

UNIVERSITATEA POLITEHNICA BUCUREȘTI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL AUTOMATICĂ ȘI INFORMATICĂ
APLICATĂ



LUCRARE DE LICENȚĂ

Aplicație pentru managementul activităților într-o companie

Coordonator:

Conf. Dr. Ing. Mihnea Moiescu

Absolvent:

Andreea Erika Vișan

BUCUREȘTI

2018

1 CUPRINS

2	Introducere	3
2.1	Context	3
2.2	Obiective	3
2.3	Rezumat.....	4
3	Studiul aplicațiilor similare.....	5
3.1	Charsima.....	5
3.2	AceProject	6
3.3	JIRA	8
4	Tehnologii folosite	10
4.1	Angular 2.4.....	10
4.2	TypeScript	11
4.3	C#	12
4.4	.Net Core	13
4.5	MSSQL.....	13
4.6	HTML.....	13
4.7	CSS – Cascading Style Sheets	15
4.8	Bootstrap	15
4.9	Protocolul HTTP	16
4.10	EntityFramework.....	18
4.11	Mediul de dezvoltare	18
5	Implementare	19
5.1	Modul de funcționare al aplicației.....	19
5.2	Prezentare funcționalități.....	20
5.3	Managementul clienților și al proiectelor.....	21

5.3.1	Nomenclator clienți.....	21
5.3.2	Nomenclator proiecte.....	26
5.3.3	Nomenclator tipuri de proiecte	28
5.3.4	Nomenclator statusuri proiecte	29
5.3.5	Nomenclator task-uri	30
5.3.6	Nomenclator tipuri de task-uri	31
5.3.7	Nomenclator priorități ale task-urilor	32
5.3.8	Nomenclator de stări ale task-urilor.....	32
5.4	Managemenet zile de concediu	33
5.4.1	Zile libere legale	33
5.4.2	Nomenclator cereri de concediu	35
6	Studiu de caz	37
7	Concluzie	41
8	Dezvoltări ulterioare	42
9	Dicționar de termeni	43
10	Bibliografie	45

2 INTRODUCERE

2.1 Context

„Povestea internetului a început printr-un e-mail. Prima utilizare publică a internetului a fost prin e-mailuri și s-a întâmplat în 1989. În același an, Tim Berners-Lee și echipa sa CERN au inventat World Wide Web, ceea ce încă mai folosim astăzi când lansăm un browser de internet.”[1]

Trei decenii mai târziu, am ajuns să trăim într-o lume în care tehnologia a luat amploare într-un mod considerabil, în fiecare secundă dezvoltându-se diferite aplicații corelate cu aproape orice domeniu.

În România, ritmul de creștere al numărului de companii în domeniul IT a atins un procent de 10-11% anual în ultima perioadă de timp, existând astfel peste 17 000 de companii în acest domeniu și peste 100 000 de angajați.[2]

Astfel, în interiorul unei firme, atât pentru o gestiune mai bună a resurselor umane cât și pentru a ține evidența numărului de ore alocat pe diferite proiecte, s-au implementat aplicațiile software pentru gestiunea activităților.

2.2 Obiective

Obiectivul prezentei lucrări este dezvoltarea unei aplicații pentru managementul activităților într-o companie IT, astfel încât să permită alocarea resurselor umane pe diverse proiecte și task-uri de către manager, de a ține o evidență a numărului de ore pentru fiecare task sau proiect sau a numărului de ore lucrate săptămânal de către fiecare angajat.

Consider că o astfel de aplicație are ca beneficiu în primul rând economisirea de timp deoarece evoluția progreselor făcută de către angajați poate fi urmărită și înregistrată zilnic, săptămânal sau lunar. Prin acest lucru, întreprinderile își reduc timpul pentru calcularea salariilor cu un procent major. De asemenea, angajații se pot concentra mai mult asupra atribuțiilor pe care le au de îndeplinit și nu asupra sarcinii suplimentare de contabilizare pentru ei înșiși.

Mai mult decât atât, simpla sortare, organizare și procesare a documentelor poate duce la un număr de ore petrecute în plus. În mod similar, costul de completare a elementelor necesare pentru a menține un sistem vechi de timp și de prezență (hârtie, pixuri / creioane) poate fi

prohibitiv pe termen lung.

2.3 Rezumat

Aplicația are ca obiectiv managementul clienților, proiectelor și task-urilor din interiorul companiei. Pe lângă aceste funcționalități, aplicația oferă posibilitatea de a organiza și crea cereri de concediu, vizualizarea istoricului acestuia, dar și aprobarea lor de către superiori.

În aplicație putem defini o serie de date grupate pe nomenclatoare. Printre cele mai importante dintre acestea se enumera următoarele:

- Clienți
- Proiecte
- Task-uri
- Angajați
- Zile libere
- Cereri de concediu

Pentru a realiza toate aceste funcționalități, am construit două aplicații, o aplicație pentru partea de Frontend și una pentru partea de Backend. Cea dintâi a fost construită folosind framework-ul AngularJS 2.4, limbajul de marcat HTML, standardul de formatare CSS și librăria Bootstrap. Cea de-a doua aplicație constă într-un API realizat cu ajutorul framework-ului .NET Core scris în limbajul C#. Aici am folosit mai multe componente precum: AutoMapper și EntityFramework.

3 STUDIUL APLICAȚIILOR SIMILARE

3.1 Charsima

O aplicație similară care promite o soluție software de resurse umane care să ajute la menținerea echilibrului atunci când punem în balanță obiectivele angajatului și cele ale companiei este „Charisma”.

Această aplicație este alcătuită din module care au scopul de a acoperi diferite problematice de resurse umane. Câteva dintre acestea sunt:

- Date personale

Acest modul include aspecte legate de activitățile de management a resurselor umane, cum ar fi: detaliile cu privire la angajații unei companii, performanța de care aceștia dau dovadă, definirea modului de organizare a unei firme etc.

- Audit

Modulul Audit permite vizualizarea istoricului în ceea ce privește modificările efectuate de utilizatori în baza de date oferind un set de rapoarte cu privire la datele în care actualizările au fost făcute, care sunt acestea și de către cine au fost executate.

- Portal HCM

Modulul HCM joacă un rol fundamental întrucât este responsabil pentru îmbunătățirea comunicării între angajați, informarea pozițiilor pe care aceștia le ocupă în cadrul firmei și alte informații cu caracter personal etc. Toate acestea au scopul de a implica și responsabiliza personalul.

- Pontaj

Acest modul este utilizat pentru a ține cont de orele de venire, respectiv plecare ale personalului, ajutând astfel la calcularea și ținerea în evidență a numărului de ore lucrate de către fiecare angajat în parte. În acest fel, managerul are de asemenea posibilitatea de a superviza fluxul persoanelor care au acces în interiorul companiei.[3]

Deși modulele abordate stârnesc curiozitatea unui posibil client, „Charisma” este o aplicație care a început să fie dezvoltată cu mai bine de zece ani în urmă. Tocmai din această cauză, mai exact a faptului că se bazează pe tehnologii vechi, viteza de lucru nu este suficient

de bună, iar numărul erorilor care apar tind să crească în mod frecvent. Spre deosebire de aceasta, aplicația DataKlasHR nu întâmpină astfel de probleme.

Tot din cauza faptului că aplicația se bazează pe tehnologii vechi, interfața nu oferă un aspect care să încante utilizatorul. Acest lucru poate fi observat în figura de mai jos:

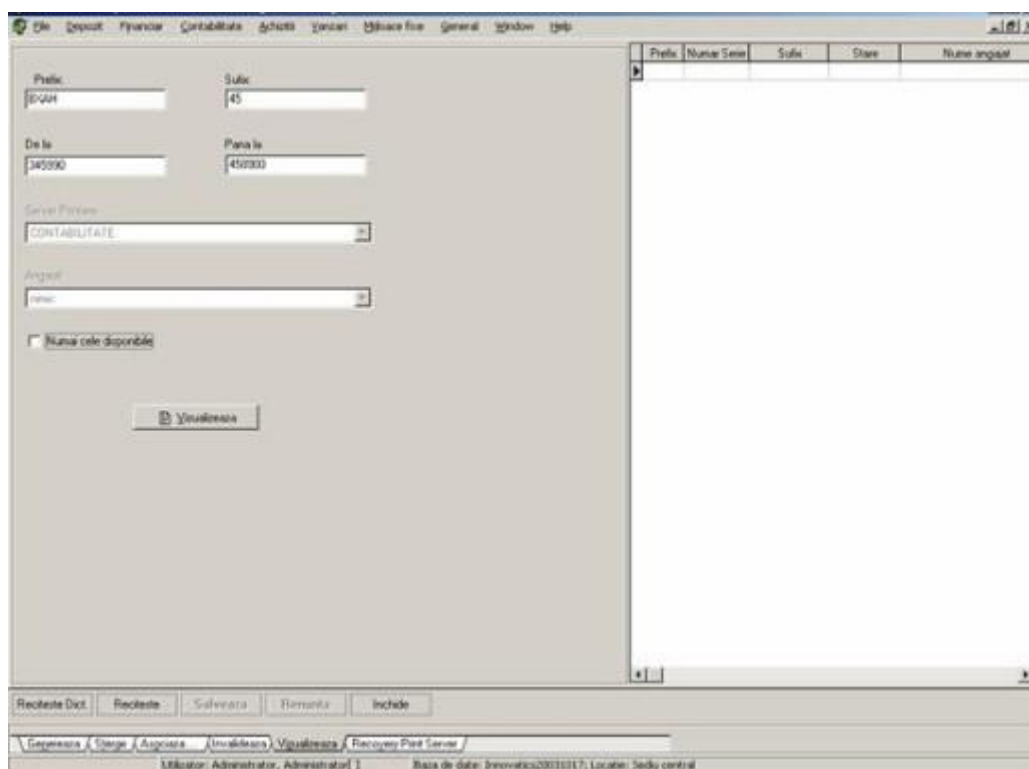


Figura 1 – Interfața aplicației „Charisma”¹

3.2 AceProject

„AceProject” este un instrument de management de proiect construit pentru a vă ajuta să vă organizați folosind o abordare colaborativă. Această soluție poate fi utilizată atât individual, cât și de echipe mici sau întreprinderi pentru a deține controlul asupra fluxului de lucru.

AceProject reușește să furnizeze un pachet robust, eficient și accesibil, care permite utilizatorilor să gestioneze o mulțime de aspecte legate de managementul proiectului, timpul de lucru, gestionarea sarcinilor, documente și multe altele.[4]

În comparație cu alte aplicații, printre care și cea descrisă mai sus, AceProject este mult mai rapidă. Cu toate că oferă numeroase facilități, utilizatorii afirmă că interfața aplicației nu

¹ <http://litesofttechs.bitballoon.com/program.html>

este foarte intuitivă și din această cauză câteva videoclipuri referitoare la utilizarea aplicației ar fi benefice.

