

Cuprins

Cuprins.....	1
Lista figurilor.....	3
Lista tabelelor.....	4
Abrevieri.....	5
Rezumat în limba engleză.....	6
1. Planificarea activității.....	11
2. Stadiul actual.....	12
2.1. JavaScript.....	13
2.2. ReactJs.....	13
2.3. NodeJS.....	14
3. Fundamentare teoretică.....	15
3.1 Front-end.....	15
3.1.1 Ce este ReactJs și de ce îl folosim.....	15
3.1.2 Material-Ui.....	17
3.1.3 HTML.....	18
3.1.4 CSS.....	20
3.2 Back-end.....	22
3.2.1 NodeJs.....	22
3.2.2 ExpressJs.....	22
3.2.3 MongoDB.....	23
3.2.4 Mongoose.....	23
4. Implementarea soluției.....	25
4.1 Crearea serverului offline folosind NodeJs, Express.....	26
4.2 Baza de date pentru gestionarea conținutului.....	27
4.3 Crearea siteului web folosind React și Material-UI.....	32
4.3.1 Navigarea în aplicația web.....	32
4.3.2 Bara de navigație.....	34
4.3.3 Modalul pentru logare.....	36
4.3.4 Pagina de înregistrare.....	37
4.3.5 Pagina de introducere a modalității de plată.....	38
4.3.6 Pagina de creare a unei licitații.....	39
4.3.7 Pagina de informații.....	39
4.3.8 Pagina principală.....	39
4.3.9 Funcționalitatea de căutare.....	40
4.4 Stilizarea paginilor web.....	41
5 Rezultate experimentale.....	43
6 Concluzii.....	48
7 Bibliografie.....	49
8 Anexe.....	50
8.1 Codul folosit pentru realizarea componentei autovehiculului.....	50
8.2 Codul folosit pentru realizarea paginii autovehiculului.....	50
8.3 Codul folosit pentru realizarea paginii principale.....	52
8.4 Codul folosit pentru realizarea paginii barei de de navigație.....	53
8.5 Codul folosit pentru pagina setării metodei de plată.....	57
8.6 Codul folosit pentru realizarea modalului de logare.....	59
8.7 Codul folosit pentru realizarea pagina de înregistrare.....	62

8.8 Codul folosit pentru realizarea paginii de crearea a licitațiilor.....	62
8.9 Codul folosit pentru realizarea afișarea întregii aplicații.....	64
8.10 Codul folosit pentru rutarea paginilor.....	67
8.11 Schema pentru autovehicule.....	67
8.12 Schema pentru utilizatori.....	67
8.13 Codul pentru crearea rutelor specifice autovehiculelor.....	68
8.14 Codul pentru crearea rutelor specifice utilizatorilor.....	69
8.15 Codul pentru crearea serverului offline.....	71
8.16 Codul pentru conectarea bazei de date la serverul offline.....	71

Lista figurilor

Figure 1.Home page.....	8
Figure 2.Search functionality.....	9
Figura 1.Structura unui document HTML.....	17
Figura 2.Structura unui element HTML.....	18
Figura 3.Alocarea unui atribut unui element HTML.....	19
Figura 4. Diagrama arhitecturii aplicației.....	25
Figura 5.Pașii pentru crearea unei licitații.....	26
Figura 6.Conectarea bazei de date la serverul offline.....	27
Figura 7.Documentul utilizatorilor.....	27
Figura 8.Documentul autovehiculelor.....	28
Figura 9.Ruta specifică a utilizatorilor.....	28
Figura 10.Ruta specifică a autovehiculelor.....	28
Figura 11.Endpoint pentru întregul document al utilizatorilor.....	29
Figura 12.Metoda de criptare a parolei	29
Figura 13. Verficarea numelor și mailurilor deja existe în baza de date.....	29
Figura 14.Metoda de salvare a unui nou uiltizator în baza de date.....	30
Figura 15.Metoda de verificare pentru logare.....	30
Figura 16.Endpoint-ul pentru salvarea metodei de plată.....	31
Figura 17.Salvarea pozelor in fișier cu extensia .jpg.....	31
Figura 18.Salvarea unei noi licitații.....	32
Figura 19.Ruta pentru fișierele statice.....	32
Figura 20.Componenta responsabilă pentru navigare.....	33
Figura 21 .Rutele pentru întreaga aplicație.....	33
Figura 22.Crearea barei de navigație pentru ecrane de dimensiuni mari.....	34
Figura 23.Bara de navigație pentru ecran de dimensiuni mari.....	34
Figura 24.Crearea barei de navigație pentru dispozitive mobile.....	35
Figura 25.Meniul de navigare pentru dispozitivele mobile.....	35
Figura 26.Crearea modalului pentru logare.....	36
Figura 27.Cererea de tip HTTP POST pentru logare.....	36
Figura 28.Modalul pentru logare.....	37
Figura 29.Pagina de crearea contului	38
Figura 30.Pagina pentru salvarea modalității de plată	38
Figura 31.Componenta pentru încărcarea imaginilor.....	39
Figura 32.Parcurgerea tuturor licitațiilor.....	40
Figura 33.Componenta unui autovehicul.....	40
Figura 34.Metoda de filtrare a datelor.....	41
Figura 35.Definirea unei clase cu atribute în CSS.....	42
Figura 36.Pagina principală Figura	43
Figura 37.Introducerea unui nume de utilizator greșit.....	44
Figura 38.Introducerea unui cont deja existent în baza de date.....	44
Figura 39.Funcționalitatea de căutare.....	45
Figura 40.Plasarea unei oferte.....	45
Figura 41.Verificarea metodei de adăugarea a minutelor.....	46
Figura 42.Serverul offline.....	46
Figura 43.Salvarea pozelor.....	47

Lista tabelelor

Tabel 1. Proprietăți compenenta Textfield din Material-Ui.....	19
Tabel 2. Tag-uri de marcare a textului în HTML[3].....	21
Tabel 3. Tipuri de date în Mongoose.....	25
Tabel 4. Operații uzuale în Mongoose.....	25

Abrevieri

1. HTML – limbaj de marcare pentru creare paginilor web(HyperText Markup Language);
2. URL – localizarea și identificarea resurselor pe Internet(Localizator uniform resurse);
3. CSS – standard pentru formatarea elementelor HTML(Cascading Style Sheets);
4. HTTP – protocol pentru transferul informației(HyperText Transfer Protocol);

Rezumat în limba engleză

Introduction

Online auctions represent an easy way to shop for valuable goods. There are hundreds of popular auction websites, especially in the United States and western European countries, such as Ebay, Copart, Sotheby's and many more. The purpose of this paper is to create a website for auto auctions in Romania, in order to promote this process of buying and selling goods. My work is different from the current national websites in the fact that it contains a timer that is added after a bid has been made. The information displayed must be organized in chronological order. The process of achieving the main goals of this paper involves adding features such as search function, and a functionality which allows the user to create an account and upload car pictures and information.

By opting for an online car auction, buyers can get cars at a better price, a bigger variety of cars to choose from, and could get a good deal on a repossessed car. As there are different types of auctions (English, Dutch, open type), I have chosen the open auction or real time bidding (RTB) for my website. Therefore, bidders place a bid against each other by offering a higher price than the previous one, and the winner is considered to be the bidder that offered the highest price, after the 30 minutes timer has passed. Buyers can place a bid more than once in the same auction, can register themselves in any available auction, and can view the latest bid placed on any car. An auction is considered complete when the timer is done.

Project Objectives

The main objective of this bachelor degree is to start an auction website for cars, so that the process of buying and selling cars is eased. The process of creating such a website involves adding the search function, login, sign up.

In order to start an auction or participate in one, users need to create an account. The login and register functions both contain an email/username and password box. However, in order to place a bid, the user must enter their card details, as the payment can only be done online. The seller must provide useful information about the car, such as pictures, the year of production, and model of the car, that would further be verified by a site administrator. Here, the search function is helpful as users can search their desired car by typing in the car model.

Technologies used

In order to achieve the goals established, I used the JavaScript scripting language and Material UI library. The reasons why I chose to work with them is because of they are easy to work with and very popular amongst software engineers.

JavaScript is used for creating web pages as a client scripting language, object oriented, which can do everything related to webpage manipulation, user interaction, and the webserver. By using JavaScript, I am able to modify styles, add new HTML to the page, change existing content, set cookies, show messages to the client and react to user interactions.

My work is split into two main parts: back end and front-end. Front-end represents the information the user interacts with. It is established on Material UI and Cascading Style Sheets programming languages. Material UI is the most efficient tool used for building an application by adding designs. It can be used with all JavaScript frameworks and libraries, such as ReactJS that I have used in creating my web page, to give a good-looking aspect and make it more responsive.

The last part is the back end, which is used to add functionalities that make the website easy to work with.

The backend part refers to the server side of development, focused mainly on how the site works. It is made out of three parts: a server, a database and an application. In this part, I am responsible of making changes and updates in addition to checking the site functionality. The code written is what communicates the database information to the browser. The backend contains anything that cannot be seen, such as databases and servers.

Node uses an event driven non-blocking model. Therefore, this means that Node is realised to handle asynchronous JavaScript code to perform reading and writing to the file system, handling requests as a web server and handling connections to database servers. Node uses a callback-based system. If an error occurs, the error argument will be an error object, and if no error occurs it will be null.

Project implementation

The main objective of this paper is to create an auction website for vehicles. In order to achieve this, I used the programming language JavaScript and the library React and Material UI, which helped me create a platform easy to use. The user can navigate the website easily due to the buttons I created. Moreover, if a user wishes to search for a specific car, this can be done the search functionality which is easy to use. Sign up is another functionality that helps the user to sell his car.

Site design specifications:

1. An auction site easy to manage
2. A dynamic application that allows advanced functionalities
3. Search functionality
4. A feature that allows the user to create an account and send data files
5. Creating a server for the site's functionalities

For most of the functionalities of the website I needed an offline server. I used NodeJS, which is an open source back end JavaScript execution environment that can run JavaScript code outside of a web browser and ExpressJS which is a framework for Node that helps structure the application on the server side. They can only be used after installing the Node program, available for free on the official website of the manufacturer. Next, you need to install the express framework on the command line, as follows in the following example:

`Npm install express --save`

The first step in creating an offline server requires importing the library into the file, such as:

```
Const express = require ('express');
const app = express ();
```

The second step is to use the Listen method in the Express library, in which we mention the desired access port.

For content management I chose to use the MongoDB database which is oriented on JSON documents. The reason I chose this database is that it supports various types of data. With the help of the Mongoose library we made the connection of the database with the offline server and two data schemes for the database: cars and user, which makes the interaction with the database easier.

Next I had to create different ENDPOINTS APIs needed for all the functionalities of the web application. An API (Application Programming Interface) allows communication between two systems, which provides a language for the interaction of the two. Each API has its own way of transmitting data, which is posted on the Internet. APIs are divided into two broad categories: SOAP and REST, both of which are used to access web services. In my application I chose to use

the REST type, which uses simple methods, using URLs in most cases, to receive and transmit information. REST uses four HTTP 1.1 methods such as: GET, POST, PUT and DELETE. The Endpoint API is simply one end of a communication channel, and each endpoint is the location from which an API can access the information it needs.

In my application I created two specific routes, one for users and one for all the auctions created. The URL of the server is <https://localhost:30001>, and if we add “/users” or “/cars” at the end it will upload the specific methods I have created. As I mentioned before, for the implementation of the website, we used the JavaScript language in correspondence with the React and Material UI libraries. I chose Material UI to make the graphic part because it already has predefined components that no longer require the design. Also, the Material-UI library allows the import and use of components in order to create a simple interface with a pleasant appearance, but also adaptable to resizing the window. In this way, we reduced the time and speeded up the realization process, for the simple fact that it does not require the writing from 0 of each element of the site.

In order to be able to create a web application with the React library, it is necessary first of all to install it through the command line: `npm create-react-app my-app` (the name of the application and the file).

This command sets a development environment in which the latest version of the JavaScript language can be used, together with all its features. Next, I had to install the MaterialUI framework in the React application file using the command in the terminal line: `npm install @material-ui / core`.

