props}/>)} />
        </Switch>
      </div>
    );
  }
}
```

```

    <Route path='/adduser/' exact={false} render={(props) => item(prop
s, <AddUser {...props}/>)}>/>

    <Route path='/addapartment/:id' exact={false} render={(props) => i
tem(props, <AddApartment {...props}/>)}>/>
    <Route path='/addapartment/' exact={false} render={(props) => item
(props, <AddApartment {...props}/>)}>/>

    <Route path='/addconsumption/:id' exact={false} render={(props) =>
item(props, <AddOtherConsumption {...props}/>)}>/>
    <Route path='/addconsumption/' exact={false} render={(props) => it
em(props, <AddOtherConsumption {...props}/>)}>/>

    <Route path='/addwaterconsumption/:id' exact={false} render={(prop
s) => item(props, <AddWaterConsumption {...props}/>)}>/>
    <Route path='/addwaterconsumption/' exact={false} render={(props)
=> item(props, <AddWaterConsumption {...props}/>)}>/>

    <Route path='/changepassword/' exact={false} render={(props) => it
em(props, <ChangePassword {...props}/>)}>/>
    <Route path='/about' exact={false} render={(props) => item(props,
<About {...props}/>)}>/>
    <Route path='/login' exact={false} render={(props) => item(props,
<Login {...props}/>)}>/>
    <Route path='/billllist' exact={false} render={(props) => item(prop
s, <Billllist {...props}/>)}>/>
    <Route path='/mansions' exact={false} render={(props) => item(prop
s, <MansionList {...props}/>)}>/>
    <Route path='/providers' exact={false} render={(props) => item(pro
ps, <ProviderList {...props}/>)}>/>
    <Route path='/users' exact={false} render={(props) => item(props,
<UserList {...props}/>)}>/>
    <Route path='/consumptions' exact={false} render={(props) => item(
props, <OtherConsumptionList {...props}/>)}>/>
    <Route path='/apartments' exact={false} render={(props) => item(pr
ops, <ApartmentList {...props}/>)}>/>
    <Route path='/waterconsumptions' exact={false} render={(props) =>
item(props, <WaterConsumptionList {...props}/>)}>/>
    <Route path='/generatedbills' exact={false} render={(props) => ite
m(props, <GeneratedBillList {...props}/>)}>/>
    <Route path='/' exact={false} render={(props) => item(props, <Main
...props}/>)}>/>
  </Switch>
</div>

);
}
}

export const configWithRouter = withRouter(RouterConfiguration);
```

#### 4.2.4 Componente

Sunt două mari grupe de componente. Componente ce conţin formulare (folosite pentru adaugare, editare) si componente folosite pentru a lista informaţii.

Exemplu componenta cu formular:

```
import * as React from 'react';
import { RouteComponentProps, Redirect } from 'react-router';
import { User } from '../../models/User';
import { Mansion } from '../../models/Mansion';
import { Apartment } from '../../models/Apartment';

interface AddApartmentState {
  users: User[];
  selectedUser?: User | null;
  mansions: Mansion[];
  selectedMansion?: Mansion;
  floor?: number;
  individualQuota?: number;
  number?: number;
  surface?: number;
  membersCount: number;
  saved: boolean;
}

export default class AddApartment extends React.Component<RouteComponentProps<any>, AddApartmentState> {
  constructor(props: RouteComponentProps<any>) {
    super(props);

    this.state = {
      users: [],
      mansions: [],
      saved: false,
      floor: undefined,
      individualQuota: undefined,
      number: undefined,
      surface: undefined,
      membersCount: 0
    }
  }

  componentDidMount() {
    const { id } = this.props.match.params;
    this.initData(id);
  }
}
```