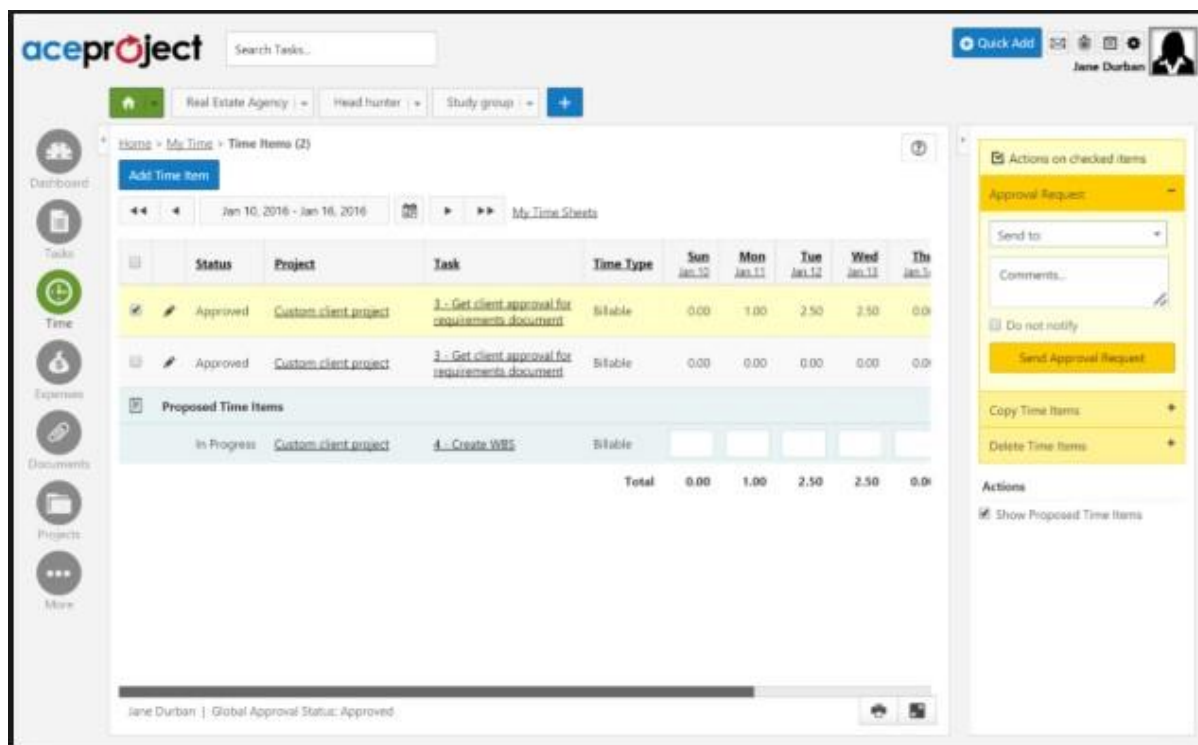


Figura 2 – Interfața aplicației „AceProject”²

Câteva îmbunătățiri pe care DataKlasHR le aduce față de AceProject sunt:

- Viteza de lucru mai rapidă datorită tehnologiilor mai noi utilizate
- Interfața este mult mai intuitivă
- Există opțiunea managementului zilelor de concediu

² <https://www.aceproject.com/>

- Este mobile friendly, diverse operațiuni se pot face direct de pe telefon (spre deosebire de aplicația „responsive” de la Ace)

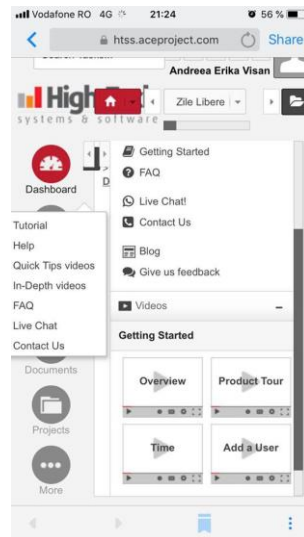


Figura 3 – Interfața mobile a aplicației „AceProject”

- Exista posibilitatea de a alege între limba română și limba engleză

3.3 JIRA

JIRA este un instrument dezvoltat de compania „Atlassian”. Acesta este utilizat pentru urmărirea erorilor, urmărirea problemelor și gestionarea proiectelor. Câteva din avantajele pe care această aplicație le oferă utilizatorilor sunt:

- Poate fi utilizată atât de echipe mici, cât și de echipe cărora le sunt alocate proiecte de dimensiuni mai mari
- Permite segmentarea proiectelor în proiecte mai mici sau multiple task-uri într-un mod care sporește eficiența și performanța
- Permite utilizatorului să completeze câmpuri precum „ore estimate” alocate pe proiect și la final îi oferă acestuia posibilitatea de a compara estimarea cu numărul de ore efectiv alocat
- Există de asemenea varianta mobile a acestei aplicații, ceea ce înseamnă că angajatul primește notificări despre modificările apărute sau task-urile care i-au fost atribuite de către manager [5]

Jira este utilizată la nivel mondial de diferite companii datorită beneficiilor pe care le prezintă, de aceea aceasta a reprezentat o sursă de inspirație pentru aplicația pe care o dezvoltăm.

„Jira este proiectat pentru a ajuta utilizatorii să capteze, să alocă și să stabilească priorități pentru munca lor. Vă permite să gestionați întregul proces de dezvoltare al aplicațiilor, asigurându-vă că toate lucrurile sunt acoperite. Interfața sa simplă și intuitivă permite colaborarea cu colegii de echipă și vă permite să vă finalizați task-urile într-o manieră eficientă.” [6]

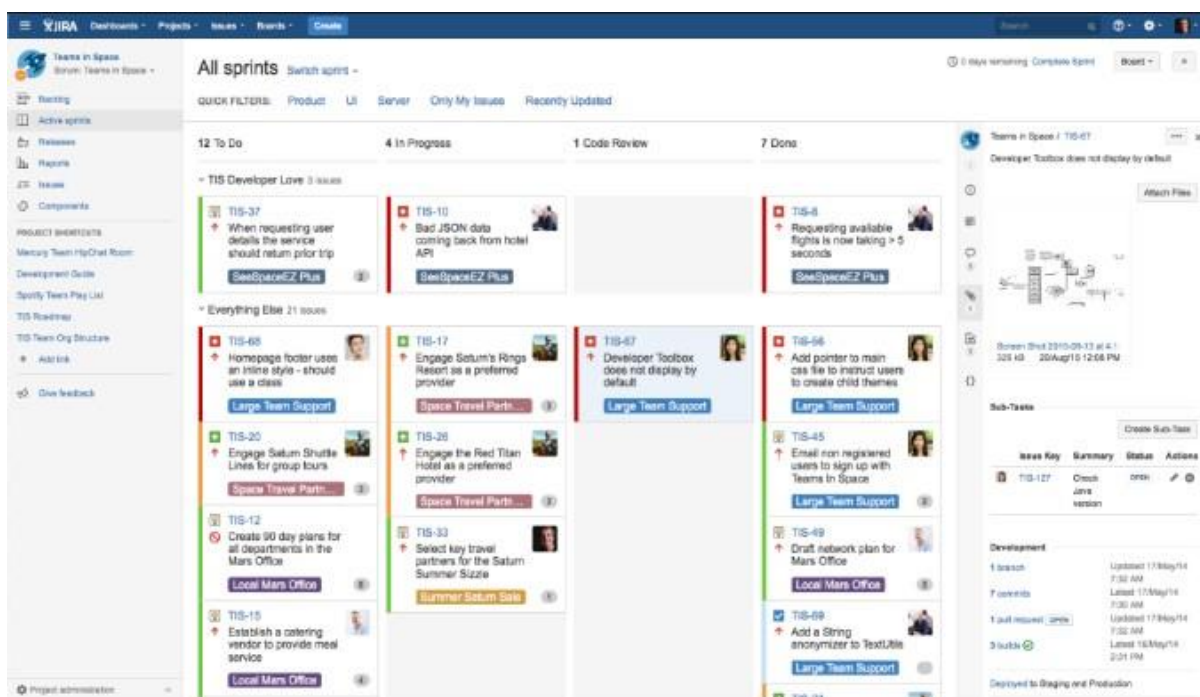


Figura 4 – Interfața aplicației „Jira”³

³ <https://www.betterbuys.com/project-management/reviews/jira/>

4 TEHNOLOGII FOLOSITE

4.1 Angular 2.4

Angular este un framework pentru dezvoltarea web a aplicațiilor care combină șabloane de-clarative, „dependency injection” și cele mai bune practici integrate pentru a putea face față provocărilor întâlnite de către dezvoltatori. Acesta este actualizat în mod constant pentru o utilizare mai ușoară și o productivitate sporită. Motivul pentru care am ales Angular 2.4 este faptul că prezintă următoarele avantaje:

- Consecvența:

Cadrul general se bazează pe componente și servicii care au aceeași structură:

1. Importul cu modulele necesare
2. Definirea componentei decorator
3. Scrierea propriu-zisă a metodelor în interiorul clasei

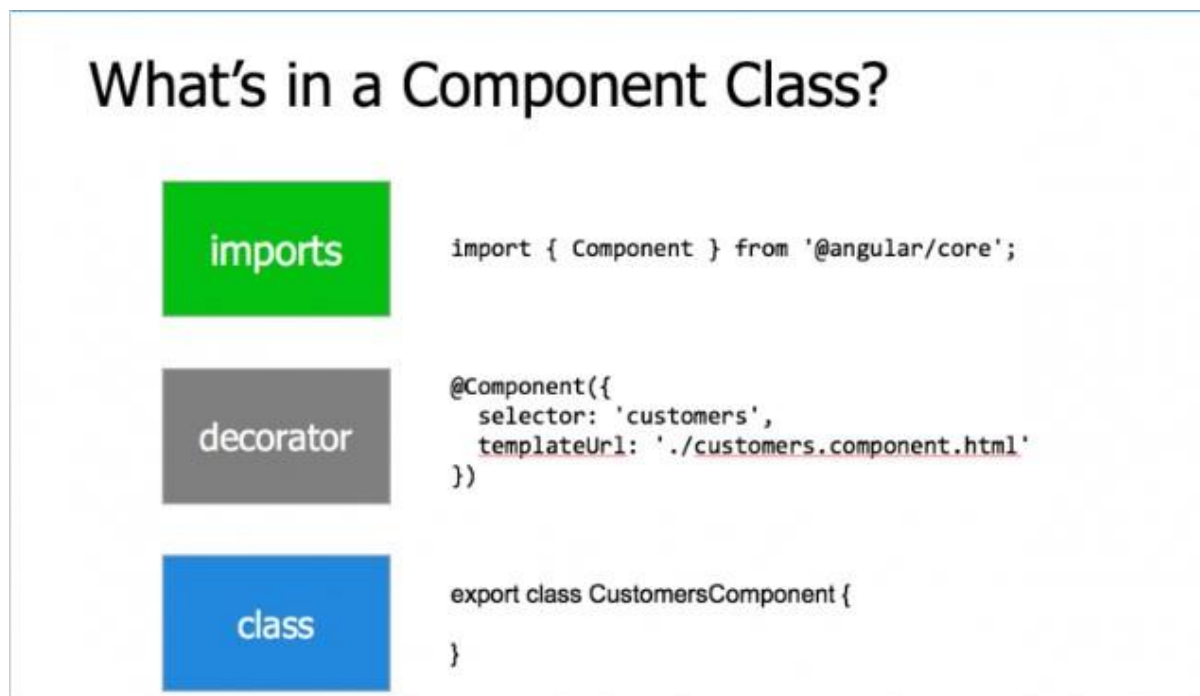


Figura 5- Cadrul general al unei componente Angular⁴

⁴ <https://blog.codewithdan.com/2017/08/26/5-key-benefits-of-angular-and-typescript/>

- Testarea: Angular 2.4 permite detecția și afișarea instantă a erorilor după fiecare salvare a codului.
- Code splitting: permite încărcarea doar a codului necesar pentru randarea paginii solicitate de către utilizator, în acest mod crescând viteza de încărcare pentru aplicațiile Angular.[7]

Elementele principale ale unei aplicații Angular sunt:

- Module

Blocurile de bază ale unei aplicații Angular sunt denumite ca „NgModules”, care asigură și permit compilarea unui set de componente. NgModules colectează și grupează codul asociat în seturi funcționale. O aplicație are întotdeauna cel puțin un modul rădăcină numit „AppModule” care asigură mecanismul de pornire care lansează aplicația.

- Componente

O componentă reprezintă de fapt o clasă care conține date și metode ce tratează evenimentele declanșate din pagina HTML afișată clientului prin intermediul browser-ului. Prin urmare, o componentă împreună cu un template alcătuiesc un Angular view.

- Servicii

Serviciul este o categorie largă care cuprinde orice valoare, funcție sau caracteristică de care o aplicație are nevoie. Un serviciu este de obicei o clasă cu un scop îngust și bine definit. O componentă nu ar trebui să definească lucruri precum modul de preluare al datelor de pe server, validarea intrărilor etc. În schimb, ea poate delega astfel de sarcini serviciilor. Definind acest tip de sarcină de procesare într-o clasă de servicii injectabile, el este pus la dispoziția oricărei componente.[8]

4.2 TypeScript

TypeScript este un super set al limbajului JavaScript care nu poate rula direct în browser și necesită să fie compilat în cod JavaScript. Compilarea lui se face cu ajutorul browser-ului sau cu ajutorul framework-ului NodeJS. În următoarea captură de ecran este exemplificată transformarea unui cod TypeScript în cod JavaScript.