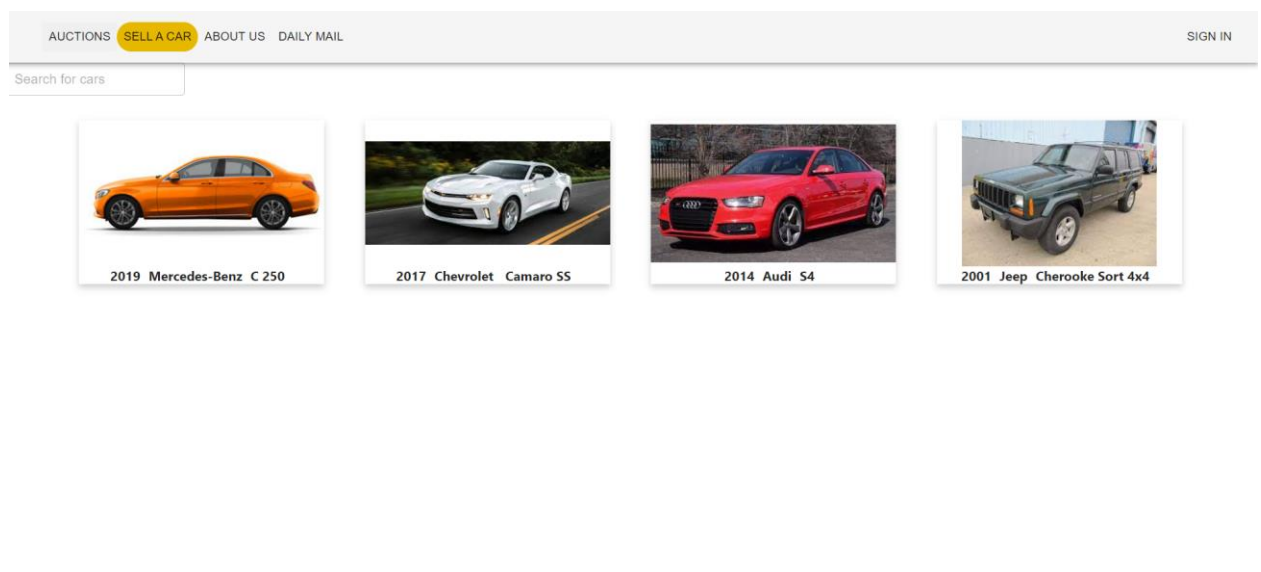


Figure 1.Home Page

The navigation bar contains the following buttons:

- Auctions; which refers to the main page
- Car sell; the page where the auctions are created
- About us; the page that contains information about the platform
- Daily mail; subscribe button
- Sign in; appears only when the user is not logged in, and when the user is logged in the account button appears, where he can change his account settings

The Sign In modal consist of a component that opens when you press the “Sign In” button on the navigation bar, if the user has not previously logged in. The component contains two data entry fields, one for the username and the other for the password. For these fields I used the predefined component of the MaterialUI `<TextField>` framework with the type = “password”

setting, for the field assigned to the password, making it not visible. user in variables specific to each field.

The payment method page allows the user to save the payment information, which can only be done by cars. The page contains all the necessary fields to complete the payment, saved with the `onChangeHandler()` method, which sends all the data introduced. The register to bid button makes a HTTP Put request on the path `http: // localhost: 3001 / users / settings` using on the offline server the `find ()` method that finds the user logged on to the site and saves the information entered correctly in the Users document of the database.

This Sell a Car component contains several data entry fields, namely all the information about the car that the user wants to sell. The upload button is in the `<form>` tag to which I have added the option to receive multipart data. Inside this element I also added an element of type `<input>`, which I set to receive several jpg files. With the help of the `onChange` method I managed to save the picture files in an object.

The HomePage component makes an HTTP Get request to the path on the offline server `http: // localhost: 3001 / cars` in the library-specific method `React componentDidMount ()` which is called before the page content is displayed, asking the server for all data saved in the Cars database as a JSON document. The component displays all auctions using the JavaScript language's `.map ()` method, which traverses the JSON document item by item. This method displays the Car component that receives the properties of each item in the received document

One of the objectives of the paper is to achieve a function that allows finding the desired vehicles to the user through the search component. To do this, we used the `<TextField>` component imported from `MaterialUI` to which we added the `onChange` method, which targets any change in the component, in this case when a user presses the keyboard. This method mentioned above calls an `updateInput ()` function that scans the object in the database and filters all data according to the written name

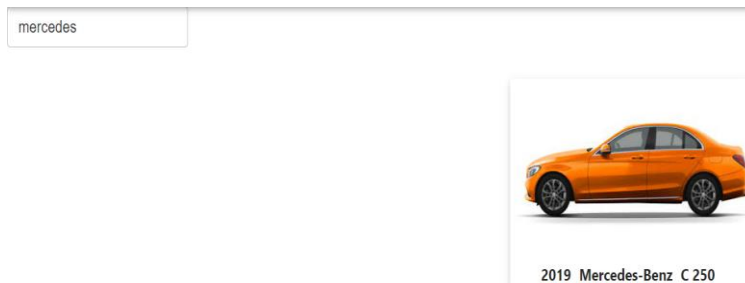


Figure 2.Search functionality

Conclusions

The website made by me is designed to conduct online auctions for vehicles, a simple and popular way of buying or selling a car online. My work is different from the existing auction website by the fact that after placing a bid, 30 minutes are added to the auction period time. In order to achieve this, I used Material UI and JavaScript to give a nice look to the website, and to add the functionalities we needed for an offline server, created by using the ExpressJs package, and to save the data we used the MongoDB database. Therefore, this project allows users to buy or bid for any type of car easily, thanks to the buttons that facilitate navigation. In this sense, the user has the opportunity to search the site for the desired car with the help of the search functionality which is very easy to use.

If a user wants to enter an auction, he must create an account and press the Place bid button within the pages of the vehicle. Another facility is to offer a car for auction, the user having to create an account as well, and to upload the necessary information and positions.

This project was carried out in several stages, trying different solutions and using several programming languages:

1. Creating a site using React, MaterialUI and CSS
2. Creating the ExpressJs offline server
3. Creating the database
4. Creating different routes in the offline server to add functionality
5. Creating the various components needed for the site
6. Functionality testing

In conclusion, by following the steps previously, the objectives of the work were met, and the result is as originally proposed.

1 Planificarea activității

Specificații proiect	Perioada	Durata(zile)
Stabilire temă proiect	20.01.2021 – 21.01.2021	1
Studiu bibliografic	21.01.2021 – 14.04.2021	83
Familiarizare cu limbajul JavaScript	22.03.2021 – 03.04.2021	12
Familiarizare cu Material-Ui	03.04.2021 – 06.04.2021	3
Realizare server offline+baza de date	06.04.2021-13.04.2021	7
Realizare pagina de front-end	06.04.2019 – 24.04.2021	18
Implementarea soluției	18.04.2021 – 25.05.2021	37
Scrierea documentației	04.04.2021 – 10.06.2021	67
Printare și legare	11.07.2021 – 12.07.2021	1
Predare proiect	14.07.2021 – 16.07. 2021	1

2. Stadiul actual

Scopul acestei lucrări de licență este realizarea unui site de licitații online pentru autovehicule bazat pe limbajul de programare JavaScript, librăriile React, Node JS, și baza de date MongoDB. Motivul principal al realizării acestui proiect este facilitarea participării la licitații într-un mod ușor și rapid prin intermediul internetului. Necesitatea acestei platforme vine în urma cererii, astfel încât licitațiile reprezintă un mod popular prin care mediul afacerilor se dezvoltă, reducând costurile de întreținere și crescând posibilitatea câștigurilor rapide din vânzarea în timp record de către oricine.

Vânzarea bunurilor în mediul online a cunoscut o creștere rapidă odată cu dezvoltarea internetului, care s-a datorat în primul rând faptului că accesul la documentația protocoalelor obligatorii a fost întotdeauna liber și gratuit. În anul 1969, informaticianul Steven Crocker a inițiat o serie de note de cercetare denumite Request for Comments (RFC), devenite cu timpul accesibile pentru oricine în mod gratuit pe internet. Însă cel mai important moment în dezvoltarea internetului s-a întâmplat în 1989 [7], când programatorul Tim Berners Lee a dezvoltat primul prototip al World Wide Web (WWW). Aplicațiile care pot fi găsite pe internet diferă de la cele în care sunt afișate mai multe informații sau cele în care imaginile și sunetele domină, apoi poșta electronică cum ar fi e-mail, chat, transferul de fișiere și informații, video, telefonie, telefonie live cu imagine prin internet, comerțul online (e-commerce), televiziune prin internet, surse media de răspândire a știrilor, muzică, jocuri, și operații bancare (internet banking).

Pentru ca toate aceste aplicații să funcționeze este necesar doar un singur program multifuncțional numit browser. Acestea sunt: Google Chrome, Internet Explorer, Mozilla Firefox, Opera și așa mai departe. Odată cu creșterea funcționalității internetului, optimizarea unei pagini web (SEO) a unui site a devenit un serviciu oferit de companii și corporații. Acest serviciu constă în totalitatea tehnicilor prin care un site web este propulsat în topul listei de rezultate date de un motor de căutare pentru diferite cuvinte-cheie. În vremurile în care internetul se afla la început, motoarele de căutare afișau în urma unei căutări, pagini în care cuvântul cheie se afla în descrierea acestora, iar proprietarii de site-uri (webmasterii), au profitat de acest lucru. Astfel, motoarele de căutare conțin algoritmi mult mai exacți, care includ:

- Textul din titlu
- Numele domeniului
- Elemente HTML
- Titlul link-ului
- Frecvența cuvântului cheie, atât în pagină cât și global
- Atributele pentru imagini ALT
- Numărul de hyperlink-uri de la și spre o pagină.

Așadar, licitațiile permit mediului de afaceri să cumpere și să vândă într-un mod securizat orice tip de bunuri. Licitațiile sunt vizibile în toate colțurile țării, iar ofertanții și clienții sunt salvați într-o bază de date prin logare, aceștia având posibilitatea de a vinde sau cumpăra într-un mod rapid. Tipurile de licitație online variază de la licitația de tip olandez, licitația inversă, licitația directă cu preț rezervat și licitația deschisă. Platforma la care voi lucra va cuprinde licitația de tip deschisă, în care:

1. Participanții licitează unul împotriva celuilalt într-un mod deschis, fiecare oferind suma cea mai mare decât suma precedentă, câștigătorul fiind acela care a oferit suma cea mai mare.
2. Aceștia au posibilitatea de a miza de mai multe ori în cadrul aceleiași licitații, cu condiția ca și cronometrul să nu fi ajuns la 0.
3. Utilizatorii se pot înscrie la orice licitație disponibilă în cadrul platformei, și pot vedea ultima sumă licitată.
4. O licitație este considerată finalizată și nu mai poate fi vizualizată de către niciun alt utilizator în momentul în care timpul de așteptare este finalizat.

Funcționalitățile considerate în realizarea acestei lucrări sunt funcția de search, logare, crearea contului, timpul alocat licitației (cronometru). Astfel încât, utilizatorii vor trebui în primul rând să își creeze un cont (email, username, parolă), după care vor avea posibilitatea de a posta timpul acordat licitației, și de a încărca informații și poze cu autovehiculul care vor fi verificate înaintea începerii licitației de către administratorul paginii. Prin funcția search, potențialii cumpărători au posibilitatea de a căuta mașina după modelul sau marca dorită.

În momentul de față în România există o asemenea platforma numită [DirektCar](#), care oferă posibilitatea deținătorilor de mașini de a pune mașina spre licitație, iar pe plan internațional un astfel de site îl reprezintă [Autorola.eu](#). Prin urmare, lucrarea mea aduce un adaos platformelor deja existente și se diferențiază de acestea prin câteva considerente. Când vine vorba de platforma mea, în momentul în care o licitație este plasată, automat se mai adaugă 30 de minute cronometrului, dând șansa oricărui posibil doritor de a contraoferta suma. În plus, platforma va fi realizată doar pe plan național, cu toate că exista unele platforme unde licitația este posibilă și în plan internațional.

2.1. JavaScript

Limbajul de programare JavaScript a apărut în anul 1996 dezvoltată de către firma Netscape, de tip script cu obiecte (object-oriented) bazat pe conceptul prototipurilor, fiind folosit mai ales pentru introducerea unor funcționalități în paginile web. Dezavantajul acestui limbaj constă în faptul că rularea durează mai mult deoarece comenzile vor fi citite de navigatorul web și procesate atunci când user-ul apelează aceste funcții. Nucleul JavaScript este format dintr-un set principal de elemente de limbaj cum ar fi operatori, instrucțiuni și structuri de control, precum și un set de obiecte predefinite cum ar fi Array, Date și Math. Apariția acestui limbaj de programare a venit din nevoia ca inteligență și logica să fie prezente și pe partea de client, nu doar pe partea de server (server-side).

Client-side JavaScript [8] permite unei aplicații să plaseze elemente într-o formă HTML și să răspundă evenimentelor generate de utilizator, oferite de navigarea în sine și completarea unui formular. Așadar, client-side extinde nucleul JavaScript furnizând obiecte pentru a controla un browser, navigator, și modelul documentului (DOM- Document Object Model). De asemenea, Server-side JavaScript permite unei aplicații să comunice cu o bază de date relațională, efectuând operații asupra unor fișiere pe server sau asigurând continuitatea informației de la o deschidere la alta a aplicației.

