UNIVERSITATEA “ALEXANDRU IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENȚĂ

Smart buildings

propusă de

Oana-Roxana Amariei

Sesiunea: *iulie, 2019*

Coordonator științific

Prof. Colab. Florin Olariu

UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IAȘI

FACULTATEA DE INFORMATICĂ

Smart buildings

Oana-Roxana Amariei

Sesiunea: *iulie, 2019*

Coordonator științific

Prof. Colab. Florin Olariu

Avizat,

Îndrumător Lucrare de Licenţă

Titlul, Numele şi prenumele _____

Data _____ Semnătura _____

DECLARAȚIE privind originalitatea conţinutului lucrării de licenţă

Subsemnatul(a) ……

domiciliul în ……..

născut(ă) la data de ……., identificat prin CNP ……..……., absolvent(a) al(a) Universităţii „Alexandru Ioan Cuza" din Iaşi, Facultatea de …….. specializarea ……., promoţia ……., declar pe propria răspundere, cunoscând consecinţele falsului în declaraţii în sensul art. 326 din Noul Cod Penal şi dispoziţiile Legii Educaţiei Naţionale nr. 1/2011 art.143 al. 4 si 5 referitoare la plagiat, că lucrarea de licenţă cu titlul: _____

_____

_____elaborată sub îndrumarea dl. / d-na _____, pe care urmează să o susţină în faţa comisiei este originală, îmi aparţine şi îmi asum conţinutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licenţă să fie verificată prin orice modalitate legală pentru confirmarea originalităţii, consimţind inclusiv la introducerea conţinutului său într-o bază de date în acest scop.

Am luat la cunoştinţă despre faptul că este interzisă comercializarea de lucrări ştiinţifice in vederea facilitării falsificării de către cumpărător a calităţii de autor al unei lucrări de licenţă, de diploma sau de disertaţie şi în acest sens, declar pe proprie răspundere că lucrarea de faţă nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

DECLARAȚIE DE CONSIMȚĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul „*Smart buildings*”, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Iași, *data*

Absolvent *Oana-Roxana Amariei*

_____

(semnătura în original)

Cuprins

Introducere

Încă de la începuturile ei, tehnologia a fost utilizată pentru a ușura viața omului. Odată cu explozia aceasta a tehnologiei, au apărut o multitudine de aplicații inteligente a căror scop este facilitarea vieții de zi cu zi a oamenilor, dar și reducerea costurilor inutile. Aceasta este și motivația pentru care am ales să construiesc o aplicație care se ocupă cu managementul inteligent al unui stoc într-o firmă. Consider că tehnologiile existente pot contribui la reducerea cheltuielilor și la anticiparea viitorului apropiat pe baza învățării din trecut, înlocuind o parte din munca manuală umană.

Motivația

Am investigat puțin piața și am găsit o mulțime de aplicații care îmbunătățesc modul de trai în interiorul unei clădiri, fie ea locuința personală sau locul de muncă, însă niciuna nu se referă la consumul regulat de produse din interiorul clădirii. Aplicațiile noi apărute oferă funcționalități precum pornirea și închiderea automată a luminilor sau de la distanță prin intermediul telefonului, găsirea unui loc de parcare, indicații către sala în care ai meeting și totodată și cel mai scurt drum până acolo, comandarea din timp a mâncării la cantină, aplicația din telefon făcând legătura cu cardul bancar, și multe alte facilități.

Cum toate acestea există deja, m-am gândit să realizez o aplicație care efectuează managementul inteligent al stocurilor de produse dintr-o firmă. Ni se întâmplă și nouă în casele noastre ca uneori să nu avem un anumit produs atunci când avem nevoie de el, iar altădată să avem prea mult, chiar să se și strice dacă este vorba de un produs perisabil. Însă tehnologiile actuale ne-ar putea oferi șansa de a cunoaște momentul în care este nevoie de o reactualizare a stocului și ar fi păcat să nu profităm de aceste oportunități. Dacă astfel de situații se întâmplă și într-o singură clădire, această aplicație ar putea fi foarte utilă pentru firmele care dețin zeci sau chiar sute de clădiri unde achiziționarea produselor implică o sumă foarte mare de bani.

Gradul de noutate

Smart Buildings este o aplicație care permite unei firme să-și adauge clădirile în sistem și să introducă în mod constant consumurile pe care acestea le înregistrează pentru fiecare produs și etaj. Pe baza acestor consumuri se va aplica un algoritm de învățare automată pentru a efectua preziceri ale consumurilor viitoare. Pentru a realiza acest lucru aplicația însumează câteva dintre cele mai utilizate tehnologii la momentul actual: Spring Framework, Angular, ML.NET, folosindu-se de o

bază de date nerelațională, MongoDB, pentru a accesa mai rapid datele, acestea fiind într-un număr foarte mare pentru a obține preziceri cât mai corecte.

Contribuții

Pentru început am realizat partea de front-end în Angular pentru a avea o imagine clară a interacțiunii pe care utilizatorul o va avea cu aplicația și modelele din Java care ajută la manipularea datelor și la trimiterea lor de la un endpoint la altul. I-am oferit utilizatorului posibilitatea de a vedea toate clădirile, de a adăuga una nouă, de a accesa informațiile despre o clădire și consumurile raportate ei, precum și șansa de a adăuga noi consumuri. Am făcut cercetări despre algoritmi de învățare și despre framework-uri care ar putea ușura procesul de antrenare și creare a modelelor, având totodată și rezultate satisfăcătoare, alegând pentru această aplicație ML.NET, un framework care se pliază perfect pe nevoile enumerate anterior. Acesta va antrena un model pe datele din MongoDB exportate într-un fișier de tip .csv, realizând preziceri pentru viitoarele consumuri. ML.NET va transmite aceste rezultate către Spring, acesta din urmă trimițându-le în Angular pentru a fi vizibile utilizatorului.

I. Descrierea problemei

Zi de zi interacționăm cu obiecte pentru a ne desfășura activitățile, fie că le producem sau le consumăm. În locurile în care mulți oameni își desfășoară activitățile consumul este, evident, mult mai ridicat și este nevoie de o grijă mai mare pentru a aproviziona și menține un stoc.

Scenariul luat în considerare este acela al unei firme care deține mai multe clădiri și oferă angajaților condițiile necesare pentru a-și prepara ceva cald de băut. În condițiile în care firma deține un număr mare de angajați, iar clădirile au mai multe etaje, monitorizarea stocurilor este foarte dificilă, personalul răspunzător de aceste servicii fiind nevoit să meargă la toate etajele pentru a verifica starea stocului și, mai apoi, în eventualitatea refacerii stocului, trebuie sa aducă din depozit cantitatea corespunzătoare din fiecare produs la fiecare etaj. Scenariile pot fi în realitate mult mai complexe și cu mai multe necunoscute care pot influența consumurile: în perioada concediilor sunt mai puțini angajați, într-un anumit sezon consumurile sunt mai mari, în zilele de sărbătoare poate consumurile de alimente sunt mai mari, șamd. Într-un ansamblu în care incertitudinea este la ordinea zilei, încercarea de a optimiza și anticipa consumurile este destul de provocatoare, dar cu atât mai utilă.