Typescript	Javascript
<pre> class Greeter { greeting: string; constructor (message: string) { this.greeting = message; } greet() { return "Hello, " + this.greeting; } } </pre>	<pre> var Greeter = (function () { function Greeter(message) { this.greeting = message; } Greeter.prototype.greet = function () { return "Hello, " + this.greeting; }; return Greeter; })(); </pre>

Figura 6 - Diferența între Typescript și Javascript⁵

TypeScript se folosește de concepte noi precum module, clase, constante, interfețe. Este un limbaj strongly typed, ceea ce ne ajută la detectarea unor posibile bug-uri. Este adevărat că efortul de dezvoltare este unul mai mare deoarece trebuie specificate tipurile de clase și de variabile, dar permite posibilitatea detectării anumitor probleme care ar apărea la run time. TypeScript este un limbaj folosit în dezvoltarea framework-ului AngularJS.

4.3 C#

C Sharp este un limbaj de programare orientat pe obiecte. Acesta seamănă foarte mult din punctul de vedere al sintaxei cu limbajul de programare C++. Este unul dintre cele mai folosite limbaje de programare deoarece majoritatea tehnologiilor Microsoft se bazează pe acesta. C Sharp este un limbaj compilat.

Ca orice alt limbaj care se bazează pe paradigma OOP (Programare orientată pe obiecte), suportă concepte ca încapsulare, moștenire, polimorfism. C Sharp-ul a ajuns până la versiunea 7.0 care vine cu îmbunătățiri precum tupluri, deconstructori, variabile de tip out etc. Aceste opțiuni ajută la scrierea unui cod cât mai clar și ușor de întreținut.[9]

Câteva dintre avantajele C Sharp-ului sunt:

- Limbaj orientat pe obiecte
- Sintaxa apropiată de limbajul natural în limba engleză
- Limbaj type-safe

⁵ <https://www.dunebook.com/typescript-vs-javascript-why-typescript-is-next-to-big-thing/>

- Conține un Garbage Colector
- Sunt disponibile multe librării compatibile care ușurează munca programatorului.

4.4 .Net Core

.NET Core este un framework open-source oferit de compania Microsoft. Acesta este scris în limbajul C# și este cross-platform, adică rulează atât pe Windows cât și pe Linux. .NET Core este succesorul framework-ului ASP .NET. Acest framework este unul modular oferit sub formă de pachete Nuget.

Nuget este un package manager open-source creat de Microsoft. Pachetele reprezintă cod care poate fi refolosit, compilat în fișiere de tip DLL. Unul din principalele avantaje ale folosirii lui .NET Core este acela că are în spate o comunitate foarte mare și se găsesc foarte rapid rezolvări ale diverselor probleme.

4.5 MSSQL

MSSQL este un SGBD (sistem de gestiune al bazelor de date) funcțional independent dezvoltat de către Microsoft. Acest produs este construit pentru funcția de stocare a datelor și poate fi rulat fie local, fie pe un alt server din rețea. MSSQL este un sistem performant care răspunde foarte repede interogărilor datorită îmbunătățirii ultimelor versiuni. Limbajul de interogare este SQL (Structured Query Language).[10]

SQL este un limbaj de programare standardizat folosit pentru interogarea bazelor de date relaționale. Cu ajutorul acestuia, putem crea tabele, insera, actualiza și șterge informații dar și alte operații precum adăugarea de indecși și alterarea tabelor. Un exemplu de interogare este următorul: „update Clients set Name = 'John' where id = 1”. Conform exemplului anterior, ne putem da seama că limbajul este unul simplu de înțeles și apropiat de limbajul natural.

În comparație cu alte sisteme de gestiune a bazelor de date, MSSQL este mai rapid și fiind contracost există suport din partea echipei Microsoft. Există și o versiune gratuită care se numește MSSQL Light, dar care bineînțeles că este mai limitată.

4.6 HTML

Pentru interfața cu utilizatorul a fost folosit limbajul de marcaj HTML (Hyper Text Markup Language). Acesta constă în mai multe etichete denumite și tag-uri care permit

organizarea elementelor dintr-o pagină web. Aceste elemente pot fi: texte, imagini, liste etc. Structura unei pagini HTML este următoarea:

Un document HTML este deschis cu tag-ul `<html>`. În secțiunea `<head>` sunt încărcate fișiere externe precum scripturi JavaScript, fișiere CSS, metatag-uri și altele. În secțiunea `<body>` sunt puse elementele pe care utilizatorul trebuie să le vadă în pagină (ex. texte, imagini).

Elementele sunt etichetate pentru identificarea ulterioară din nevoia de a le stiliza sau modifica. Există o gamă largă de tag-uri cum ar fi: `<p>` pentru paragrafe, `` pentru imagini, `/` pentru liste, `<header>`, `<article>` etc. Asupra tag-urilor din HTML, putem seta diferite atribute prin care le putem diferenția. Două dintre cele mai comune atribute utilizate sunt „id” și „class”. Atributul de tip id este unic într-o pagină HTML, folosirea acestuia de două ori reprezentând o eroare. Atributul de tip class se poate regăsi la mai multe elemente în același timp.

În standardul CSS, putem face referire la elemente cu un anumit id sau cu o anumită clasă și le putem modifica proprietăți precum: culoare, așezare, font etc. Pentru o mai bună organizare, în versiunea HTML5 au fost introduse tag-uri precum `<article>`, `<header>`, `<footer>` și `<nav>`.

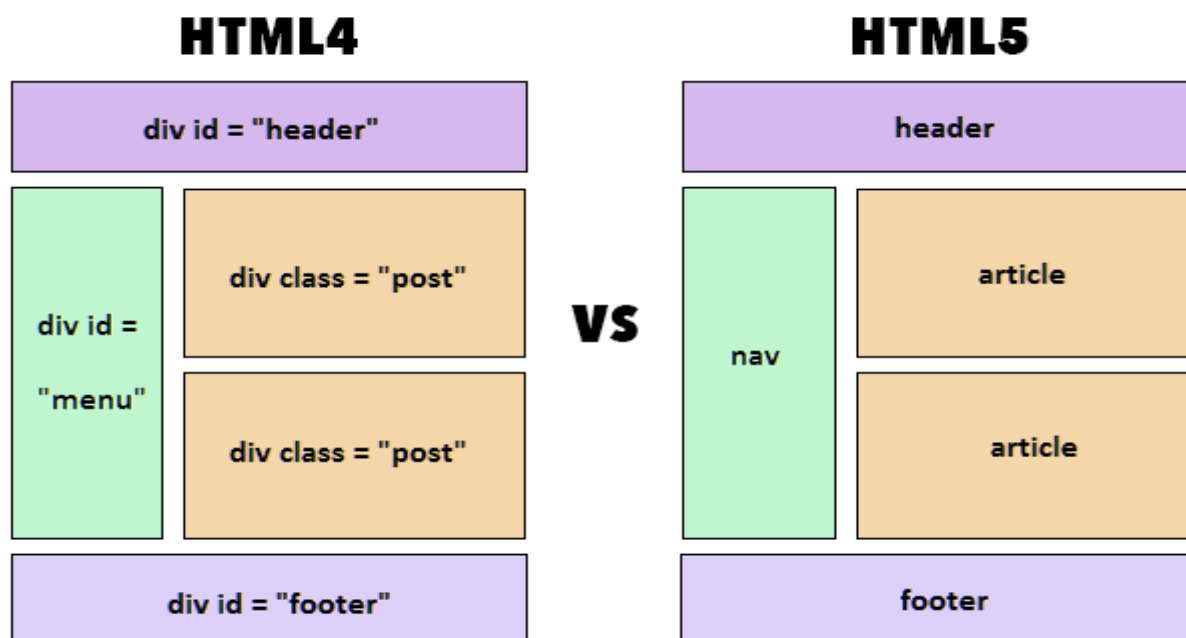


Figura 7 – Diferența dintre HTML4 și HTML5⁶

4.7 CSS – Cascading Style Sheets

Pentru a descrie prezentarea paginilor este utilizat limbajul CSS care include proprietăți precum culori, fonturi, aspect. Acesta permite adaptarea prezentării pentru diferite rezoluții.

Câteva dintre avantajele acestui limbaj sunt:

- Compatibilitate cu diferite browsere
- Conținutul este separat de prezentare
- O încărcare mai rapidă a paginilor: În CSS, nu este necesar să folosești attribute pe fiecare tag HTML, proprietățile sunt aplicate pe o clasă definită de programator. Aceeași clasă se poate afla pe mai multe elemente din diferite pagini de HTML, rezultatul fiind observabil pe fiecare dintre acestea. Mai puțin cod înseamnă o încărcare mai rapidă a paginii.

4.8 Bootstrap

Bootstrap este cel mai cunoscut framework de tip open-source și a fost dezvoltat de către echipa Twitter. Bootstrap este o combinație de CSS, HTML și JavaScript care ajută la

⁶ <https://www.besanttechnologies.com/difference-html-html5-css-css3>

construirea ușoară a interfețelor pentru utilizatori. Datorită funcțiilor sale, se pot crea design-uri responsive în funcție de rezoluția fiecărui dispozitiv în parte.

Avantaje [11]:

- Economisești timp datorită claselor predefinite
- Este ușor de folosit
- Poate fi descărcat gratuit
- Are compatibilitate cu browsere moderne precum Mozilla Firefox, Google Chrome, Safari, Internet Explorer și Opera.

Una dintre cele mai importante componente ale acestei librării este aceea care ne ajută să poziționăm elementele într-o pagină, așa-zisul Grid System. Acesta se bazează pe noțiunile de rânduri și coloane. Sistemul are la bază dimensionarea coloanelor de pe un anumit rând. Aceasta se face în mod dinamic în funcție de dimensiunile device-ului pe care este randată aplicația. Folosind clasele de mai jos, putem specifica dimensiunile în funcție de lățimea ecranului:

- col-xs: pentru rezoluții mai mici de 768px
- col-sm: pentru rezoluții mai mari sau egale de 768px
- col-md: pentru rezoluții mai mari sau egale de 992px
- col-lg: pentru rezoluții mai mari sau egale de 1200px

Această librărie ne oferă diferite widget-uri și structuri HTML gata făcute precum calendar, box-uri stilizate, liste stilizate, butoane etc.

4.9 Protocolul HTTP

Prescurtarea de HTTP provine de la Hypertext Transfer Protocol care permite comunicarea între o varietate de host-uri și clienți și suportă un amestec de configurații de rețea. Această comunicare se realizează prin intermediul unei perechi de tip cerere-răspuns: clientul trimite o cerere către server, iar apoi așteaptă ca acesta să îi trimită un răspuns cu privire la solicitarea făcută. [12]

