****

**Facultatea de Automatică și Calculatoare**

**Ciclu Licență: CTI Română**

**APLICAȚIE WEB PENTRU CAZAREA STUDENȚILOR ȘI REZERVAREA LOCURILOR ÎN CĂMINELE STUDENȚEȘTI**

**Proiect de diplomă**

Student:

**CRISTIAN-SORIN POP**

Profesor îndrumător:

Timișoara, 2019

***Cuprins***

[1. Introducere 4](#_Toc18573325)

[1.1 Context 4](#_Toc18573326)

[1.2 Motivație 5](#_Toc18573327)

[1.3 Scop 6](#_Toc18573328)

[2. Tehnologii folosite 6](#_Toc18573329)

[2.1 Prezentare generală a unei Aplicații Web 6](#_Toc18573330)

[2.2 HTML (HyperText Markup Language) 7](#_Toc18573331)

[2.3 CSS ( Cascading Style Sheets ) 9](#_Toc18573332)

[2.4 Bootstrap 9](#_Toc18573333)

[2.5 JavaScript 10](#_Toc18573334)

[2.6 TypeScript 11](#_Toc18573335)

[2.7 Angular 11](#_Toc18573336)

[2.8 Firebase 13](#_Toc18573337)

[3. Prezentarea aplicației 14](#_Toc18573338)

[4. Implementarea aplicației 15](#_Toc18573339)

[4.1 Prezentarea generală a *componentelor* 15](#_Toc18573340)

[4.1.1 ‘app’ component 17](#_Toc18573341)

[4.1.2 ‘header’ component 19](#_Toc18573342)

[4.1.3 ’home’ component 20](#_Toc18573343)

[4.1.4 ’contact’ component 20](#_Toc18573344)

[4.1.5 ’register’ component 22](#_Toc18573345)

[4.1.6 ’login’ component 24](#_Toc18573346)

[4.1.7 ‘home-log-in’ component 25](#_Toc18573347)

[4.1.8 ’not-found’ component 26](#_Toc18573348)

[4.2 Modulul de rutare & navigare ( *routing module )* 26](#_Toc18573349)

[4.3 Serviciile aplicației 28](#_Toc18573350)

[4.3.1 ‘auth’ service 29](#_Toc18573351)

[4.3.2 ‘get-update’ service 34](#_Toc18573352)

[5. Îmbunătățiri 38](#_Toc18573353)

[6. Concluzie 38](#_Toc18573354)

[7. Bibliografie 38](#_Toc18573355)

# Introducere

## Context

Trăim în secolul XXI, numit de altfel și ”secolul tehnologiei” fiindcă în ultimii 20 de ani tehnologia a avansat extrem de mult aducând o sumedenie de facilități omenirii, ușurând : munca, utilizarea de servicii, accesul la informație etc.

Printre aceste facilități se numără și ***W****orld* ***W****ide* ***W****eb*, sau în termeni uzuali, ***I****nternetul*. Trebuie menționat , de altfel, că acești doi termeni definesc două lucruri diferite :

* ***I****nternet:* reprezintă un sistem global de rețele de calculatoare interconectate care interschimbă date prin pachete utilizând suita standard **I**nternet **P**rotocol (TCP/IP).
* ***W****orld* ***W****ide* ***W****eb:* reprezintă un spațiu infomațional în care elementele de interes, denumite de altfel resurse, sunt identificate prin intermediul identificatorilor globali numiți ***U****niform* ***R****resouce* ***I****dentifier (****URI****).*

***W****orld* ***W****ide* ***W****eb* a apărut inițial ca o viziune a informaticianului englez *Tim Berners-Lee*, în martie 1989,pe când muncea ca inginer software la laboratorul pentru fizica particulelor din cadrul organizației***CERN*** *(Conseil Européen pour la Recherche Nucléaire)* ,ca urmare a nevoii de a partaja informații aflate pe diferite sisteme de calcul. *Tim Berners-Lee* a început să lucreze la viziunea lui în septembrie 1990 folosindu-se de un calculator **NeXT**, unul dintre primele calculatoare produse de către Steve Jobs (anul 1988).

La sfârșitul lunii ocrombrie 1990, *Tim Berners-Lee,* a reușit să scrie trei tehnologii fundamentale care stau ca piatră de temelie și astăzi pentru tot ce semnifică ***W****orld* ***W****ide* ***W****eb*. Cele 3 tehnologii, despre care o să vorbim în detaliu și în capitolele ce urmează, dezvoltate de *Tim Berners-Lee* sunt următoarele :

* **HTML : H**yper**T**ext **M**arkup **L**anguage – limbaj de formatare prin care browser-ul care încarcă pagina web să știe cum va afișa conținutul paginii respective.
* **URI : U**niform **R**resouce **I**dentifier – cunoscut de altfel sub abrevierea **URL**. Reprezintă adresa unică a unei resurse aflate pe web.
* **HTTP : H**ypertext **T**ransfer **P**rotocol – metodă de accesare a datelor de pe Internet.

În 1994 *Tim Berners-Lee* a fondat comunitatea ***W****orld* ***W****ide* ***W****eb* ***C****onsortium (****W3C****)* ce are ca scop dezvoltarea și stabilirea standardelor web pentru a putea beneficia la maxim de tot ceea ce oferă ***W****orld* ***W****ide* ***W****eb.* Printre aceste standarde enumerăm :

* *Descentralizarea :* libertarea de a posta pe Web fără a fi nevoie de aprobarea unei autorități centrale.
* *Non-discriminarea :* cunoscut și sub denumirea de *Net Neutrality* , se referă la faptul că doi utilizatori ce sunt conectați la internet prin intermediul unor servicii de calitate diferită pot comunica între ei la același nivel.
* *Bottom-up Design :* accesul gratuit la codul paginilor Web.
* *Universalitate :* pentru a putea publica orice pe Web , toate calculatoarele implicate trebuie să vorbească aceiași limbă.
* *Consesn :* pentru ca standardele universale să funcționeze, toată lumea trebuie să fie de acord cu aceste standarde.

Acum că ne-am familiarizat cu tehnologiile și standardele ***W****orld* ***W****ide* ***W****eb,* ne vom canaliza atenția, în capitolele următoare, asupra ***Aplicațiilor Web****.* Vom da o definiție a ceea ce reprezintă o aplicație web și cum este compusă aceasta.

## Motivație

În viața de zi cu zi, fie că mergi la magazin să faci cumpărături , fie că ai de plătit o factură la una din serviciile de care beneficiezi, fie că ai de plătit o amendă (din păcate) sau chiar să poti afla anumite informații este nevoie să stai la coadă. Un lucru neplăcut, mai ales dacă ești contra timp, te grăbești și timpul nu îți permite să pierzi nici măcar un minut așteptând până îți va veni răndul. Însă datorită ***Internetului***și ***Aplicațiilor Web*** această problemă a fost rezolvată in foarte multe domenii și servicii scutindu-te de cozi interminabile, nervi și timp pierdut. Acum cumpăraturile stau la cateva click-uri distanță și nici măcar nu trebuie să te deplasezi până la magazinul respectiv deoarece comanda ta îți vine direct acasă. Tot la fel este si pentru plata serviciilor de care beneficiezi, amenzi sau altele. Câteva minute și două, trei click-uri și ai comandat ce ți-ai dorit, ai plătit ce era de plătit, ai aflat informațiile de care aveai nevoie și ți-ai făcut rezervarea la hotel pentru mini vacanța de weekend. Toate aceste facilități sunt oferite astazi de catre ***Aplicații Web*** sau ***Aplicații Mobile.***

**De ce o Aplicație Web ?**

De ce am ales să dezvolt o Aplicație Web? Sunt multe avantaje pe care o aplicație web le poate oferi și am să enumăr câteva dintre ele:

* Comparativ cu alte tipuri de aplicații, aplicațiile web sunt mai ușor de întreținut și se pot dezvolta foarte ușor fără a fi nevoie de anumite programe software specializate.
* Poate fi utilizată pe orice sistem de operare: Windows, Linux, Mac fiindcă toate aceste sisteme suportă browser-e moderne.
* Poate fi modificată oricând fără ca utilizatori să fie nevoiți să facă vre-un update.
* Nu necesită aprobarea vre-unei autorități superioare pentru a putea face accesibilă aplicația web.
* Poate fi accesată atât de pe smart phone-uri cât și de pe orice PC.

Acestea sunt doar câteva avantaje principale care fac aplicațiile web foarte utilizate și căutate. Totodată aceste avantaje m-au determinat și pe mine să aleg dezvoltarea acestei aplicații web.

## Scop

Atras de partea aceasta de Web Development, Web Design, Web Application , observând nevoia și oportunitatea de a crea o nouă aplicație web ce ar putea ușura și fluidiza cazarea studenților și rezervarea locurilor în căminele studențești, am hotarât în urma unei experiente personale să dezvolt o aplicație web în acest sens. Această idee mi-a venit la începutul anului IV de faculate când am fost nevoit să stau 2 zile la coadă ca să pot să îmi fac cazarea în cămin. Așadar, m-am gândit să dezvolt o aplicație web prin care studenții să poată fi cazați mult mai ușor în căminele studențești.

Scopul principal al aplicației web dezvoltate este de a facilita cazare studenților în căminele studențești și posibilitatea de a-și rezerva locul intr-o cameră de cămin alături de prieteni sau colegi de facultate fără a mai sta la cozi interminabile și salvând timp pentru alte activități. Această aplicație nu doar că va avantaja studenții ci și administratorii de cămine, care v-or putea gestiona mult mai ușor studenții cazați, repartizarea acestora sau eventualele probleme întâmpinate.

# Tehnologii folosite

În acest capitol vom cuprinde tehnologiile folosite pentru dezvoltarea acestei aplicații web.Vom prezenta de altfel date și exemple pentru o prezentare cât mai bună .

Dezvoltarea acestei aplicații am realizat-o cu ajutorul editorului de cod/text ***Visual Studio Code***, un editor foarte puternic, ușor de folosit care beneficiază de o multitudine de extensii utile pentru orice limbaj de programare, de formatare, de stilizare fiind disponibil pentru toate sistemele de operare existente.

## Prezentare generală a unei Aplicații Web

O aplicație web este o aplicație ce se dorește a fi folosită prin intermediul unui browser de mai mulți utilizatori, concomitent, punând la dispoziție anumite servicii, beneficii sau informații aflate pe web prin intermediul unei rețele de internet. O aplicație web de asemenea este o aplicație software care a fost dezvoltată într-un limbaj de programare suportat de browser-ele existente.

Orice aplicație web este compusă din trei layere de bază (plus diferite framework-uri), asemenea unui tort (fig 1), fără de care o aplicație web nu ar putea exista.

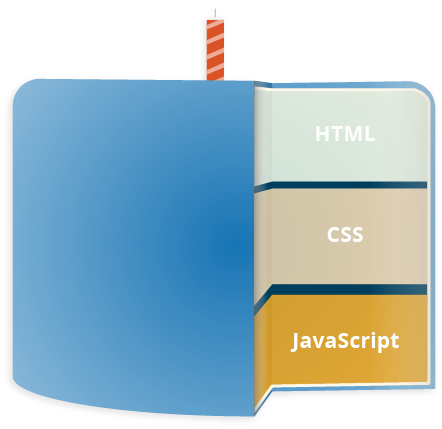


Fig 1. Layere unei aplicații web 1

## HTML (HyperText Markup Language)

După cum am precizat și în introducerea lucrării acest limbaj de formatare a fost creat în anul 1990 de către *Tim Berners-Lee* inițial cu scopul de a partaja documente știintifice, însă apoi a devenit un standard utilizat pentru strucurarea paginilor web.

Conform [1] ***HTML*** reprezintă abrevierea pentru **H**yper**T**ext **M**arkup **L**anguage ce permite utilizatorului să creeze aplicații web bine structurate. *HTML* nu este un limbaj de programare deoarece nu poate crea funcționalități dinamice ci este un limbaj de formatare a textului aflat între marcaje, marcaje numite in limba engleză *tag-uri*.

Să dăm o definiție mai completă a cuvintelor **H**yper**T**ext **M**arkup **L**anguage conform [2], [3] , adică ce semnifică fiecare dintre aceste cuvinte:

* **H**yper**T**ext – este metoda prin ”navigăm” pe web – se referă la acele *link-uri (hyperlinks)* prin care putem accesa sau conecta paginile web aflate in **WWW** .
* **M**arkup – repezintă formatarea pe care *tag-urile* ***HTML*** o fac textului ce se află în interiorul *tag-ului* respectiv. Marcandu-l ca fiind un text de un anumit tip.
* **L**anguage – se referă la faptul că *HTML* este un limbaj ca oricare alt limbaj și de altfel are cuvinte și sintaxă proprie .

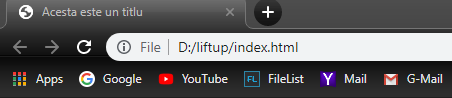
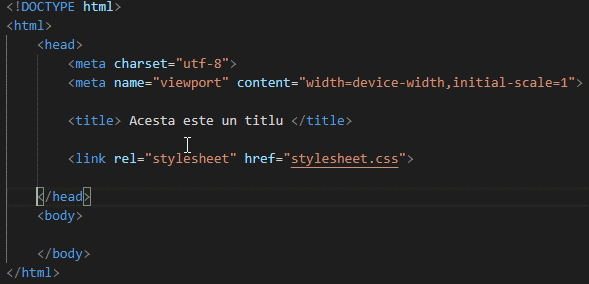
În definițiile anterioare am folosit cuvântul ***tag*** . Ce semnifică acest cuvânt? Conform [3], **tag-urile,**numite si **elemente**, sunt marcaje ce ajută încadrearea unui anumit text unui anumit tip și ajută browser-ul să redea conținutul acestuia utilizatorului. Tag-urile sunt folosite pentru a crea o structură bine definită si ierarhică a tuturor elementelor care compun pagina web, adică va structura felul în care elementele v-or fi afișate în browser .