```
initData = async (id?: number) => {
  if(sessionStorage.getItem('authToken') != null) {

    const usersFromDb: User[] = await fetch(`/users`, {
      headers: {
        'Authorization': sessionStorage.getItem('authToken')
      }
    }) as RequestInit).then(response => {
      if(response.ok) {
        return response.json();
      }

      return undefined;
    });

    const mansionsFromDb: Mansion[] = await fetch(`/mansions`, {
      headers: {
        'Authorization': sessionStorage.getItem('authToken')
      }
    }) as RequestInit).then(response => {
      if(response.ok) {
        return response.json();
      }

      return undefined;
    });

    if(id) {
      const item: Apartment = await fetch(`/apartments/${id}`, {
        headers: {
          'Authorization': sessionStorage.getItem('authToken')
        }
      }) as RequestInit).then(response => {
        if(response.ok) {
          return response.json();
        }

        return undefined;
      });

      if(item) {
        this.setState({
          mansions: mansionsFromDb,
          users: usersFromDb,
          floor: item.floor,
          individualQuota: item.individualQuota,
          number: item.number,
          surface: item.surface,
```

```
        selectedMansion: mansionsFromDb.find(x => x.id === item.mansionId),
        selectedUser: usersFromDb.find(x => x.userId === item.userId),
        membersCount: item.membersCount
      })
    }
  } else {
    this.setState({mansions: mansionsFromDb, users: usersFromDb});
  }
}
}

submit = (e: React.FormEvent<HTMLFormElement>) => {
  e.preventDefault();

  const item: Apartment = {
    mansionId: this.state.selectedMansion?.id,
    userId: this.state.selectedUser?.userId,
    floor: this.state.floor as number,
    individualQuota: this.state.individualQuota as number,
    number: this.state.number as number,
    surface: this.state.surface as number,
    userName: this.state.selectedUser?.name as string,
    mansionName: this.state.selectedMansion?.address as string,
    apartmentId: this.props.match.params.id,
    membersCount: this.state.membersCount
  }

  fetch('/apartments', {
    method: 'POST',
    body: JSON.stringify(item),
    headers: {
      'Content-Type': 'application/json',
      'Authorization': sessionStorage.getItem('authToken')
    }
  }) as RequestInit).then(result => {
    if(result.ok) {
      this.setState({saved: true});
      return;
    }
    return result.json();
  }).then(error => {
    if(error) alert(error);
  });
}

renderMansions = () => {
```

```
        return this.state.mansions && this.state.mansions.map((mansion: Mansion, index: number) => {
            const isSelected = this.state.selectedMansion && this.state.selectedMansion?.id === mansion.id;
            return <option key={`_${mansion.id}-${index}`} value={mansion.id} selected={isSelected}>{mansion.address}</option>
        })
    }

    selectMansion = (e: React.ChangeEvent<HTMLSelectElement>) => {
        const selectedOptionId = parseInt(e.target.selectedOptions[0].value);
        const selectedM = this.state.mansions.find(x => x.id === selectedOptionId);

        if(selectedM) {
            this.setState({selectedMansion: selectedM, selectedUser: undefined}, () => {
                const selectUser = document.getElementById('userselect') as HTMLSelectElement;
                selectUser.selectedIndex = 0; // first option is selected
            }, or
            // -
            1 for no option selected
        });
    }

    renderUsers = () => {
        if(this.state.selectedMansion) {
            return this.state.users &&
                this.state.users
                    .filter(x => x.mansionId === this.state.selectedMansion?.id)
                    .map((user: User, index: number) => {
                        const isSelected = this.state.selectedUser?.userId === user.userId;
                        return <option key={`_${user.userId}-${index}`} value={user.userId} selected={isSelected}>{user.name}</option>
                    });
        }

        return this.state.users &&
            this.state.users
                .map((user: User, index: number) => {
                    return <option key={`_${user.userId}-${index}`} value={user.userId}>{user.name}</option>
                });
    }
}
```

72