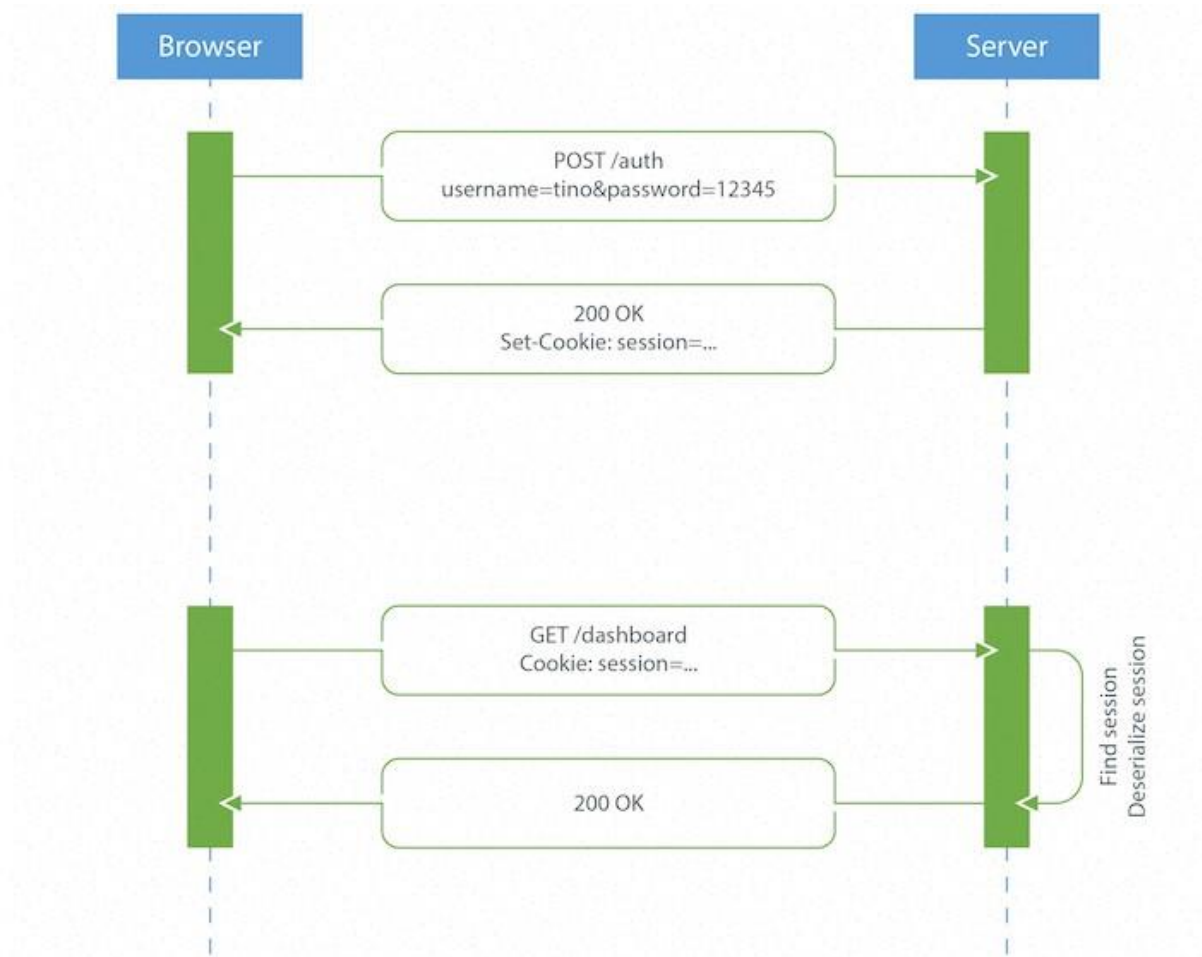


Figura 8 – Comunicarea client-server printr-o cerere HTTP⁷

Unele dintre cele mai utilizate tipuri de cerere sunt:

- GET: Metoda HTTP GET solicită o reprezentare a resursei specificate. Solicitățile folosind această metodă ar trebui să recupereze numai date.
- POST: Metoda HTTP POST trimite date către server. O solicitare de acest tip este trimisă prin intermediul unui formular și are ca rezultat o modificare asupra datelor din server.
- DELETE: Prin folosirea acestei metode, se trimite o cerere către server prin care datele solicitate vor fi șterse.

⁷ <https://www.toptal.com/web/cookie-free-authentication-with-json-web-tokens-an-example-in-laravel-and-angularjs>

4.10 EntityFramework

Denumirea completă a termenului de ORM este Object Relational Mapping și reprezintă un mecanism care permite abordarea, accesarea și manipularea obiectelor fără a lua în considerare modul în care obiectele respective se raportează la sursa de date [13]. ORM-ul este utilizat pentru a face operațiile CRUD (Create, Read, Update și Delete) la baza de date mult mai simple și eficiente.

Entity Framework este un framework ORM open-source pentru aplicațiile .NET suportate de Microsoft. Acesta permite dezvoltatorilor să lucreze cu datele prin intermediul claselor specifice, fără a se concentra atât de mult pe tabelele și coloanele bazei de date în care sunt stocate datele. Prin intermediul Entity Framework-ului, programatorii pot lucra la un nivel mai înalt întrucât este posibilă o scriere mai puțină a codului comparativ cu aplicațiile „tradiționale”.

4.11 Mediul de dezvoltare

Suita de aplicații folosită pentru dezvoltarea acestui proiect este oferită de Microsoft. Pentru aplicația de Backend, respectiv web Api, am folosit Visual Studio 2017, iar pentru cea de Frontend, am utilizat Visual Studio Code. Pentru gestionarea bazei de date m-am servit de Microsoft Server Management Studio, iar pentru versionarea de cod TFS.

Visual Studio este un mediu de dezvoltare profesional care asigură scrierea rapidă a codului. Acesta pune la dispoziție diferite tool-uri care ajută la crearea unor aplicații moderne. Fiind o tehnologie Microsoft, se integrează foarte bine cu TFS (Team Foundation Server) care este o soluție de gestionare a proiectelor oferită tot de Microsoft. Soluția permite centralizarea fișierelor, organizarea lor pe versiuni și modificarea lor de către mai mulți utilizatori în același timp. [14]

Visual Studio Code este un mediu de dezvoltare asemănător cu Visual Studio, dar care, spre deosebire de acesta, poate rula și pe Linux sau pe MacOS. Am folosit Visual Studio Code pentru scrierea aplicației de Frontend, fiind o versiune mai light a mediului Visual Studio. [15]

Microsoft Server Management Studio reprezintă o aplicație cu ajutorul căreia ne putem conecta la o bază de date MSSQL și prin care putem opera pe aceasta. Interogările se pot face fie folosind interfața grafică cu evenimente de click, dar și scriind limbaj SQL.

5 IMPLEMENTARE

5.1 Modul de funcționare al aplicației

Aplicația este structurată pe două componente: partea de Frontend și partea de Backend. Interfața grafică este interpretată de browser-ul web care citește și interpretează codul HTML din fișierele de pe server. Partea de Frontend este constituită sub forma unei aplicații scrise utilizând framework-ul AngularJS 2.4. Principiul de funcționare este urmatorul:

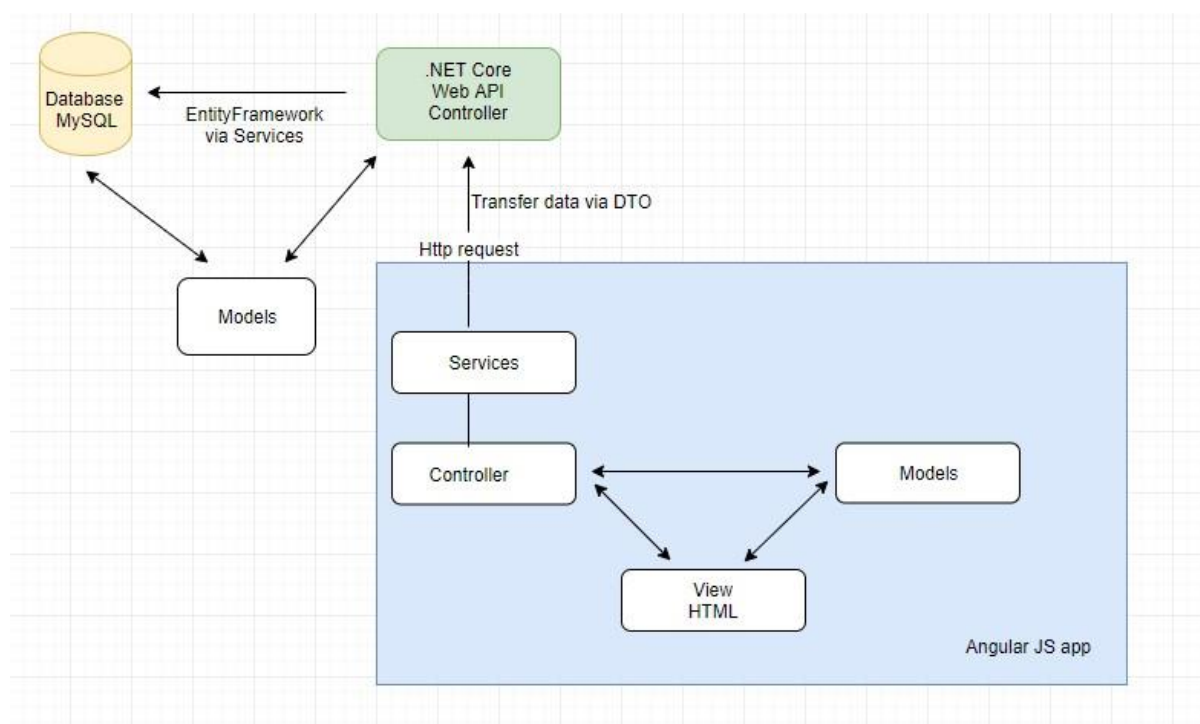


Figura 9 – Fluxul datelor

Acțiunile utilizatorilor sunt consumate în aplicația de Angular care interoghează baza de date prin intermediul web API-ului. Datele introduse de către utilizator ajung în Controllerul de Angular, iar mai departe, prin intermediul unor metode aflate în diverse servicii, ajung să fie transferate printr-un request HTTP către aplicația de Backend, adică web Api. Datele sunt abstractizate sub forma unor obiecte de tip DTO. Mai departe, în web API se lucrează cu baza de date folosind un ORM și anume EntityFramework. Baza de date este una de tip relațională, MSSQL. Am folosit această bază de date deoarece se integrează foarte ușor cu .NET Core, ambele fiind tehnologii Microsoft.

Conexiunea dintre tabele a fost făcută prin intermediul cheilor primare și cheilor străine. Cheia primară constă într-o coloană (uneori se poate folosi o grupare a mai multor coloane) ale cărei date conținute sunt utilizate pentru identificarea unică a fiecarui rând din tabelă. Cheia secundară este la rândul ei o coloană care face referire la cheia primară dintr-o altă tabelă. Legătura se poate realiza fie în Diagramă sau prin scrierea de cod.

Folosind C#, în Visual Studio, am creat o clasă denumită entitate care conține aceleași proprietăți cu cele ale unei tabele. Maparea dintre obiectele de tip DTO și obiectele ce reprezintă entitățile din baza de date este făcută automat folosind componenta „Automapper”.

Automapper-ul este o componentă care ne ajută să completăm proprietățile unui obiect cu date dintr-un alt obiect. Acesta depistează proprietățile comune ale celor două obiecte și dispensează programatorul de o muncă suplimentară, adică aceea de a atribui valori fiecărei proprietăți în parte. Un exemplu de utilizare a Automapper-ului este afișat în printscreen-ul de mai jos.

```
CreateMap<ClientDto, Client>();  
CreateMap<Client, ClientDto>()  
    .ForMember(x => x.CreatedOn, y => y.MapFrom(z => z.CreatedOn.ToString("dd.MM.yyyy")));
```

Se poate vedea cum este completată o entitate de tip Client folosind valorile obținute în urma unui transfer de date folosind un obiect de tip DTO. Automapper-ul oferă posibilitatea de a specifica cum să fie transferate valorile respective. În imaginea de mai sus, se dorește maparea valorii dintr-un câmp de tip DateTime într-un câmp de tip String făcând conversia necesară cu ajutorul metodei „ToString” al proprietății CreatedOn.

5.2 Prezentare funcționalități

Acest proiect își propune să țină evidența numărului de ore lucrate de către fiecare angajat, numărului de zile de concediu rămase pentru fiecare dintre aceștia, clienților care colaborează cu noi și pentru care dezvoltăm diverse proiecte și de asemenea a proiectelor și task-urilor alocate de către manager.

Principalele funcționalități ale aplicației sunt:

- Crearea și modificarea datelor unui client
- Adăugarea și editarea detaliilor unui proiect
- Adăugare tip proiect

- Adăugare status proiect
- Adăugare status task
- Adăugare și modificare task
- Adăugare tip task
- Adăugare prioritate task
- Adăugare zile libere legale
- Vizualizare cereri de concediu

5.3 Managementul clienților și al proiectelor

5.3.1 Nomenclator clienți

Tabela clienți conține câmpuri precum: Nume care este de tip string și este obligatoriu, Descriere: câmp opțional de tip string, Valid de tip boolean și de asemenea câmpurile CreatedOn care indică data în care adăugarea a fost făcută, CreatedByOperator care arată persoana care a realizat inserarea, ModifiedByOperator și ModifiedOn reprezentând data, respectiv persoana care a modificat datele despre client.

Ultimele patru câmpuri nu sunt vizibile pentru utilizator, întrucât acest lucru nu este relevant. În cazul în care câmpul Valid nu este bifat, cu alte cuvinte are valoarea zero, clientul se consideră ca fiind invalid și nu va mai face parte din lista de clienți valabilă în cadrul celorlalte pagini.