SSJS (Server Side JavaScript) este o versiune extinsă a limbajului JavaScript care permite părții de backend să acceseze bazele de date, fișierele și serverele. Partea de server JavaScript, este un cod JavaScript care rulează peste resursele locale ale unui server, de exemplu Node.JS. Cu ajutorul Node.JS putem scrie codul JavaScript să programăm partea de server, iar acel cod poate fi văzut ca și un simplu C#, C, sau orice alt limbaj pe partea de server. Mai mult decât atât, cu codul pe partea de server, putem trimite JavaScript pe partea de client, însă codul scris pe partea de client este restricționată de resursele clienților, precum permisiuni și puterea de calcul. În acest sens, partea de client scrisă în JavaScript nu poate accesa hard disk-ul clientului, iar cu partea de server poți accesa fără nicio problemă. Primul și cel mai important avantaj de a scrie pe partea de server este posibilitatea de a customiza răspunsul, pe baza cererii clientului, drepturile de acces și interogările în bazele de date.

2.2. REACT JS.

ReactJS [1] a fost creat de un inginer software de la Facebook, Jordan Walke, care a fost implementat în news feed în 2011. De atunci, sute de mii de website-uri și aplicații, precum Instagram sunt susținute de către această librărie. React, considerat un JavaScript framework, este o bibliotecă de tip open source, folosită pentru interfețe web mari, complexe, cât și pentru

aplicații single-paged. Principalul motiv pentru care am ales acest limbaj este modul simplu de operare. În plus, această librarie aduce un beneficiu prin prezentarea sa separată de semantica sa, oferind posibilitatea de a construi o bibliotecă de componente și interfețe din componente.

2.3. NODEJS

Creată în 2009, Node.js [5] este un ecosistem de JavaScript de tip open-source, cross-platform, back-end care rulează pe motorul V8 de la Chrome și execută cod JavaScript în afara unui browser web. Cu ajutorul Node.js se pot scrie instrumente de linie de comandă și pentru scripturi de server, mai precis rularea de scripturi de pe server pentru a produce conținut dinamic paginii web înainte ca pagina web să fie trimisă în browserul web al utilizatorului. Cel mai popular pachet îl constituie ExpressJS, deoarece oferă posibilitatea creării unei rute HTTP foarte ușor.

ExpressJs este un cadru de aplicații web minimal și flexibil care facilitează dezvoltarea aplicațiilor web și mobile. Aplicațiile web sunt ajutate de dezvoltarea rapidă bazată pe noduri. Principalele motive pentru care am ales acest ecosistem sunt:

- a) Redarea dinamică a paginilor HTML pe baza transmiterii argumentelor la șabloane
- b) Configurarea middlewares pentru a răspunde solicitărilor HTTP
- c) Definirea tabelor de rutare care ajută la efectuarea diferitelor acțiuni pe baza metodei HTTP și URL

3. Fundamentare teoretică

Realizarea paginii web necesită împărțirea muncii în două parti. Prima, care deja există și nu este atât de dificilă constă în partea de front-end, unde informația este deja disponibilă utilizatorului cu care interacționează. Aceasta se bazează pe limbajele de programare web JavaScript, HTML și CSS. Cea de-a doua parte este cea de back-end, care necesită o varietate de cunoștințe mult mai avansate, fiind mai complexă, care ajută la adăugarea unor funcționalități noi. În final, cele două părți trebuie legate între ele prin anumite procedee.

3.1 Front-end

Front-end este modul prin care producem limbajele HTML, CSS și JavaScript pentru un site web sau aplicație web, pentru a permite utilizatorului interacțiunea cu ele. Prin crearea design-ului unui site, utilizatorii vizualizează informația într-un format relevant și accesibil atunci când accesează pagina. Ca urmare, este necesară afișarea corectă a site-ului în browsere diferite (cross-browser), sisteme de operare (cross-platform) și dispozitive (cross-device), deoarece utilizatorii folosesc o gamă largă de dispozitive care variază în rezoluție sau mărimea ecranului.

3.1.1 Ce este ReactJS și de ce îl folosim?

ReactJS[1] permite crearea unor interfețe (UI) într-un timp mult mai scurt, datorită abilității bibliotecii de a refolosi și componente din altă interfață. Poate fi folosit în dezvoltarea unei interfețe a unei aplicații atât într-o aplicație de tip web, dar și într-o aplicație de tip Mobile. ReactJS oferă soluții persistente numeroaselor probleme front-end, permițând crearea unei aplicații web dinamice și interactive cu ușurință.

Am ales acest cadru de dezvoltare front-end deoarece:

1. Asigură crearea cu ușurință a unor aplicații dinamice deoarece necesită mai puțin cod scris și multă funcționalitate decât JavaScript, unde totul poate deveni foarte complex.
2. Performanță: React utilizează Virtual DOM, în acest fel creând aplicații cu rapiditate. În acest sens, Virtual DOM compară statutul precedent al componentelor și actualizează doar funcționalitățile aflate în Real DOM care au fost schimbate, în loc să actualizeze toate componentele din nou, cum se întâmplă cu aplicațiile web convenționale.
3. Componente reutilizabile: Componentele reprezintă construcția de bază a oricărei aplicații React, și o singură aplicație consistă din mai multe componente. Acestea au propria logică și controale, care pot fi refolosite în cadrul aplicației, care în schimb reduce dramatic timpul folosit în realizarea aplicației.
4. Unelte dedicate pentru depanare: Facebook a lansat o extensie Chrome care poate să indentifice erorile în componentele React.
5. Poate fi folosită atât în crearea aplicațiilor web cât și pentru aplicațiile mobile. Cu ajutorul React Native, derivat din React, este foarte populară în realizarea aplicațiilor mobile.
6. Ușor de învățat: deoarece combină noțiuni de bază HTML și concepte JavaScript cu câteva adăugări benefice.
7. Flux de date nedirecțional: din moment ce datele se îndreaptă într-o singură direcție, devine mult mai ușor să găsești erori și să afli de unde vine problema într-o aplicație.

React folosește VDOM (virtual DOM) care face ca aplicația web să funcționeze mult mai rapid decât alte aplicații web dezvoltate cu altă bibliotecă alternativă. React desparte o interfață complexă a unui utilizator în componente individuale, permițând mai multor utilizatori să lucreze simultan pe aceeași componentă, facilitând și grăbind timpul necesar dezvoltării aplicației.

Modul prin care React leagă datele într-un singur sens face ca totul să se desfășoare modular și rapid. Fluxul de date unidirecțional se referă la faptul că de obicei, programatorii adaugă

componente copii în componente părinți. În acest fel, se pot afla unde și când se produce o eroare, permițând și facilitând un control mai bun al întregii aplicații.

Componentele reprezintă componentele de bază al oricărei aplicații React. De obicei acestea reprezintă o parte a interfeței utilizatorului. React separă interfața în părți independente, reutilizabile care pot fi procesate separat.

- Componente funcționale: aceste componente mai pot fi numite și componente fără stare, deoarece nu au propria lor stare și conțin doar o metodă de afișare. Ele mai pot de asemenea să dobândească date ca și proprietăți de la alte componente.
- Componente de tip clasă: aceste componente, numite și componentă de stare, pot păstra și administra starea lor și au o metodă de afișare separată pentru a readuce JSX pe ecran.

Starea este un obiect React built-in care este folosit în a menține date sau informații despre componente. Starea unei componente se poate schimba de-a lungul timpului, și de fiecare dată când asta se întâmplă, componenta se reafișează. Schimbarea se produce în urma unei acțiuni a utilizatorului sau în urma unui eveniment generat de către sistem, acestea determinând comportamentul componentei sau felul în care se redă.

Prescurtate de la proprietăți, props sunt un obiect React built-in care stochează valoarea unui atribut și lucrează într-un mod asemănător cu atributele HTML. Permite o modalitate de a trimite date de la o componentă la altele în același fel în care argumentele sunt trimise într-o funcție.

JSX este o sintaxă pentru JavaScript, asemănătoare XML/HTML, pentru a reda HTML din codul JavaScript și pentru a descrie cum va arăta interfața. React transformă JSX în JavaScript pentru browser și folosindu-se de instrumentele oferite codul HTML și al site-urilor în JSX, acesta permite ca codul să se asocieze cu ușurință, deoarece se aseamănă cu scrierea unui cod HTML. JSX nu trebuie să fie neapărat utilizat cu React, ci și doar JavaScript, fiind un instrument foarte puternic care ajută la definirea structurilor de arbori cu atribute dar și ct atribuirea acestora.

JSX permite scrierea elementelor HTML în JavaScript și plasarea acestora în DOM fără nici o metodă **createElement()** sau **appendChild()**. JSX schimbă tag-uri HTML în elemente react.

React păstrează o reprezentare ușoară a DOM (Document Object Model) în memorie, cunoscută ca 'virtual' DOM (VDOM). VDOM reprezintă o modalitate mult mai ușoară și rapidă decât DOM deoarece nu se schițează nimic pe ecran. Când starea unui obiect se schimbă, VDOM schimbă doar obiectul în DOM în loc să actualizeze toate obiectele.

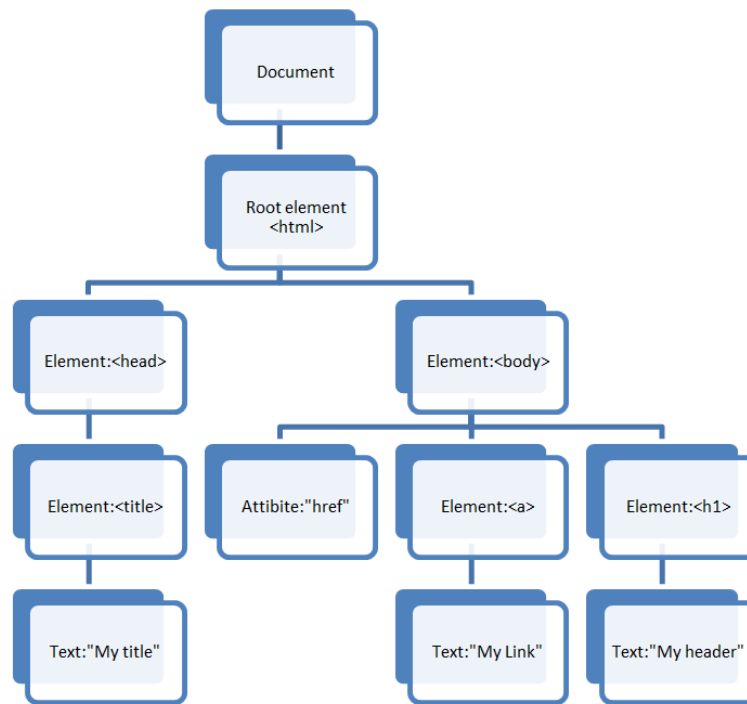


Figura 1.Structura unui document HTML

Așadar, DOM tratează un document XML sau HTML ca o structură de tip arbor în care fiecare nod reprezintă un obiect care face parte din document. Prin importanța sa caracteristică de a actualiza doar obiectele aflate în DOM, VDOM reprezintă o cale mult mai ușoară și rapidă.

3.1.2 MaterialUI

MaterialUI [2] este un framework de tip front-end și open-source pentru componente React. Este bazat pe limbajul de design Google pentru a furniza o calitate înaltă a experienței digitale. A fost creată pentru paginile web, focalizată pe planuri bazate pe rețea și animații receptive. Folosind Material-UI, este posibilă elaborarea unei colecții compacte și personalizabile de componente și utilități React, realizând texturi în urma concentrării sale asupra umbrelor și luminilor. Presupune folosirea Less (Leaner Style Sheets), un limbaj extensie pentru CSS.

Am ales Material UI în lucrarea mea deoarece:

- Componentele se mențin consistente când vine vorba de desing și tonurile culorilor, care oferă paginii web un aspect plăcut.
- Rămâne la zi cu noile actualizări.
- Deține o documentație detaliată, ușurând procesul de lucru.

Librăria Material-UI este formată din mai multe componente React pentru navigare, aspect, forme și diverse widget-uri. Una dintre cele mai ușoare componente este Button.

<Button>: Produce un efect de ondulare atunci când este dat click pe el. Importat prin @material-ui/core/Button. La fel ca și HTML <button>, Button poate conține text, pictograme sau orice alt nod React. Componenta Button suportă proprietăți comune precum className, href, disabled, aria-label. Proprietățile importante ale Button sunt:

- Culoarea: una dintre primary, secondary sau default, care e aceeași culoare dacă rămâne gol.

- Mini: dacă varianta este setată cu fab (floating action button), atunci mărimea butonului este redusă
- Varianta: stilul vizual al componentei, fie contained, outlined, fab, sau empty pentru stilul linkului în mod implicit.