Dacă personalul ar cunoaște exact momentul în care este necesară o actualizare a stocului, dar și locul și produsul, s-ar economisi mult timp, iar sarcina ar putea fi îndeplinită de un număr mai mic de persoane. Totodată, achiziționarea stocului ar fi mult mai precisă, evitând atât excesul, cât și deficitul.

Evoluția rapidă a tehnologiei din ultimul timp a dus la apariția multor aplicații care facilitează viața omului. O ramură care s-a extins foarte mult este învățarea automată, folosită în multe domenii la momentul actual. Cel mai cunoscut exemplu pentru care s-a folosit învățarea automată este prezicerea prețurilor. Algoritmii necesită numai date de intrare pe care să antreneze un model, pentru ca, mai apoi, să realizeze preziceri. Cum există o evidență a stocurilor achiziționate anterior, s-ar putea folosi un astfel de algoritm și pentru a prezice cantitatea care urmează a fi consumată.

II. Abordări anterioare

Cele mai multe aplicații de monitorizare a consumului se referă numai la tipurile de energie dintr-o locuință. Câteva exemple de astfel de aplicații ar fi următoarele:

- Onnergy: permite controlarea termostatului de la distanță, a luminilor și a dispozitivelor electronice

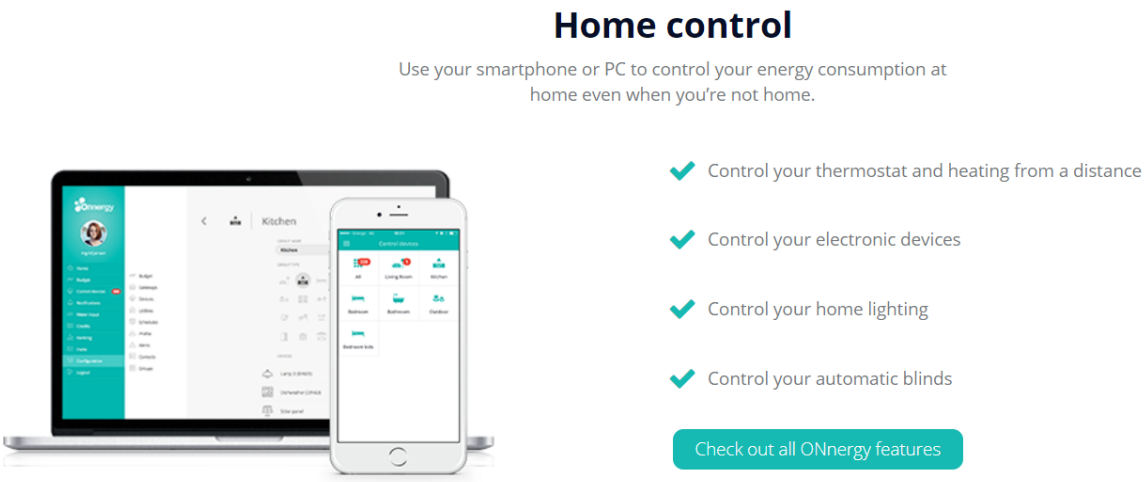

Figura 1 : Aplicația Onnergy

- Energy Consumption Analyzer: permite introducerea consumului de electricitate, apă sau gaz și afișează grafice ale mediilor consumului per oră, zi, săptămână sau lună.

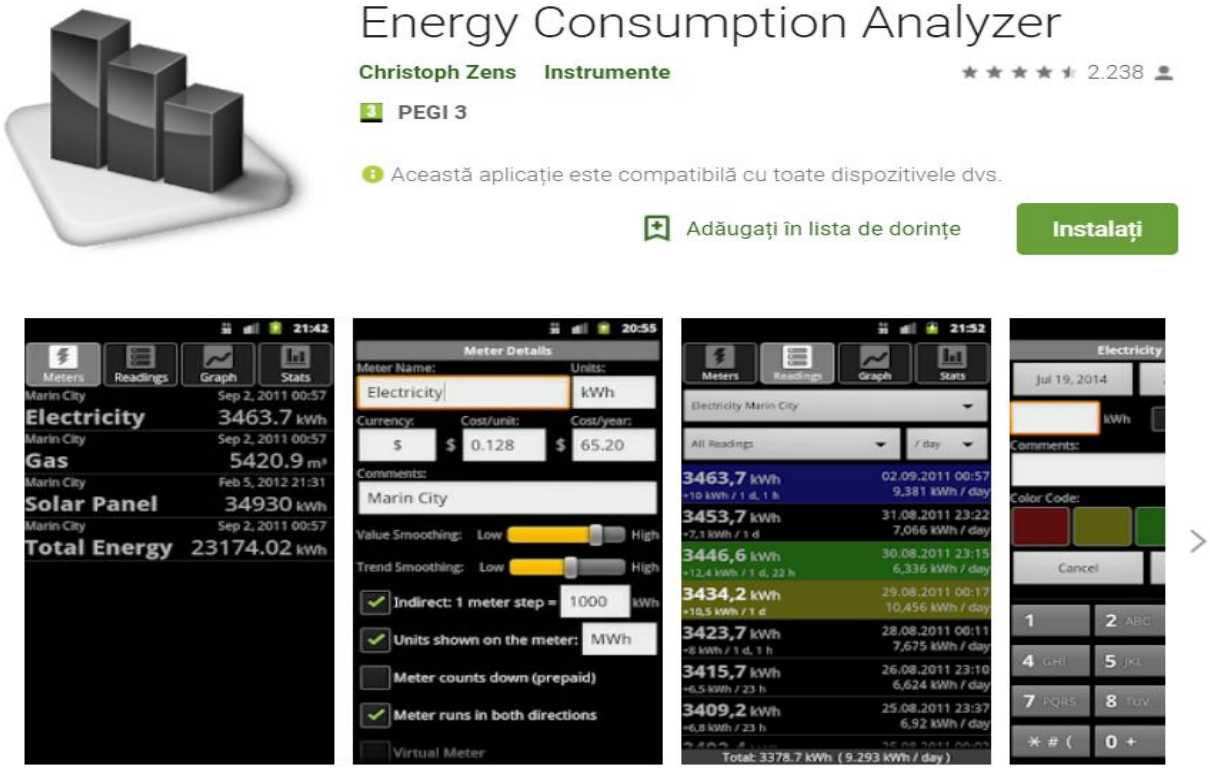

Figura 2 : Previzualizare a aplicației Energy Consumption Analyzer

Mendix este o platformă software low-code foarte rapidă și simplă, folosită pentru dezvoltarea aplicațiilor web și mobile. În ultimul deceniu aplicațiile din domeniul afacerilor au evoluat semnificativ, ele neputând rămâne pasive, așteptând date de intrare de la utilizator. Așa că, împreună cu partenerii lor, cei de la Mendix modelează următoarea generație de aplicații de afaceri, folosind date de dimensiuni foarte mari și algoritmi de învățare. Ei au facilitat managementul produselor unei companii prin amplasarea câtorva mii de senzori în locuri speciale, cum ar fi aparatele de cafea, dozatoarele de săpun sau suporturile de prosoape de hârtie. Aplicația a fost mult mai eficientă decât verificarea manuală efectuată de către personalul răspunzător de aceste servicii, contribuind la diminuarea costurilor și la îmbunătățirea serviciilor. Acum personalul primește notificarea și cunoaște exact locul și produsul care necesită reumplere. Este o aplicație foarte utilă, însă toți acei senzori plasați în puncte cheie implică un cost inițial foarte mare.