Se observă faptul că se poate efectua căutarea atât în fiecare coloană, cât și în toate simultan. Căutarea este realizată pe client și se face prin intermediul componentei „PrimeNG” care ne pune la dispoziție această opțiune. Sortarea datelor și filtrarea acestora este posibilă datorită atributelor [sortable] și [filter] care au valoarea „true”. Paginarea este făcută astfel încât să apară zece înregistrări pe pagină, însă acest lucru se poate modifica având opțiuni de afișare a 5, 10 sau 20 de înregistrări.

Întrucât nu se dorește modificarea inline a datelor, atributul [editable] are valoarea „false”. Listarea clienților se face folosind componenta <p-dataTable> </p-dataTable> în interiorul căreia sunt definite proprietățile.

```

<p-dataTable [value]="clients" [editable]="false" [rows]="10" [paginator]="true" [pageLinks]="3" [rowsPer
  <p-column field="name" header="{{ 'Admin.Generic.Name' | translate }}" [editable]="false" sortable="t
  <p-column field="createdOn" header="{{ 'Admin.Generic.Date' | translate }}" [editable]="false" sortab
  <p-column field="valid" header="{{ 'Admin.Client.ClientValid' | translate }}" [editable]="false" sortabl
    <ng-template let-col let-valid="rowData" pTemplate="body">
      <p-checkbox [(ngModel)]="valid[col.field]" binary="true" [disabled]="true"></p-checkbox>
    </ng-template>
  </p-column>
  <p-column styleClass="col-button">
    <ng-template pTemplate="header">
      <button type="button" pButton icon="fa-pencil-square-o"></button>
    </ng-template>
    <ng-template let-client="rowData" pTemplate="body">
      <button class="reqView" type="button" pButton (click)="viewClient(client)" label="{{ 'Admi
    </ng-template>
  </p-column>
</p-dataTable>

```

Informațiile despre clienți sunt aduse în componentă, specificând prin intermediul atributului [value] lista ce conține datele. Lista este definită în Controllerul cu numele „Clients” căreia îi sunt atribuite datele apelând metoda „getAllClient” din serviciul „clientService”. Aceasta întoarce o listă de obiecte de tip Client. Mai departe, în atributul „value” este asociată această listă: [value]=”clients”.

Metoda „getAllClient” face un request de tip GET către web Api. Metoda are următoarea structură care cuprinde tipul de răspuns, mai exact „json” și token-ul care permite autentificarea. Rezultatul întors este o listă de obiecte de clienți de tip „Observable”.

```

getAllClient(): Observable<Client[]>{
  let headers = new Headers();
  headers.append('Content-Type', 'application/json');
  let authToken = localStorage.getItem('auth_token');
  headers.append('Authorization', `Bearer ${authToken}`);

  return this.http.get(this.baseUrl + "/client", {headers})
    .map(response => response.json())
    .catch(this.handleError);
}

```

„Observable” este un design pattern care definește o relație de dependență de 1 la n, între un obiect care joacă rolul de subiect și celelalte obiecte care sunt denumite „observatori”. În cazul în care obiectul-subiect își schimbă starea, celelalte obiecte care sunt dependente sunt notificate și actualizate în mod automat. Avantajul acestui pattern este acela de a facilita o comunicare mai bună între clase în funcție de relațiile dintre acestea.

Design Pattern-ul Observer se utilizează în cazul în care o clasă reprezintă componenta de bază, iar celelalte folosesc doar rezultatele acesteia. Obiectul-subiect, menține referințe către obiectele observatoare, însă nu știe ce funcție îndeplinesc acestea. [16]

Pe pagina de listare a clienților, există butonul „Adăugare client” care deschide un pop up în care se află un formular. Acesta conține câmpurile: „Nume”, „Descriere” și „Valid”. Câmpul Nume are atributul de „required”, în acest mod, numele fiind obligatoriu de completat. Tot pe acest câmp, am adăugat validări, astfel că butonul de „Salvare” nu poate fi accesat în cazul în care numele nu este completat sau nu conține lungimea minimă necesară, aici fiind de cel puțin două caractere. Pentru această opțiune, am folosit aliasul „clientName” căruia i-a fost asociată clasa „alert”, după cum se poate observa în figura de mai jos:

```
<div class="ui-grid-row">
  <div class="ui-grid-col-4">
    <label for="name">{{ 'Admin.Generic.Name' | translate }}</label>
  </div>
  <div class="ui-grid-col-8">
    <input pInputText required minlength="2" maxlength="250" id="name" name="clientName" [(ngModel)]="clientToCreate.name" #clientName="
  </div>
</div>

<div class="alert alert-danger" *ngIf="clientName.errors && (clientName.dirty || clientName.touched)">
  <small class="text-danger" [hidden]="!clientName.errors.required">
    {{ 'Admin.Validators.ClientName.Required' | translate }}
  </small>
  <small class="text-danger" [hidden]="!clientName.errors.minlength">
    {{ 'Admin.Validators.ClientName.MinLength' | translate }}
  </small>
  <small class="text-danger" [hidden]="!clientName.errors.maxlength">
    {{ 'Admin.Validators.ClientName.MaxLength' | translate }}
  </small>
</div>
```

Opțiunea de adăugare a fost realizată prin intermediul metodei „addNewClient” care trimite o cerere de tip POST către web Api. Spre deosebire de metoda „getAllClient” prezentată mai sus, „addNewClient” conține în plus variabila „body” care include json-ul convertit din obiectul model.

```
addNewClient(model: Client): Observable<WebAPIResult>{
  let body = JSON.stringify(model);
  let headers = new Headers({ 'Content-Type': 'application/json' });
  let authToken = localStorage.getItem('auth_token');
  headers.append('Authorization', `Bearer ${authToken}`);
  let options = new RequestOptions({ headers: headers });
  this: this
  return this.http.post(this.baseUrl + "/client", body, options)
    .map(res => res.json())
    .catch(this.handleError);
}
```


Cererea de tip POST este trimisă către Controllerul din web Api, unde se află acțiunea de adăugare. În exemplul de mai jos se poate observa metoda de creare a unui nou client:

```
// POST api/client
[HttpPost]
[Authorize]
[Authorize(Policy = "ApiUser")]
// References: 0 requests | 0 exceptions
public async Task<IActionResult> Post([FromBody] ClientDto model)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

    var currentUser = await _identityService.GetCurrentPersonIdentityAsync();
    if (currentUser == null)
        return new BadRequestObjectResult(new { Error = true, Message = await _localizationService.GetResource("BackEnd.User.InvalidUser") });

    if (currentUser.Person == null)
        return new BadRequestObjectResult(new { Error = true, Message = await _localizationService.GetResource("BackEnd.User.InvalidPerson") });

    var newClient = new Client
    {
        Name = model.Name,
        Description = model.Description,
        Valid = model.Valid,
        CreatedOn = DateTime.Now,
        ModifiedByOperator = null,
        ModifiedOn = null,
        CreatedByOperator = currentUser.Person.Id
    };

    await _clientService.CreateClientAsync(newClient, currentUser.Person);

    return new OkObjectResult(new { Message = await _localizationService.GetResource("BackEnd.Client.ClientCreated"), Success = true });
}
```

Atributul [HttpPost] indică faptul că cererea făcută de către metodă este de tip Post.

Atributul [Authorize] verifică dacă persoana care apelează metoda este autorizată.

Atributul [Authorize(Policy = "ApiUser")], unde „ApiUser” reprezintă rolul utilizatorului, verifică dacă utilizatorul care dorește să facă inserarea datelor clientului are permisiunea pentru acest lucru. Se observă că acțiunea primește ca parametru un obiect de tip DTO care permite transferul de informații dintre AngularJS și .NET Core. Se verifică dacă modelul este valid și în caz contrar se întoarce o eroare de tip „BadRequest”.

Așa cum îi spune și denumirea, variabila „currentUser” conține datele user-ului curent. Se realizează din nou o verificare a faptului că user-ul există. În continuare se face asocierea datelor, obiectul „newClient” primind valorile introduse de către utilizator prin intermediul browser-ului. Prin metoda „CreateClientAsync” care conține cei doi parametrii, din serviciul „ClientService” se realizează inserarea în baza de date. Se va returna un mesaj care anunță faptul că datele despre client au fost introduse cu succes.

Prin acționarea butonului „Vezi client” se deschide o nouă pagină care conține mai multe taburi realizate cu ajutorul componentei <p-tabPanel></p-tabPanel>. Pe lângă cele trei câmpuri menționate ulterior, este afișat câmpul „Data” reprezentând ziua în care clientul a fost

adăugat. Numele, Descrierea și opțiunea de Valid pot fi modificate, iar noile valori vor fi salvate în baza de date prin acționarea butonului „Editează client”.

Modificarea este efectuată cu ajutorul metodei „updateClient” din serviciul Angular, metodă asemanatoare cu „addNewClient”, singura diferență fiind reprezentată de rută.

În ceea ce privește metoda de „Update” din controllerul din .NET Core, aceasta aduce mai întâi obiectul curent, asociază proprietăților obiectului noile valori, după care obiectul este salvat. Aplicația conține o tabelă de evenimente ale utilizatorilor unde se stochează fiecare acțiune a utilizatorilor. Această tabelă se numește „History”.

Un exemplu de salvare a unei acțiuni făcute de către utilizator îl găsim chiar în metoda de update a entității „client” despre care am discutat mai sus. După ce clientul a fost actualizat, se creează un obiect nou de tip „History” care conține id-ul entității modificate, data modificării și id-ul operatorului curent. Acesta se inserează în baza de date pentru o mai bună trasabilitate a acțiunilor. Funcționalitatea este utilă pentru a depista de exemplu un utilizator care a făcut o acțiune greșită în aplicație.

```
historyChange = new History()
{
    FormTypeId = formType.Id,
    ModifiedOn = DateTime.Now,
    ModifiedByOperator = person.Id,
    EntityId = model.ClientId
};
```

Un alt tab prezent în această pagină este acela de „Listă contacte clienți” prin care afișez datele de contact ale clientului. Tabela corelată acestei pagini include câmpuri precum Nume, Descriere, Număr de telefon, Mobil, Email și Valid, cel dintâi fiind obligatoriu de completat.

Inclusiv aici, am folosit validări pentru toate câmpurile, mai puțin Descriere și Valid, asigurându-mă în acest mod că datele completate sunt valide (spre exemplu, mobilul conține zece cifre și emailul este de forma „ceva@ceva.com”). De asemenea, este posibilă adăugarea de date pentru respectivul client și modificarea acestora pe același principiu explicat anterior.

Următorul tab conține lista de proiecte alocată clientului curent, cu posibilitatea de adăugare și editare a datelor referitoare la proiect.

Ultimul tab denumit „Istoric Modificări” cuprinde istoricul modificărilor efectuate de către utilizatori. Acesta afișează numele utilizatorului, data și ora la care au fost făcute

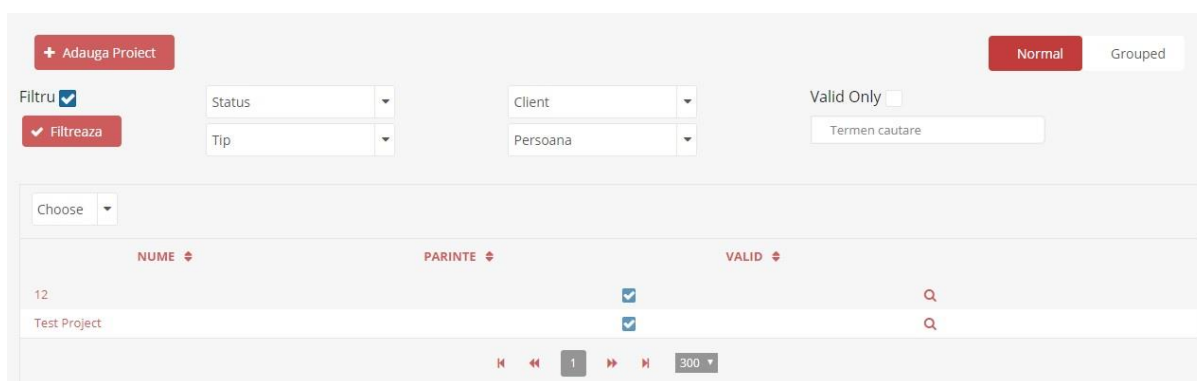
modificările, numele câmpurilor cărora le sunt atașate valorile vechi și cele noi pentru ca modificarea să poată fi ușor observată.