```
    </div>
    <div className="form-group">
      <label>Number</label>
      <input
        type="number"
        min="0"
        onChange={(e) => this.setState({number: parseInt(e.target.value)})} )
        className="form-control"
        required
        defaultValue={this.state.number}
      />
    </div>
    <div className="form-group">
      <label>Floor</label>
      <input
        type="number"
        min="0"
        onChange={(e) => this.setState({floor: parseInt(e.target.value)})} )
        className="form-control"
        required
        defaultValue={this.state.floor}
      />
    </div>
    <div className="form-group">
      <label>Surface</label>
      <input
        type="number"
        step="any"
        onChange={(e) => this.setState({surface: parseFloat(e.target.value)})} )
        className="form-control"
        required
        defaultValue={this.state.surface}
      />
    </div>
    <div className="form-group">
      <label>Individual Quota</label>
      <input
        type="number"
        step="any"
        onChange={(e) => this.setState({individualQuota: parseFloat(e.target.value)})} )
        className="form-control"
        required
        defaultValue={this.state.individualQuota}
      />
    </div>
```

```
        <button type="submit" className="btn btn-  
primary">Submit</button>  
      </form>  
    )  
  }  
}
```

În componentele create pentru a lista s-a folosit libraria react-bootstrap-table ce oferă o componentă principală de Tabel foarte customizabilă ce aduce multe beneficii:

- Filtrare avansată
- Export to CSV
- Search avansat
- Ordonare
- Design responsive

Pentru elementele de selecție de data s-a folosit libraria react-datepicker ce oferă o componentă principală pentru a avea un selector de date avansat în care sunt multe opțiuni de setare (format, minDate, maxDate, utc or not, selectare an, selectare zi etc).

Exemplu componenta de listare:

```
import * as React from 'react';  
import { RouteComponentProps } from 'react-router';  
import { BootstrapTable, TableHeaderColumn } from 'react-bootstrap-table';  
import { Link } from 'react-router-dom';  
import { Apartment } from '../../models/Apartment';  
  
interface ApartmentListState {  
  apartments: Apartment[];  
  reload: boolean;  
}  
  
export default class ApartmentList extends React.Component<RouteComponentProps  
<any>, ApartmentListState> {  
  constructor(props: RouteComponentProps<any>) {  
    super(props);  
  
    this.state = {  
      apartments: [],  
      reload: false  
    }  
  }  
  
  componentDidMount() {  
    this.initData();  
  }  
}
```

```
componentDidUpdate(prevProps: any, prevState: ApartmentListState) {
  if(prevState.reload !== this.state.reload && this.state.reload) {
    this.initData();
  }
}

initData = () => {
  if(sessionStorage.getItem('authToken') !== null) {
    fetch(`/apartments`, {
      headers: {
        'Authorization': sessionStorage.getItem('authToken')
      }
    } as RequestInit).then(response => {
      if(response.ok) {
        return response.json();
      }

      return undefined;
    }).then((result: Apartment[]) => {
      this.setState({ apartments: result, reload: false })
    });
  }
}

enumFormatter = (cell: any, row: any, enumObject: any) => {
  return enumObject[cell];
}

getMansionsEnum = () => {
  const mansions: any = {};
  this.state.apartments && this.state.apartments.forEach(item => {
    if(!(`${item.mansionId}` in mansions)) {
      mansions[`${item.mansionId}`] = item.mansionName
    }
  });

  return mansions;
}

deleteRow = (item: Apartment) => {
  if(sessionStorage.getItem('authToken') !== null) {
    fetch(`/apartments/${item.apartmentId}`, {
      method: 'DELETE',
      headers: {
        'Authorization': sessionStorage.getItem('authToken')
      },
    } as RequestInit).then(response => {
      if (response.ok) {
        this.setState({reload: true});
      }
    });
  }
}
```