Tag-urile sunt de forma : **< tag\_name > text < /tag\_name >** ,spre exemplu: **<title>** *Acesta este un titlu* **< /title>** este tag-ul prin care browser-ul setează numele paginii web, așa cum se poate observa în imaginea (fig 1.1).Tot în imaginea (fig 1.1) se poate observa structura unui document *HTML* (fișier cu extensia **.html** sau **.htm**).

Toate elementele, tag-urile, *HTML* pot avea ***atribute*** care furnizează informații suplimentare despre un element. ***Atributele*** sunt declarate în perechi ***nume/valoare***: **nume=”valoare”** , în tag-ul de început al fiecărui element**.** În fig 1.1 se pot observa câteva atribute ce apar în tag-ul de început al câtorva elemente.

Fiecare fisier de tipul *HTML* va trebui să inceapă cu o declarație de tipul **<!DOCTYPE>**, această declarație nereprezentând un *tag*, ci fiind mai degrabă o instrucțiune ce are menirea de a furniza browser-ului datele necesare prin care află ce versiune de *HTML* este utilizată. Tag-ul **<head> </head>** repezintă antetul paginii unde sunt descrise diverse aspecte ale paginii web. Tag-ul **<body></body>** reprezintă conținutul propriu-zis al paginii, ceea ce utilizatorul vede. În interiorul tag-ul **<html></html>** sunt cuprinse antetul si conținutul paginii web, delimitând începutul și sfărșitul paginii web. Limbajul *HTML* este unul de tipul *whitespace-insesitive*, ceea ce înseamnă că permite spațierea fară a exista eventuale probleme.

Fig 1.1Setarea titlului unei pagini HTML; structura unui document HTML



**Declararea unui fișier .css**

## CSS ( Cascading Style Sheets )

Conform [4] ***C****ascading* ***S****tyle* ***S****heets* este un limbaj de stilizare ce oferă un mod eficient și ușor pentru a controla selectiv aspectul elemetelor *HTML* și de asemenea aspectul paginii în întregime. Este un limbaj folosit exclusiv pentru web design.

Acest limbaj de stilizare se scrie și se salvează în fișiere de tipul ***nume\_fișier.css*** . Pentru ca unui document *HTML* să îi fie aplicat un document *CSS*, adică să beneficieze de toate stilurile definite în fișierul .css , acesta trebuie introdus ca referință în interiorul tag-ului **<head>** (fig 1.1).

Un fișier *CSS* este compus din mai multe *seturi de reguli* [5], fiecare regulă avănd următoarea structură (fig 1.2):

* **Selectoru-ul** – numele prin care se identifică elementul căruia i se aplică o *regulă* (numele poate să fie: tag-ul *HTML* , id-ul unu element *HTML* sau clasa atribuită unui element *HTML*).
* **Declarația** – este un câmp de tipul: ‘*propietate: valoare\_proprietate’.* Fiecare *regulă* poate conține mai multe *declarații* separate prin ’;’.

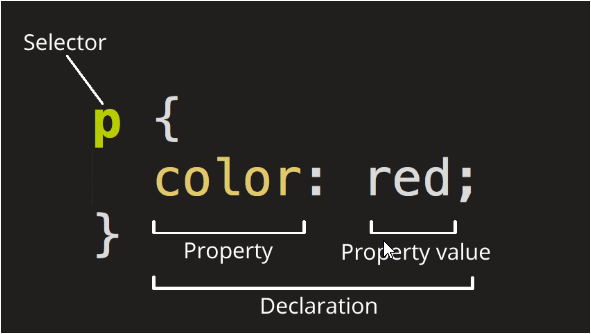


Fig 1.2 Anatomia unei reguli CSS

## Bootstrap

Conform definiției date pe site-ul oficial al ***Bootstrap*** [6]: *Bootstrap* este un framework pentru fornt-end , este un toolkit open source pentru dezvoltarea rapidă de aplicații web și aplicații mobile dinamice, capabile să se rearanjeze, redimensioneze sau să aibă un alt aspect în funcție de dimensiunea ecranului dispositivului de pe care se utilizează aplicația.

Acest front-end framework a fost creat la mijlocului anului 2010 în cadrul companiei Twitter de către designer-ul *Mark Otto* și developer-ul *Jacob Thornton*. Inițial a fost dezvoltat ca un framework open-sourced pentru Twitter, însă în urma unui concurs numit ”*Hack Week*” la care au participat o mulțime de dezvoltatori talentați, în 19 august 2011, *Bootstrap* a devenit oficial un framework pentru *HTML*, *JavaScript* și în special un framework pentru *CSS.*

*Bootstrap* ajută orice dezvoltator de aplicații web sau aplicații mobile să creeze aplicații cu un aspect deosebit, aplicații dinamice care oferă utilizatorului o experiență mult mai plăcută. *Bootstrap* ofera numeroase stiluri *CSS* de baza, cum ar fi butoane, tabele, navigație, formulare, imagini sau alte componente de interfață ce pot fi usor de ajustat si care se pot adapta. *Bootstrap* vine de asemenea si cu cateva componente de *JavaScript* sub forma de plugin-uri *jQuery*. Acestea asigurand elemente grafice aditionale cum ar fi casutele de dialog sau carusele.

## JavaScript

A fost dezvoltat inițial de către *Brendan Eich* [7] de la Netscape Communications Corporation ,în jurul anului 1995, sub numele de Mocha, apoi LiveScript, și denumit în final JavaScript.

Conform [8] ***JavaScript*** (de regulă prescurtat ca ***JS***) este un limbaj simplu, orientat pe obiecte având funcții *first-class* cunoscut ca limbajul de scripting pentru paginile web, dar care este utilizat în multe medii fără browser așa cum sunt *node.js* sau *Apache CouchDB*. Este un limbaj de scriptare bazat-pe-prototip, multi-paradigmatic care este dinamic și care oferă stiluri de programare funcționale, orientate pe obiect și imperative.

*JavaScript* este un limbaj de programare care îți permite să implementezi lucruri complexe pe paginile web de fiecare dată când o pagină web face mai mult decât să afișeze informații statice pentru ca tu să te uiți afișând actualizări de conținut în timp util sau hărți interactive, grafice animate 2D/3D, video jukebox, etc. poți să fii sigur că *JavaScript* este implicat. Este cea de-a treia componentă din tehnologiile web standard , *HTML, CSS, JavaScript.*

Codul *JavaScript* poate fi implementat în interiorul tag-ului *HTML* atâta timp cât sunt specificate tag-urile **<script>** *JavaScript Code***</script>**, însă o mai bună practică este acela de a salva codul *JavaScript* în fișiere cu extensia ***.js*** și legând aceste fișiere de codul *HTML,* prin tag-ul : ***<link href=”nume\_fișier\_javascript.js”>.***

În ciuda faptului că limbajul JavaScript este unul foarte popular și are foarte multe avantaje, are și două dezavantaje destul de mari, și anume: nu poate funcționa fără existența unui program gazdă și acela că nu poate garanta o securitate de nivel inalt.

## TypeScript

***TypeScript*** [9] este un limbaj de programare utilizat pentru programarea la scară largă a aplicațiilor *JavaScript* , este un superset sintactic al limbajului *JavaScript.*

***TypeScript*** [10] pornește de la aceeași sintaxă și semantică pe care milioane de dezvoltatori *JavaScript* o cunosc astăzi. Utilizați codul *JavaScript* existent, încorporați bibliotecile *JavaScript* populare și apelații codul TypeScript de la *JavaScript*. *TypeScript* compilează pentru a curăța codul *JavaScript* simplu, care rulează pe orice browser, în Node.js sau în orice motor *JavaScript* care acceptă ECMAScript 3

***TypeScript***[9] are nevoie de un compilator care poate converti sintaxa *TypeScript* în *JavaScript* standard. Și acest compilator se numește transpiler. Transpilerul este conceput pentru a converti un limbaj de programare în altul. Veți observa că fișierele TypeScript au extensie .ts. Odată ce transpilerul compilează *fișierele* ***.ts***, veți primi *fișiere* ***.js*** ca ieșire. Un alt lucru bun despre *TypeScript* este că vă permite să accesați cadre și biblioteci *JavaScript* comune. Acest lucru este posibil prin asigurarea integrării perfecte a fișierelor de declarație.

## Angular

***Angular*** [11] este o platformă și un framework pentru construirea de *reactive* *Single Page Application* (datorită serviciului de *Routing* pus la dispoziție) în HTML și TypeScript. *Angular* este scris în TypeScript. Implementează funcționalități de bază și opționale ca un set de biblioteci TypeScript pe care le importă în aplicații. Blocurile de bază ale unei aplicații *Angular* sunt NgModules, care oferă un context de compilare pentru componente. NgModules colectează codul aferent în seturi funcționale; o aplicație *Angular* este definită de un set de NgModules. O aplicație are întotdeauna cel puțin un modul rădăcină care permite bootstrapping-ul și, de obicei, are multe alte funcții.

* ***Componentele*** definesc vizualizările, care sunt seturi de elemente de ecran pe care Angular le poate alege și modifica în funcție de logica și datele programului.
* *Componentele* folosesc ***servicii***, care oferă funcționalități specifice ce nu sunt legate direct de vizualizări. Furnizorii de *servicii* pot fi injectați în componente ca dependențe, ceea ce face ca codul să fie modular, reutilizabil și eficient.

Atât *componentele*, cât și *serviciile* sunt pur și simplu clase, cu decoratori care marchează tipul lor și oferă *metadate* care îi spun aplicației *Angular* cum să le folosească. *Metadata* pentru o clasă de componente o asociază cu un șablon care definește o vizualizare. Un șablon combină *HTML*-ul obișnuit cu directivele *Angular* și marcarea obligatorie care permite ca *Angular* să modifice *HTML* înainte de a-l face pentru afișare. *Metadatele* pentru o clasă de servicii furnizează informațiile de care *Angular* are nevoie pentru a le pune la dispoziția componentelor prin *dependency injection* ( *DI* - este o tehnică prin care un obiect furnizează dependențele unui alt obiect ).

Un template combină *HTML*-ul cu marcajul *Angular* care poate modifica elementele *HTML* înainte ca acestea să fie afișate. Instrucțiunile de template oferă logica programului, iar marcarea obligatorie conectează datele aplicației și ***DOM***-ul (***D****ocument* ***O****bject* ***M****odel*).

***D****ocument* ***O****bject* ***M****odel* [12] este reprezentarea datelor asupra obiectelor care cuprind structura și conținutul unui document de pe web, cu alte cuvinte este o interfață de programare pentru documentele *HTML* ce permite ca un program să poată schimba structura, stilul și conținutul unui document *HTML.* Există două tipuri de legare/conectare a datelor:

* ***Event Binding*** – permite aplicației să răspundă oricăror schimbări produse de către utilizator prind update-ul datelor aplicației.
* ***Property Binding*** *–* permite interpolarea valorilor ce sunt evaluate, calculate sau schimbate în interiorul aplicației în documentrul *HTML.*



Fig 1.3 Structura unei Aplicații **Angular**

Angular acceptă *two-way data binding* (legarea în două sensuri a datelor), ceea ce înseamnă că modificările în *DOM*, cum ar fi alegerile utilizatorului, sunt reflectate și în datele programului. În fig 1.3 se poate observa în linii mari structura unei aplicații Angular și tot ce am prezentat până în prezent.

## Firebase

***Firebase*** [13] este o platformă ***Google*** de dezvoltare software a aplicațiilor care ne ajută să construim noi aplicații, să înbunătățim aplicațiile existente și să putem extine/crește aplicația. *Firebase* este o platformă de dezvoltare softwarece pune la dispoziția dezvoltatorilor o vastă gamă de instrumente care acoperă o mare parte din serviciile, pe care în mod tradițional fiecare dezvoltator ar fi trebuit să le construiască singur, prin care pot construi aplicații robuste și rapide. Aceste instrumente se referă la servicii precum : servicii de analiză, servicii de autentificare, baze de date, servicii pentru stocare de fișiere. Toate aceste servicii sunt găzduite în *Cloud* și au componentele de *backend* complet întreținute si opare de *Google*. *Firebase* pune la dispoziția clienților un *SDK (* ***S****oftware* ***D****eveloper* ***K****it )* ce interacționează direct cu aceste servicii *backend* fără a fi nevoie de niciun program intermediar între aplicații și servicii. Prin acest *SDK* , *Firebase* , te scutește de de crearea unui *backend* și îți oferă mai mult timp pentru *frontend* prin care se invocă doar punctele finale ale *API-ului* expuse de *backend*. Diferența dintre dezvoltarea tradițională și ce prin intermediul *Firebase* se poate observa în fig. 1.4.