TextField, importat din @material-ui/core/TextField se comportă ca o componentă input HTML standard și susține următoarele proprietăți:

Proprietate	Semnificație
autoComplete	Are capacitatea de autofill
autoFocus	Elementul se focusează în pagină
fullWidth	Ocupă toată lățimea elementului în care se află
multiline	Acceptă text pe mai multe linii
placeholder	Afișează valoarea nominală
type	Tipul de date
size	Mărimea elementului

Tabel 1. Proprietăți componenta TextField din Material-Ui

Material-UI folosește o metodă bazată pe JavaScript pentru a da o tematică componentelor sale numita CSS-in-JS. Folosind această metodă, numele de clasa CSS sunt generate prin obiecte JavaScript. Astfel, Material-UI este un mod simplu de a înfrumuseța componentele paginii React cu un minim de efort.

3.1.3 HTML

HTML (Hyper Text Markup Language) [3] reprezintă limbajul de marcaj standard în crearea paginilor web, elementele acestuia transmițând browser-ului să afișeze conținutul. Limbajul HTML descrie structura unei pagini web, fiind format dintr-o serie de elemente care numesc piesele conținute cu: ‘acesta este titlul’; ‘acesta este un paragraf’ ‘acesta este un link’. Partile principale ale unui element sunt:

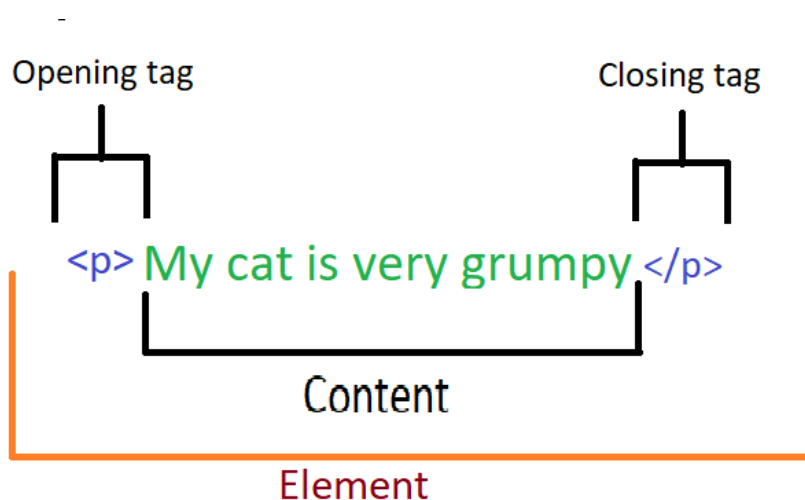


Figura 2. Structura unui element HTML

- Opening tag= numele elementului (p), împrejmuit de paranteze unghiulare. Acesta arată unde începe sau unde se termină obiectul în acest caz unde începe paragraful.
- Closing tag= este la fel ca si opening tag, doar ca include / înaintea numelui elementului. Acesta arată unde se încheie elementul, deci unde se încheie paragraful. Una dintre cele mai comune greșeli este aceea de a uita să se închidă tag-ul, ceea ce poate duce la un rezultat nedorit.
- The content= acesta reprezintă conținutul elementului.
- The element= tag-ul de deschidere, de închidere și tot conținutul, care formează așadar întregul element.

Elementele mai pot avea următoarele atribute:



Figura 3. Alocarea unui atribut unui element HTML

Atributele conțin informații în plus despre element, care nu vrem să apară în conținut. De exemplu, class este atributul 'nume' și editor-note este atributul 'valoarea'. Atributul 'clasa' ne permite să îi oferim elementului un identificator non-unic care poate fi folosit în a-l semnală (și orice alt element cu aceeași valoare 'class') cu informații despre stil și așa mai departe.

Un atribut trebuie să conțină:

- Spațiu între el și numele elementului (sau atributul precedent dacă sunt mai multe)
- Numele atributului urmat de semnul egal
- Valoarea atributului înconjurat de ghilimele (de deschidere, închidere)

Nesting elements se referă la elementele aflate în interiorul altor elemente. Dacă vrem să scoatem un cuvânt în evidență, putem adăuga cuvântul într-un element <spam>, care deosebește cuvântul ales de celelalte.

Elementele goale_sunt considerate elemente goale acelea care nu au conținut. Spre exemplu .

Structura unui document HTML este:

- <!DOCTYPE html> doctype- este o introducere necesară pentru a ne asigura ca browserul web folosește versiunea corectă.
- <html></html> elementul <html> dispune întregul conținut pe toată pagina și mai este uneori cunoscut ca și elementul de baza.
- <head></head> elementul <head> se comportă ca un recipient pentru tot conținutul inclus pe pagina HTML, care nu este arătat vizitatorului paginii.

Acestea pot fi cuvinte cheie si descrierea paginii care apare in rezultatele căutărilor, CSS care stilizează pagina.

- `<meta charset="utf-8">` acest element setează caracterele documentului, care conține majoritatea caracterelor limbajelor scrise (UTF-8). Prin setarea acestui cod, putem evita eventualele probleme nedorite, aduce încă un plus prin faptul că poate manevra orice conținut textual.
- `<title></title>` elementul `<title>` setează titlul paginii care apare in tab-ul browser-ului pe care este încărcată pagina.
- `<body></body>` elementul `<body>` este format din tot conținutul pe care vrem să îl arătăm utilizatorilor web când vizitează pagina, fie text, video, imagine, joc, melodii.

Elementele headings ne permit să specificăm că anumite părți din conținut sunt titluri sau subtitluri.

<code><h1></code>	Formatează textul ca titlu principal
<code><h2></code>	Formatează textul ca titlu nivel 2
<code><h3></code>	Formatează textul ca titlu nivel 3
<code><h4></code>	Formatează textul ca titlu nivel 4
<code><h5></code>	Formatează textul ca titlu nivel 5
<code><h6></code>	Formatează textul ca titlu nivel 6(cel mai puțin important)

Tabel 2.Tag-uri de marcare a textului în HTML[3]

Elementele `<p>` sunt folosite pentru a alcătui paragrafele textului.

În cazul în care textul conține liste, HTML are un element special. Marcarea acestora constă in folosirea a două elemente. Cele mai comune sunt listele ordonate sau neordonate:

- Listele ordonate sunt listele în care ordinea este importantă; acestea sunt scrise ca un element ``
- Listele neordonate sunt listele în care ordinea nu este importantă; acestea fiind scrise ca un element ``

Fiecare articol din cadrul listei este scris în interiorul unui element `` (list item)

Pentru a adaugă un link, avem nevoie de un elemnt simplu `<a>`. adăugarea linkului este foarte importantă pentru că asta face ca un site web să existe.

3.1.4 CSS

CSS (Cascading Style Sheets) [4] este un limbaj pentru design cu scopul de a simplifica metodele de a face paginile web să arate prezentabil, care descrie felul în care elementele HTML trebuie să fie afișate pe ecran, hârtie, sau orice alt material media. Acesta salvează lucrări de mari dimensiuni, poate controla aspectul mai multor pagini web în același timp, dar și stoca fișiere de stil externe. Folosind CSS, putem controla culoarea, fontul și stilul textului, spațierea dintre paragrafe, felul în care coloanele sunt prezentate si mărimea lor, culorile sau imaginile fundalului, aspectul design-ului, variațiile prezentării pentru diferitele device-uri si mărimea acestora. Fundatia CSS este reprezentată de HTML, iar CSS are simpla sarcină de a indica variantele estetice.

Avantajele CSS sunt:

- Ușor de folosit: putem scrie CSS și refolosi aceeași pagină in mai multe foi HTML. Putem defini un stil pentru fiecare element HTML si să îl aplicăm la un număr nelimitat de pagini web.

- Usor de întreținut: pentru a schimba ceva, putem cu ușurință să schimbăm stilul, iar toate elementele din paginile web vor fi actualizate automat.
- Paginile se încarcă rapid: dacă folosim CSS, nu avem nevoie să scriem tag-uri HTML de fiecare dată. Prin simpla aplicare a unei reguli CSS unui tag la fiecare aplicare a acestuia, timpul de descărcare este mai rapid.
- Un design mai amplu decât HTML: CSS posedă o varietate mai largă de atribute decât HTML, deci aduce avantajul unei pagini mai aspectuoase.
- Compatibil cu mai multe dispozitive: stilul foilor permit conținutului să fie optimizat pentru mai multe tipuri de device-uri. Prin folosirea aceluiași document HTML, diferite versiuni ale aceluiași site web pot fi prezentate pentru telefoane, sau pentru a fi printate.

CSS este un limbaj bazat pe reguli, acestea fiind definite prin specificarea diferitelor grupuri de stiluri care trebuie să fie aplicate unor elemente particulare sau elementelor din pagina web. Are o sintaxă simplă și folosește un număr de cuvinte cheie în limba engleză pentru a specifica numele diferitelor proprietăți ale stilului. Fiecare regulă sau set de reguli conține unul sau mai mulți selectori și un bloc declarativ. Selectorii declară care parte de marcaj a unui stil este aplicat prin potrivirea tag-urilor și atributelor în marcaj.

Selectorii se aplică:

- Elementelor specificate de către atribut, în mod special:
 - o Id: un identificator unic din document, identificat cu un prefix hashtag, spre exemplu #id
 - o Clasa: un identificator care poate adnota multiple elemente într-un document, identificat cu prefixul punct, spre exemplu .classname
- Elemente care depind de modul în care sunt plasate în comparație cu altele în documentul arbore

Clasele și Id-urile sunt un caz sensibil, încep cu litere și pot conține caractere alfanumerice și cratime. O clasă poate fi aplicată mai multor numere de elemente, iar un ID poate fi aplicat unui singur element.

Moștenirea [4] este o caracteristică importantă pentru CSS, deoarece este bazat pe relația dintre urmaș și succesor pentru a opera. Moștenirea este un mecanism prin care proprietățile sunt aplicate nu doar elementului specificat, dar și predecesorilor săi. Se bazează pe un document arbore, care este ierarhia elementelor XHTML într-o pagină. Elementele descendente pot moșteni proprietăți valorice ale CSS de la orice element predecesor pe care îl conțin. De obicei, elementele descendente moștenesc proprietăți legate de text, pe când elementele legate de mărimea elementului nu sunt moștenite. Proprietățile care pot fi moștenite sunt culoarea, fontul, spațierea dintre litere, stilul listelor, indentarea, alinierea, vizibilitatea, spațierea dintre cuvinte.

Proprietățile care nu pot fi moștenite sunt fundalul, marginea, stilul textului, mărimea, poziția, afișarea, înălțimea minimă și maximă, conturul, lățimea. Moștenirea poate evita declararea unor proprietăți anume în mod repetat într-o pagină de stil, permițând scrierea mai compactă a CSS.

Sunt trei modalități de implementare CSS [3]:

1. Intern

Acest stil CSS este un mod eficient de a stiliza o singură pagină. Totuși, folosind acest stil pentru mai multe pagini necesită mult timp deoarece regulile CSS trebuie să fie aplicate fiecărei pagini a site-ului web. Este necesară adăugarea tag-ului <style> în secțiunea <head> a documentului HTML.

2. Extern

Cu CSS extern, putem asocia paginilor web un fișier cu extensia .css, care poate fi creată prin adăugarea oricărui editor de text. Acest stil este mai eficient, mai ales pentru stilizarea unui site web de mari dimensiuni. Prin editarea unui fișier .css, este posibilă schimbarea întregului site o dată.

3. În linie

Acest stil CSS este folosit pentru a stiliza un anumit element HTML. Pentru a-l folosi, trebuie adăugat stilul atributului fiecărui tag HTML, fără a folosi selectori. Deoarece tag-ul HTML trebuie să fie stilizat individual, acest stil CSS nu este tocmai recomandat. Totuși, poate fi folosit în unele situații, când nu avem acces la fișierele CSS sau când trebuie să aplicăm stilul unui singur element.

3.2 Back-end

Back-end sau partea server reprezintă partea site-ului care nu interacționează în mod direct cu utilizatorul. Aceasta se ocupă de stocarea și managementul datelor prin intermediul bazelor de date, logica de business a aplicației și scrierea de API-uri care transpun datele din Back-end către front-end. Back-end-ul este format din trei părți: un server, o aplicație de interfață și o bază de date. Marea parte a logicii de business este scrisă în back-end, pentru a evita accesarea logicii site-ului de către utilizatori. Pentru a ne feri de atacuri de tip malware la nivelul bazei noastre de date, comunicarea cu bazele de date se face exclusiv în back-end.

3.2.1 NodeJs

Node.js [5] sau simplu Node, este un mediu de rulare de tip open-source, și o multiplatforma care permite dezvoltatorilor să creeze și aplicații și unelte în JavaScript. Timpul de rulare este destinat pentru folosirea în afara browser-ului, de exemplu direct pe computer sau pe un server OS. Așadar, mediul omite API-uri specifice browser-ului JavaScript și adaugă suport pentru mai multe API-uri tradiționale OS, inclusive HTTP și biblioteci de sistem ale fișierelor.