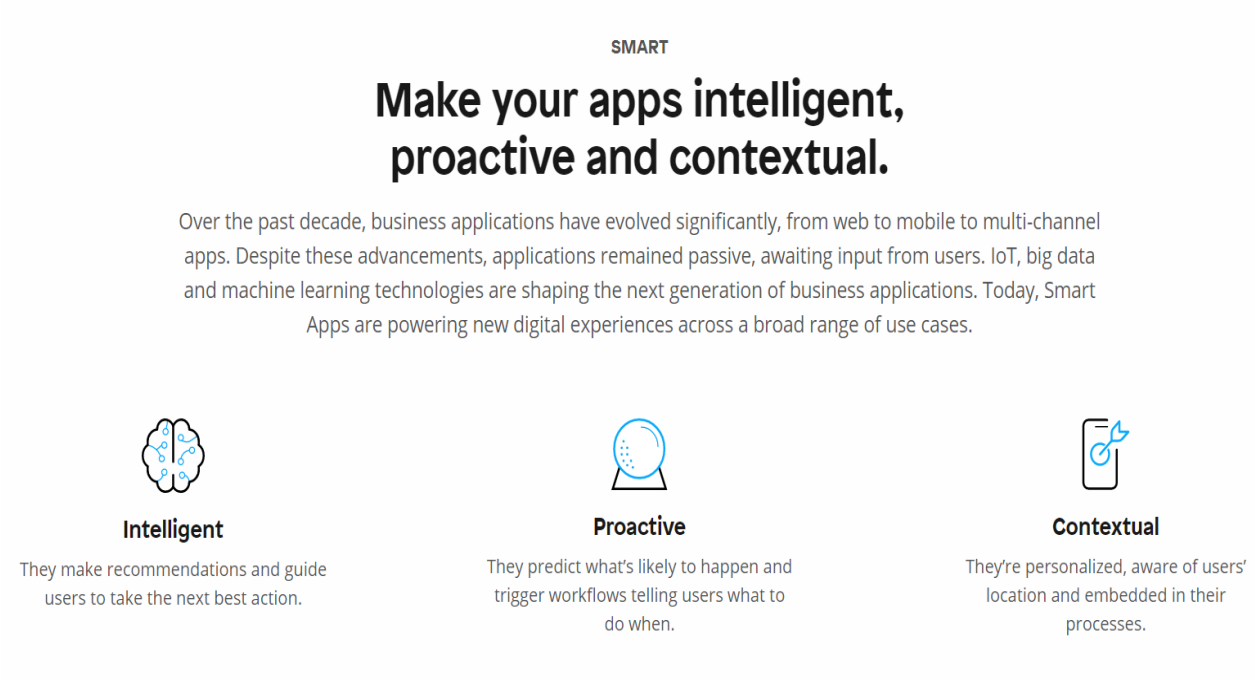

Figura 3 : Funcționalități ale platformei Mendix

Aplicația creată de mine are nevoie doar de câteva date de intrare cu privire la consumul precedent de produse, date a căror evidență oricum este ținută, deoarece stocul trebuie aprovizionat în permanență.

III. Descrierea soluției

III.1. Tehnologii utilizate

III.1.1. Angular

Angular este un framework de tip open-source pentru dezvoltarea aplicațiilor web bazat pe limbajul TypeScript. Principala caracteristică o reprezintă modularitatea, majoritatea funcționalităților fiind împărțite în module. Angular include programarea orientată pe obiecte folosind clase, programare generică, programare lambda, iteratori, bucle For/If, încărcare dinamică.

Arhitectura Angular este împărțită în următorul fel: module, componente, servicii.

Crearea unei componente se realizează prin intermediul comenzii: ”ng generate component [name]”, având următoarea structură:

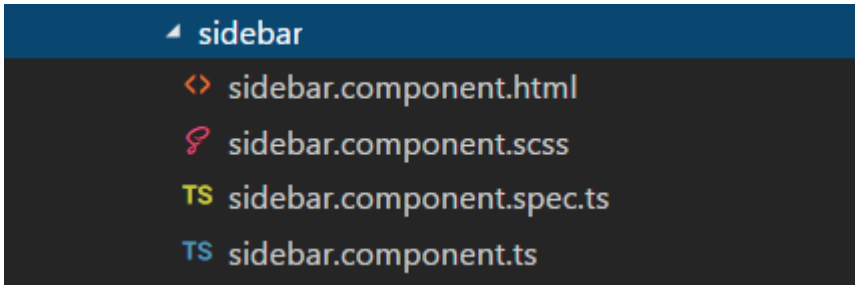

Figura 4 : Structura unei componente în Angular

Pentru a crea un serviciu există comanda: ”ng generate service [name]”, care creează o clasă cu următoarea structură:

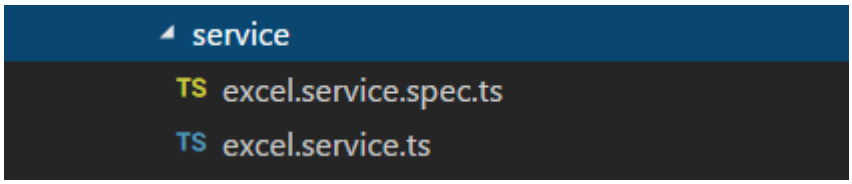

Figura 5 : Structura unui serviciu în Angular

III.1.2. Spring

Spring Framework este o platformă open-source pentru realizarea aplicațiilor în Java. Spring oferă o varietate de funcționalități, împărțite în module: Spring Core Container (modulul de bază, care

furnizează containerele spring) , accesul la date (sisteme de administrare a bazelor de date relaționale) , model-view-controller (un framework bazat pe HTTP care furnizează instrumente pentru aplicațiile web și serviciile REST) , clase suport pentru scrierea unit-testelor și a testelor de integrare.