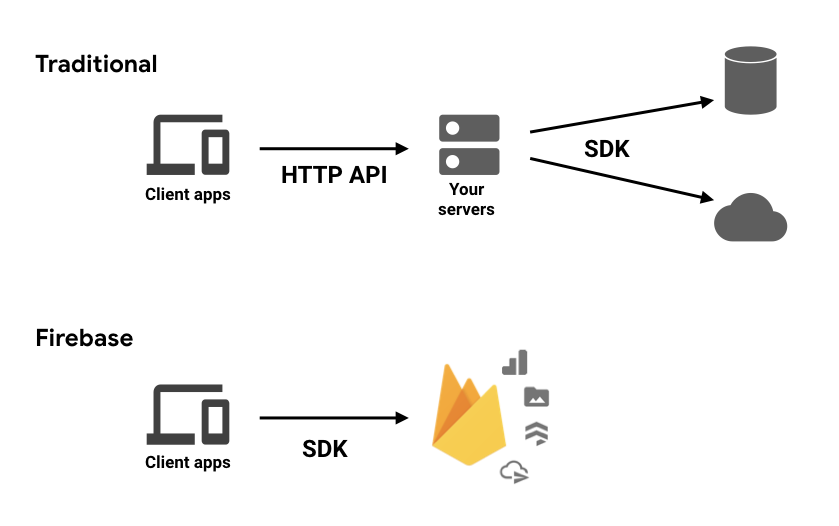


Fig1.4 Backend tradițional / Backend cu **Firebase**

*Firebase* pune la dispoziție 17 servicii individuale (fig 1.5) dintre care pentru dezvolatea aplicației noastre vom folosi serviciul *Auth* și *Realtime Database*. Despre aceste două servicii vom discuta mai pe larg în capitolul X al lucrării.

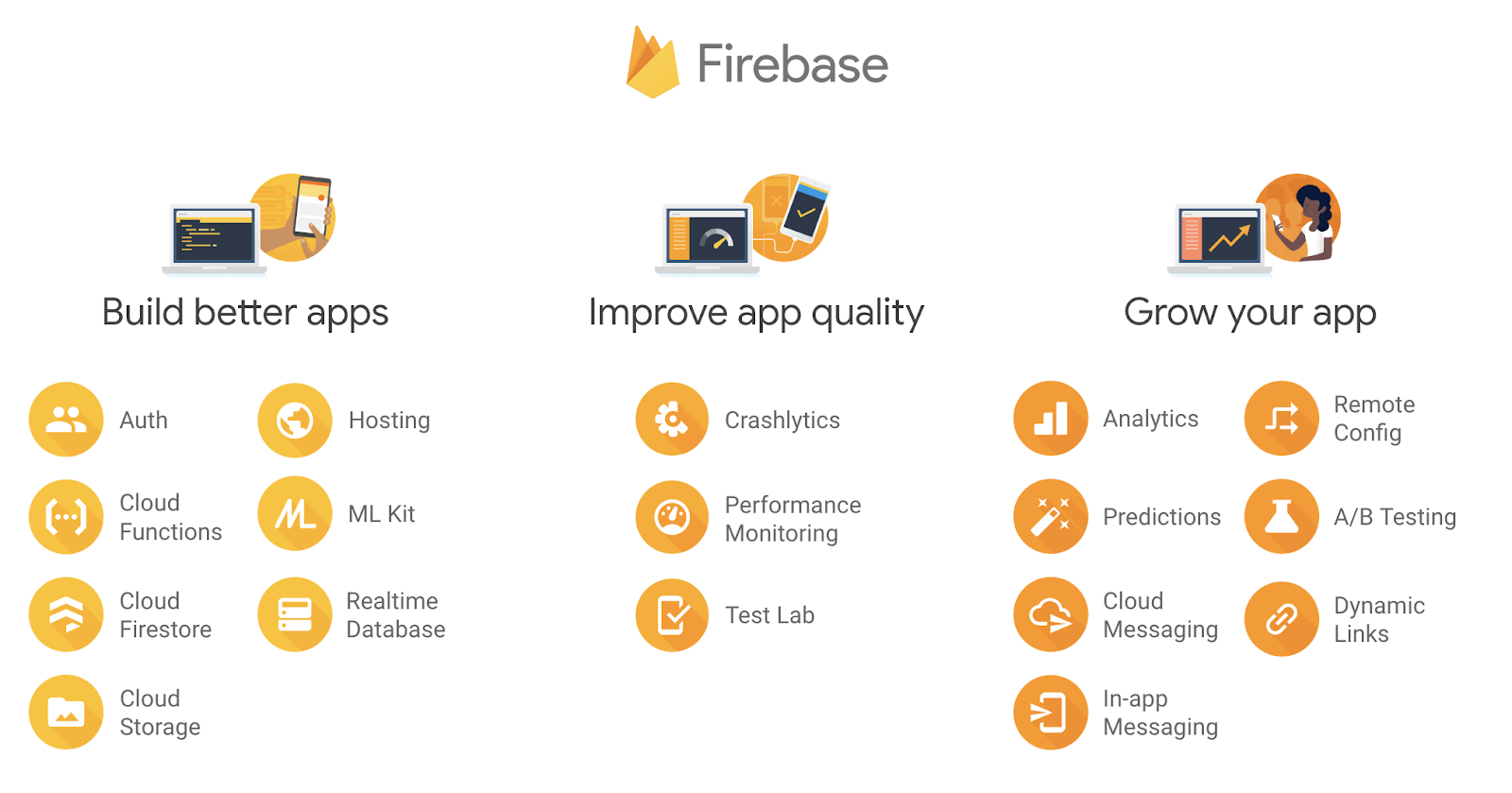


Fig 1.5 Serviciile **Firebase**

# Prezentarea aplicației

Prezentarea aplicației o v-oi face prin intermediul răspunsurilor date la întrebări de tipul: *cine?, ce? cum?* și *de ce?.* Să începem deci, cu prima întrebare.

***Către ce tip de utilizatori se adresează această aplicație ? Cine o va folosi?***

După cum spuneam și în *subcapitolul 1.3* din *Introducere*, această aplicație a fost dezvoltată și este destinată pentru a ajuta studenții din cadrul unei universități sau facultăți ce pune la dispoziție studenților locuri de cazare pe perioada anului universitar și nu numai. Așadar orice student admis în cadrul unei universități sau facultăți va putea utiliza această aplicație pentru aș ușura procesul de cazare în căminele studențești evitând cozile kilometrice și testarea răbdării așteptând să-i vină rândul.

***Ce funcționalități/ servicii oferă această aplicație ?***

Ei bine, aplicația pune la dispoziția studenților câteva servicii simple, ușor de folosit, iar aceastea sunt :

* Vizualizarea datelor personale prin care se face cazarea în cămin.
* Vizualizara camerelor din cămin disponibile pentru rezervare.
* Vizualizarea viitorilor colegi de cameră și a câtorva informații legate de aceștia (precum : număr de telefon, e-mail, facultatea la care este admis).
* Posibilitatea de rezervare în camera dorită.
* Vizualizarea noutăților/anunțurilor privitoate la căminele studențești.

***Cum vor putea studenții să utilizeze aplicația ?***

După cum spuneam și în răspunsul primei întrebări, doar studenți admiși la o facultate din cadrul unei universități sau facultăți individuale vor putea utiliza această aplicație prin intermediul unui cont de utilizator de tipul : *email / parolă* . Studenților li se va da posibilitatea să își creeze astfel de conturi pe baza ***numărului matricol / număr unic de identificare / id*** furnizat de către facultate la care studentul a fost admis.

***De ce să utilizeze studenții aplicația? Care sunt beneficiile aduse prin utilizarea acesteia?***

În primul rând, așa cum am mai menționat, scutește studenții de timp pierdu stând la coadă dându-le posibilitate de a utiliza acel timp în alte scopuri mai benefice. Un al doilea beneficiu este acela de a putea fi la curent cu anunțurile și năutățile privitoare la căminele studențești. Un alt beneficiu este acela de a putea vizualiza cine sunt colegii de cameră și date de contact pentru a putea lua legătura cu ei .

# Implementarea aplicației

Dezvoltarea aplicației de față a fost făcută prin intermediul framework-ului *Angular* ce utilizează toate tehnologiile descrise la punctul 2 creeând un mediu de dezvoltare foarte plăcut, ușor de folosit și scutind programatorul de o mulțime de legături, import-uri, conexiuni care ar fi putu îngreuna procesul de dezvoltare și chiar crearea de eventual erori. Angular mai pune la dispoziți foarte multe librării și pachete care ușurează și mai mult procesul de dezvoltare. Cu ajutorul acestui framework putem crea cunoscutele aplicații reactive,adaptabile ce poartă denumirea de *reactive Single Page Application*. De ce se numesc așa ? Ei bine, ele poartă această denumire pentru că orice modificare a informației prezente în *DOM* (*Document Object Module*), fie schimbarea unei simple imagini, adăugarea unui text, navigarea spre o altă pagină a aplicației se realizează în interiorul browser-ului nu pe un anumit server. Asta înseamnă ca nu mai avem servere ? Nicidecum, ci acestu lucru înseamnă ca randarea aplicației, porcesarea informației provenit de la server (informație ce poate fi și este de altfel stocată în background la runtime) sau informație provenită de la utilizator se realizează în interiorul browser-ului. Acest lucru face ca aplicația să fie foarte rapidă din punct de vedere al performanței dar în mod special oferă o experință inedită utilizatorului.

## Prezentarea generală a *componentelor*

Orice proiect *Angular* este compus din 3 elemente principale: *module, componente* și *servicii.*

***Componentele*** sunt elementele cheie ale aplicației. Acestea sunt clase *JavaScript* ce poartă au o calitate specială date de *decoratorul* ***@Component***. Cu ajutorul *componentelor* practic construim întreaga aplicație web. *Componentele* pot fi create de programator după nevoile sale și de altfel după preferință, însă toate componentele create individual vor fi utilizate intr-o *componentă rădacină*, *componentă de bază* numită *root component* sau *app component* după cum se va putea vedea ,de altfel, pe parcursul acestui capitol. Fiecare componentă are propriul *template*, proproiul cod *HTML*, propriul mod de stilizare și propria sa logică de funcționare. Acest lucru facilitează crearea de aplicații foarte complexe într-un mod foarte simplu,permite reutilizarea unei componente de câte ori este necesar, evitând duplicarea de cod , care ar îngreuna lizibilitate codului și crearea de fișiere imense ca dimensiune și linii de cod. Până acum am discutat în mod teoretic despre *componente* ,haideți acum să vedem practic cum arată o astfel de componentă. Pentru aceasta mă v-oi folosi de o componentă din aplicație și anume componenta *header* (fig 1.6).

Pe scurt voi descrie cele 3 elemente din interiorul decoratorului *@Component*, despre care am menționat mai devreme:

* *‘selector’ :* reprezintă numele prin care vom putea selecta componenta, în cazul nostru componenta *header*. Acest nume poate fi dat după preferință însă în practică se obișnuiește să se dea un nume descriptiv de forma: ‘*app-****numeleComponentei***’.
* *‘templateUrl’ :* în acest câmp se specifică numele documentului *HTML* ce conține *template-ul* care se dorește a fi afișat utilizatorului.
* *‘styleUrls’ :* parantezele drepte ne spun că acest câmp este un *array* ce are ca elemente numele fișierului sau fișierelor *CSS* ce conțin modul de stilizare al *template-ului* specificat în *templateUrl*.

Tot în figura 1.6 și logica componentei *header,* adică metodele pe care aceasta le are, atributele, constructorul clasei și în partea de sus, import-urile necesare ce pot fi după caz: alte componente, funcții, librării sau alte servicii.

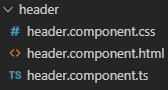


Fig 1.6 Structura unei **componente**

Acum că suntem familiarizați cu termenul de *componentă* am să enumăr *componentele* pe care le-am utilizat în aplicația dezvoltată:

* *contact*
* *header*
* *home*
* *home-log-in* – este o componentă mai complexă ce are la rândul ei alte componente și anume:
* *account-info*
* *home-contact*
* *home-menu*
* *rooms*
* *user-room*
* *login*

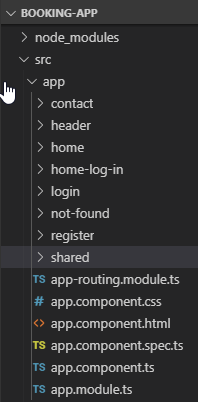


Fig 1.7 Lista **componentelor**

* *not-found*
* *register*

În folderul *‘shared’* se găsesc servicii, modele și alte fișiere ce sunt utilizate de mai multe componente nu doar de o singură componentă.

În continuare, în următoare subcapitole ale acetui subcapitol voi descrie pe scurt rolul fiecare componente și după caz voi și ilustra componenta pentru o descriere mai ușoară. Trebuie specificat faptul că fiecare dintre componentele enumerate se află într-un folder propriu, după cum se poate observa în figura alăturată, care conține trei tipuri de fișiere:

* *nume-componentă.component****.css***
* *nume-componentă.component****.html***
* *nume-componentă.component****.ts***

În sub-subcapitolele următoare vă voi prezenta pe scurt fiecare componentă iar pentru o înțelegere cât mai ușoară a acestora mă voi folosi de imagini cu părți din codul componentelor (code-sniped) și aspectul câtorva dintre aceste componente.