Motive pentru care am ales Node.js:

- A fost proiectat să optimizeze scalabilitatea în aplicațiile web și este o soluție bună pentru multe probleme des întâlnite în dezvoltările web.
- Beneficiază îmbunătățiri în design-ul limbajului comparativ cu alte limbaje tradiționale (Python, PHP). Multe alte limbaje noi și populare se convertesc în JavaScript, de exemplu TypeScript, CoffeeScript, Scala.
- Node Package Manager (NPM) oferă acces către sute de mii de pachete reutilizabile. Are de asemenea cea mai bună rezoluție de dependență din clasă și poate fi utilizat și pentru automatizarea majorității lanțurilor de construcție.
- Este portabil. Este disponibil atât pe Microsoft Windows, macOS, Linux, Solaris, WebOS, NonStop OS. Totodată este suportat pe majoritatea furnizorilor web, care oferă infrastructură și documentație specifică pentru găzduirea site-urilor web.

Alte task-uri comune nu sunt însă suportate de către Node însuși. Dacă dorim adăugarea unor manipulări specifice pentru diferite verbe HTTP (GET, POST, DELETE), este necesară gestionarea separată a cererilor la diferite căi URL, servirea fișierelor statice sau utilizarea șabloanelor pentru a crea un răspuns dinamic.

3.2.2 Express

Express [5] este cel mai popular pachet Node, fiind librăria de bază pentru un număr mare de cadre web Node. Oferă mecanisme:

- Scrie manipulatori de cereri cu diferite verbe HTTP către diferite sute URL.
- Setează diferite caracteristici ale aplicației web, cum ar fi locul șabloanelor care sunt utilizate pentru redarea răspunsului.
- Integrează motoarele de redare 'view' pentru a genera răspunsuri prin inserarea datelor în șabloane.

Pentru a gestiona aproape orice problemă cu dezvoltarea aplicațiilor web, dezvoltatorii au creat pachete middleware compatibile. Există librării care sunt destinate lucrării cu cookies, sesiuni, logarea utilizatorului, URL, parametri, headers de Securitate, etc.

3.2.3 MongoDB

MongoDB[6] este o bază de date NoSQL, apărută în mijlocul anilor 2000, folosită pentru stocarea datelor de volum mare. MongoDB folosește colecții și documente, în loc să folosească tabele și rânduri ca în bazele de date tradiționale. Documentele consistă în perechi cheie valoare, care sunt unitatea de bază în MongoDB. Colecțiile conțin funcții și seturi de documente care sunt echivalentul tabelor bazelor de date relaționale.

Caracteristici:

- fiecare baza de date conține colecții, care conțin la rândul lor documente. Fiecare document poate fi diferit, variind de la domeniu la domeniu. Conținutul și mărimea fiecărui document diferă.
- Documentele, sau rândurile cum mai pot fi ele numite, nu au nevoie de o schema definită dinainte. În schimb, domeniile pot fi create din mers.
- Structura documentului este în linie cu felul în care dezvoltatorii construiesc clasele și obiectele în limbajele de programare respective.
- Modelul disponibil al datelor în MongoDB ne permite reprezentarea ierarhică a relațiilor, și stocarea matricelor

3.2.4 Mongoose

Mongoose[6] este o librărie ODM (Object Data Modeling) pentru MongoDB and Node.js. Gestionează relațiile dintre date, dispune validarea schemelor, și poate fi folosit pentru traducerea dintre obiecte și coduri și reprezentarea acelor obiecte în MongoDB.

Motive pentru care am ales MongoDB:

- Modelele în Mongoose execută cea mai mare parte a stabilirii valorilor implicite pentru proprietățile documentelor și validarea datelor.
- MongooseJS dispune de un strat abstract deasupra bazei de date care elimină procesul folosirii colecțiilor
- Funcțiile pot fi atașate modelelor în Mongoose. Acesta permite încorporarea noilor funcționalități cu ușurință.
- Interogările folosesc mai degrabă înlanțuirea funcțională ceea ce ajută la scrierea unui cod mai ușor de citit, de întreținut și mai flexibil.

Mongoose folosește scheme pentru a modela datele pe care o aplicație trebuie să le stocheze și să le manipuleze în MongoDB. Aceasta include caracteristici de validare, construirea interogărilor, tipurilor de date.

Această schemă descrie atributele proprietăților care vor fi manipulate de către aplicație. Aceste atribute includ:

- Tipul de date (ex: string number)
- Dacă este necesară sau opțională
- Dacă valoarea este unică, ceea ce înseamnă că baza de date are voie să conțină doar un document cu valoarea respectivă în proprietate.

Tip	Semnificație
string	Șir de caractere
number	Numerele reale
date	data în format YYYY-MM-DD
buffer	Orice tip de fișiere
boolean	True sau false
array	Șir pentru stocarea mai multor informații într-o singură variabilă

Tabel 3. Tipuri de date în Mongoose

Comandă	Semnificație
findById	Căutarea unui document după id
findByIdAndUpdate	Căutarea unui document după id și modificarea acestuia
save	Adaugă datele în document
remove	Șterge înregistrări dintr-un tabel
findOneAndUpdate	Updatează înregistrea dintr-un document
replaceOne	Înlocuiește primul document bazat pe condiția dată

Tabel 4. Operații uzuale în Mongoose

4. Implementarea soluției

Scopul acestei lucrări este de a facilita licitațiile de autovehicule în mediul online, în așa fel încât acest proces să poată fi realizat de oriunde, cu ușurință. Proiectul permite utilizatorilor să participe la licitații și să își caute mașina dorită, sau să ofere o mașină spre licitație. Cu ajutorul limbajului de programare JavaScript împreună cu librăria React JS și MaterialUI am realizat partea de front-end în care se realizează interacțiunea utilizatorului cu aplicația, iar pentru partea de back-end am folosit pachetul Express pentru Node JS, care realizează legătura dintre site și baza de date, unde sunt stocate datele utilizatorilor și informații ale licitațiilor.

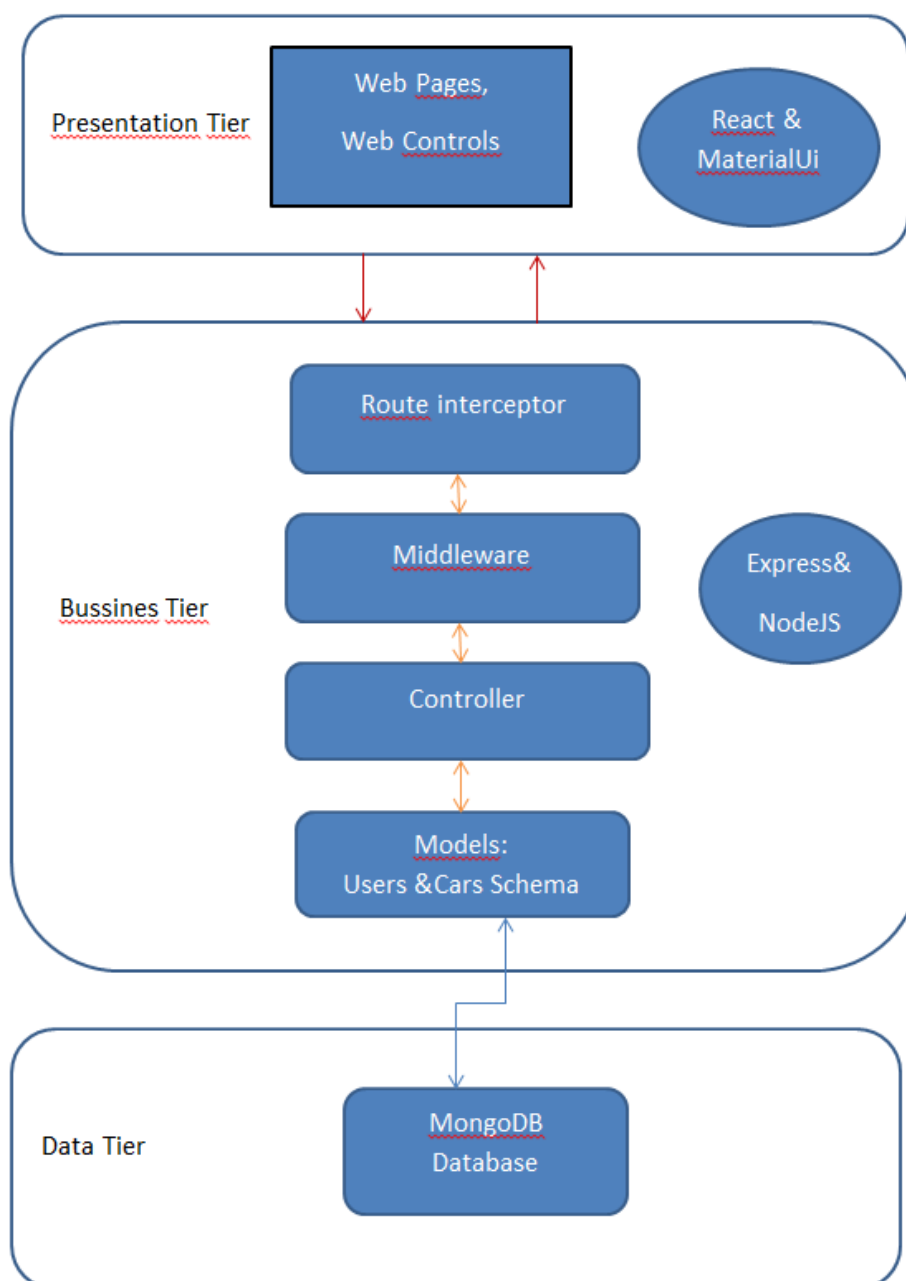


Figura 4. Diagrama arhitecturii aplicației

În figura de mai sus, se poate observa reprezentarea comunicării dintre nivelele sistemului. La comunicarea nativă se mai adaugă o relație cu aplicații, servicii API externe pe care le poate folosi sistemul pentru a întreține toate funcționalitățile necesare.

Proiectul permite utilizatorului să navigheze în cadrul site-ului cu ușurință, deoarece există butoane care facilitează navigarea. Mai mult, dacă un utilizator dorește să caute o anumită mașină pe site are această posibilitate, deoarece am adăugat funcționalitatea de căutare care este foarte ușor de folosit. Altă facilitare de care beneficiază, este aceea de a crea un cont nou pe site și a putea să își pună autovehiculul spre vânzare.

Specificațiile de realizare a site-ului:

1. Un site de licitații ușor de administrat și gestionat
2. O aplicație dinamică care permite funcționalități avansate
3. Adăugarea funcționalității de căutare
4. Adăugarea unei funcționalități care permite utilizatorului să își creeze un cont și să trimită date și fișiere
5. Realizarea unui server pentru funcționalitățile site-ului

În continuare vă prezint o diagramă care arată pașii necesari pentru a publica o licitație:

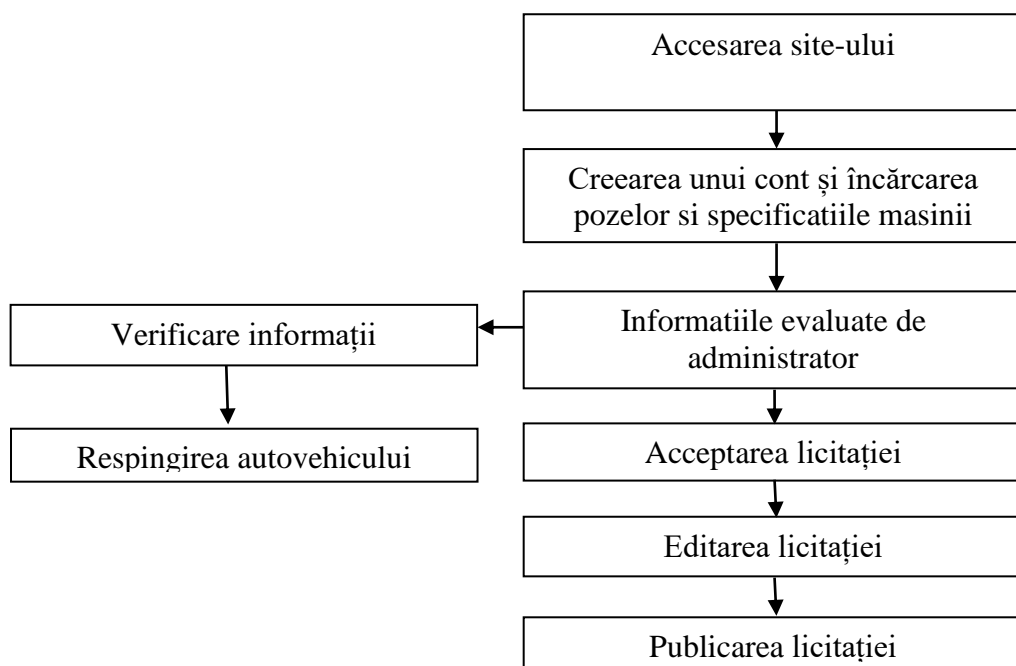


Figura 5. Pașii pentru crearea unei licitații

4.1 Crearea serverului offline folosind NodeJS, Express

Pentru majoritatea funcționalităților site-ului am avut nevoie de un server offline. Am utilizat NodeJS care este un mediu de execuție JavaScript de tip back-end open source care poate să ruleze cod JavaScript înafara unui browser web și ExpressJS care este un framework pentru Node care ajută la structurarea aplicației pe partea de server. Acestea pot fi folosite doar după instalarea programului Node, disponibil gratuit pe site-ul oficial al producătorului. În continuare, este necesară instalarea framework-ului express în linia de comandă, după cum urmează în următorul exemplu:

Npm install express –safe

Primul pas în crearea unui server offline necesită importarea librăriei în fișier, precum:

```
Const express = require('express');
```

```
const app = express();
```

Cel de-al doilea pas constă în folosirea metodei Listen din librăria Express, în care menționăm portul de acces dorit

```
App.listen(3001);
```

4.2 Baza de date pentru gestionarea conținutului

Pentru gestionarea conținutului am ales să folosesc baza de date MongoDB care este orientată pe documente JSON. Motivul pentru care am ales aceasta bază de date este faptul că această suportă tipuri variate de date.