```
    }
  });
}

actionsFormatter = (cell: any, row: Apartment) => {
  return <>
    <Link to={` /addapartment/${row.apartmentId}`} className="fas fa-
edit"></Link>
    <i
      className="fas fa-trash-alt ml-3"
      onClick={() => this.deleteRow(row)}></i>
  </>;
}

render() {
  if(sessionStorage.getItem('authToken') == null) {
    return <div>
      Nu esti logat!
    </div>
  }

  const mansionsType = this.getMansionsEnum();

  return (
    <div className="container consumptiontypelist-container">
      <Link to={` /addapartment`} className="btn btn-
info">Add apartment</Link>

      <BootstrapTable data={this.state.apartments} containerClass="m
t-3"
        striped hover
        exportCSV
        search
        version='4'
        options={{
          noDataText: 'No data!' ,
          defaultSortName: 'number',
          defaultSortOrder: 'desc',
          sortIndicator: false,
          sizePerPage: 5,
          sizePerPageList: [ {
            text: '5', value: 5
          }, {
            text: '10', value: 10
          }, {
            text: '25', value: 25
          } ],
        }}
      </BootstrapTable>
    </div>
  )
}
```

```
        pagination
      >
      <TableHeaderColumn isKey hidden dataField='apartmentId'>ID
    </TableHeaderColumn>
      <TableHeaderColumn hidden dataField='userId'>userId</Table
HeaderColumn>
      <TableHeaderColumn
        dataField='mansionId'
        dataSort={true} filterFormatted dataFormat={ this.enum
Formatter }

        formatExtraData={ mansionsType }
        filter={ { type: 'SelectFilter', options: mansionsType
} }

      >Mansion Name</TableHeaderColumn>
      <TableHeaderColumn
        dataField='membersCount'
        dataSort={true}
        filter={ { type: 'NumberFilter', delay: 1000, numberCo
mparators: [ '=', '>', '<=' ] } }
      >Members</TableHeaderColumn>
      <TableHeaderColumn
        dataField='number'
        dataSort={true}
        filter={ { type: 'NumberFilter', delay: 1000, numberCo
mparators: [ '=', '>', '<=' ] } }
      >Number</TableHeaderColumn>
      <TableHeaderColumn
        dataField='surface'
        dataSort={true}
        filter={ { type: 'NumberFilter', delay: 1000, numberCo
mparators: [ '=', '>', '<=' ] } }
      >Surface</TableHeaderColumn>
      <TableHeaderColumn
        dataField='floor'
        dataSort={true}
        filter={ { type: 'NumberFilter', delay: 1000, numberCo
mparators: [ '=', '>', '<=' ] } }
      >Floor</TableHeaderColumn>
      <TableHeaderColumn
        dataField='individualQuota'
        dataSort={true}
        filter={ { type: 'NumberFilter', delay: 1000, numberCo
mparators: [ '=', '>', '<=' ] } }
      >Individual quota</TableHeaderColumn>
      <TableHeaderColumn dataField='userName' filter={ { type: '
TextFilter' } } dataSort={true}>User Name</TableHeaderColumn>
      <TableHeaderColumn dataField="actions" dataFormat={this.ac
tionsFormatter}></TableHeaderColumn>
    </BootstrapTable>
```

```
    </div>
  )
}
```

Se poate observa că pentru a face un request de API, s-a folosit “fetch” metoda ce face parte din Fetch API (integrat cu majoritatea browser-urilor).

#### 4.2.5 Stiluri

Stilurile sunt create folosind CSS. Integrat este şi libraria de bootstrap aplicată pentru a crea un design responsive.

```
.navigation-container {
  background-color: rgb(228, 247, 252);
  padding: 0 15px;
  margin: 5px;
  display: flex;
  flex-direction: row;
  align-items: center;
  justify-content: space-between;
  width: 100%;
}