Cererile HTTP sunt îndeplinite de o clasă cu adnotarea de ”controller” care apelează alte clase, a căror funcționalitate a fost injectată (dependency injection) în clasa controller.

```
@CrossOrigin(origins = "*", allowedHeaders = "*")
@RestController
@RequestMapping()
public class BuildingController {

    @Autowired
    private BuildingRepository buildingRepository;

    public BuildingController(BuildingRepository buildingRepository) { this.buildingRepository = buildingRepository; }

    @GetMapping("buildings/all")
    public List<Building> getAll() throws IOException {
        List<Building> buildings = this.buildingRepository.findByActive(true);

        return buildings;
    }
```

Figura 6 : RestController în Spring

Spring Framework oferă adnotări și pentru maparea obiectelor din baza de date la clasele din Java. În următoarea figură este reprezentată maparea unui document dintr-o bază de date nerelațională:

```
@Document(collection = "Buildings")
public class Building {
    @Id
    private String id;
    private String name;
    private String address;
    private Boolean active;
    private List<Product> products;
```

Figura 7 : Clasă în Java care mapează un document

III.1.3. ML.NET

ML.NET este un framework ce furnizează un mecanism de învățare automată în .NET. Framework-ul oferă suport pentru următoarele probleme: analiza sentimentelor, prezicerea prețurilor, detectarea fraudelor, clasificarea imaginilor, identificarea/detectarea obiectelor, prezicerea vânzărilor. ML.NET permite refolosirea cunoștințelor și librăriilor existente în .NET, facilitând integrarea algoritmilor de învățare automată cu aplicațiile web, mobile sau desktop. De asemenea, framework-ul oferă și instrumente de antrenare automată a modelelor (AutoML) , fiind necesară numai încărcarea datelor de antrenament, după cum ilustrează și următoarele imagini:

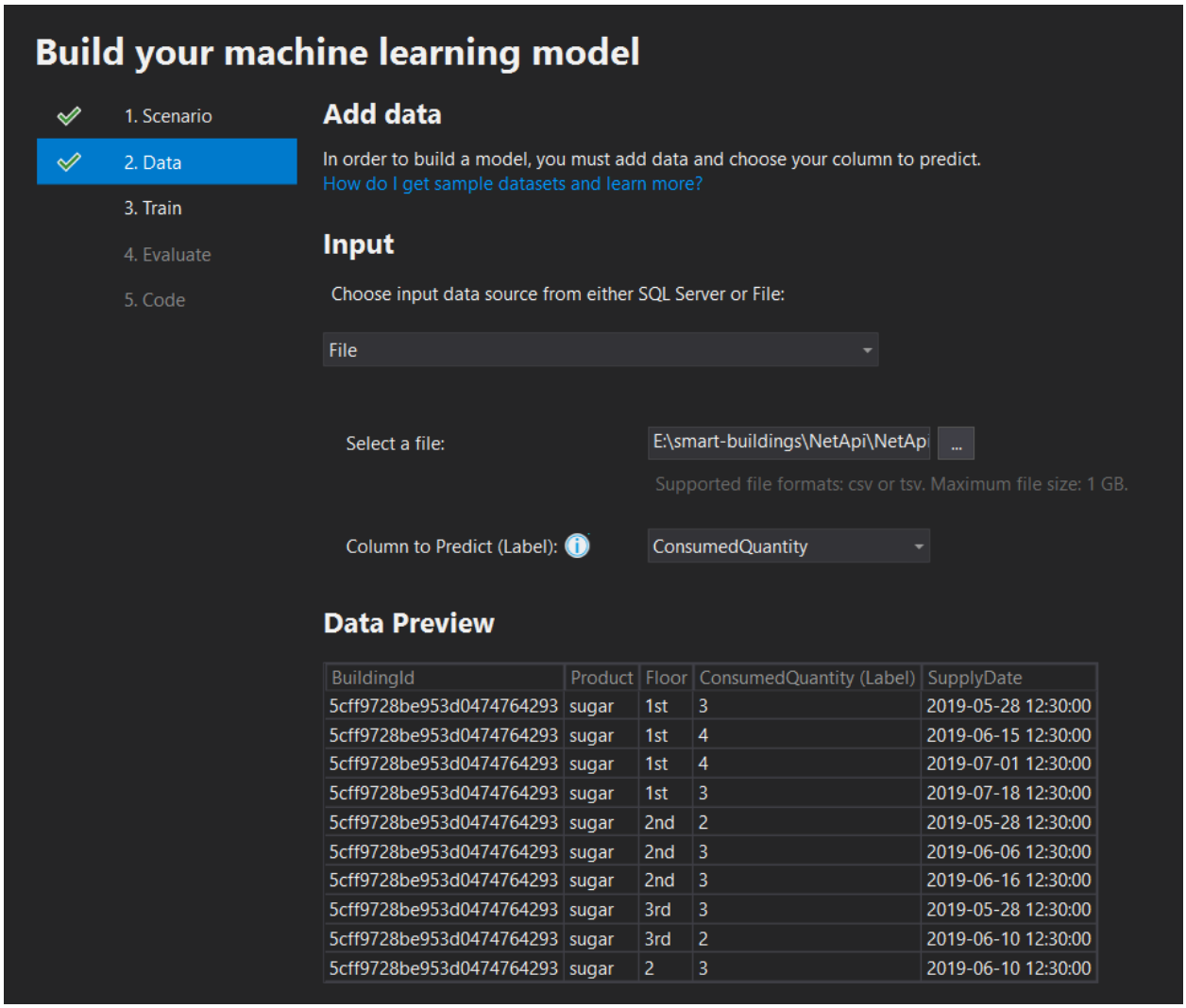

Figura 8 : AutoML: încărcarea datelor de antrenament

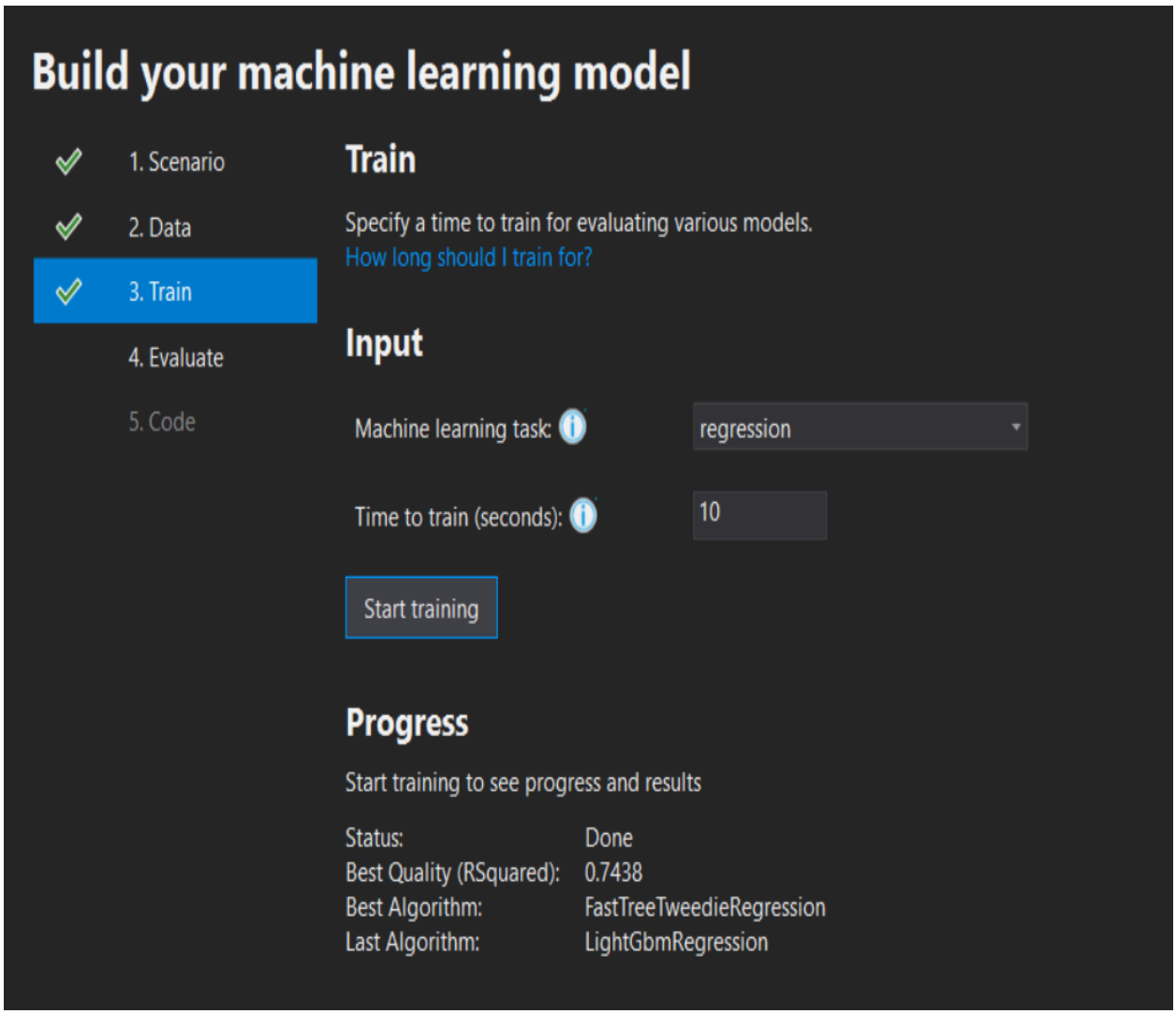

Figura 9 : AutoML: antrenare și detectarea celui mai bun algoritm

III.1.4. MongoDB

MongoDB este o bază de date nerelaționale open-source ce are la bază documente. Stocarea datelor nu se realizează prin tabele, ca într-o bază de date relațională, ci în colecții prin documente (JSON).

Figura 10 : Structura unei clădiri ca document JSON

III.2. Arhitectura aplicației

Tehnologiile folosite și interacțiunea dintre ele pot fi observate în figura de mai jos:

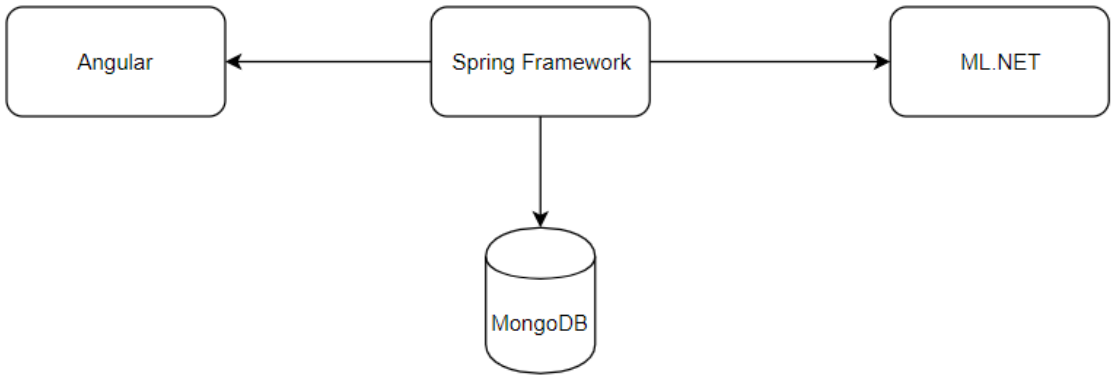

Figura 11 : Tehnologiile utilizate și interacțiunea dintre ele

În centrul aplicației se află frameworkul Spring care interacționează cu Angular pentru a putea oferi vizibilitate utilizatorului și cu ML.NET pentru a accesa funcționalitatea de bază și anume predicțiile, folosindu-se totodată de MongoDB pentru a stoca date.

În figura următoare este prezentată arhitectura aplicației:

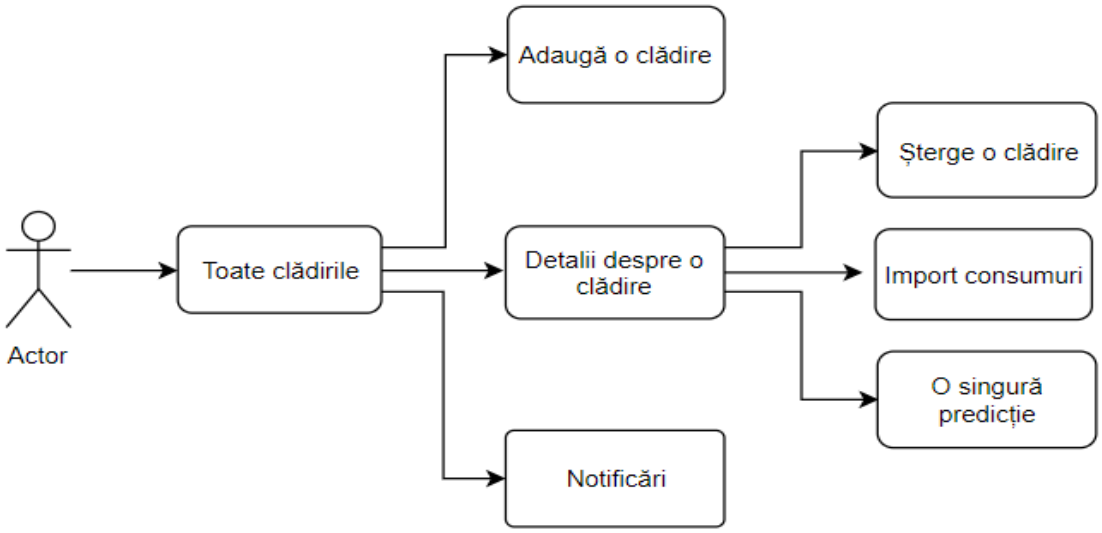

Figura 12 : Arhitectura aplicației Smart Buildings

III.3. Structura aplicației

Atunci când se pornește aplicația, pe pagina principală vor fi afișate toate clădirile din baza de date pentru care se vor face ulterior predicții. Utilizatorul poate naviga în aplicație cu ajutorul unui sidebar poziționat în partea stângă, care îi permite să acceseze formularul pentru adăugarea unei clădiri, să verifice pagina cu notificări, unde vor fi afișate rezultatele algoritmului de învățare, sau să revină la pagina cu toate clădirile.