5.3.2 Nomenclator proiecte

Tabela denumită „Projects” conține următoarele câmpuri: Nume, Tip proiect, Status proiect, Client, Părinte, Culoare proiect, Descriere, Valid, Dată început estimată, Dată sfârșit estimată, Total ore estimate, Dată început actuală, Dată sfârșit actuală și Total ore actuale.

Pagina „Proiecte” conține lista proiectelor adăugate până în momentul de față. Se poate observa checkbox-ul denumit „Filtru” care permite filtrarea proiectelor în funcție de: status, tip, client, persoană sau dacă acestea sunt valide.

Filtrarea este posibilă prin intermediul variabilei „filterVisible” care este de tip „binary”, ceea ce înseamnă că nu poate lua valori decât de adevărat sau fals. În cazul în care checkbox-ul este bifat, dropdown-urile sunt vizibile pentru utilizator. Fiecare „div” care conține elementele după care se poate efectua filtrarea include atributul [hidden] care primește valoarea variabilei „filterVisible”. În cazul în care variabila are valoarea „false”, „div”-ul curent va fi ascuns.



The screenshot displays the 'Projects' management interface. At the top left is a red button labeled '+ Adauga Proiect'. To its right are two tabs: 'Normal' (selected) and 'Grouped'. Below these are filter controls: a 'Filtru' checkbox (checked), a red 'Filtreaza' button, and several dropdown menus for 'Status', 'Tip', 'Client', and 'Persoana'. A 'Valid Only' checkbox is also present, followed by a 'Termen cautare' input field. Below the filters is a table with columns 'NUME', 'PARINTE', and 'VALID'. The table contains one visible row for 'Test Project' with a status of 'Valid' (indicated by a blue checkmark). A pagination bar at the bottom shows '1' of 300 items.

Figura 10 – Pagina de listare a proiectelor

Butonul „Adăugare proiect” conduce utilizatorul către o nouă pagină în care sunt afișate câmpurile referitoare la proiect. Link-ul „Înapoi la lista de proiecte” duce utilizatorul către pagina precedentă. Am realizat acest lucru utilizând componenta „routerLink” care conține link-ul către pagina dorită.

Din câmpurile tabelului, primele cinci sunt obligatorii de completat, fapt pentru care în partea de HTML, am adăugat atributul [required] și de asemenea validări, astfel încât utilizatorul să fie atenționat printr-un mesaj că respectivul câmp este necesar de completat.

Pentru o siguranță sporită, datorită faptului că toate datele se află în interiorul unui formular, pe butonul de „Salvează proiect” am pus atributul [disabled] care nu permite acționarea butonului decât în momentul în care toate datele obligatorii au fost completate.

În ceea ce privește câmpul „Descriere”, am asociat un editor cu ajutorul componentei din „Pri-meNG” <p-editor> care permite formatarea textului. Editorul este vizibil în momentul în care se face click pe „Click to edit”, eveniment realizat cu ajutorul componentei „pInplaceDisplay” și <pInplaceContent>. Exemplul se poate observa în imaginea de mai jos:

```
<div class="form-group" >
  <label for="description" class="col-xs-12 col-sm-4 col-md-4 col-lg-3">{{'User.Projects.Description' | translate}}</label>
  <div class="col-xs-12 col-sm-12 col-md-10 col-lg-7">
    <p-inplace closable="closable">
      <span pInplaceDisplay>
        <span class="fa fa-pencil-square-o"></span><span style="margin-left:8px">Click to edit</span>
      </span>
      <span pInplaceContent>
        <p-editor name="description" [(ngModel)]="projectToCreate.description" [style]="{'height':'220px'}"></p-editor>
      </span>
    </p-inplace>
  </div>
</div>
```

O altă componentă este <p-colorPicker> folosită pentru a asocia fiecărui proiect câte o culoare. Utilizatorul vizualizează un editor de culori, selectează o culoare, iar componenta salvează în proprietatea „Culoare proiect” codul culorii care este de fapt un string.

Pentru datele estimate și cele actuale, am utilizat componenta <p-calendar> care afișează un calendar ce permite selectarea datei. Datorită atributelor [monthNavigator], [yearNavigator] care au valoarea „true” și a atributului „yearRange”, se poate naviga prin calendar, schimbând lunile sau anii inclusiv, anul minim care poate fi selectat este 1930 și anul maxim 2030 în acest caz. În această pagină, există de asemenea două input-uri denumite „Total Ore Estimate” și „Total Ore Actuale” pentru a ține evidența diferenței numărului de ore estimate pentru un proiect și numărul real al acestuia.

În ceea ce privește fluxul datelor, acesta este asemănător cu cel de la pagina de clienți. Atât pentru adăugare, cât și pentru editare, există metode în Controllere care apelează serviciile din Angular prin intermediul cărora se creează conexiunea cu Controllerele din web Api. Mai departe, serviciile din C# comunică cu baza de date folosind EntityFramework.

În adăugarea de proiect avem posibilitatea de a mapa tipuri și statusuri de proiecte. Maparea tipurilor de proiecte pe proiecte se realizează în metoda „submitProject()” care se află în Controllerul denumit „newProject”. Putem observa că obiectul de tip „NewProject” implementează clasa „project” și preia câmpurile „typeId” și „statusId”. În acestea sunt atribuite

valorile statusurilor și tipurilor selectate de utilizator. În dropdown-urile din care utilizatorul poate selecta, datele ajung prin intermediul unui request HTTP către web Api. Metodele din aplicația de Frontend care realizează aceste request-uri se numesc „getProjectStatuses()” și „getProjectTypes()”. Acestea reîntorc o listă de obiecte de tip „SelectedItem” care conțin proprietățile „label” și „value”, proprietăți ce ne ajută la crearea unui combo-box din care utilizatorul poate selecta opțiunea dorită.

Componenta care formează acest dropdown din care utilizatorul poate selecta face parte tot din suita de componente „PrimeNG” și se numeste <p-dropdown>. Ea preia listele de opțiuni ca și valoare a atributului „[options]”, iar prin intermediul atributului „ngModel” este specificată proprietatea pe care trebuie să o completeze în urma acțiunii user-ului.

Pe lângă acestea, există și metode prin intermediul cărora sunt aduse listele de Clienti, Tip proiect, Status proiect, Persoană pentru că sunt necesare atunci când dorim să efectuăm filtrarea în funcție de un anumit element. De asemenea, avem și metodele de „viewProject” și „addNewProject” care nu conțin altceva decât rutele către paginile respective. Numele funcțiilor sunt atribuite butoanelor de adăugare și editare prin evenimentul (click).

```
viewProject(projectListItem : ProjectListItem){
  this.router.navigate(['user/projects/edit/', projectListItem.id])
}

addNewProject(){
  this.router.navigate(['user/projects/new'])
}
```

5.3.3 Nomenclator tipuri de proiecte

Această pagină conține lista tipurilor de proiecte și este permisă adăugarea și modificarea acestora. Nomenclatorul conține următoarele câmpuri: „Nume” de tip string și „Valid” de tip boolean. Pagina de listare este sub formă de tabel și permite filtrare, căutare și paginare. Toate acestea sunt realizate cu ajutorul componentelor „PrimeNG”.

Nomenclatorul tipuri de proiecte a fost implementat din nevoia de a structura proiectele pe anumite tipuri din diferite domenii de activitate (ex. eCommerce, CRM, Contabilitate).

Principiile de funcționare ale operațiunilor de Create, Read, Update și Delete sunt aceleași ca cele prezentate mai sus. Metodele din aplicația client sunt următoarele: „getProjectType()”, „updateProjectType()”, „addNewProjectType()”. Toate acestea la rândul

lor se folosesc de servicii prin intermediul cărora creează request-uri HTTP pentru a trimite datele către aplicația server, cea din urmă inserând sau modificând datele din bază.

5.3.4 Nomenclator statusuri proiecte

Această pagină conține lista statusurilor de proiecte și este permisă adăugarea și modificarea acestora. Nomenclatorul conține următoarele câmpuri: „Nume” de tip string, „Tip” de tip string și „Valid” de tip boolean. Pagina de listare este sub formă de tabel și permite filtrare, căutare și paginare. Toate acestea sunt realizate cu ajutorul componentelor „PrimeNG”.

Nomenclatorul statusuri de proiect a fost implementat din nevoia de a gestiona și a prioritiza proiectele în vederea unui management cât mai eficient. Statusurile predefinite sunt următoarele:

- Presales: reprezintă analiza făcută de către departamentul de vânzări.
- To do: proiecte care așteaptă să fie începute
- Analiza: reprezintă pasul în care proiectul este analizat de către un arhitect
- Planificare: momentul în care se stabilesc etapele dezvoltării
- Dezvoltare: etapa în care se lucrează efectiv la proiect
- Completed: etapa în care proiectul este finalizat
- Cancelled: proiectul intră în acest status doar în cazul în care se decide, din diverse motive, renunțarea la dezvoltare.

DataKlas HR Lista Status Proiecte

NUME	TIP	VALID	
Presales	Project Waiting Statuses	<input checked="" type="checkbox"/>	<button>Vezi Status Proiect</button>
To do	Project Waiting Statuses	<input checked="" type="checkbox"/>	<button>Vezi Status Proiect</button>
Analiza	Project In Progress Statuses	<input checked="" type="checkbox"/>	<button>Vezi Status Proiect</button>
Planificare	Project In Progress Statuses	<input checked="" type="checkbox"/>	<button>Vezi Status Proiect</button>
Dezvoltare	Project In Progress Statuses	<input checked="" type="checkbox"/>	<button>Vezi Status Proiect</button>
Completed	Project Completed Statuses	<input checked="" type="checkbox"/>	<button>Vezi Status Proiect</button>
Cancelled	Project Completed Statuses	<input checked="" type="checkbox"/>	<button>Vezi Status Proiect</button>

Figura 11 – Lista de statusuri de proiecte

Principiile de funcționare ale operațiunilor de Create, Read, Update și Delete sunt aceleași ca cele prezentate mai sus. Metodele din aplicația client sunt următoarele: „getProjectStatus()”, „updateProjectStatus()”, „addNewProjectStatus()”. Toate acestea la rândul lor se folosesc de servicii prin intermediul cărora creează request-uri HTTP pentru a trimite datele către aplicația server, cea din urma inserând sau modificând datele din bază.

5.3.5 Nomenclator task-uri

Tabela „Taskuri” cuprinde următoarele coloane: Proiect, Nume, Descriere, Tip, Prioritate, Dată început estimată, Dată sfârșit estimată, Ore estimate, Dată început actuală, Dată sfârșit actuală, Ore efective și Task părinte.

La fel ca în cazul proiectelor, datele și numărul de ore estimate, respectiv actuale sunt folosite pentru a ține evidența diferenței între numărul de ore considerate a fi necesar pentru terminarea task-ului și numărul de ore real (durata de timp propriu-zisă a task-ului până când acesta a fost definitivat.)

În cazul în care un task este de o dimensiune mare, acesta poate fi împărțit în mai multe subtask-uri. Task-ul rădăcină prezintă task-ul părinte.

Avem posibilitatea de listare a task-urilor sub formă de tabel. Acestea pot fi căutate după „Nume” și sortate după „Status”, „Stare”, „Client” etc. Elementele din pagină au fost realizate ca în exemplele anterioare folosind componentele „PrimeNG”. Pagina de listare ne pune la dispoziție filtrarea task-urilor după Nume proiect, Nume client, Nume persoană, Tip task, Prioritate și Status. La bifarea opțiunii „Filtru”, toate aceste liste de opțiuni din care utilizatorul poate selecta filtrele dorite, vor apărea deasupra tabelului. Principiul de funcționare este similar cu cel din pagina de Proiecte.