Cu ajutorul librăriei Mongoose am realizat conectarea bazei de date cu serverul offline și două scheme de date pentru baza de date: cars și user, care face ca interacțiunea cu baza de date să fie mai ușoară. Conectarea bazei de date la serverul offline am realizat-o scriind codul următor:

```
1  const mongoose = require('mongoose');
2
3  module.exports = () => {
4    mongoose.connect('mongodb://localhost/licenta', {
5      useNewUrlParser: true,
6      useUnifiedTopology: true
7    })
8    .then(() => {
9      console.log("Mongodb connected...")
10     });
11  }
```

Figura 6. Conectarea bazei de date la serverul offline

În documentul users se salvează toate informațiile utilizatorilor:

```
1  const mongoose = require('mongoose');
2
3  const UsersSchema = mongoose.Schema({
4    email: { type: String, required: true },
5    password: { type: String, required: true },
6    username: { type: String, required: true },
7    name: { type: String, required: true },
8    zip: { type: String, required: true },
9    cardNumber: { type: String, required: true },
10   date: { type: Date, required: true },
11   cvc: { type: String, required: true },
12   phone: { type: String, required: true },
13  });
14
15
16  module.exports = mongoose.model('Users', UsersSchema);
```

Figura 7. Documentul utilizatorilor

În documentul cars se salvează toate licitațiile postate pe site:

```

2
3  const CarsSchema = mongoose.Schema({
4      name: { type: String, required: true },
5      number: { type: String, required: true },
6      year: { type: String, required: true },
7      make: { type: String, required: true },
8      model: { type: String, required: true },
9      vin: { type: String, required: true },
10     mileage: { type: String, required: true },
11     features: [ { type: String, required: true } ],
12 })
13
14 module.exports = mongoose.model('Cars', CarsSchema);

```

Figura 8.Documentul autovehiculelor

În continuare a fost nevoie să creez diferite API ENDPOINTS necesare pentru toate funcționalitățile aplicației web. Un API(Application Programming Interface) permite comunicarea între două sisteme, acesta asigurând un limbaj pentru interacțiunea celor două. Fiecare API are propria modalitate de a transmite date, acestea fiind postate pe internet. API-urile sunt împărțite în două categorii mari: SOAP și REST, amândouă fiind utilizate pentru a accesa servicii web. În aplicația mea am ales să le folosesc pe cele de tip REST, care utilizează metode simple, folosind URL-uri în cele mai multe cazuri, pentru a primi și transmite informații. REST folosește patru metode HTTP 1.1 cum ar fi : GET, POST, PUT și DELETE. API Endpoint reprezintă pur și simplu un capăt al unui canal de comunicare, iar fiecare endpoint este locația din care un API poate să acceseze informația de care are nevoie.

În aplicația mea am creat două rute specifice, prima fiind cea pentru utilizatori:

```
app.use('/users', UsersRoute);
```

Figura 9.Ruta specifică a utilizatorilor

Și cea de-a doua pentru toate licitațiile create:

```
app.use('/cars', CarsRoute);
```

Figura 10.Ruta specifică a autovehiculelor

URL-ul serverului este, după cum am menționat mai sus, <https://localhost:30001> . Dacă adăugăm “/users” sau “/cars” la finalul acestuia o să se încarce metodele specifice fiecăruia pe care le am creat.

Pentru ruta “/users” am creat următoarele endpointuri:

1. <https://localhost:3001/users>

Acest endpoint se ocupa de toate cererile de tip HTTP GET. Pe aceasta rută folosim metoda `.find()` pentru a găsi toți utilizatorii din baza de date `users`. Rezultatul este returnat în format JSON `res.json(users)`.

```
router.get('/', async (req, res) => {
  try {
    const users = await Users.find();
    res.json(users);
  } catch (err) {
    res.json({ message: err });
  }
});
```

Figura 11.Endpoint pentru întregul document al utilizatorilor

2. <https://localhost:3001/users/register>

Acest endpoint se ocupă de toate cererile de tip post. Aceasta rută este folosită în momentul creării unui nou cont pe site. Această rută primește în corpul cererii adresa de mail, parola și numele de utilizator. În primul rând, parola este criptată cu ajutorul algoritmului de criptare `bcrypt`:

```
const salt = await bcrypt.genSalt(10);
const hashedPassword = await bcrypt.hash(req.body.password, salt);
```

Figura 12.Metoda de criptare a parolei

În continuare se verifică dacă emailul și numele de utilizator primite pe cerere se mai regăsesc în baza de date, și în acest caz se trimite pe răspuns o eroare:

```
const emailExist = await Users.findOne({ email: req.body.email });
if (emailExist) return res.status(400).json({ message: "email already exists", statusCode: 400, emailError: true });

const usernameExist = await Users.findOne({ username: req.body.username });
if (usernameExist) return res.status(400).json({ message: 'username already exists', statusCode: 400, userNameError: true });
```

Figura 13. Verificarea numelor și mailurilor deja existe în baza de date

Iar în cazul în care nu este găsită nici o eroare se creează o instanță a modelului `User` și ulterior se salvează noul utilizator cu metoda `save()`

```

const user = new Users({
  email: req.body.email,
  password: hashedPassword,
  username: req.body.username
});

try {
  await user.save();
  res.status(200).json({ message: "ok", statusCode: 200 })
} catch (err) {
  res.status(400).send(err)
}

```

Figura 14. Metoda de salvare a unui nou utilizator în baza de date

2. <https://localhost:3001/users/login>

Acest endpoint are rolul de a verifica datele primite pe cerere în baza de date atunci când un utilizator vrea să se logheze pe site, trimițând o eroare sau un mesaj afirmativ că utilizatorul se poate loga cu datele pe care le-a introdus.

```

const user = await Users.findOne({ email: req.body.email });
if (!user) return res.status(400).send({ message: 'email not found', isAuthenticated: false });

const validPass = await bcrypt.compare(req.body.password, user.password);
if (!validPass) return res.status(400).send('Invalid password')

res.status(200).send({ isAuthenticated: true })

```

Figura 15. Metoda de verificare pentru logare

3. <https://localhost:3001/users/settings>

Acest endpoint servește la stocarea informațiilor despre metoda de plată pe care utilizatorul vrea să o folosească. Aceasta rută folosește metoda PUT a protocolului HTTP care stochează datele introduse pe baza numelui de utilizator care este logat pe site.

```

Users.findOneAndUpdate({ email: req.body.email },
{
  $set: {
    name: req.body.name,
    zip: req.body.zip,
    cardNumber: req.body.cardNumber,
    date: req.body.date,
    cvc: req.body.cvc,
    phone: req.body.phone,
  }
},
{ new: true },
(err, doc) => {
  if (err) {
    console.log("Something wrong when updating data!");
  }

  console.log(doc);
});

```

Figura 16.Endpoint-ul pentru salvarea metodei de plată

Pentru ruta “/cars” unde se creează licitațiile care o să fie publicate pe site am creat următorul endpoint:

5. <https://localhost:3001/cars/submit>

Pe acest endpoint am folosit librăria multer, care m-a ajutat să stochez imaginile preluate de pe front-end și să le salvez într-un director. Toate pozele care sunt transmise de utilizator sunt salvate cu numele utilizatorului și numerotate în ordine crescătoare, toate având extensia .jpg

```

var storage = multer.diskStorage({
  destination: function (req, file, cb) {
    cb(null, 'uploads')
  },
  filename: function (req, file, cb) {
    console.log(file.files)

    for (let x = 1; x <= req.body.nr; x++) {
      if (typeof (file) === "object") {
        cb(null, req.body.name + x + '.jpg')
      }
    }
  }
})

```

Figura 17.Salvarea pozelor in fișier cu extensia .jpg

Pe aceasta rută se mai salvează și toate datele despre mașinile care o să fie publicate pe site în documentul Cars pe care l-am menționat mai sus.

```
const car = new Cars({
  name: req.body.name,
  number: req.body.number,
  year: req.body.year,
  make: req.body.make,
  model: req.body.model,
  vin: req.body.vin,
  mileage: req.body.mileage,
  features: req.body.features,
});
```

Figura 18.Salvarea unei noi licitații

În final am creat ruta prin care se trimite părții de front-end pozele folosite în componenta de licitație a mașinii, ca obiecte statice:

```
app.use("/image", express.static(path.join(__dirname, "uploads")));
```

Figura 19.Ruta pentru fișierele statice

4.3 Crearea site-ului web folosind React si Material UI

După cum am mai menționat, pentru implementarea site-ului web, am folosit limbajul JavaScript în corespondență cu librăriile React si Material UI. Am ales Material UI pentru realizarea părții grafice deoarece are deja componente predefinite care nu mai necesită realizarea design-ului. De asemenea, librăria Material-UI permite importul și folosirea componentelor în scopul creării unei interfețe simple și cu un aspect plăcut, dar totodată și adaptabile la redimensionarea ferestrei. În acest fel, am redus din timp și am grăbit procesul de realizare, pentru simplul fapt că nu necesită scrierea de la 0 a fiecărui element din site.

Pentru a putea realiza o aplicație web cu librăria React este nevoie în primul rând de instalarea acesteia prin linia de comandă: `npm create-react-app my-app` (numele aplicației și a fișierului). Aceasta comandă setează un mediu de dezvoltare în care se poate folosi ultima versiune a limbajului JavaScript, împreună cu toate caracteristicile sale. În continuare a fost nevoie să instalez și frameworkul MaterialUI în fișierul aplicației React cu ajutorul comenzii în linia terminalului: `npm install @material-ui/core`.

4.3.1 Navigarea în aplicația web

Pentru a putea naviga cu ușurință pe toate paginile aplicației am avut nevoie de o librărie numită React-Router-DOM instalată cu ajutorul comenzii: `npm install react-router-dom`. După ce am instalat această librărie a fost nevoie să import componenta `<Switch>` pentru a putea declara înăuntrul acesteia diferitele rute de care am avut nevoie.


```
function App() {
  return (
    <div>
      <Switch>
      </Switch>
    </div>
  )
}
```

Figura 20. Componenta responsabilă pentru navigare

Mai departe am creat rutele specifice fiecărei pagini din aplicație cu ajutorul componentei Route care primește ca parametrii următoarele:

path – în acest parametru declarăm ruta pentru fiecare pagină
 component - acest parametru primește componenta specifică fiecărei rute

Pentru a putea fi afișată în orice pagină a aplicației, componenta barei de navigații trebuia plasată în afara componentei <Switch>

```
<div>
  <NavBar />
  <Switch>
    <Route exact path='/signup' component={SignUpPage} />
    <Route exact path='/settings' component={SettingsPage} />
    <Route exact path='/submit' component={SubmitPage} />
    <Route exact path='/' component={HomePage} />
    <Route exact path='/car' component={CarPage} />
  </Switch>
</div>
```

Figura 21 .Rutele pentru întreaga aplicație

Site-ul web conține următoarele pagini:

1. Home
2. Short presentation
3. Register
4. Payment info
5. Submit car

6. Login
7. Account

Fiecare pagină web reprezintă un fișier cu extensia .js în care este scris codul folosit pentru a crea interfața grafică în care îi este afișată utilizatorului.