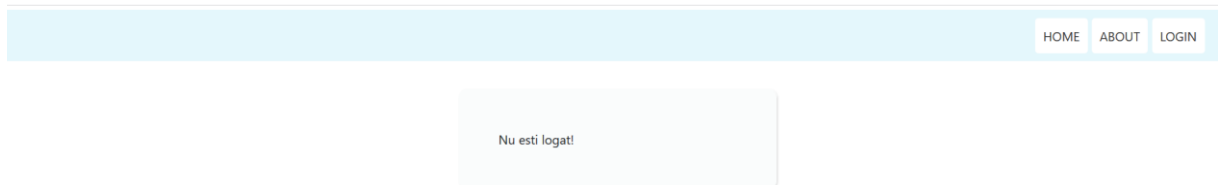
ul.navigation-content {
  display: flex;
  flex-direction: row;
  margin: 10px;
}

.navigation-content li {
  display: flex;
  padding: 10px;
  border-radius: 4px;
  background-color: white;
  margin-right: 5px;
  text-transform: uppercase;
}

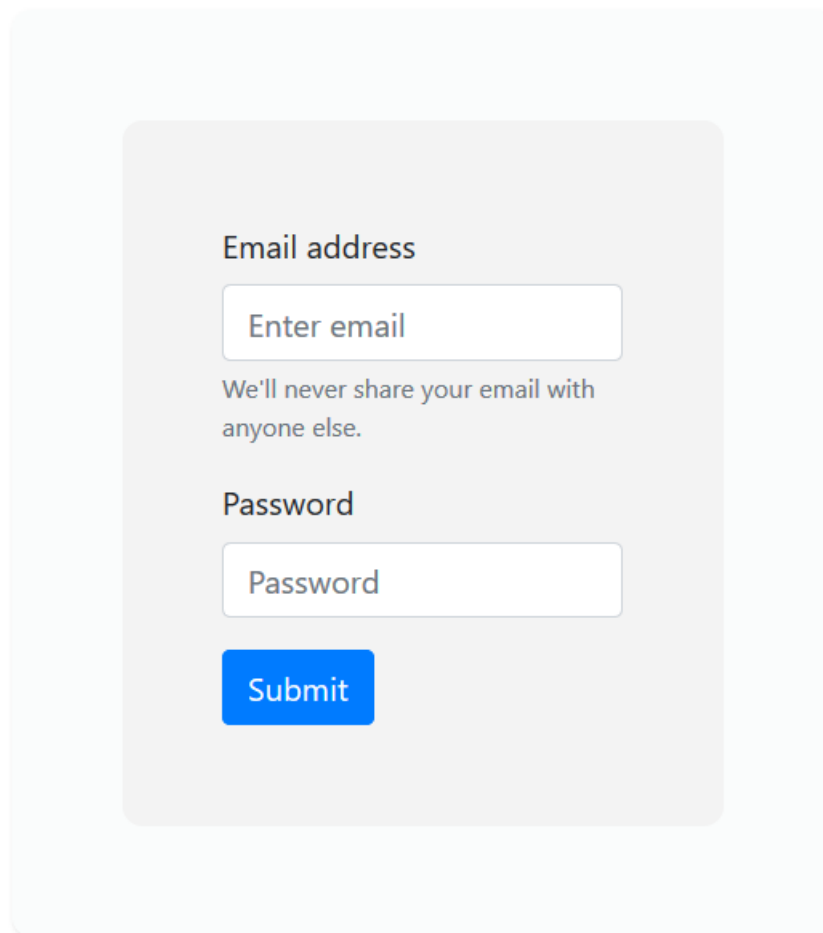
a {
  color: #333;
  text-decoration: none;
}