Figura 12 – Filtrarea task-urilor

La apăsarea butonului „Adaugă task”, se deschide o noua pagină cu un formular care permite introducerea unui nou task. Câmpurile care necesită completarea sunt următoarele:

- Proiect: unde sunt puse la dispoziție toate proiectele definite anterior.
- Nume: reprezintă numele task-ului.
- Descriere: unde avem la dispoziție un editor de text.
- Tip: unde sunt puse la dispoziție tipurile de task-uri din nomenclatorul „Tipuri de task-uri”.
- Prioritate: de unde putem selecta opțiunile predefinite din nomenclatorul de priorități.

Cu excepția descrierii, toate câmpurile sunt obligatorii.

5.3.6 Nomenclator tipuri de task-uri

Această pagină conține lista tipurilor de task-uri și este permisă adăugarea și modificarea acestora. Nomenclatorul conține următoarele câmpuri: „Nume” de tip string și „Valid” de tip boolean. Pagina de listare este sub formă de tabel și permite filtrare, căutare și paginare. Toate acestea sunt realizate cu ajutorul componentelor „PrimeNG”.

Nomenclatorul tipuri de task-uri a fost implementat din nevoia de a organiza task-urile pe anumite categorii. Putem enumera câteva categorii de task-uri:

- Bug: reprezintă că task-ul respectiv se va ocupa de repararea unei probleme depistate în timpul testării.
- Implementare nouă: înseamnă că acest task va duce la crearea unei noi funcționalități în aplicație.
- Analiză: acțiunea determinată de acest task este de a pune cap la cap informațiile venite din specificații și de stabilire a pașilor de dezvoltare.
- Testare: se referă la testarea unei funcționalități.

Principiile de funcționare ale operațiunilor de Create, Read, Update și Delete sunt aceleași ca cele prezentate mai sus. Metodele din aplicația client sunt următoarele: „getTaskType()”, „updateTaskType()”, „addNewTaskType()”. Toate acestea la rândul lor se folosesc de servicii prin intermediul cărora creează request-uri HTTP pentru a trimite datele către aplicația server, cea din urmă inserând sau modificând datele din bază.

5.3.7 Nomenclator priorități ale task-urilor

Această pagină conține lista priorităților de task-uri și este permisă adăugarea și modificarea acestora. Nomenclatorul conține următoarele câmpuri: „Nume” de tip string și „Valid” de tip boolean. Pagina de listare este sub formă de tabel și permite filtrare, căutare și paginare. Toate acestea sunt realizate cu ajutorul componentelor „PrimeNG”.

Nomenclatorul de priorități ale task-urilor a fost implementat pentru a stabili ordinea în care se vor pune în dezvoltare diferite task-uri, unele fiind mai importante decât altele. Exemple de priorități:

- Critic: ceea ce înseamnă că necesitatea de rezolvare a task-ului este urgentă.
- Ridică
- Normală
- Scăzută
- Foarte scăzută

Principiile de funcționare ale operațiunilor de Create, Read, Update și Delete sunt aceleași ca cele prezentate mai sus. Metodele din aplicația client sunt următoarele: „getTaskPriority()”, „updateTaskPriority()”, „addNewTaskPriority()”. Toate acestea la rândul lor se folosesc de servicii prin intermediul cărora creează request-uri HTTP pentru a trimite datele către aplicația server, cea din urmă inserând sau modificând datele din baza.

5.3.8 Nomenclator de stări ale task-urilor

Această pagină conține lista stărilor de task-uri și este permisă adăugarea și modificarea acestora. Nomenclatorul conține următoarele câmpuri: „Nume” de tip string, „Cod” de tip string, „Ordine” de tip int și „Valid” de tip boolean. Pagina de listare este sub formă de tabel și permite filtrare, căutare și paginare. Toate acestea sunt realizate cu ajutorul componentelor „PrimeNG”.

Nomenclatorul de stări ale task-urilor a fost implementat pentru a depista stadiul în care se află un task. Exemple de stări ale task-urilor sunt următoarele:

- Backlog: reprezentând lista de task-uri ce vor fi făcute pe viitor, care nu au o prioritate foarte ridicată.
- Analiza: momentul în care task-ul este proiectat.
- Development: se asociază acest status în momentul în care task-ul a intrat în dezvoltare.

- Testing: momentul în care dezvoltările realizate de către programator încep a fi testate.
- Done: momentul în care task-ul a fost finalizat.

Principiile de funcționare ale operațiunilor de Create, Read, Update și Delete sunt aceleași ca cele prezentate mai sus. Metodele din aplicația client sunt următoarele: „getTaskStatus()”, „updateTaskStatus()”, „addNewTaskStatus()”. Toate acestea la rândul lor se folosesc de servicii prin intermediul cărora creează request-uri HTTP pentru a trimite datele către aplicația server, cea din urmă inserând sau modificând datele din bază.

5.4 Management zile de concediu

5.4.1 Zile libere legale

Nomenclatorul „Zile libere legale” permite utilizatorului să vizualizeze lista zilelor libere pe anul curent. Acesta conține câmpul „Nume” de tip string care cuprinde denumirea zilei libere (ex. Ziua Națională a României) și câmpul „Data” de tip datetime care indică ziua, luna și anul acesteia. Pagina de listare conține posibilitatea de căutare după fiecare câmp sau în toate coloanele.

Pagina permite adăugarea, modificarea sau ștergerea unei zile libere. Prin accesarea butonului „Adaugă o zi liberă legală nouă” se va deschide un pop up prin intermediul căruia se poate completa numele și selecta o zi din calendar. În componenta HTML, pe tag-ul destinat afișării pop up-ului, a fost adăugat atributul [(vizible)] care primește valoarea variabilei „displayDialogToCreateHoliday”. Aceasta este declarată în componenta TypeScript corespunzătoare celei de HTML și este de tip boolean. În momentul în care aceasta are valoarea „true”, pop up-ul este vizibil pentru utilizator.

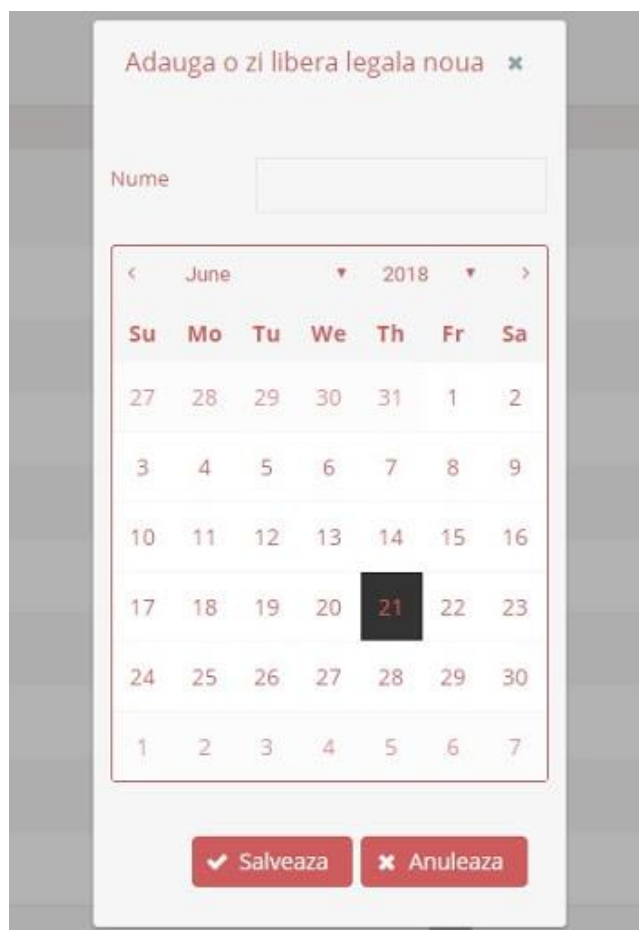


Figura 13- Pop up-ul pentru adăugarea unei noi zi libere legale

Dacă se face click pe iconița „X”, se va deschide un nou pop up care va întreba utilizatorul dacă este sigur că dorește ștergerea permanentă a zilei respective. Aceasta se va efectua doar în cazul în care user-ul va apăsa butonul „Continuă”, în caz contrar pop-ul se închide și se revine la pagina de listare.

Ștergerea propriu-zisă se realizează prin intermediul funcției „deletePublicHoliday” care apelează serviciul din Angular responsabil pentru această acțiune. Serviciul comunică cu Controllerul din web Api care conține metoda de delete. Cea din urmă, verifică dacă modelul este valid și dacă este, aduce în variabila „toDelete” modelul cu id-ul corespunzător, cu ajutorul serviciului „GetPublicHolidayById(model.Id)”. Ștergerea efectivă a datelor din baza de date se face apelând metoda „DeletePublicHolidayAsync” care primește ca parametru variabila declarata anterior „toDelete”, variabilă ce conține toate informațiile referitoare la ziua care se dorește a fi ștearsă. Apoi se va afișa un mesaj care înștiințează user-ul că ștergerea a fost făcută cu succes.

```

[HttpPost("delete")]
[ClaimRequirement(Constants.Strings.JwtClaimIdentifiers.Permission, nameof(ClaimRequirementEnum.AdminHierarchicalLevelUpdate))]
- references | O requests | O exceptions
public async Task<IActionResult> Delete([FromBody]PublicHolidayDto model)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }
    else
    {
        var toDelete = await _publicHolidayService.GetPublicHolidayById(model.Id);
        await _publicHolidayService.DeletePublicHolidayAsync(toDelete);
        return new OkObjectResult(new { Message = await _localizationService.GetResource("BackEnd.PublicHoliday.PublicHolidayDeleted"), Success = true
    }
}

```

5.4.2 Nomenclator cereri de concediu

Nomenclatorul „Cereri de concediu” afișează lista cererilor făcute de angajații companiei. Câmpurile vizibile sunt: Nume, Interval, Număr de zile, Firmă, Tip, Manager și Status manager. Este permisă căutarea atât în fiecare coloană cât și în toate coloanele simultan.

Tipul cererii de concediu este de tip string și are următoarele opțiuni:

- Concediu odihnă
- Concediu medical
- Concediu fără plată
- Ocazii speciale

În această pagină este posibilă „Căutarea complexă”. Prin bifarea acestei opțiuni, utilizatorului îi este permisă filtrarea în funcție de firma în cadrul căreia angajatul lucrează și selectarea unui interval de timp.

Cele doua calendare cu ajutorul cărora se poate selecta data de început și data de sfârșit au fost construite cu ajutorul componentei „PrimeNG” <p-calendar>. În Controller-ul din Angular, a fost implementată o funcție de conversie a datei selectate din tip „date” în tip „string”. Acest lucru este necesar pentru a putea transmite datele către web Api, întrucât C# nu știe să citească formatul de dată din Angular.

```

convertDateToString(date: Date){
    var curr_date = date.getDate();
    var curr_datestr = date.getDate().toString();

    if (curr_date < 10)
        curr_datestr = '0' + curr_date;

    var curr_month = date.getMonth() + 1; //Months are zero based
    var curr_monthstr = (date.getMonth() + 1).toString(); //Months are zero based

    if(curr_month < 10)
        curr_monthstr = '0' + curr_month;

    var curr_year = date.getFullYear();

    return curr_datestr + '.' + curr_monthstr + '.' + curr_year;
}

```

Filtrarea este realizată prin intermediul funcției „filterGridData()” care apelează serviciul denumit „filterHolidayRequest()” și care primește ca parametrii id-ul firmei, data de început și data de sfârșit. Acesta comunică cu metoda din Controllerul din web Api. În metoda de aici, se realizează din nou conversia din „string” în „DateTime”.

```

public async Task<ActionResult> Get(string startDate, string endDate, long firmId)
{
    DateTime? datestart = null;
    DateTime? dateEnd = null;
    if (DateTime.TryParseExact(startDate, "dd.'MM'.'yyyy", CultureInfo.InvariantCulture, DateTimeStyles.None, out DateTime filterStartDate))
        datestart = filterStartDate;

    if (DateTime.TryParseExact(endDate, "dd.'MM'.'yyyy", CultureInfo.InvariantCulture, DateTimeStyles.None, out DateTime filterEndDate))
        dateEnd = filterEndDate;

    var holidays = await _holidayRequestService.FilterHolidayRequests(datestart, dateEnd, firmId);
    var holidayDtos = _mapService.Map<List<HolidayRequestSmallDto>>(holidays);

    return new OkObjectResult(holidayDtos);
}

```

Se observă că este precizat formatul în care data se dorește a fi inserată. Clasa „CultureInfo” deține informații specifice culturii, cum ar fi limba asociată, țara/regiunea, calendar și convenții culturale. Această clasă oferă, de asemenea, acces la instanțe specifice: DateTimeFormatInfo, NumberFormatInfo, CompareInfo și TextInfo.