III.3.1. Adaugă o clădire

Pe această pagină utilizatorul poate adăuga o nouă clădire completând formularul cu numele și adresa noii clădiri. Produsele, etajele și consumurile vor fi adăugate ulterior de pe pagina cu detalii a respectivei clădiri.

III.3.2. Detaliile unei clădiri

Pe această pagină se poate ajunge apăsând pe numele unei clădiri de pe pagina principală. Aici utilizatorul poate vedea numele clădirii, adresa, produsele împreună cu toate etajele la care se găsesc acestea. De asemenea, pe această pagină se află 3 butoane, după cum se poate vedea în figura următoare:

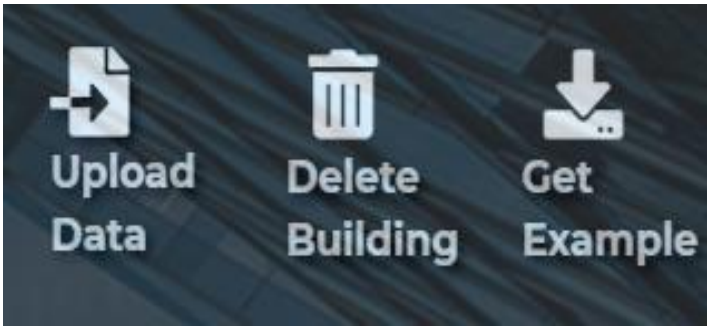

Figura 13 : Butoane pentru interacțiunea utilizatorului cu aplicația

Primul buton îi oferă utilizatorului posibilitatea de a adăuga consumuri pentru clădirea respectivă. Tot ce trebuie să facă utilizatorul este să încarce un fișier de tipul excel din propriul calculator, în spate realizându-se un request către API-ul din Spring, care va parsa fișierul primit ca input și va adăuga noile date documentului din MongoDB corespunzător clădirii respective. În cazul în care documentul are câmpuri goale, utilizatorul va fi anunțat printr-un pop-up ce linii trebuie sa completeze din fișier pentru a-l putea încărca iarăși.

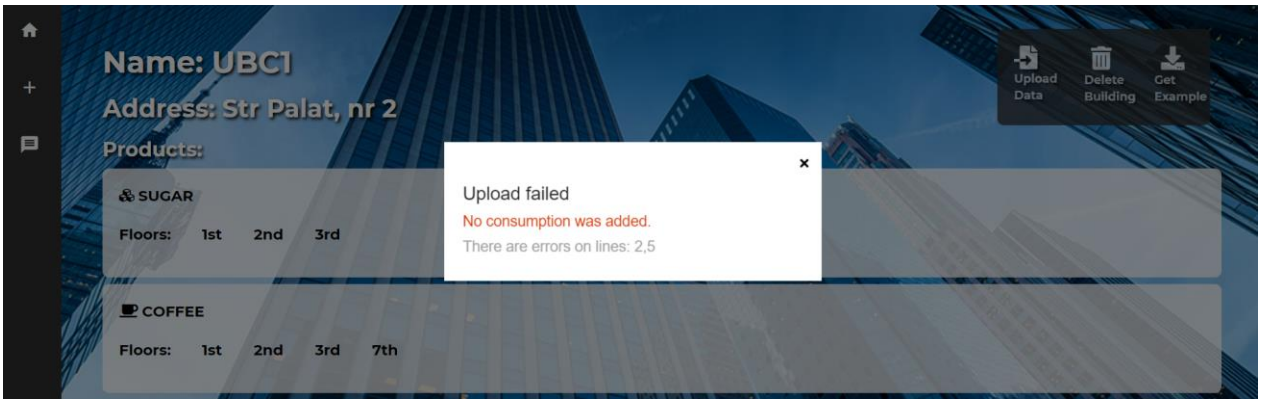

Figura 14 : Pop-up pentru încărcarea unui fișier incorect

Pentru a ști cum trebuie să arate fișierul care trebuie încărcat, utilizatorul poate accesa al treilea buton pentru a descărca un model. După ce a descărcat modelul, acesta poate observa atât câmpurile ce trebuie completate, cât și cum anume trebuie completate cu ajutorul a două exemple de acolo.

În cazul în care utilizatorul nu mai deține o clădire, aceasta poate fi ștearsă cu butonul din mijloc, însă va rămâne în continuare în baza de date nerelațională pentru a nu influența algoritmul de învățare automată. Respectivul imobil nu va mai apărea pe pagina cu toate clădirile și nu se vor mai face predicții pentru acesta.

III.3.3. Notificări

Când utilizatorul apasă pe iconița de notificări se apelează API-ul din Spring, care face export colecției de clădiri din MongoDB într-un fișier de tip .csv și apelează la rândul lui API-ul din .NET.

```
@GetMapping(value = "/algorithm")
public String startMLAlgoritm() throws IOException {
    return mlService.startMLAlgorithm();
}
```

Figura 15 : Endpoint în Spring pentru pornirea algoritmului de învățare automată

```
public String startMLAlgorithm() throws IOException {
    try {
        exportMongoDbToCSV();
        final String uri = "http://localhost:52199/algorithm";
        RestTemplate restTemplate = new RestTemplate();
        String predictions=restTemplate.getForObject(uri, String.class);
        return getPredictions(predictions);
    } catch (IOException e) {
        e.printStackTrace();
        return null;
    }
}
```

Figura 16 : Funcție pentru apelarea endpoint-ului din .NET

Cu ajutorul framework-ului ML.NET, se creează un model, se antrenează pe datele obținute din baza de date nerelațională și se fac predicții.

```
//Machine Learning model to load and use for predictions
private const string MODEL_FILEPATH = @"../../../NetApiML.Model/MLModel.zip";

//Dataset to use for predictions
private const string DATA_FILEPATH = @"E:\smart-buildings\NetApi\NetApi\data\consumption.csv";
private const string TEST_FILEPATH = @"E:\smart-buildings\NetApi\NetApi\data\prediction.csv";

1 reference
public static String MLAlgorithm()
{
    MLContext mlContext = new MLContext();

    // Training code used by ML.NET CLI and AutoML to generate the model
    ModelBuilder.CreateModel();

    ITransformer mlModel = mlContext.Model.Load(GetAbsolutePath(MODEL_FILEPATH), out DataViewSchema inputSchema);
    var predEngine = mlContext.Model.CreatePredictionEngine<ModelInput, ModelOutput>(mlModel);

    String message = makePredictions(mlContext, TEST_FILEPATH, predEngine);
    return message;
}
```