.navigation-container li:hover {
  background-color: rgb(112, 200, 216);
}
```

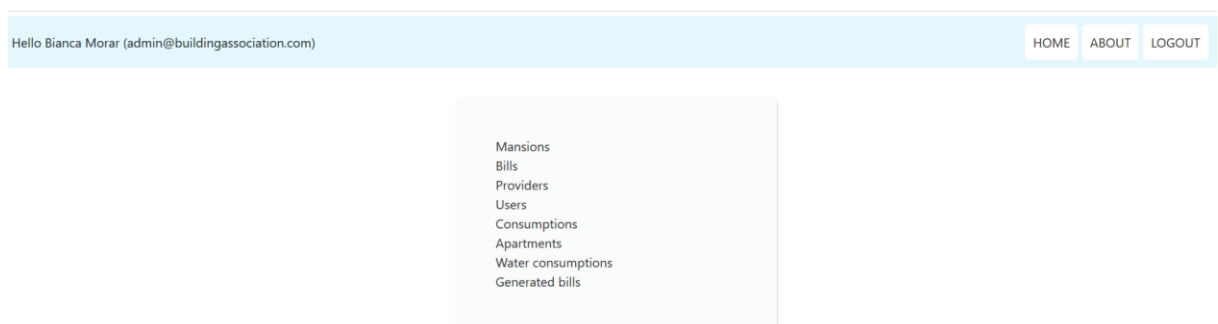
## 4.3 Screenshots



**Fig. 21** Cand nu exista user logat



**Fig. 22** Formular logare













**Fig. 23** Meniu admin

Add apartment

Export to CSV

Search

Mansion Name	Members	Number	Surface	Floor	Individual q...	User Name	
Select Mans...	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	Enter User N...	
Plopului 3 BV	3	6	95	3	3.2	Adrian Ilie	 
Plopului 3 BV	1	5	75	3	2.2	George Ando...	 
Plopului 3 BV	1	4	78	2	2.2	Vlad Popescu	 
Plopului 3 BV	1	3	85	2	2.9	Miruna Straila	 
Plopului 3 BV	4	2	85	1	2.9	Mircea Vasile	 

5

12>

Fig. 24 Pagina in care se listeaza date



## 5. Concluzii

Folosind tehnologiile Microsoft şi limbajele de programare avansate s-a putut crea o aplicaţie aparent existentă dar care vine cu îmbunătăţiri pe partea de infrastructură şi “business logic”. Cu ajutorul frameworkului .NET aplicaţia a fost structurată atent astfel următoarele versiuni vor putea fi dezvoltate uşor.

Folosirea limbajului de programare C# şi a framework-urilor ca LINQ şi Entity, performanţa acestei aplicaţii este mai mare ca a multor alte aplicaţii asemănătoare deoarece apelurile la baza de date sunt mult mai bine structurate şi optimizate.

Codul detaliat al aplicaţiei cât şi comentariile necesare se găsesc în CD-ul atasat acestei lucrări.

## Bibliografie

- [1] [https://en.wikipedia.org/wiki/Web\\_service](https://en.wikipedia.org/wiki/Web_service)
  - [2] <https://www.roweb.ro/ro/tehnologii/ReactJS>
  - [3] <https://www.tutorialsteacher.com/webapi/what-is-web-api>
  - [4] <https://www.tutorialsteacher.com/webapi/web-api-tutorials>
  - [5] <https://www.todaysoftmag.ro/article/81/restful-web-services-folosind-jersey>
  - [6] <https://www.math.uaic.ro/~cgales/csharp/Curs1.pdf>
  - [7] <https://www.tutorialsteacher.com/ioc/unity-container>
  - [8] <https://www.c-sharpcorner.com/UploadFile/4d9083/dependency-injection-using-microsoft-unity-framework/>
  - [9] <https://www.tutorialsteacher.com/linq/what-is-linq>
  - [10] <https://www.guru99.com/sql-server-introduction.html>
  - [11] <https://azimutvision.ro/unit/a-ce-este-html/>
  - [12] <https://getbootstrap.com/>
  - [13] <https://web.ceiti.md/lesson.php?id=16>
  - [14] <https://js.locoman.ro/>
  - [15] <https://ro.sawakinome.com/articles/programming/difference-between-javascript-and-typescript.html>
  - [16] <https://facebook.github.io/jsx/>
  - [17] <https://reactjs.org>
- [www.cs.ubbcluj.ro/~vcioban/Bistrita/Manuale/CursDotNetSassu.pdf](http://www.cs.ubbcluj.ro/~vcioban/Bistrita/Manuale/CursDotNetSassu.pdf)

# ANEXE

Controlere:

## 1. USER CONTROLLER