### ‘app’ component

În subcapitolul 4.1 specificam faptul că această componentă este componenta de *bază* sau componenta *rădăcină* a proiectului. În această componenta utilizăm toate celelalte componente pe care le-am creat iar la final această componentă va fi cea pe care o vom folosi în fișierul de bază , *index.html*, al aplicației, fișier ce va fi citit și procesat de către browser-ul nostru.

Spuneam că în *app-component* ne vom folosi de celelalte componente create. De unde știe această *componentă de bază* care sunt celelalte componente pe care să le utilizeze și mai exact de unde poate lua aceste componente ? După cum am mai spus, orice aplicație *Angular* are 3 elemente de bază *componente, servicii* și *module.* În aplicața dezvoltă am folosit un singur modul, modul de bază ce se găsește sub denumirea de *app.module.ts*. Desigur, într-o aplicație Angluar se pot crea mai multe *module* însă pentru aplicația de față acest lucru nu fost necesar.

În orice aplicație *Angular,* orice componentă, *serviciu* sau *modul* trebuie specificate în modulul de bază menționat mai sus deoarece *Angular* nu va prelua în mod automat aceste *componentele, serviciile* sau *modulele* create de noi până nu specificăm noi că am creat astefl de elemente. Cu ajutorul figurii 1.8 voi prezenta aceast *modul app.module.ts* , însă înainte de a face acest lucru vom mai specifica câteva aspecte legate de acest *modul.*

Un *modul* este de altfel tot o clasă dar o clasă specială deoarece are *decoratorul* ***@NgModule*** care preia un obiect de tip *metadata* care descrie cum să compileze *template-ul* unei componente și cum să creeze un *injector* la runtime (timpul de execuție). Prin intermediul acestui *decorator* ***@NgModule*** identificăm componentele, directivele, serviciile și pipe-urile modului, făcându-le publice pe unele dintre aceastea prin proprietate *export* , astfel încât componentele externe să le poată utiliza. ***@NgModule*** poate adăuga, de asemenea, furnizori de servicii în *injectoarele de dependențe* ale aplicației (*application dependency injectors).* Unele librării importate precum *FormsModule, HttpClientModule* sau *RouterModule* sunt de altfel tot module de tipul *NgModule* (fig. 1.9). Așadar *NgModule* ajută la configurarea injectorului și a compilatorului ajutând la organizarea elementelor și legarea acestora împreună.

Acum că știm care e rolul modului nostru și ce face acest *decorator* ***@NgModule*** să vedem ce conține defapt acest *decorator* :

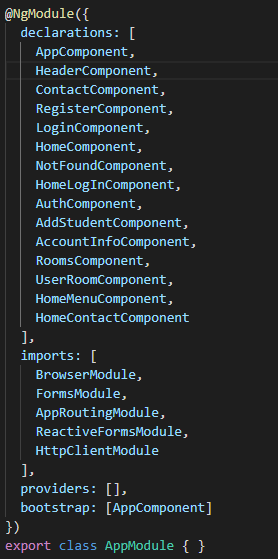


Fig. 1.8 Decoratorul **@NgModel**

* *‘bootstrap’* : este un array care conține toate componentele pe care aplicația *Angular* ar trebui să le folosească atunci când fișierul *index.html* este analizat, la pornirea aplicației.
* *‘declarations’ :* este un array care conține toate componentele pe care le-am declarat și le-am creat și de altfel componente pe care le-am utilizat pentru crearea acestei aplicații.
* *‘imports’* : este un array în care putem importa,specifica și alte *module* de care avem nevoie pentru dezvoltarea aplicației ( de exemplu *BrowserModule* este un modul care ne oferă funcțioalitățile de bază pentru ca aplicația să porneasca ).
* *‘providers’*: este un array ce conține servicii. Aceste servicii,dacă sunt declarate în acest array, pot fi utilizate de către toate celelalte componente ale aplicației. Există și o alternativă pentru specificarea unui serviciu și acesta este prin utilizarea *decoratorului* ***@Injectable*** prin parametrul *providedIn: ‘root’;* (fig. 2).

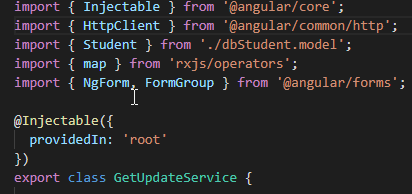


Fig. 2 Utilizarea decoratorului **@Injectable** într-un serviciu

Pe lângă faptul că trebuie să specificăm în ***@NgModule*** componentele,serviciile și alte module este nevoie să specificăm și calea către aceastea deoarece *TypeScript* nu știe de unde să ia elementele specificate și nici nu va putea să facă legătura între ele. Din acest considerent în orice clasă a undei componente, a unui serviciu sau al unui modul este nevoie să specificăm cu ajutorul cuvântului cheie *import* toate elementele pe care le folosim în clasa respectivă. În figura 1.9 se pot observa astefl de *import-uri* ,prezente și în aplicația noastră în modului *app.module.ts* despre care am vorbit mai devreme.

### ‘header’ component

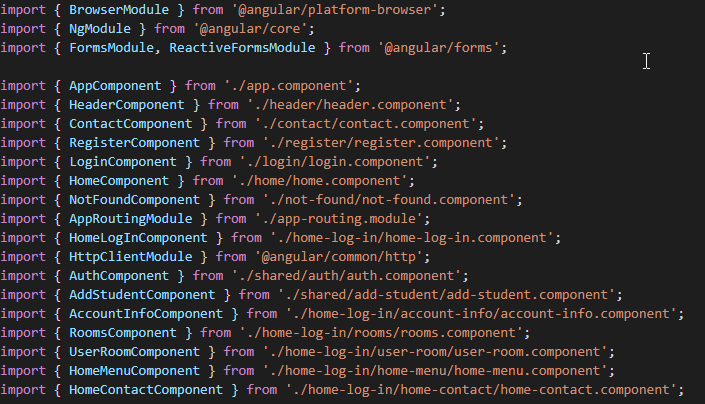


Fig. 1.9 Import-urile din modulul **app.module.ts**

Este componenta ce reprezintă header-ulaplicației, header ce conține de altfel bara principală de navigație a aplicației noastre. În figura de mai jos (fig. 2.1) se poate observa codul *HTML* pentru această bară de navigație.

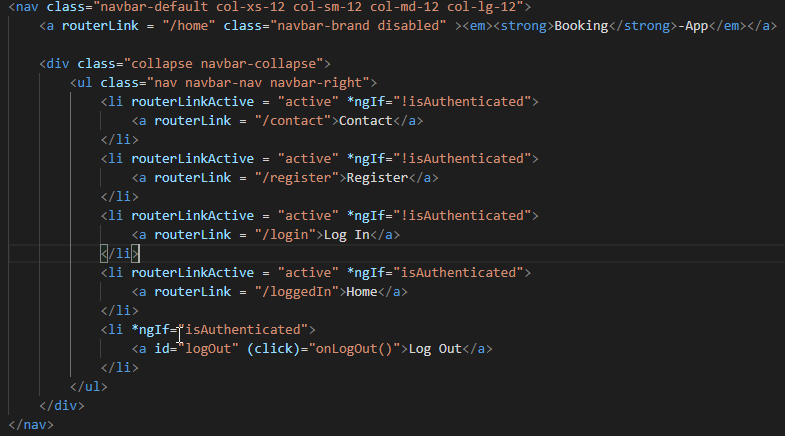


Fig. 2.1 Codul HTML pentru bara de navigație

Această bară de navigație (fig. 2.2) conține în partea stângă un *logo*, care poate fi, după caz, o imagine cu un logo personalizat sau un logo simplu, de tip text ca și în cazul de față. În partea dreaptă se pot observa, pentru început, 3 butoane care poartă câte un nume specific pentru funcționalitatea lor și anume :

* butonul *‘Contact’* : acest buton duce utilizatorul către un formular de contact prin care poate specifica orice tip de probleme care le întâmpină utilizând aplicația.
* butonul ‘Register’: acest buton duce utilizatorul către un formular de înregistrare prin care se poate înregistra în baza de date a aplicație pentru ca ulterior să poată utiliza aplicația.
* butonul ‘Log In’: acest buton duce utilizatorul către componenta de logare în aplicație , de unde acesta se va putea loga prin email și parolă.

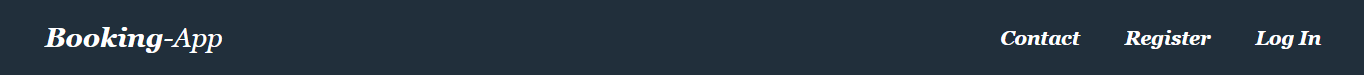


Fig. 2.2 Bara de navigație din componenta **header**

### ’home’ component

Această componentă reprezintă, după cum sugerează și numele acesteia, pagina de *acasă* a aplicației care poate fi modificată după preferința fiecărei facultăți, oferind detalii, anunțuri, imagini sau anumite direcții privitoare la utilizarea aplicației. În momentul de față, această componentă este foarte simplă, conține un mesaj de bun-venit și o imagine ce se vrea a reprezenta o facultate/cămin studențesc.

### ’contact’ component

*Componenta de* *contact – contact component -* este o componentă care conține un formular (fig. 2.3) ce poate fi completat de studenți în cazul în care întâmpină probleme la înregistrare, la logare, pe parcursul utilizării aplicației, posibile funcționalități care nu execută corect anumite servicii sau funcții. Toate aceste formulare completate vor fi salvate în baza de date *Firebase* pentru a putea fi apoi gestionate de către administratori sau persoanele desemnate pentru rezolvarea problemelor.

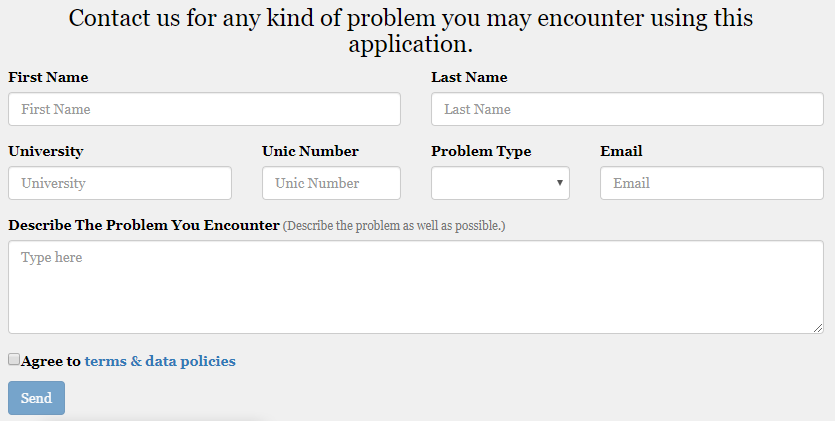


Fig 2.3 Formularul de contact

Pentru crearea acestui formular am folosit una din cele două metode de abordare a formularelor oferite de *Angular* și anume metoda ***Reactive.*** Acastă metodă este o metodă mai complexă deoarece formularul va fi creat pe cale programatică care necesită o pregătire mai amplă. Pentru început se va crea formularul și structura sa în cod *TypeScript*. Al doilea pas presupune scrierea codului *HTML* pentru formularul dorit. Al treilea, și ultimul pas, este legarea manuală a formularului creat în *TypeScript* și codul *HTML.* Această metodă de abordare a forumlarelor oferă control total si complet asupra unui formular. Această metodă de creare a formularelor este pusă la dispoziție de *modulul ReactiveFormsModule* (fig. 1.9) ce conține toate uneltele si elementele necesare pentru a conecta formularul creat în *TypeScrip* și cel din codul *HTML.*

*Cum creăm un formular în TypeScript?* Un astfel de formular este un obiect de tipul *FormGroup* conține mai multe elemente de control de forma ‘***key: value****’*, în care ***‘value’*** este de asemenea un obiect însă obiect de tipul *FormControl,* iar ***‘key’*** reprezintă numele prin care se va putea accesa valoarea obicetului. Codul *TypeScript* responsabil pentru crearea fromularului nostru este redat în figura 2.4 de mai jos. *FormControl* este un obiect care poate avea mai multe argumente, dar în cazul nostru am utilizat doar două argumente și anume : argumentul *value (primul argument)– prin care setăm valoarea inițială a obiectului* și argumentul *validators(al doilea argument) – acest argument poate conține o singură valoara sau un array de valori.* Argumentul *validators* se referă la niște funcții de validare predefinite sau definite manual care ai rolul de a verifica anumite aspecte legate de formular sau valoarea unui control al formularului. Pentru acest formular am folosit obiectul *Validators* care pune la dispoziție mai multe funcții de verificare. Dintre funcțiile puse la dispoziție de acest obiect, am folosit doar două: *Validators.required – acestă funcție face ca acest control să fie obligatoru, adică să fie completat de către student; Validators.email – această funcție verifică dacă valoarea controlului se conformează unui model de tip* ***email*** *.*

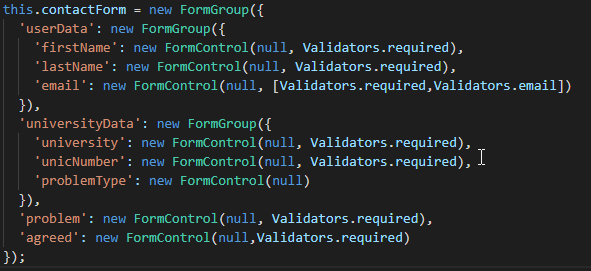


Fig 2.4 Crearea formularului în cod TypeScript

*Cum creăm un formular în cod HTML și cum legăm acest formular de cel din codul TypeScript?* Un formular în cod *HTML* este creat în interiorul unui element ce are *tag-ul* ***<form></form>*** și cu ajutorul elementelor de tip ***input*** ce sunt create cu ajutorul *tag-ului* ***<input>***. Pentru a putea lega formularul din codul *HTML* de cel din codul *TypeScript* în interiorul *tag-ului* ***<form></form>*** vom adăuga o directivă numită ***formGroup*** (fig. 2.5). Acestei directive îi vom specifica prin *property binding* faptul că dorim să creăm un formular care are structura definită în codul *TypeScript*. Ca formularul nostru să fie complet și funcțional mai trebuie să specificăm care dintre *input-urile* din codul *HTML* trebuie legate la elemetele de control din formularul creat în *TypeScript.* Acest lucru îl realizăm tot printr-o directivă și anume directiva ***formControlName*** prin care specificăm numele controlului căruia dorim să îi atribuim valoarea unui *input* (fig. 2.5).Numele trebuie să fie unul dintre numele cheii ***key*** date la crearea formularului în codul *TypeScript* . După realizarea acestor două etape cele două formulare sunt legate și sincronizate.