4.3.2 Bara de navigație

Pentru a realiza bara de navigație a site-ului web care este afișată doar pe dispozitivele cu o rezoluție mai mare de 600 de pixeli pe lățime, am folosit componenta AppBar împreună cu Hidden, care verifică dimensiunea ferestrei.

```
<AppBar position="sticky" color="transparent" className={classes.appBar}>
  <Toolbar >
    <Hidden xsDown >
      <div className={classes.navButton}>
        <Button disableElevation="true" disableFocusRipple="true" disableRipple="true" >Auctions</Button>
        <Button disableElevation="true" disableFocusRipple="true" disableRipple="true" className={classes.sellButton}>Sell a Car</Button>
        <Button disableElevation="true" disableFocusRipple="true" disableRipple="true" >About Us</Button>
        <Button disableElevation="true" disableFocusRipple="true" disableRipple="true" >Daily Mail</Button>
      </div>
      <div className={classes.navSearch}>
        <TextField id="standard-start-adornment" variant="outlined" size="small" placeholder="Search for cars" fullWidth="true" />
      </div>
      <div>
        {user ? <div> <Button onClick={handleClick}>
          <AccountBoxIcon fontSize='large'></AccountBoxIcon>
        </Button>
          <Menu id="simple-menu" anchorEl={anchorEl} keepMounted open={Boolean(anchorEl)} onClose={handleClose} className="dropdown" >
            <MenuItem onClick={handleClose} className="dropdown-btn" >
              <Button href="/settings"> Settings </Button>
            </MenuItem>
            <MenuItem onClick={handleClose} className="dropdown-btn">My Listings</MenuItem>
            <MenuItem onClick={() => { localStorage.removeItem('user'); setUser(null) }} className="dropdown-btn" id="sign-out" >Sign Out</MenuItem>
          </Menu>
        </div>
        : <Button disableElevation="true" disableFocusRipple="true" disableRipple="true" className="nav" onClick={() => setSignIn(true)} >Sign In</Button>
      </div>
    </Hidden>
    <IconButton color="inherit" aria-label="open drawer" edge="start" onClick={handleDrawerToggle} className={classes.menuButton} >
      <MenuIcon />
    </IconButton>
  </Toolbar>
</AppBar>
```

Figura 22.Crearea barei de navigație pentru ecrane de dimensiuni mari

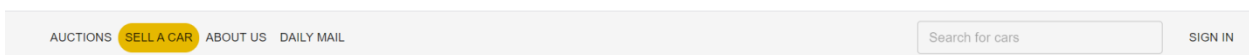


Figura 23.Bara de navigație pentru ecran de dimensiuni mari

Pentru dispozitivele cu ecran mai mic am folosit componenta drawer împreună cu List si Listitem am folosit următorul cod:

```

<div className={classes.drawer}>
  <Hidden smUp implementation="css">
    <Drawer
      container={container}
      variant="temporary"
      anchor={theme.direction === 'rtl' ? 'right' : 'left'}
      open={mobileOpen}
      onClose={handleDrawerToggle}
      classes={{
        paper: classes.drawerPaper,
      }}
      ModalProps={{
        keepMounted: true, // Better open performance on mobile.
      }}
    >
      {drawer}
    </Drawer>
  </Hidden>

```

Figura 24.Crearea barei de navigație pentru dispozitive mobile

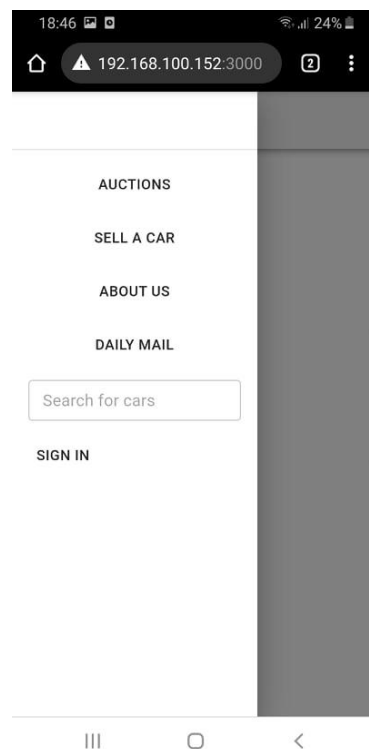


Figura 25.Meniul de navigare pentru dispozitivele mobile

Bara de navigație conține următoarele butoane:

- Auctions; care trimite la pagina principală
- Sell a car; pagina unde se creează licitațiile
- About us; pagina care conține informații despre platformă
- Daily mail; butonul de abonare
- Sign in; apare doar când utilizatorul nu este logat, iar atunci când utilizatorul este logat apare butonul de account, unde acesta poate să își modifice setările contului.

4.3.3 Modalul pentru logare

Acesta constă într-o componentă care se deschide la apăsarea butonului “Sign In” aflat pe bara de navigație, în cazul în care utilizatorul nu s-a logat anterior. Componenta conține două campuri de introducere a datelor, una pentru numele de utilizator, iar cealalta pentru parola. Pentru aceste câmpuri am folosit componenta predefinită din framework-ul MaterialUI `<TextField>` cu setarea `type="password"`, pentru câmpul alocat parolei, acesta facand să nu fie vizibilă. Cu ajutorul metodei realizate de mine `onChangeHandler()` pot să stochez datele introduse de utilizator in variabile specifice fiecărui câmp.

```

<Typography>
  >
    Email address
  </Typography>
  <TextField name="email" required autoFocus variant="outlined"
    fullWidth size="small" onChange={onChangeHandler} error={error.emailStatus}
    helperText={error.errorMessage.includes("email") ? error.errorMessage : " "}
  </TextField>
</div>


Figura 26. Crearea modalului pentru logare



La apăsarea butonului Sign In din această componentă se apelează metoda handleSubmit() care face o cerere de tip HTTP pentru a trimite datele pe care utilizatorul le-a introdus la endpoint-ul https://localhost:3001/users/login, care trimite la rândul lui un mesaj de confirmare sau un mesaj cu erori. Dacă datele introduse de utilizator sunt corecte, această metodă salvează numele utilizatorului în spațiul local al browserului.



```

const handleSubmit = event => {
 event.preventDefault()
 fetch('http://localhost:3001/users/login', {
 method: 'POST',
 headers: {
 'Accept': 'application/json',
 'Content-type': 'application/json'
 },
 body: JSON.stringify({
 email: inputValue.email,
 password: inputValue.password,
 })
 }).then(res => res.json()).then(res => {
 console.log(res.isAuthenticated)
 if (res.isAuthenticated === true) {
 localStorage.setItem('user', inputValue.email)
 window.location.reload()
 }
 })
}

```



Figura 27. Cererea de tip HTTP POST pentru logare



36


```

Componenta de logare mai contine și un link către pagina de sign-up în cazul în care utilizatorul respectiv nu are cont pe site-ul web.

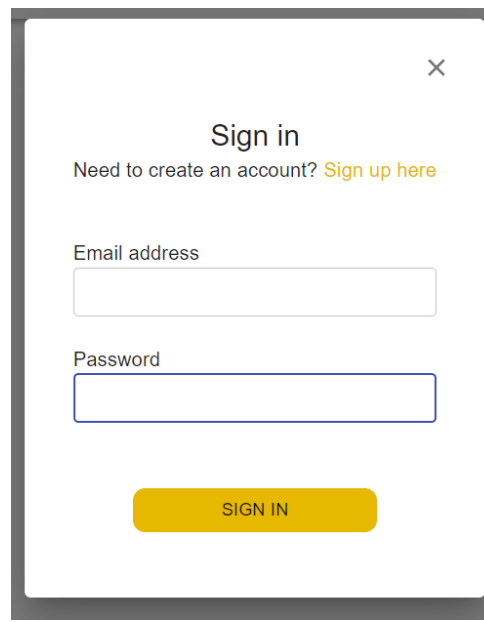



Figura 28.Modalul pentru logare

4.3.4 Pagina de înregistrare

Aceasta pagina constă în 3 câmpuri de text, mai exact componenta `<TextField>` importată din framework-ul MaterialUI, acesta fiind: email, username, și password. Cu ajutorul funcției `onChangeHandler()` menționată mai sus se salvează datele introduse de către utilizator în variabilele specifice fiecăruia. Butonul Create Account face o cerere de tip HTTP POST la endpointul din serverul offline pe ruta <http://localhost:3001/users/register>, care trimite datele salvate din câmpuri în corpul cererii. Această componentă poate să primească mesaje de tip eroare și să le afișeze în momentul în care utilizatorul a introdus un email sau un username deja salvat în baza de date Users, în așa fel încât să nu se creeze două conturi cu același nume de utilizator sau email.



Sign Up

Email address

Password

Create a Username (public name that others see)

CREATE ACCOUNT

Figura 29. Pagina de crearea contului

4.3.5 Pagina de introducere a modalității de plată

Aceasta componentă permite utilizatorului să își salveze informațiile metodei de plată pe care o dorește, plata făcându-se doar cu cardul bancar. Pagina conține toate câmpurile necesare pentru o plată cu cardul, acesta fiind salvat tot cu metoda `onChangeListener()` menționată mai sus, aceasta trimițând toate datele introduse. Butonul Register to bid face o cerere de tip HTTP Put pe ruta <http://localhost:3001/users/settings> folosind pe serverul offline metoda `find()` care găsește utilizatorul logat pe site și salvează informația introdusă corect în documentul Users al bazei de date.


Credit Card Information

Name on card

Zip or Postal Code

Credit Card Number

Expiration



CVC (3 or 4 digit code)

Contact Information

Phone Number (in the event you win an auction)

REGISTER TO BID

Figura 30. Pagina pentru salvarea modalității de plată

4.3.6 Pagina de creare a unei licitații

Această componentă conține mai multe câmpuri de introducere a datelor, și anume toate informațiile despre mașina pe care utilizatorul vrea să o vândă. Butonul de încărcare a pozelor se află în eticheta <form> la care am adăugat opțiunea de a primi datele de tip multipart. În interiorul acestui element am mai adăugat și un element de tip <input>, pe care l-am setat să primească mai multe fișiere de tip jpg. Cu ajutorul metodei onChange am reușit să salvez fișierele de tip poză într-un obiect.

```
<form encType="multipart/form-data" >
  <input
    accept=".jpg"
    id="file"
    multiple
    type="file"
    onChange={(event) => { const files = event.target.files; setImages(files) }}
  />
</form>
```

Figura 31.Componenta pentru încărcarea imaginilor

Toate datele pe care utilizatorul le introduce, inclusiv fișierele pe care le încarcă, sunt salvate într-un obiect predefinit al limbajului Javascript de tip FormData. La apăsarea butonului de submit acesta face o cerere de tip HTTP POST cu datele salvate pe ruta <http://localhost:3001/users/submit>, iar fișierele de tip poză sunt salvate cu ajutorul librăriei multer după cum am menționat mai sus.

4.3.7 Pagina de informații

Această componentă este o pagină statică în care se afișează informații utile despre modul în care funcționează licitațiile, și crearea acestora.

4.3.8 Pagina principală

Această componentă face o cerere de tip HTTP Get la ruta de pe serverul offline <http://localhost:3001/cars> în metoda specifică a librăriei React componentDidMount() care se apelează înainte ca conținutul paginii să fie afișat, prin care cere serverului toate datele salvate în documentul din baza de date Cars sub forma unui document de tip JSON. Componenta afișează toate licitațiile cu ajutorul metodei .map() al limbajului JavaScript, care parcurge element cu element documentul de tip JSON. Prin această metodă se afișează componenta Car care primește proprietățile fiecărui element din documentul primit, așa cum reiese din acest cod:

```

<div className="preview">
  {this.state.users.map(({ id, ...props }) =>
    <Car id={this.state.users.id} {...props} />
  )}
</div>

```

Figura 32.Parcurgerea tuturor licitațiilor

Aceasta componentă Car conține prima poza pe care utilizatorul a încărcat-o, și informații utile despre mașină, cum ar fi: anul producției, marca și modelul.