```
using Services.Contracts;
using System;
using System.Net.Http;
using System.Web.Http;
using System.Web.Http.Cors;
using Website.Helpers;
using Website.Extensions;
using System.Linq;
using Website.ViewModels;

namespace Website.Controllers
{
    [EnableCors(origins: "http://localhost:3000", headers: "*", methods: "*")]
    [BasicAuthentication]
    public class UsersController : ApiController
    {
        private IUserService _userService;

        public UsersController(IUserService userService)
        {
            _userService = userService;
        }

        // GET api/users
        [MyAuthorize(Roles = "Admin")]
        public HttpResponseMessage Get()
        {
            var items = _userService.GetAll().Select(x => x.ToViewModel());
            return Request.CreateResponse(System.Net.HttpStatusCode.Accepted, items);
        }

        // GET api/users/5
        public HttpResponseMessage Get(long id)
        {
            var item = _userService.Get(id).ToViewModel();
            return Request.CreateResponse(System.Net.HttpStatusCode.Accepted, item);
        }

        public HttpResponseMessage Post([FromBody]UserViewModel item)
        {
            try
            {
                var userEntity = item.FromViewModel();
                userEntity.Roles = "User";

                if (userEntity.UniqueId.HasValue)
                {
                    _userService.Update(userEntity);
                }
                else
                {

```

```
        _userService.Insert(userEntity);
    }

    return Request.CreateResponse(System.Net.HttpStatusCode.Accepted);
}
catch (Exception)
{
    return
Request.CreateResponse(System.Net.HttpStatusCode.InternalServerError);
}
}

// DELETE api/users/5
public HttpResponseMessage Delete(long id)
{
    try
    {
        _userService.Delete(id);
        return Request.CreateResponse(System.Net.HttpStatusCode.Accepted,
"Bravo patratel");
    }
    catch
    {
        return
Request.CreateResponse(System.Net.HttpStatusCode.InternalServerError, "Fi atenta!");
    }
}
}
}
```

## 2. PROVIDER CONTROLLER

```
using Services.Contracts;
using System;
using System.Linq;
using System.Net.Http;
using System.Web.Http;
using System.Web.Http.Cors;
using Website.ViewModels;
using Website.Extensions;
using Website.Helpers;

namespace Website.Controllers
{
    [EnableCors(origins: "http://localhost:3000", headers: "*", methods: "*")]
    [BasicAuthentication]
    public class ProvidersController : ApiController
    {
        private IProviderService _providerService;

        public ProvidersController(IProviderService service)
        {
            _providerService = service;
        }

        // GET api/providers
        public HttpResponseMessage Get()
        {
            var items = _providerService.GetAll().Select(x => x.ToViewModel());
            return Request.CreateResponse(System.Net.HttpStatusCode.Accepted, items);
        }
    }
}
```

```
// GET api/providers/5
public HttpResponseMessage Get(long id)
{
    var item = _providerService.Get(id).ToViewModel();
    return Request.CreateResponse(System.Net.HttpStatusCode.Accepted, item);
}

public HttpResponseMessage Post([FromBody]ProviderViewModel item)
{
    try
    {
        var entity = item.FromViewModel();

        if (entity.UniqueId.HasValue)
        {
            _providerService.Update(entity);
        }
        else
        {
            _providerService.Insert(entity);
        }

        return Request.CreateResponse(System.Net.HttpStatusCode.Accepted);
    }
    catch (Exception)
    {
        return
Request.CreateResponse(System.Net.HttpStatusCode.InternalServerError);
    }
}

// DELETE api/providers/5
public HttpResponseMessage Delete(long id)
{
    try
    {
        _providerService.Delete(id);
        return Request.CreateResponse(System.Net.HttpStatusCode.Accepted,
"Bravo patratel");
    }
    catch
    {
        return
Request.CreateResponse(System.Net.HttpStatusCode.InternalServerError, "Fi atenta!");
    }
}
}
```

### 3. NAVIGATION COMPONENT