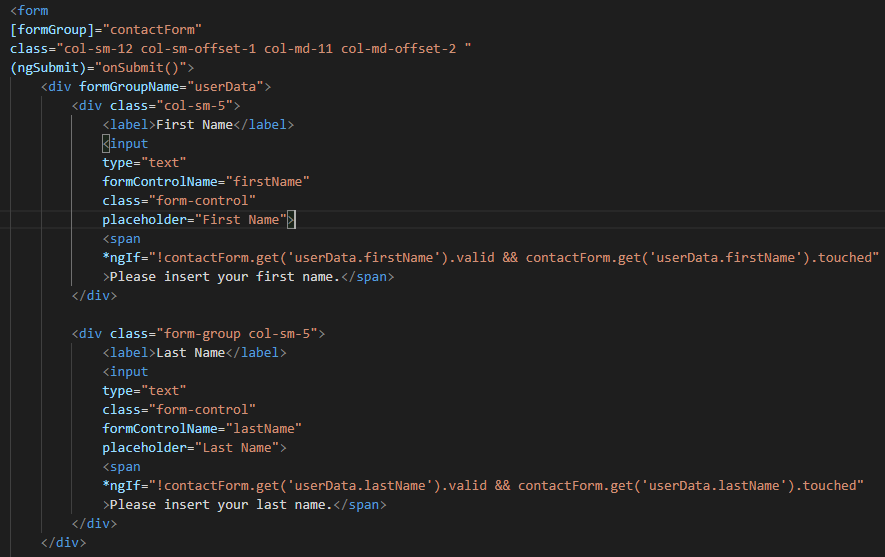


Fig. 2.5 Parte din codul HTML prin care s-a creat formularul de contact

Tot în *template-ul* acestei componente avem două elemente *HTML* vizibile doar în două cazuri:

* *cazul de eroare –* dacă o eroare a apărut aceasta va fi semnalată vizibil afișând un mesaj de eroare corespunzător.
* *cazul de succes –* dacă formularul a fost trimis cu succes se va afișa un mesaj de succes corespunzător.

Pe lângă aceste elemente mai există un element ce poate conține informații (fig. 2.3 – încadrat cu albastru-închis) precum : termeni & condiții , termeni & politici despre prelucrarea datelor cu caracter personal, reguli sau alte informații în funcție de diferite circumstanțe.

### ’register’ component

*Componenta de înregistrare – register component –* este formată tot dintr-un formular (fig. 2.6), însă după cum sugerează și numele acestei componente, acesta este un formular de înregistrare care permite studenților să își creeze un cont bazat pe *email* și *parolă* prin care să poată folosi aplicația.

Pentru crearea acestui formular am folosit cea de-a doua metodă de abordare a formularelor oferită de *Angular* și anume metoda ***Template-Driven (TD).*** Această metodă presupune crearea formularului în *template,* în cod HTML, iar *Angular* va deduce automat structura formularului, toate *elementele de control* existente.

*Cum reușește Angular să realizeze aces lucru? Angular* pune la dispoziție un modul special creat pentru formulare și anume modulul *FormsModule*.Acest modul folosit de *Angular* va crea în mod automat un formular, mai degrabă un obiect *JavaScript* care va fi defapt o reprezentare a formularului, atunci când in *template-ul* componentei *Angular* va detecta un element/tag de tipul ***<form></form>.*** *Angular* va crea un astfel de obiect însă acesta va fi compelt gol fiindcă *Angular* nu va detecta în mod automat si elementele de *input* din interiorul formularului sau alte elemente care s-ar putea să conțină informații pe care dorim totuși să le salvăm sau să nu le salvăm în formularul nostru. De aceea este nevoie ca să furnizăm manual informații cu privire la structura formularului nostru și acest lucru îl facem în *template-ul* componentei.

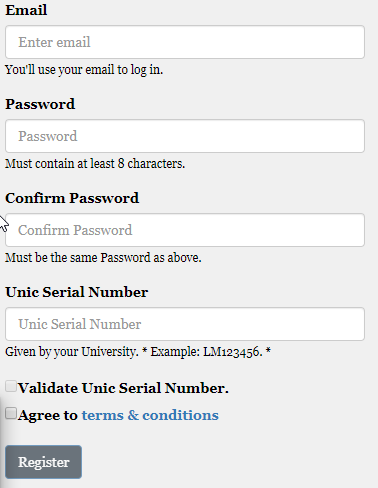


Fig. 2.6 Formularul de înregistrare

*Cum se realizază acest lucru ?* În primul rând, trebuie să anunțăm *Angular* ce input sau ce element dormi să fie un element de control al formularului prin directiva ***NgModule*** pusă la dispoziție de modulul importat , modulul *FormsModule.* Pe lângă această directivă este nevoie să specificăm și numele elementului de control care va conține informația și prin care putem să acesăm informația pe care acesta o deține. Acest lucru se realizează prin atributul *HTML* numit ***name***. În figura 2.7 se poate observa o parte din codul *HTML* ,din acest formular, în care s-au utilizat directivele si atributul despre care am vorbit mai sus.

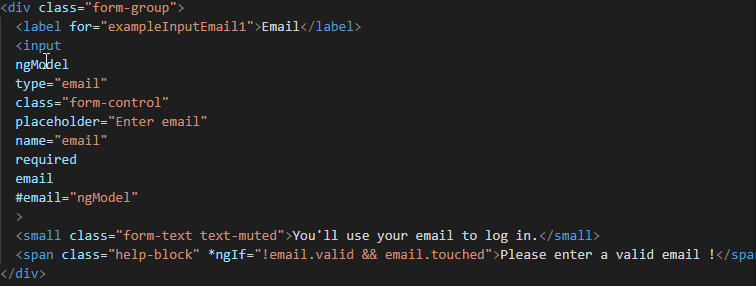


Fig. 2.7 Secvență din crearea formularului de înregistrare

Tot în figura 2.7 putem observa 3 elemente noi despre care nu am dat nici o informație, și anume : ***required , email , #email=”ngModel”***. Primele două elemente, ***require*** și ***email*** sunt două funcții de validare , la fel ca cele despre care am vorbit în componenta *contact*. ***Require*** cere ca *input-ul,* în care acesta este specificat, să fie completat obligatoriu. ***Email*** de asemenea cere ca *input-ul*, în care acesta este specificat, să fie de forma unui *email* valid. Cel de-al treilea element ***#email=”ngModel”*** este o variabilă locală prin care putem accesa anumite proprietăți ale formularului, proprietăți care ne furnizează informații referitoare la formular și prin care putem stiliza formularul. Spre exemplu, tot în figura 2.7 în cadrul elementului ***<span>*** avem un mesaj de informare prin care anunțăm studentul că email-ul introdus nu este valid sau nu a introdus nici un email în câmpul *email* din formular. *Cum se realizează afișarea acestui mesaj?* Afișarea acestui mesaj se face prin accesarea proprietăților formularului prin variabila locală ***#email***. Prin această variabilă accesăm două propietăți și anume: proprietatea ***valid*** și proprietatea ***touched***. Prin proprietate ***valid*** verificăm dacă *input-ul* este valid, adică dacă studentul a introdus un email valid, iar prin proprietatea ***touched*** verificăm dacă studentul a vizitat câmpul respectiv și nu a completat acel câmp. Dacă expresia ***!email.valid && email.touched*** este evaluată și returnează *true* mesajul elementului ***<span>*** va fi afișat, iar dacă se returnează *false* acesta nu va fi afișat. Acest lucru este realizat printr-o directivă structurală (directivă prin care modificăm structura ***DOM-ului*** ) și anume directiva ***ngIf*** prin care putem afișa în mod condiționat anumite elemete *HTML*, în cazul de față mesajul prin care anunțăm studentul că nu a introdus un email sau nu a introdus un email valid.

Înregistrarea unui student se va face în mod condiționat, și anume, dacă studentul este admis în cadrul undei facultăți și i s-a furnizat un *cod unic de identificare*, acesta va putea să se înregistreze și să utilizeze aplicația. Prin această condiție evităm crearea conturilor multiple sau accesarea aplicației de utilizatori care nu sunt studenți admiși în cadrul facultății.

### ’login’ component

*Componenta de login* este o componentă relativ simplă ,tot un formular, ce conține două câmpuri și un buton (fig. 2.8). Un câmp pentru *email* și un câmp pentru *parolă*. Dacă acest două câmpuri sunt completate corect cu un email valid și o parolă care sunt înregistrate în baza de date *Firebase* studentului i se va permite accesul în aplicație și va fi redirecționat către meiul principal al aplicației ce îi va permite să utilizeze mai multe servicii si să acceseze mai multe informații.



Fig. 2.8 Panoul de Login al aplicației

La fel ca și în componenta *contact* această componentă *login* conține elemente *HTML* ce vor conține mesaje de eroare corespunzătoare, spre exemplu: dacă studentul a introdus emailul sau parola greșit sau dacă emailul nu este înregistrat sau daca cumva legătura cu baza de date nu s-a putut realiza. Afișarea acestor elemente *HTML* se va face condiționat, doar în cazul în care o eroare apare. Afișarea condiționată se realizază prin directiva structurală ***ngIf*** despre care am vorbit mai pe larg în componenta *register*. În figura 2.8 se poate observa faptul că butonul este dezactivat, adică acesta nu va putea fi apăsat decât după ce câmpurile email și password vor fi completat cu date corespunzătoare, adică formularul va deveni valid.

### ‘home-log-in’ component

Această componentă este o componentă ce conține mai multe componente, componente ce vor fi accesate doar de studenții înregistrați. Am să prezint pe scurt fiecare subcomponentă din aceasta componentă *home-log-in*.

1. *home-menu component –* este componenta ce conține meniul principal al aplicației (fig. 2.8.1) prin care studenții vor putea accesa diferite servicii și informații.
2. *account-info component* – este o componentă ce furnizează infomații despre studentul care s-a logat în aplicație și permite modificarea anumitor infomații, de către student, în cazul în care acesta constată că nu sunt corecte (fig 2.8.1).
3. *home-contact component –* este o componentă prin care studentul poate raporta anumite probleme sau poate contacta unul din personalul autorizat al aplicației pentru ajutor sau anumite informații (fig 2.3).
4. *rooms component –* este componenta care va afișa camerele disponibile din căminul în care studentul a fost repartizat și îi va oferi posibilitatea să rezerve un loc în camera dorită de acesta.
5. *user-room component –* este componenta prin care studentul va putea vedea mai multe informații despre camera în care s-a cazat și informații suplimentare despre colegii de cameră, precum: număr de telefon, email, faculate .