Figura 17 : Funcție în .NET pentru efectuarea prezicerilor

Funcția CreateModel() are următoarea structură:

```
public static void CreateModel()
{
    // Load Data
    IDataView trainingDataView = mlContext.Data.LoadFromTextFile<ModelInput>(
                                    path: TRAIN_DATA_FILEPATH,
                                    hasHeader: true,
                                    separatorChar: ',',
                                    allowQuoting: true,
                                    allowSparse: false);

    // Build training pipeline
    IEstimator<ITransformer> trainingPipeline = BuildTrainingPipeline(mlContext);

    // Evaluate quality of Model
    Evaluate(mlContext, trainingDataView, trainingPipeline);

    // Train Model
    ITransformer mlModel = TrainModel(mlContext, trainingDataView, trainingPipeline);

    // Save model
    SaveModel(mlContext, mlModel, MODEL_FILEPATH, trainingDataView.Schema);
}
```

Figura 18 : Crearea modelului pentru algoritmul de învățare automată

Aceste predicții sunt trimise înapoi la API-ul din Spring care formează mesajul și îl trimite utilizatorului pe front-end. Există o predicție pentru fiecare pereche (clădire, produs, etaj), deci numărul notificărilor va fi unul destul de mare. Pentru a găsi mai ușor predicțiile pentru o anumită clădire sau pentru a identifica toate locurile în care este nevoie de un anumit produs am adăugat acestei pagini și bară de căutare.

Figura 19 : Pagina de notificări cu toate prezicerile effectuate

III.3.4. ML.NET

Pentru a alege cel mai bun algoritm de învățare, am observat timp de câteva zile acuratețea rezultată la rularea algoritmilor pe date de antrenament din ce în ce mai mari. Am efectuat teste cu algoritmi de tip regresie și clasificare multiplă, iar de cele mai multe ori scorul a fost unul mai bun pentru cei de clasificare multiplă, motiv pentru care am și ales să folosesc un astfel de algoritm în predicțiile mele. În următoarele imagini sunt câteva rezultate ale testelor efectuate de mine:

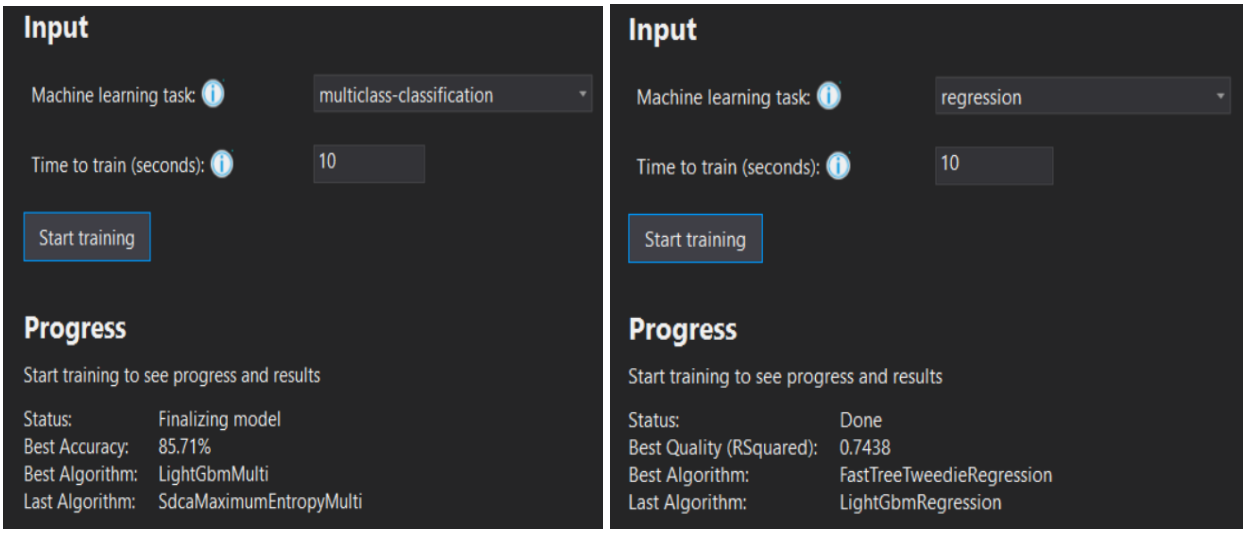

Figura 20 : Teste de tip regresie și clasificare multiplă cu rezultate foarte bune

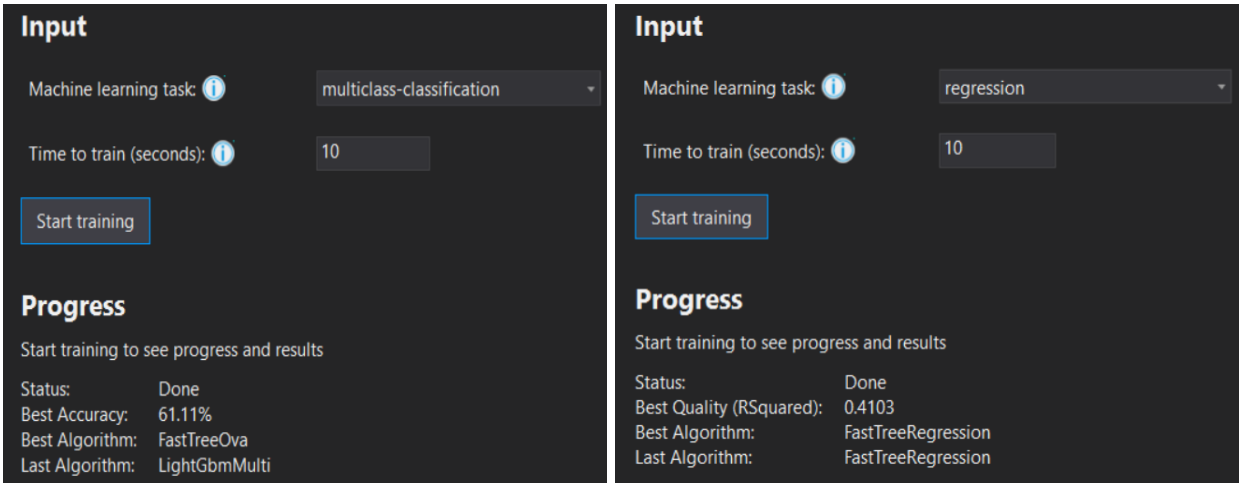

Figura 21 : Teste de tip regresie și clasificare multiplă cu rezultate satisfăcătoare

III.3.5. O singură predicție

Pentru a ușura și mai mult interacțiunea utilizatorului cu aplicația am implementat și posibilitatea de a prezice un singur element, identificat de perechea (clădire, produs, etaj). Pentru a face aceasta, utilizatorul trebuie să apese pe etajul din dreptul unui produs în pagina ce conține detaliile unei clădiri. Raționamentul este același ca și în cazul tuturor prezicerilor: se trimit datele necesare din front-end în back-end către API-ul din Spring, iar acesta din urmă va apela API-ul din .NET.

```
@PostMapping(value = "/singlePrediction")
public String singlePrediction(@RequestBody Prediction prediction) throws IOException {
    return mlService.getSinglePrediction(prediction.getBuildingId(), prediction.getProduct(),prediction.getFloor());
}
```