```
import * as React from 'react';
import './style.css';
import {
    RouteComponentProps,
} from 'react-router';

import { Link } from 'react-router-dom';
```

```
interface NavigationProps
{
  userFullName?: string;
  username?: string;
}

export default class Navigation extends React.Component<RouteComponentProps<any> & NavigationProps> {
  logout = () => {
    sessionStorage.clear();
  }
  render() {
    const locationState: any = this.props.location.state;

    return (
      <div className="navigation-container">
        <div>
          {locationState && locationState.userFullName && <span>Hello {`${locationState.userFullName} (${locationState.username})`}</span>}
        </div>
        <ul className="navigation-content">
          <li>
            <Link to="/">Home</Link>
          </li>
          <li>
            <Link to="/about">About</Link>
          </li>
          <li>
            {sessionStorage.getItem('authToken') !== null
              ? <Link to={{
                pathname: '/',
                state: {from: 'logout'}
              }} onClick={this.logout}>Logout</Link>
              : <Link to="/login">Login</Link>}
          </li>
        </ul>
      </div>
    )
  }
}
```

#### 4. Add component

```
import * as React from 'react';
import { RouteComponentProps, Redirect } from 'react-router';
import { Mansion } from '../../models/Mansion';

interface AddMansionState {
  address: string;
```

```
    totalFunds: number;
    saved: boolean;
  }

export default class AddMansion extends React.Component<RouteComponentProps<any>, AddMansionState> {
  constructor(props: RouteComponentProps<any>) {
    super(props);

    this.state = {
      address: "",
      totalFunds: 0,
      saved: false
    }
  }

  componentDidMount(){
    const { id } = this.props.match.params;
    if(id) {
      this.getItem(id);
    }
  }

  getItem = (id: number) => {
    if(sessionStorage.getItem('authToken') != null) {
      fetch(`/mansions/${id}`, {
        headers: {
          'Authorization': sessionStorage.getItem('authToken')
        }
      }) as RequestInit).then(response => {
        if(response.ok) {
          return response.json();
        }

        return undefined;
      }).then((result: Mansion) => {
        this.setState({ totalFunds: result.totalFunds as number, address: result.address });
      });
    }
  }

  submit = (e: React.FormEvent<HTMLFormElement>) => {
    e.preventDefault();

    const mansion: Mansion = {
      address: this.state.address,
      totalFunds: this.state.totalFunds,
      bills: [],
    }
  }
}
```

```
      users: [],
      consumptions: [],
      id: this.props.match.params.id
    }

    fetch('/mansions', {
      method: 'POST',
      body: JSON.stringify(mansion),
      headers: {
        'Content-Type': 'application/json',
        'Authorization': sessionStorage.getItem('authToken')
      }
    })
  } as RequestInit).then(result => { this.setState({saved: true})});
}

render() {
  if(sessionStorage.getItem('authToken') == null) {
    return <div>
      Nu esti logat!
    </div>
  }

  if(this.state.saved)
  {
    return <Redirect to="/mansions" />
  }

  return (
    <form className="container addmansion-
container" onSubmit={this.submit}>
      <h3>Add mansion</h3>
      <div className="form-group">
        <label>Address</label>
        <input
          type="text"
          onChange={(e) => this.setState({address: e.target.valu
e}) }

          className="form-control"
          defaultValue={this.state.address}
          required
        />
      </div>
      <div className="form-group">
        <label>Total Funds</label>
        <input
          type="number"
          step="any"
          onChange={(e) => this.setState({totalFunds: parseFloat
(e.target.value)}) }
        />
      </div>
    </form>
  )
}
```



```
        className="form-control"
        defaultValue={this.state.totalFunds}
        required
      />
    </div>
    <button type="submit" className="btn btn-
primary">Submit</button>
  </form>
)
}
}
```