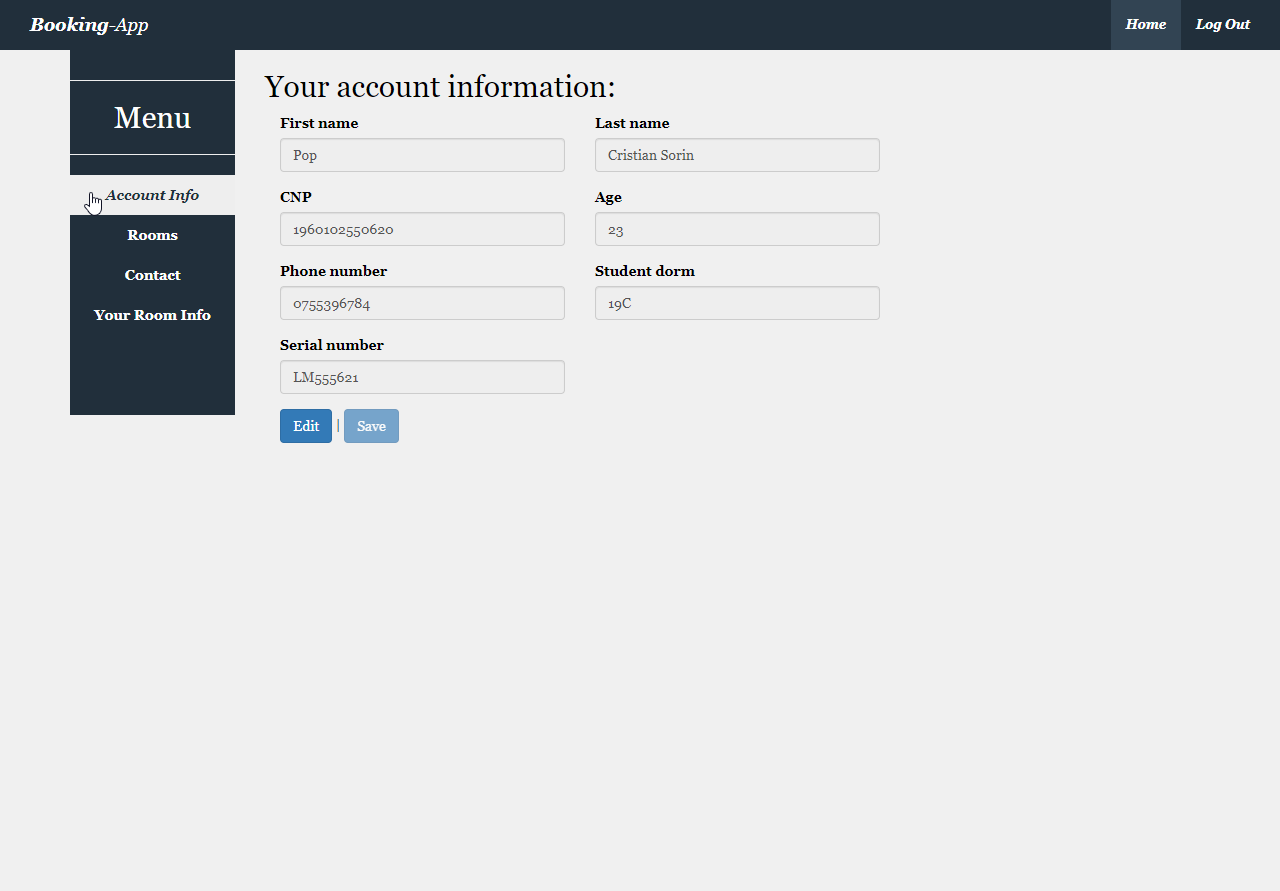


Fig. 2.8.1 **home-log-in** component

### ’not-found’ component

Este o componentă care conține un mesaj prin care studentul este înștiințat că a incercat să acceseze o componentă sau o parte a aplicației care nu există. Spre exemplu, dacă voi încerca să accesez [***http://localhost:4200/menu***](http://localhost:4200/menu), acest link mă va duce către această componentă (fig. 3 ) deoarece această cale nu este definită în aplicația noastră în *modulul de routing*, despre care o să vorbim în capitolul următor.



Fig. 3 **Not-found** component

## Modulul de rutare & navigare ( *routing module )*

Gestionarea tranzițiilor de stare este una dintre cele mai grele părți atunci când vine vorba de construirea unei aplicații, în mod special al unei aplicații web, unde fiecare stare (pagină diferită vizitată) trebuie să fie reflectată într-o anumită măsură și în adresa *URL*. În cazul nostru aplicația este construită, așa cum am văzut până acum, din mai multe componente pe care dorim să le afișăm în *DOM* doar atunci când studentul dă click pe unul din butoanele din bara de navigație sau când dorește să acceseze direct (prin URL) o anumită pagină a aplicației. În mod practic, la fiecare schimbare a *DOM-ului* noi nu ne dorim să părăsim pagina, să nu încărcăm o nouă pagină, ci doar schimbăm conținutul acesteia, oferind doar impresia că ne aflăm intr-o nouă pagină când defapt să rămânem în aceiași pagină. Dorim să acoperim 3 metode de accesare a unei pagini în aplicația dezvoltată și anume:

* Dacă introducem o adresă *URL* în bara de adrese, browser-ul să fie capabil să acceseze pagina corespunzătoare.
* Dacă dăm click pe un anume buton sau element ce conține un *link* , browser-ul să poată să ne ducă la pagina dorită sau să ne ofere o nouă pagină.
* Dacă dăm click pe butonul de *back* sau *forward* al browser-ului, să putem accesa paginile vizitate înainte sau înapoi.

Serviciul de *routare Angular* rezolvă această problemă ocupându-se de stările aplicației, de tranzițiile între acestea, încărcarea componentelor specifice cât și afișarea diferitelor stări în adresa *URL.* Pentru început este nevoie să stabilim care sunt defapt *rutele* din aplicația noastră pentru că serviciul de *rutare* nu are rute prestabilite sau detectate automat ci doar dacă le definim noi. În acest sens va trebui să definim un array care să conțină aceste rute pe care le putem accesa. Acest array va fi un array de tipul *Routes* prin care informăm aplicația *Angular* de structura pe care o să o avem pentru rutele pe care le definim în interiorul array-ului. Structura acestui array este prezentată în figura 3.1.

Fiecare *rută* are structura: **{ *path: ‘calea’ , component: numeleComponentei* }***.* ***Path*** reprezintă calea, adresa *URL* prin care se accesează. Spre exemplu, luând din figura 3.1 a doua cale: ***{path:'contact', component: ContactComponent},*** adresa *URL* pe care o definește aici***path*** este: *http://localhost:4200/****contact ,*** iar ***component*** repezintă acțiune ce se va face atunci când adresa definită de ***path*** este accesată, și anume, va încărca componenta ***ContactComponent*** în browser. Tot în acest array avem încă două rute care parcă nu se conformează structurii pe care am definit-o mai sus. Am să încep cu ultima rută din array. Ruta ***{path:'\*\*', redirectTo:'/not-found'}*** se ocupă de cazurile în care se încearcă (prin adresă *URL*) să se acceseze rute inexistente, nedefinte de noi. În astfel de cazuri utilizatori vor fi redirectionați spre ruta ***not-found*** care va încărca componenta ***NotFoundComponent*** ce va informa utilizatorul că adresa *URL* pe care încearcă să o acceseze, nu există.

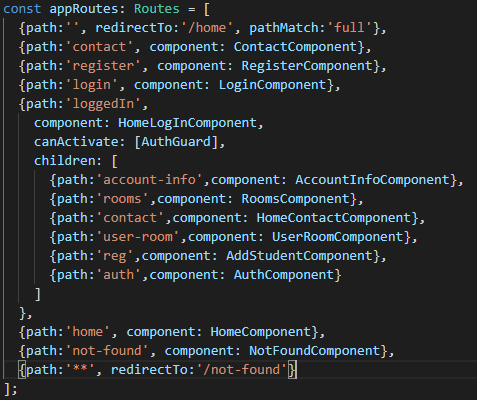


Fig. 3.1 Rutele aplicației

Cea de a doua rută este defapt tot o rută însă o rută care are și rute-copii, adică pentru a accesa, spre exemplu, ruta *account-info,* va trebui ca adresa *URL* să fie de forma *http://localhost:4200/****loggedIn/account-info,*** o rută compusă, multiplă. În plus față de celelalte, pe lângă argumentul ***children***, avem argumentul ***canActivate***. ***canActivate*** specificăm un serviciu prin care setăm un ”*gardian”* care permite accesul la aceste rute doar sub anumite condiții. ***AuthGuard*** este un serviciu prin care implementăm faptul că doar utilizatori înregistrați care s-au conectat vor putea accesa aceste rute, iar în cazul în care utilizatorul nu s-a conectat dar încearcă prin adresa *URL* să acceseze rutele, ”*gardianul”* îl va redirecționa spre pagina de conectare.

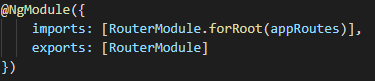


Fig. 3.2 Codul prin care se inițializează rutele pentru întreg programul

După ce am creat rutele aplicație, acestea trebuie să fie făcute cunoscute întregii aplicații și acest lucru se poate realiza prin adăugarea următoarei porțiuni de cod, din figura 3.2, în serviciul de *rutare* creat.

Pentru ca aceste rute să poată fi accesate prin intermediul unor butoane avem două directive pe care le atribuim unui buton pentru ca prin apăsarea acestuia, utilizatorul să fie dus la pagina dorită (fig. 3.3b):

* ***RouterLink –*** ne permite să specicăm conexiunea către o anumită rută din cele definite în aplicație
* ***RouterLinkActive –*** ne permite să adăugăm o clasă *CSS* unui element atunci când acesta devine activ

Pe lână toate etapele enumarate mai sus, trebuie să mai facem încă un lucru pentru ca totul să funcționeze așa cum ne dorim și anume trebuie să îi spunem lui *Angular* unde să încarce aceste rute, iar acest lucru îl facem în *app-component*, componenta de bază a aplicației. Ne vom folosi de o altă directivă numită ***router-outlet*** (fig 3.3a) care acționează ca un fel de înlocuitor, marcator al locului unde *Angular* va încărca dinamic conținutul *DOM-ului* în funcție de starea de rutare în care se află utilizatorul la momentul respectiv.

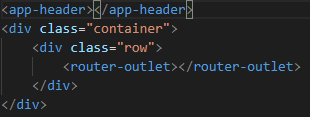


Fig. 3.3 a) Declararea directivei router-outlet

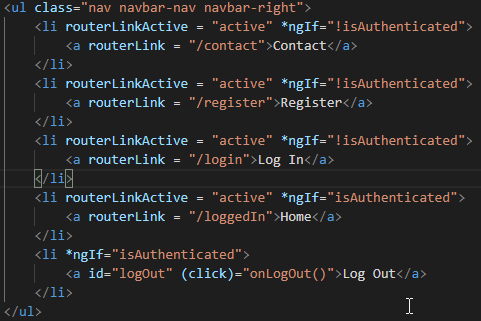


Fig. 3.3 b) routerLink & routerLinkActive

## Serviciile aplicației

*De ce avem nevoie de servicii ?*

*Componentele* unei aplicați *Angular* nu ar trebui să se ocupe în mod direct de salvarea sau aducerea datelor de pe un anumit server și cu siguranță nu ar trebui să afișeze date false. Componentele ar trebui să se ocupe de prezentarea datelor și de crearea unui mediu plăcut prin care utilizatorii să poată acesa datele unui *serviciu*. *O componentă* trebuie să poată delega anumite sarcini *serviciilor*, precum: preluarea datelor de pe server, validarea *input-urilor* utilizatorului sau afișarea directă în consolă. Definind astfel de activități de procesare într-o clasă de servicii *injectabile*, facem aceste sarcini disponibile oricărei componente. *Angular* ne ajută să urmăm aceste principii facilitând modul prin care controlăm logica aplicației în *servicii* și de altfel ne ajută să le punem la dispoziția componentelor prin *dependency injection*.

De asemenea *Angular* face o distincție între *componente* și *servicii* pentru a crește modularitate și reutlizarea acestora. Prin această distingere între *componente* și *servicii* putem crea clase simple și eficinente, foarte ușor de reutilizat.

***Serviciile*** [14] reprezintă o categorie largă ce cuprinde valori, funcții sau caracteristicile de care are nevoie o aplicatie. De obicei un *serviciu* este o clasă ce are un scop restrâns, bine definit, către un lucru specific pe care ar trebui să îl facă cât de bine posibil. *Serviciile* pot depinde de alte *servicii*, adică pot să utilizeze funcțiile altor servicii pentru a putea să ducă la capăt funcțiile proprii.

*Cum creăm un serviciu ?*

După cum spuneam, *serviciile* sunt defapt niște clase cărora le atribuim niște sarcini specifice pe care le putem facem disponibile tuturor *componentelor* care s-ar putea să aibe nevoie de serviciile pe care aceaste clase le oferă. Pentru a răspunde mai concret la acestă întrebare voi prezenta în continuare *serviciile* puse la dispoziție de aplicația dezvoltată, însă înainte de a face acest lucru voi mai specifica câteva aspecte importante.

Front-end-ul aplicațiilor web construite cu ajutorul framework-ului *Angular* comunică cu serverele din back-end prin ***REST APIs***( care în cazul nostru este pus la dispoziție de *Firebase*) ce sunt bazate pe *protocoale HTTP,* prin care trimitem anumite cereri către server. Pentru a putea face astfel de cereri ne vom folosi de modulul *HttpClientModule* pus la dispoziție de *Angular*. Acest modul pun le dispoziție un serviciu numit *HttpClient* care se ocupă de trimiterea cererilor către server. Pentru aplicația noastră vom folosi două tipuri de cereri puse la dispoziție de serviciul *HttpClient* și anume:

* Cerere de tip ***get()*** – prin care vom putea aduce date de pe server.
* Cerere de tip post() - prin care vom putea salva date pe server.

### ‘auth’ service

*‘auth’ service* este serviciul care se ocupă doar de autentificare și înregistrarea studenților, precum și conectarea la aplicație și deconectarea de la aplicație a studenților. Funcțiile,metodele puse la dispoziție de către acest *serviciu* sunt:

* ***signUp****( email: string, pwd: string )*
* ***login****( email: string, pwd: string )*
* ***private handleAuthentication****(*

*email: string,*

*userId: string,*

*token: string,*

*expiresIn: number )*

* ***autoLogin****()*
* ***logout****()*
* ***autoLogout****( expirationDuration: number )*
* ***handleError****(errorRes: HttpErrorResponse)*

1. ***signUp(*** *email: string, pwd: string* ***) method***

Această metodă este responsabilă pentru salvarea și crearea conturilor studenților, fiind apelată din interiorul componentei *register* atunci când studentul a introdus datele corete și a apăsat butonul de înregistrare. În figura 3.4 putem observa codul funcției și pe baza acestei figuri vom mai face câteva precizări. Orice cerere de tipul *post* are nevoie de două informații foarte importante și anume : adresa *API-ului*  și tipul de informație pe care îl trimitem. După cum am specificat ne vom folosi de platforma *Firebase* și serviciile puse la dispoziție, cel de autentificare și cel de baze de date/server. Așadar pentru a acesa acest serviciu trebuie să facem o cerere prin care creăm un cont de tipul *email/password* și să îl salvăm.