Butonul „Vezi cerere” conduce utilizatorul către o pagină care prezintă detaliile cererii de concediu. Datele de aici au ca atribut [disabled]=”true” care permite user-ului doar vizualizarea acestora și nu modificarea lor. Pentru fiecare dropdown din această pagină, în componenta Angular există câte o funcție care întoarce lista necesară asociată fiecărui dropdown.

6 STUDIUL DE CAZ

Primul pas pe care utilizatorul trebuie să îl îndeplinească este acela de a se loga. Acest lucru se realizează prin completarea câmpurilor ce pot fi observate în figura de mai jos și anume: selectarea firmei corespunzătoare angajatului, numele atribuit și parola.



The image shows a login interface for 'High-Tech systems & software'. At the top is the company logo, which consists of three vertical bars (grey, orange, red) followed by the text 'High-Tech' in a large, bold, sans-serif font, and 'systems & software' in a smaller, lighter font below it. Below the logo is a dropdown menu with 'HT55' selected. Underneath the dropdown are two text input fields: the first contains the username 'erikav' and the second contains a masked password represented by eight dots. At the bottom of the form is a grey button with the word 'LOGIN' in red capital letters.

Figura 14- Logare

Prima pagină afișată este cea care conține datele personale ale user-ului, câmpurile sunt de tip readonly, ceea ce înseamnă că acestea pot fi doar vizualizate și nu se pot actualiza direct din această pagină.

În partea stângă se poate observa meniul care cuprinde toate nomenclatoarele. Sunt cinci containere principale, mai exact: Time, Clients, Projects, Tasks și Administration, care la rândul lor includ alte nomenclatoare, ale căror iconițe sunt reprezentative. Deasupra meniului, este afișat un mesaj de întâmpinare adresat utilizatorului.

Figura 15 – Datele utilizatorului

Prin selectarea nomenclatorului „Clienți” se va afișa lista de clienți sub formă de tabel care are câmpurile: Nume, Dată (reprezentând data la care a fost inserat în baza de date clientul respectiv) și Valid. Apăsând săgețile care se află în dreptul fiecărui nume al coloanei, se poate realiza ordonarea înregistrărilor în funcție de câmpul dorit.

De asemenea este permisă filtrarea datelor. Căutarea se poate efectua în funcție de un anumit câmp sau în luându-se în considerare toate coloanele. Prin intermediul paginației, utilizatorul poate selecta numărul de înregistrări afișate în pagină, opțiunile fiind de cinci, zece sau douăzeci de rânduri.

DataKlas HR Lista clienti

NUME	DATA	CLIENT VALID	
Cauta dupa nume	Cauta dupa data	Client valid	
Client1 test sedinta2	23.02.2018	<input checked="" type="checkbox"/>	Vezi client
Client23	23.02.2018	<input checked="" type="checkbox"/>	Vezi client
Client3	23.02.2018	<input type="checkbox"/>	Vezi client
Client4	26.02.2018	<input checked="" type="checkbox"/>	Vezi client
Client5636	28.02.2018	<input checked="" type="checkbox"/>	Vezi client
Client Dragoș	09.03.2018	<input checked="" type="checkbox"/>	Vezi client
doc.ro	11.06.2018	<input checked="" type="checkbox"/>	Vezi client

1
10
5
15
20

Figura 16 – Lista clienților

Butonul „Adăugare client” va deschide un pop up ce va conține aceleași câmpuri menționate anterior și care îi va permite user-ului inserarea în baza de date a unui nou client. După inserare, noua înregistrare va fi afișată în lista de clienți.

Se poate observa că în cazul în care utilizatorul nu completează un câmp care este obligatoriu, va apărea un mesaj de informare. Mai mult decât atât, în cazul necompletării datelor obligatorii, butonul care permite salvarea nu poate fi accesat.

Figura 17 – Fereastră pentru adăugarea unui nou client

Prin apăsarea butonului „Vezi client”, utilizatorul este redirecționat către o pagină care conține detaliile referitoare la client și permite actualizarea acestora. Și în acest caz, lipsa datelor care sunt necesare va conduce către imposibilitatea de salvare a datelor editate ulterior.

DataKlas HR Editeaza client - Client1 test sedinta2 [Inapoi la lista de clienti](#)

Editeaza client Lista contacte client Lista proiecte Istoric Modificari

Nume

Numele este obligatoriu!

Descriere

detalii client

Data

23.02.2018

Valida ☒

+ Editeaza client

Figura 18 – Pagina cu detalii ale clientului

Deasupra tabelului, este expus numele paginii pe care utilizatorul se poziționează, mai exact „Editează client” în dreptul căruia se află numele clientului ale căror detalii urmează a fi actualizate. Pe același rând se situează link-ul care conduce user-ul înapoi către lista de clienți.

Tot în această pagină, prin selectarea unui alt tab din cele patru disponibile, se pot afișa informații referitoare la contactele clientului, proiectele acestuia și istoricul modificărilor. Cel din urmă include date despre câmpurile care au fost editate, data modificării și numele utilizatorului care a efectuat operațiunea.

Întrucât este o aplicație care se bazează pe nomenclatoare, deci pe stocarea și gestiunea datelor, principiul de funcționare al celorlalte pagini este asemănător cu cel prezentat în cadrul nomenclatorului „Clienți”.

7 CONCLUZIE

DataKlasHR este o soluție software pentru resurse umane care rezolvă într-un mod simplu plurivalența capitalului uman punând la dispoziție instrumente cu ajutorul cărora se poate analiza și urmări performanța angajaților, managementul competențelor și evoluția carierei în cadrul companiei.

Această aplicație promite menținerea și loializarea angajaților valoroși care pot face diferența între o companie și competitorii ei prin intermediul funcțiilor care sunt puse la dispoziție.

Cu ajutorul tehnologiilor noi utilizate pentru dezvoltarea soluției DataKlasHR și prin intermediul interfeței intuitive, utilizatorul poate vizualiza listele și datele clienților asociați firmei, dar și a proiectelor aflate în progres folosindu-se de nomenclatoarele pe care aplicația le pune la dispoziție.

Mai mult decât atât, se poate ține evidența zilelor libere legale și a zilelor de concediu consumate atât de către angajat, cât și de superiorii acestuia, lucru ce conduce la diminuarea timpului alocat aflării acestor informații și de asemenea a cheltielilor indirecte destinate acestui capitol.

8 DEZVOLTĂRI ULTERIOARE

Din punct de vedere al implementărilor ulterioare se propune îmbunătățirea fluxului de lucru și posibilitatea de adaptare în funcție de necesitățile companiei care utilizează aplicația și implementarea unui modul nou de gestionare a task-urilor folosind metoda Kanban. Această metodă este utilizată în managementul proiectelor pentru a monitoriza atribuirea task-urilor și stadiul de completare al acestora.

Metoda Kanban se bazează pe noțiunea de tabele, coloanele acestuia reprezentând stări ale sarcinilor. Trei posibile stări sunt următoarele: „de făcut”, „în curs de dezvoltare” și „finalizate”.

La începutul proiectului, toate task-urile se află pe prima coloană („de făcut” sau „To do”), în momentul în care sunt atribuite, acestea trec în statusul „în curs de dezvoltare” reprezentat de cea de-a doua coloană urmând să fie transferate în ultima coloană (finalizate) în momentul în care sunt terminate.

Doresc să implementez acest sistem într-o formă adaptată necesităților utilizatorului. O facilitate importantă, în acest sens, este posibilitatea de a transfera task-urile dintr-un status în altul cu evenimentul „drag and drop”.

9 DICȚIONAR DE TERMENI

- **Framework**

O platformă ce ajută la crearea rapidă de aplicații. Aceasta pune la dispoziție cod reutilizabil.

- **Frontend**

Parte a aplicației web cu care utilizatorul interacționează în mod direct. De obicei, aceasta este făcută folosind limbajul HTML și standardul CSS, dar și limbajul JavaScript pentru o mai bună interacțiune.

- **Backend**

Parte a aplicației web care procesează datele, conține logica de business și are legătură directă cu baza de date.

- **Api**

Colecție de metode care alcătuiesc o interfață prin intermediul căreia putem comunica din punct de vedere programatic cu aplicația.

- **Tag**

Cuvânt cheie ascuns în pagina web care definește modul în care browser-ul trebuie să afișeze și formateze conținutul.

- **Browser**

Aplicație software pentru căutarea, prezentarea și traversarea informațiilor din World Wide Web.

- **Bug**

Eroare de funcționalitate a aplicației din motive de proiectare sau dezvoltare.

- **Open-source**

Soft dezvoltat de către o comunitate la care poate participa oricine dorește. Codul este liber pentru modificări.

- **Drag and drop**

Termenul de „drag and drop” este utilizat pentru a descrie acțiunea de selectare și mutare a unui obiect pe ecran cu ajutorul pointer-ului.

- **Dropdown**

Meniu sub formă de listă ce permite utilizatorului să aleagă o opțiune.

- **Task**

Activitate (de obicei în domeniul tehnic) ce necesită să fie rezolvată începând cu o anumită dată și care trebuie finalizată până la un anumit moment prestabilit.

- **Nomenclator**

Listă de opțiuni predefinite ce sunt folosite în mai multe secțiuni dintr-o aplicație.

- **CRUD**

Abrevierea de la Create Read Update Delete. Reprezintă un set de acțiuni ce se pot efectua pe un set de date. Așa cum sugerează și denumirile acțiunilor „Create” reprezintă acțiunea de inserare a datelor, „Read” reprezintă acțiunea de citire a datelor, „Update” reprezintă acțiunea de actualizare și „Delete” reprezintă acțiunea de ștergere.

- **BackLog**

Listă de funcționalități acumulate în timp și care așteaptă să fie dezvoltate.

10 BIBLIOGRAFIE

- [1] Olivia Waring, „When was the Internet invented?” 2018 [Online]. Valabil: <https://metro.co.uk/2018/03/22/when-was-the-internet-invented-7408002/>
Accesat: 16.06.2018
- [2] <https://republica.ro/industria-it-din-romania-17-000-de-companii-pest-100-000-de-angajati-si-venituri-totale-de-4-8-miliarde>. Accesat: 16.06.2018
- [3] <https://www.softlead.ro/aplicatii-software/charisma-hcm.html> Accesat: 10.05.2018
- [4] <https://www.aceproject.com/>. Accesat: 10.05.2018
- [5] <https://www.trustradius.com/products/jira-software/reviews/pros-and-cons?f=50>.
Accesat: 20.06.2018
- [6] <https://reviews.financesonline.com/p/jira/> Accesat: 20.06.2018
- [7] <https://angular.io/guide/architecture> Accesat: 18.05.2018
- [8] Asim Hussain, „Angular from theory to practice”, ediția Kindel, 2017
- [9] <https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/> Accesat: 16.06.2018
- [10] https://www.tutorialspoint.com/ms_sql_server/index.htm Accesat: 17.06.2018
- [11] <https://www.tutorialrepublic.com/twitter-bootstrap-tutorial/bootstrap-introduction.php>
Accesat: 17.06.2018
- [12] Sarath Pillai, „HTTP (Hypertext Transfer Protocol) Request and Response” 2014
[Online]. Valabil: <https://www.slashroot.in/http-hypertext-transfer-protocol-request-and-response> Accesat: 20.05.2018
- [13] Margaret Rouse, „Object Relational Mapping” 2015 [Online].
Valabil: <https://searchwindevelopment.techtarget.com/definition/object-relational-mapping>
Accesat: 9.06.2018
- [14] <https://visualstudio.microsoft.com/downloads/> Accesat: 12.06.2018

[15] <https://code.visualstudio.com/docs/editor/whyvscode> Accesat: 12.06.2018

[16] Theodor Stoican, „Design Patterns” 2017. Valabil:

<http://elf.cs.pub.ro/poo/laboratoare/design-patterns> Accesat: 10.06.2018