Figura 22 : Endpoint în Spring pentru o singură predicție

```java
public String getSinglePrediction(String buildingId, String product, String floor){
    try {
        exportMongoDbToCSV();
        exportSingleRow(buildingId,product,floor);
        final String uri = "http://localhost:52199/singlePrediction";
        RestTemplate restTemplate = new RestTemplate();
        String singlePrediction=restTemplate.getForObject(uri, String.class);
        String[] predictionArray=singlePrediction.split( regex: "@");
        String message = "In " + buildingRepository.findById(predictionArray[0]).get().getName() + predictionArray[1];
        return message;
    } catch (IOException e) {
        e.printStackTrace();
        return null;
    }
}
```

Figura 23 : Funcție pentru a solicita o singură predicție din .NET

```java
public static String getSinglePrediction()
{
    MLContext mlContext = new MLContext();

    // Training code used by ML.NET CLI and AutoML to generate the model
    //ModelBuilder.CreateModel();

    ITransformer mlModel = mlContext.Model.Load(GetAbsolutePath(MODEL_FILEPATH), out DataViewSchema inputSchema);
    var predEngine = mlContext.Model.CreatePredictionEngine<ModelInput, ModelOutput>(mlModel);

    // Create sample data to do a single prediction with it
    ModelInput sampleData = CreateSingleDataSample(mlContext, SINGLE_FILEPATH);

    // Try a single prediction
    ModelOutput predictionResult = predEngine.Predict(sampleData);

    String message = sampleData.BuildingId + "@ building, at the " + sampleData.Floor + " floor, " + predictionResult.Prediction +
        " packs of " + sampleData.Product + " are needed.";
    return message;
}
```

Figura 24 : Generarea unei preziceri în .NET

Rezultatul predicției va fi afișat prin intermediul unui pop-up:

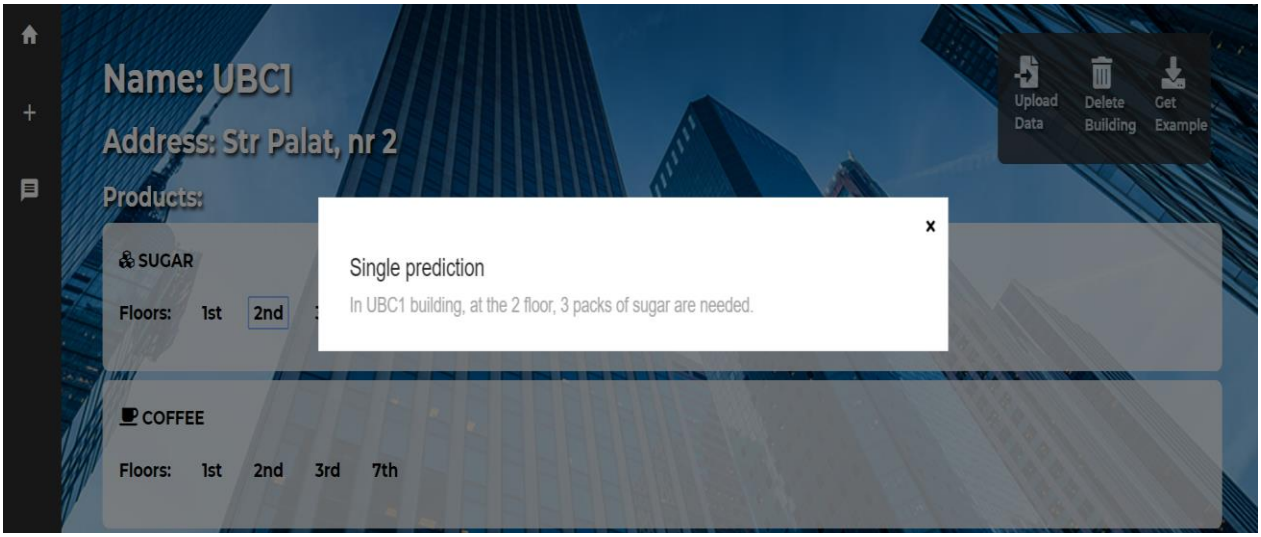

Figura 25 : Rezultatul unei predicții în Angular

Concluzii

Smart Buildings este o aplicație care contribuie la menținerea unui stoc de produse într-o firmă, evitând atât excesul, cât și deficitul, folosind unul dintre cele mai importante și apreciate domenii de cercetare la ora actuală și anume învățarea automată. Aplicația facilitează monitorizarea clădirilor deținute de o anumită firmă, a consumurilor generate de acestea, precum și reactualizarea permanentă a stocurilor. Algoritmul de învățare automată oferă rezultate destul de bune, aplicația fiind o modalitate foarte eficientă de a automatiza munca umană, conducând astfel la diminuarea costurilor inutile.

Aplicația prezentată ar putea fi îmbunătățită prin următoarele:

- adăugarea unui login pentru a putea fi folosită simultan de mai multe firme;
- sistemul de notificări ar putea fi implementat fără să fie nevoie de apelarea manuală a iconiței, ci automat, la anumite perioade de timp, și adăugarea unei alerte pe telefon sau pe email;
- posibilitatea de a genera statistici sau grafice pe baza consumurilor din baza de date.

Bibliografie

1. https://angular.io/docs
2. https://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/mvc.html
3. https://docs.mongodb.com/manual/
4. https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet
5. https://dotnet.microsoft.com/learn/machinelearning-ai/ml-dotnet-get-started-tutorial/intro
6. https://docs.microsoft.com/en-us/dotnet/machine-learning/tutorials/github-issue-classification
7. https://docs.microsoft.com/en-us/dotnet/machine-learning/tutorials/predict-prices
8. https://developer.mozilla.org/ro/docs/Web/HTML
9. https://developer.mozilla.org/en-US/docs/Web/CSS
10. https://devdocs.io/typescript/
11. https://docs.sheetjs.com/#workbook-object
12. https://www.mendix.com/smart-apps/?fbclid=IwAR1iBfs14kdePSQP3G4ogDStg2d7EVyIrTeBgMAAGdqkBk0setTWlEdC9Jk
13. http://onnergy.com/onnergy-functions/
14. https://play.google.com/store/apps/details?id=at.topfen.ecas&hl=en_US
15. https://stackoverflow.com/
16. https://www.baeldung.com/spring-data-mongodb-tutorial
17. https://www.baeldung.com/spring-controllers
18. https://www.w3schools.com/
19. https://www.e4developer.com/2018/08/06/spring-boot-best-practices/
20. https://www.freecodecamp.org/news/best-practices-for-a-clean-and-performant-angular-application-288e7b39eb6f/
21. https://www.geeksforgeeks.org/machine-learning/