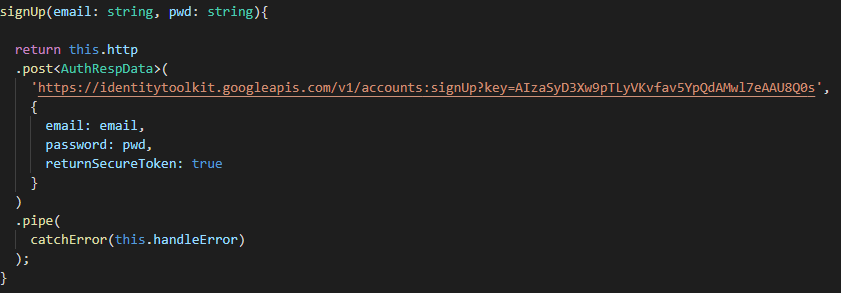


Fig. 3.4 **signUp()** method

În funcția noastră am folosit adresa pentru *API-ul* pus la dispoziție de *Firebase* cât și tipul/formatul informației transmise prin aceasă cerere. Am legat la această cerere prin metoda *pipe()* funcția *handleError* care în cazul unei errori survenite va face cunoscută această eroare studentului.

Această cerere va returna anumite informații sub o anumită structură, iar pentru aceasta am creat inerfața *AuthRespData* care are strucutra informației returnate de cererea *post()* și va reține aceste informații pentru ca apoi să poată fi utilizate dacă este cazul.

Această metodă este utilizată după cum am specificat și mai sus de către componenta *register* unde pentru ca această cerere să fi trimisă, pe lângă apelarea acestei metode, mai trebuie să folosim metoda *.subscribe()* care se va ocupa de trimiterea acestei cereri. În interiorul metodei *.subscribe()* putem face 3 lucruri: putem manipula datele returnate în urma cererii, putem prinde eventualele errori care apar și putem executa o funcție sau un cod în cazul în care cererea s-a finalizat cu succes. În figura 3.5 puteți observa utilizarea acestei metode .*subscribe()* în interiorul componentei *register.*

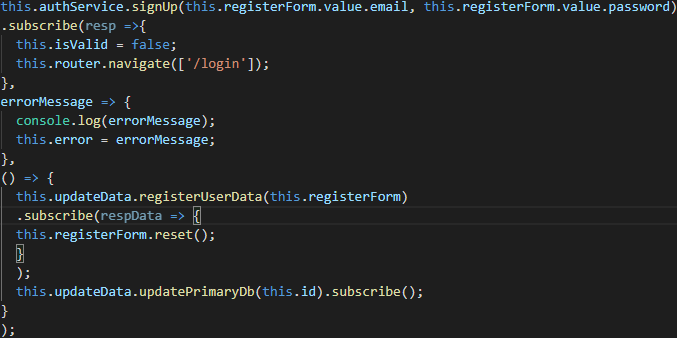


Fig. 3.5 **subscribe()** method

1. ***login(*** *email: string, pwd: string* ***) method***

Acestă metodă se ocupă de conectare stundeților la aplicația propriu-zisă prin *email și parolă* și după cum deja ați bănuit este apelată in interiorul componentei *login,* unde , de asemenea, am folosit metoda *.subscribe* despre care am vorbit în cadrul primei metode descrise, metoda *signUp(1).* În această metodă facem tot o cerere de tip *post* însă spre un alt tip de *API* pus la dispoziție de serviciul de autentificare *Firebase.* În urma acestei cereri vom primit o informație de tipul *AuthRespData* pe care o vom prelucra în metoda *pipe()* prin *operatorul tap()* transmițând mai departe funcției *handleAuthentication()* care va prelucra mai departe datele primite (fig. 3.6).

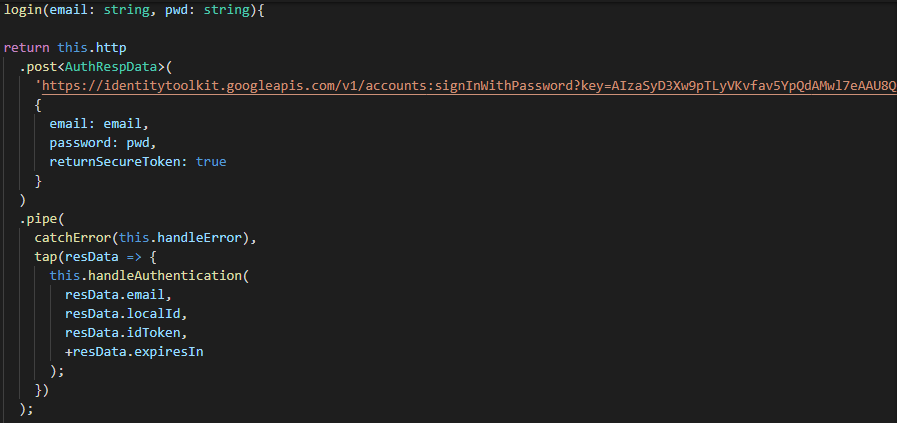


Fig. 3.6 **login()** method

1. ***handleAuthentication() method***

Această metodă este o metodă privată prin care prelucrăm datele primite în urma conectării studentului la aplicație. Printre aceste date preluate avem două date care conțin două informații importante, și anume: *token – care conține un șir de caractere alfanumerice care reprezintă un id unic de identificare a studentului conectat. Acest id se oferă în mod unic, automat și aleatorde fiecare dată, fiecărui student care se conectează la aplicație; expiresIn – care conține o valoare numerică (3600 de secunde) ce reprezintă timpul în care* ***token-ul*** *este valid.*

Prelucrăm toate datele primite pe care le salvăm într-un obiect de tip *User,* pe care l-am definit intr-o altă clasă, apelăm metod *autoLogout(),* iar apoi salvăm în memoria locală a browser-ului aceste date care apoi vor fi preluate de către metoda *autoLogin(),* unde vom vedea cu ce scop vor fi utilizate acesta date. Codul metodei prin care s-a realizat întreg procesul descris mai sus se poate vedea în figura 3.7.

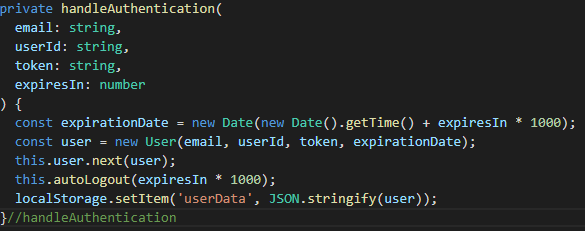


FIg. 3.7 **handleAuthentication()** method

1. ***autoLogin() method***

Această metodă (fig. 3.8) este responsabilă pentru menținerea conectării studentului pe toată durata de timp în care *toke-ul* este valid. În cazul în care aplicația este reîncărcată în browser sau pagina aplicației este închisă și apoi reaccesată pe durata de timp validă a *token-ului*, această metodă se ocupă de conectarea automată a studentului. Acest lucru se realizează prin intermediul datelor salvate în memoria locală a browser-ului în momentul în care studentul s-a conectat pentru a utiliza aplicația.

Medota aceasta verifică dacă avem date salvate în memoria locală privitoare la un student care s-a conectat. Dacă există astfel de date în memorie, se salvează într-o constantă locală aceste date din memorie, pe care le transmitem mai departe unei alte constate locale de tip *User*. Verificăm existența unui *token*, iar apoi transmitem mai departe perioada de timp rămasă în care *token-ul* este valid și o transmitem mai departe metodei *autoLogout().* Metoda *autoLogin()* este apelată de fiecare dată când apicația tranzitează dintr-o stare în altă stare.

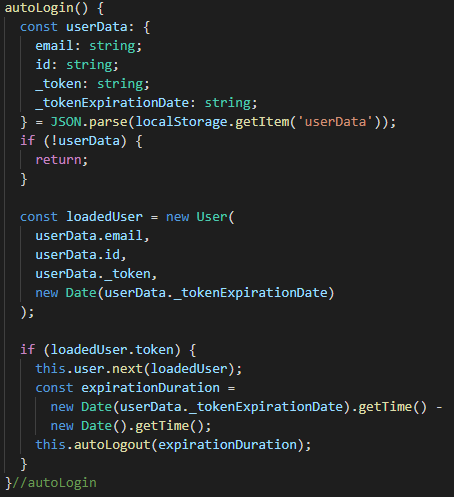


Fig. 3.8 **autoLogin()** method

1. ***logout****()* ***method***

Prin intermediul acestei metode se realizează deconectarea studentului de la aplicație, eliminarea datelor salvate în memoria locală a browser-ului, resetarea timer-ului și redirecționarea studentului către pagina de start a aplicației. În figura alăturată puteți vedea codul sursă al acestei metode (fig. 3.9).

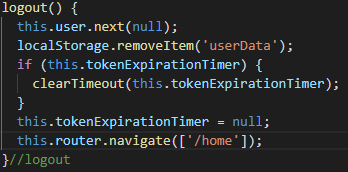


Fig. 3.9 **logout()** method

1. ***autoLogout(*** *expirationDuration: number* ***) method***

*autoLogout* (fig. 4)este o metodă care în momentul în care este apelată pornește un timer care după un timp specificat va executa metoda *logout().* Timpul după care va fi executată metoda este defapt perioad de validitate a *token-ului* furnizat în momentul conectării de către *expiresIn* și transmisă ca parametru acestei funcții.

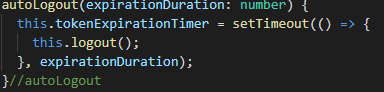


Fig. 4 **autoLogout()** mehtod

1. ***handleError****(errorRes: HttpErrorResponse)* ***method***

Reprezintă *handler-ul* serviciului *auth* care se ocupă de gestionarea erorilor și transmiterea mai departe a mesajelor de eroare corespunzătoare pentru a informa studentul sau dezvoltatorul de anumite probleme ce au survenit în timpul comunicării cu serverul *Firebase* (fig. 4.1)*.*

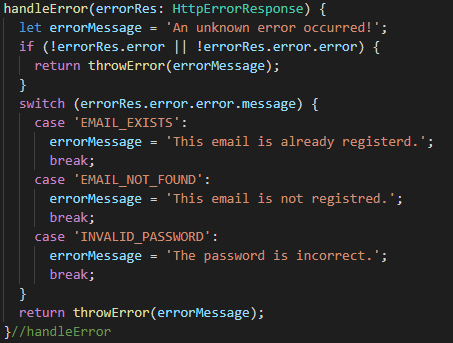


Fig. 4.1 **handleError()** method

### ‘get-update’ service

*‘get-update’ service* este un serviciu care se ocupă în mod special cu aducerea datelor din baza de date și actualizarea datelor din baza de date a serverului nostru pus la dispoziție de *Firebase.* Funcțiile puse la dispoziție de cătrea acest *serviciu* sunt:

* ***getStudents****()*
* ***updateStudentData****( key:string, data:FormGroup )*
* ***registerUserData****( registerForm : NgForm )*
* ***updatePrimaryDb****( key:string )*
* ***getRegistredUsers****()*
* ***registerProblem****(contactForm: FormGroup)*

În cazul celui de-al doilea serviciu vom lucra cu 3 baze de date care vor conține diferite date. Aceste baze de date sunt (fig. 4.2):

* *‘primaryDb’* – este baza de date principală, de regulă baza de date a facultății sau universității, ce conține toți studenții care au fost admiși si sunt distribuiți într-un cămin. Fiecare student va fi descris prin intermediul a 8 câmpuri cu nume specific tipului de informație/date pe care îl va deține : *firstName, lastName, CNP, age, phoneNumber, usNumber(numărul matricol), studentDorm(căminul în care a fost repartizat), registred ( câmp cu valoare booleană care va indica dacă studentul respectiv și-a creat cont sau nu ).*
* *‘problemsDb’ –* bază de date în care salvăm toate problemele trimise de către studenți prin intermediul fomularului de contact din cele două componente. Fiecare problemă înregistrată va fi caracterizată de 8 câmpuri cu nume sugestive care vor indica ce informație/date dețin acestea: *fistName, lastName, email, unicNumber ( numărul matricol ), university, problemType ( tipul problemei întâmpinate; va fi ales dintr-o listă predefinită de tipuri de probleme ), agreed ( câmp cu valoara booleană ce indică faptul că studentul care a completat formularul este de acord cu termenii și politica despre prelucrare datelor cu caracter personal ), problem ( câmp ce va conține problema descrisă de către student ).*
* *‘registredUsers’ –* bază de date în care salvăm conturile create cu succes de către studenți pentru aș-i putea recupera parola sau email-ul contului. Fiecare cont creat cu succes și salvat va vi descris de 5 câmpuri cu nume sugestive care vor indica tipul de informație/date pe care îl deține: *email, password, confirmedPassword, unNumber ( numărul matricol ), checked ( câmp cu valoare câmp cu valoara booleană ce indică faptul că studentul care a completat formularul este de acord cu termenii și condițiile impuse de universitate/faculate ).*



Fig. 4.2 Bazele de date **Firebase** utilizate și structura acestora

În figura 4.2 de mai sus se poate observa structura bazelor de date utilizate pe care le-am prezentat mai devreme. Se poate observa că fiecare înregistrare a fiecărei baze de date are asociat un șir de caractere alfa-numerice. Aceste șiruri de caractere alfa-numerice sunt defapt niște id-uri unice generate în mod automat de *Firebase*. Ne vom folosi și noi de aceste id-uri unice pentru a efectua anumite operații asupra înregistrărilor dorite. În continuare vom descrie rolul funcțiilor/metodelor din acest serviciu și unde sunt ele folosite.

1. ***getStudents****()* ***method***

Metoda *getStudents()* ne pune la dispoziție posibilitate de a accesa studenții din baza de date *primaryDb.* Acest lucru se realizează printr-o cerere *HTTP* de tipul *get()* în care trebuie să indicăm locul (adresa *URL,* în cazul nostru) unde se află baza de date. În figura 4.3 putem observa faptul că în adresa *URL* există la final extensia ***.json.*** Această extensie este prezentă deoarece baza de date este salvată sub acest format care este înțeles și de către *Angular*. Apoi se observă că am folosit și metoda *pipe()* ce pune la dispoziție operatorul *map()* prin care putem modifica datele pe care le primim în urma cereri. În interiorul acestui operator *map()* am transformat datele primite de la server adăugându-le într-un array normal care poate fi accesat print-un simplu index. În plus față de câmpurile din baza de date am mai adăugat încă un câmp ce conține acel id alfa-numeric unuic. Metoda este utilizată în componenta *account-info* pentru a afișa datele despre studentul care este conectat și în componenta *register* pentru a verifica dacă numărul matricol furnizat de stundet la înregistrare se află în baza de date principală, adică dacă utilizatorul este defapt un student al universității/facultății.

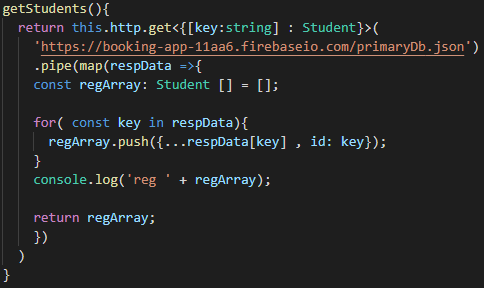


Fig. 4.3 **getStudents()** method

1. ***updateStudentData****( key:string, data:FormGroup )*

Este o metodă utilizată în componenta *account-info* ce este apelată atunci când studentul modifică datele care sunt afișate și le salvează (fig 4.4). Pe baza chei *key,* transmise ca parametru funcției, care conține id-ul unic de identificare a studentului în baza de date *primaryDb* se face actualizarea tuturor datelor prin parametrul *data* care este defapt un formular pentru că modificarea făcută a fost efectuată într-un formular. Actualizarea se realizează printr-o cerere *HTTP* de tipul *patch()* care actualizează doar câmpurile specificate, celelalte rămânând nemodificate. Cererea de tip *patch()* este la fel ca și cea de tip *post()*, fiind necesar să furnizăm calea către înregistrarea pe care dorim să o actualizăm și un obiect care conține câmpurile impreună cu datele noi care sunt de forma *key/value.*

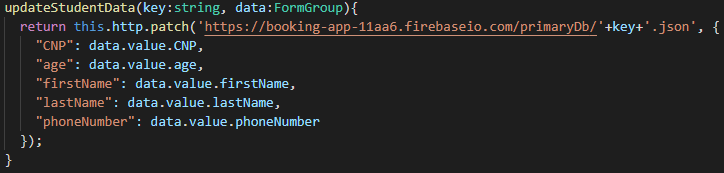


Fig 4.4**updateStudentData()** method

1. ***registerUserData****( registerForm : NgForm )* ***method***

Prin intermediul acestei metode (fig. 4.5) se salvează în baza de date *registredUsers* toate înregistrările efectuate cu succes, fiind utilizată de componenta *register*. Salvarea se realizează tot printr-o cerere *HTTP* de tipul *post()* în care frunizăm calea către baza de date și datele care dorim să le salvăm, acestea fiind salvate într-un formular.

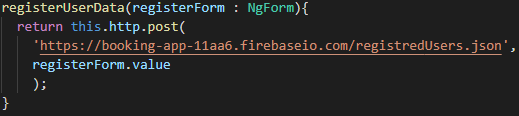


FIg. 4.5 **registerUserData()** method

1. ***updatePrimaryDb****( key:string )* ***method***

Cu ajutorul acestei metode actualizăm câmpul *registred* din baza de date *primaryDb* atunci când un student s-a întregistrat cu succes fiind apelată din componenta *register*. Actualizarea se face la fel ca și la metoda *updateStudentData()* printr-o cerere *HTTP* de tipul *patch()* care actualizează doar câmpurile specificate, celelalte rămânând nemodificate, adică actualizăm doar câmpul *registred* de la valoarea inițială *false* la valoarea *true.*

1. ***getRegistredUsers****()* ***method***

Este o metodă prin care accesăm baza de date *registredUsers* în care avem toate conturile înregistrate împreună cu celelalte informații privitoare la cont. Accesul se face la fel ca și în metoda *getStudents()* printr-o cerere *HTTP* de tipul *get()* după care modificăm informațiile returnate salvându-le intr-un array pentru a putea manipula datele mult mai ușor.

1. ***registerProblem****( contactForm: FormGroup )* ***method***

Această metodă efectuază salvarea tuturor formularelor de contact care au fost completate cu scopul de a raporta o problemă întâmpinată și au fost trimise cu succes, tot printr-o cerere *HTTP* de tipul *post()* în care am indicat adresa către baza de date *problemsDb* și am transmis datele completate în formular. Metoda este apelată în ambele componente de contact, *componenta contact* și *componenta home-contact.*

Pentru ca toate acest metode prezentate ce sunt furnizate de către serviciul *get-update* să fie apelate cu succes este nevoie să ne ”abonăm” la ele, adică sa folosim metoda  *.subscribe()* în toate componentele în care utilizăm una dintre acest metode ale serviciului

# Îmbunătățiri

Îmbunătățirile ce s-ar putea aduce acestei aplicații sunt diverse aducând un beneficiu și mai mare, în primul rând studenților, iar apoi administratorilor de cămine și de altfel și universității/facultății. O parte din aceste îmbunuătățiri s-au dorit a fi implementate până în moemetul de față, însă, din păcate, acest lucru nu s-a putut realiza. Am să prezint pe rând aceste îmbunătățiri ce pot fi aduse, voi da câteva detalii și beneficii pe care le aduce prin implementarea lor.

1. *Implementarea serviciului de plată online* – prin implementarea acestui serviciu se dorește oferirea posibilității de a efectua plata aferentă fiecărei luni a sumei stabilite la încheierea contractului, online prin card bancar. Acest serviciu aduce din nou un beneficiu pentru studenți deoarece le oferă posibilitate de a se ocupa mult mai ușor și mult mai rapid de această responsabilitate, evitarea cozilor și îi scutește de faptul că au un timp limitat în care pot efectua plata căminului. De asemenea, acest serviciu nu vine doar în beneficiul studenților ci și în beneficiul administratorilor cărora le va ușura munca și le va oferi mai mult timp pe care să îl poată investi în alte aspecte ale coordonării și îngrijirii căminulor de care sunt responsabili.
2. *Implementarea serviciului de gestionare a chitanțelor –* necesitatea acestui serviciu este impusă de cel de la punctul 1, deoarece în urma orcărei efectuări a undei plăți este nevoie și de o dovadă că acea plată s-a efectuat cu succes. Acest serviciu își dorește stocarea în format electronic a tuturor chitanțelor ce se vor genera în mod automat atunci când plata s-a efectuat cu succes. Acest lucru se va efectua pentru fiecare student în parte și va permite accesul la aceste chitanțe pentru a putea fi vizualizate, verificate și utilizate în caz de nevoie în orice moment de timp. Beneficiul pe care acest serviciu îl aduce este mai mult unul ecologic, doarece elimină în mare parte utilizarea hârtiei pentru tipărirea acestor chitanțe, tușuri pentru imprimante. Însă acesta nu este singurul beneficiu, mai oferă și accesul la aceste chitanțe oriune și oricând se dorește sau este nevoie.
3. *Implementarea serviciului de generare automată a contractului de închiriere –* acest serviciu are scopul de a genera în mod automat contractul de închiriere în format electronic, completat cu toate datele necesare, care au fost verificate de către student. Acest serviciu va pune la dispoziția studentului contractul în format electronc, completat cu datele sale și îi va oferi posibilitate de a citi contractul respectiv și prin exprimarea acordului său se va salva acest contrat generat în contul personal al aplicației. La fel ca și beneficiul adus de serviciul de la punctul 2, ne ajută să folosim mai puțină hârtie ajutând ecosistemul.
4. *Implementarea unui serviciu de tip* ***reminder*** *–* prin acest serviciu dorim în primul rând să înștiințăm fiecare stundet ( printr-un email, SMS ) atunci când data scadentă pentru efectuara plătii se apropie (cu 3-4 zile înainte) iar acesta nu a efectuat încă plata. Se dorește ca prin implementarea acestui serviciu să ajutăm studenții să nu devină restanțieri ( și aici ), iar pentru cei care au depășit data scadentă și nu au efecutat plata să fie înștiințați zilnic în următoarele 3-4 zile prin email sau SMS.

# Concluzie

În finalul acestei lucrări de diplomă aș dori să formulez o scurtă concluzie asupar acestei aplicații și sper că până aici nu v-am plictisit și v-am oferit informații suficiente și utile despre această aplicație.

Acum, să concluzionez această lucrare. Lucrarea de față și aplicația prezentată își dorește eficientizarea procesului de rezervare a locurilor și cazare a stundeților în căminele studențești prin metode tehnologie avansate.

# Bibliografie

***INTRODUCERE***

[1] https://webfoundation.org/about/vision/history-of-the-web/

[2] https://home.cern/about/who-we-are/our-history

[3] https://www.w3.org/TR/webarch/

[4] https://www.guru99.com/difference-web-application-website.html

***THNOLOGII FOLOSITE***

[1] https://www.hostinger.com/tutorials/what-is-html

[2] https://www.yourhtmlsource.com/starthere/whatishtml.html

[3] https://developer.mozilla.org/en-US/docs/Web/HTML

[4] https://ro.wikibooks.org/wiki/Cascading\_Style\_Sheets\_(CSS)

[5] https://developer.mozilla.org/ro/docs/Learn/Getting\_started\_with\_the\_web/CSS\_basics

[6] https://getbootstrap.com

[7] https://developer.mozilla.org/en-US/docs/Web/JavaScript/About\_JavaScript

[8] https://developer.mozilla.org/ro/docs/Web/JavaScript

[9] https://dev.to/singhdigamber/what-is-typescript-jig

[10] https://www.typescriptlang.org/index.html

[11] https://angular.io/guide/architecture

[12]

https://developer.mozilla.org/en-US/docs/Web/API/Document\_Object\_Model/Introduction

[13] https://medium.com/firebase-developers/what-is-firebase-the-complete-story-abridged-bcc730c5f2c0

[14] https://angular.io/guide/architecture-services

# Lista figurilor

[Fig 1] https://mdn.mozillademos.org/files/13502/cake.png

[Fig 1.1] Setarea titlului unei pagini *HTML*; structura unui document *HTML*

[Fig 1.2] Anatomia unei reguli *CSS*

[Fig 1.3] https://angular.io/generated/images/guide/architecture/overview2.png

[Fig 1.4] https://miro.medium.com/max/839/0\*DylbZPWyXT7S0Fn5

[Fig 1.5] https://miro.medium.com/max/1600/0\*HORJhBhTELtW9qQw

[Fig 1.6] Structura unei **componente**

[Fig 1.7] Lista **componentelor**

[Fig 1.8] Decoratorul **@NgModel**

[Fig 1.9] Import-urile din modulul **app.module.ts**

[Fig 2] Utilizarea decoratorului **@Injectable** într-un serviciu

[Fig 2.1] Codul *HTML* pentru bara de navigație

[Fig 2.2] Bara de navigație din componenta **header**

[Fig 2.3] Formularul de contact

[Fig 2.4] Crearea formularului în cod *TypeScript*

[Fig 2.5] Parte din codul *HTML* prin care s-a creat formularul de contact

[Fig 2.6] Formularul de înregistrare

[Fig 2.7] Secvență din crearea formularului de înregistrare

[Fig 2.8] Panoul de *Login* al aplicației

[Fig 2.9] **home-log-in** component

[Fig 3] **Not-found** component

[Fig 3.1] Rutele aplicației

[Fig 3.2] Codul prin care se inițializează rutele pentru întreg programul

[Fig 3.3a] *routerLink* & *routerLinkActive*

[Fig 3.3b] Declararea directivei *router-outlet*

[Fig 3.4] **signUp()** method

[Fig 3.5] **subscribe()** method

[Fig 3.6] **login()** method

[Fig 3.7] **handleAuthentication()** method

[Fig 3.8] **autoLogin()** method

[Fig 3.9] **logout()** method

[Fig 4] **autoLogout()** mehtod

[Fig 4.1] **handleError()** method

[Fig 4.2] Bazele de date **Firebase** utilizate și structura acestora

[Fig 4.3] **getStudents()** method

[Fig 4.4] **updateStudentData()** method

[Fig 4.5] **registerUserData()** method