Figura 33.Componenta unui autovehicul

4.3.9 Funcționalitatea de căutare

Unul dintre obiectivele lucrării este realizarea unei funcții care permite găsirea autovehiculelor dorite utilizatorului prin componenta de căutare. Pentru a face acest lucru, am folosit componenta `<TextField>` importată din `MateriaUI` la care i-am adăugat metoda `onChange` care vizează orice schimbare în componentă, în acest caz când un utilizator apasă pe tastatură. Aceasta metodă menționată mai sus apelează o funcție `updateInput()` care parcurge obiectul din baza de date și flitrează doar datele în funcție de numele scris de utilizator. Am creat două variabile, una pentru numele mărcii, iar cealaltă după modelul mașinii.


```

updateInput = (event) => {
  const filteredMake = this.state.defaultUsers.filter(cars => {
    return cars.make.toLowerCase().includes(event.target.value.toLowerCase())
  })

  const filteredModel = this.state.defaultUsers.filter(cars => {
    return cars.model.toLowerCase().includes(event.target.value.toLowerCase())
  })

  const unfilteredList = [...filteredMake, ...filteredModel]
  const filteredList = [...new Set(unfilteredList)];
  this.setState({
    users: filteredList
  })
}

```

Figura 34. Metoda de filtrare a datelor

4.4 Stilizarea paginilor web

Chiar dacă Material UI dispune de componente predefinite și stilizate, am ales să folosesc și limbajul CSS pentru a-i oferi site-ului un aspect cât mai plăcut și pentru ca utilizatorul să vizualizeze cât mai bine conținutul. Pentru a realiza acest lucru, am avut nevoie de fișiere cu extensia .css (foaie de stil externă) în care acord fiecărui element atributele pe care le-am vrut, în funcție de clasa în care se află.

Pentru fiecare pagină a site-ului am creat fișiere CSS specifice fiecăruia, în care am realizat mai multe clase precum: .btn .redirect .avatar. Ca să utilizezi o anumită clasă în pagina web aceasta trebuie să fie localizată în componentă.

În următoarele rânduri o să exemplific cum se identifică o secțiune, o clasă sau un element al unei pagini.

```

<div className="card-item"></div>
<span></span>
<p id="sign-out"></p>

```

```
.card-item{  
  display: flex;  
  flex-direction: column;  
  height: 200px;  
  align-items: center;  
  margin: 30px 25px;  
  box-shadow: 0 4px 8px 0 □ rgba(0,0,0,0.2);  
  transition: 0.3s;  
}
```

Figura 35. Definirea unei clase cu attribute în CSS

Pentru aplicarea stilului dorit este necesară importarea fișierelor direct în fișierele sursă ale paginilor web.

Această legătură arată astfel:

import './styles.css'

5.Rezultate experimentale

Principalele obiective ale acestei lucrări au fost îndeplinite cu succes, iar în următoarele paragrafe voi prezenta rezultatele experimentale obținute în accesării site-ului web.

Unul dintre obiective a fost crearea unui site pentru licitații auto cu un aspect plăcut și ușor de folosit pentru orice utilizator. În acest sens, pagina nu deține opțiuni ascunse, și este accesibilă atât pentru vânzători cât și pentru posibili cumpărători.

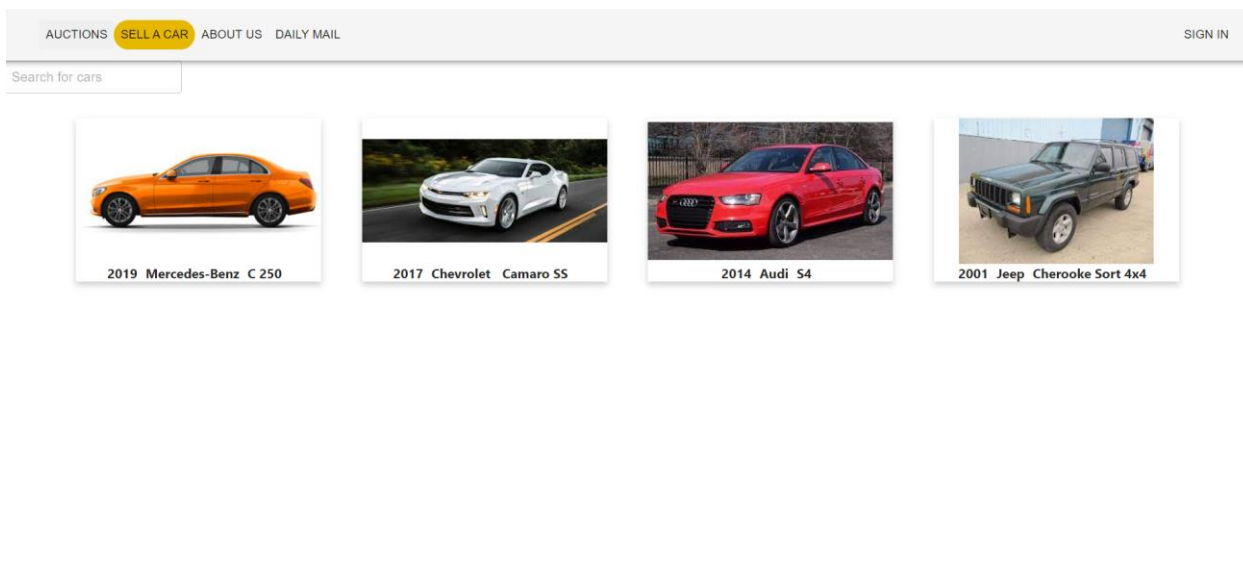



Figura 36.Pagina principală

Printre obiectivele lucrării de licență se numără și organizarea în ordine cronologică a postărilor. Așadar, când o licitație este postată, aceasta este plasată ultima pe site din punct de vedere cronologic. Aceasta funcționalitate ajută potențialii clienți să vadă prima dată licitațiile cu cel mai puțin timp rămas.

Un alt obiectiv a fost acela de a implementa funcționalitatea de înregistrare și logare. Aceasta permite utilizatorului să creeze un cont și să îl acceseze oricând, fiind absolut necesară în cazul în care cineva dorește să participe la o licitație sau să creeze una. Înregistrarea sau crearea unui cont se poate face prin accesarea paginilor Login și Register din meniul site-ului. La logare datele introduse sunt verificate de fiecare dată în baza de date, iar la înregistrare se verifică dacă numele utilizatorului sau mail-ul introdus sunt deja prezente în baza de date.



Sign Up

Email address


email already exists

Password

Create a Username (public name that others see)

CREATE ACCOUNT

Figura 37. Introducerea unui nume de utilizator greșit



Sign in

Need to create an account? [Sign up here](#)

Email address

email not found

Password

SIGN IN

Figura 38. Introducerea unui cont deja existent în baza de date

Funcționalitatea de căutare ajută utilizatorii să își găsească mașina dorită, după marcă sau model. Dacă există mai multe rezultate pe baza căutării, utilizatorul are posibilitatea de a alege varianta pe care îl interesează.

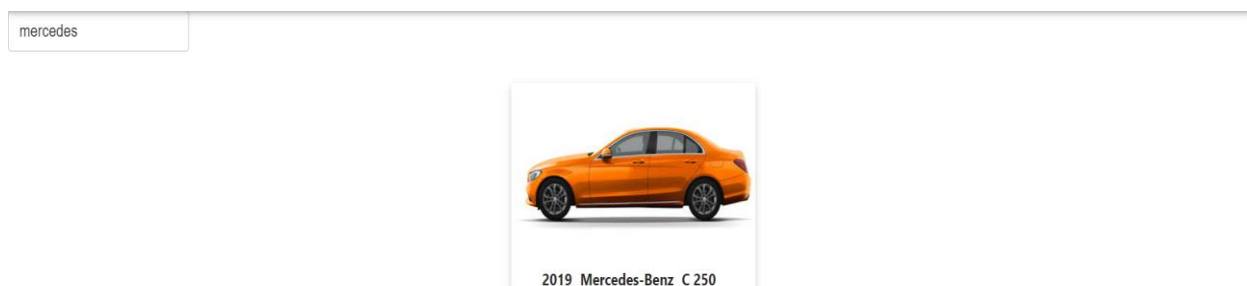


Figura 39.Funcționalitatea de căutare

Alta funcționalitate importantă pe care am reușit să o îndeplinesc este adăugarea celor 30 de minute la fiecare licitație depusă. În primul rând, aceasta face diferența dintre site-ul creat de către mine și alte site-uri pentru licitații de autovehicule deja existente. În al doilea rând, funcționalitatea dă șansa licitației să continue și după o ofertă mai mare decât ultimul preț este plasată. Acesta pune în dezavantaj licitatorii care vor să liciteze o mașină în ultimele secunde rămase.

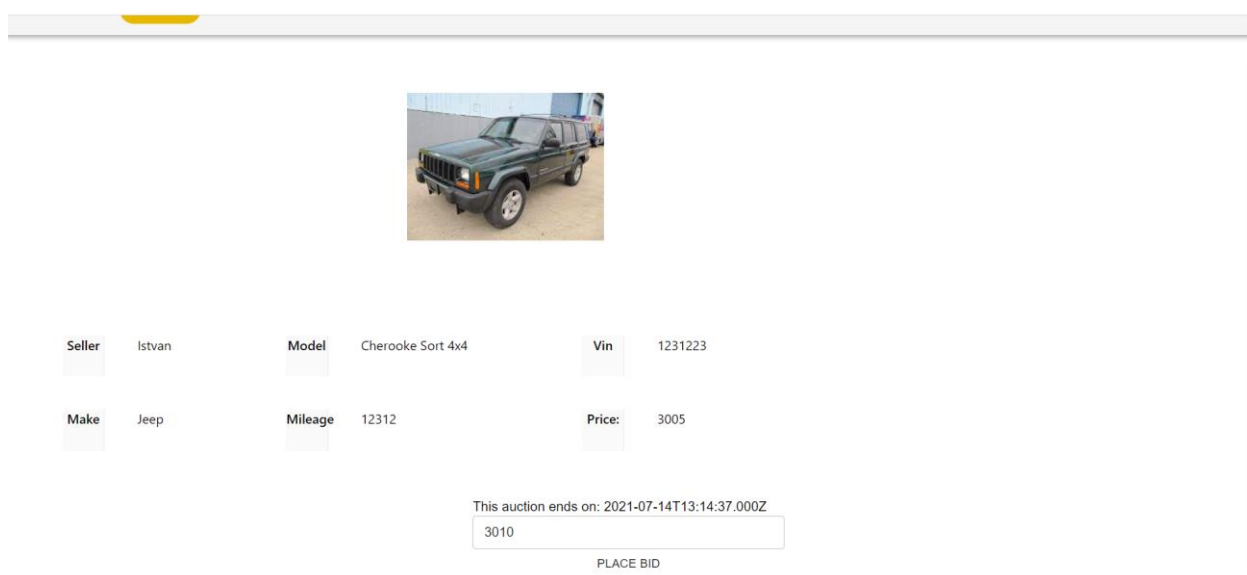


Figura 40.Plasarea unei oferte



Seller	Istvan	Model	Cherooke Sort 4x4	Vin	1231223
Make	Jeep	Mileage	12312	Price:	3010

This auction ends on: 2021-07-14T13:44:37.000Z

Figura 41.Verificarea metodei de adaugera a minutelor

Un alt rezultat este realizarea server-ului offline care stochează toate datele introduse care pot fi accesate din orice componentă a aplicației, inclusiv bazele de date și informațiile încărcate. Acesta presupune o baza de date care contine toate informatiile utilizatorilor si licitatiilor create.Fara realizarea acestui server nu as fi putut introduce functionalitatile necesera a aplicatiei web

models	6/30/2021 3:42 PM	File folder	
node_modules	6/30/2021 6:26 PM	File folder	
routes	7/1/2021 4:30 PM	File folder	
uploads	7/7/2021 3:14 PM	File folder	
app	7/2/2021 3:59 PM	JavaScript File	1 KB
initDB	7/4/2021 9:43 PM	JavaScript File	1 KB
package.json	6/30/2021 6:26 PM	JSON File	1 KB
package-lock.json	6/30/2021 6:26 PM	JSON File	67 KB

Figura 42.Serverul offline

Un alt rezultat ar fi stocarea pozelor incarcate de catre utilizator la crearea unei noi licitatii si salvarea lor in fisierul dorit.Salvarea acestor se face cu numele utilizatorului urmat de un numar in ordine crescatoare pornind de la 1 cu extensia .jpg



Robert1



Robert2

Figura 43.Salvarea pozelor

6. Concluzii

Lucrarea realizată prezintă un site web de licitații autovehicule, un mod simplu și cunoscut de a achiziționa o mașină în mediul online. Așadar, acest tip de site mai există atât în țară cât și în străinătate, un adaos adus de către mine fiind timpul adăugat automat după ce o licitație este făcută.

Realizarea site-ului s-a făcut cu ajutorul librăriei React și Material UI, care oferă un aspect plăcut, iar pentru adăugarea funcționalităților a fost nevoie de un server offline, pe care l-am creat cu ajutorul pachetului ExpressJs, iar pentru salvarea datelor am folosit baza de date MongoDB.

Așadar, acest proiect permite utilizatorilor să cumpere sau să ofere spre licitație orice tip de mașină cu ușurință, datorită butoanelor care facilitează navigarea. În acest sens, utilizatorul are posibilitatea de a căuta pe site mașina dorită cu ajutorul funcționalității de căutare care este foarte ușor de folosit. În cazul în care un utilizator dorește să se înscrie într-o licitație, acesta trebuie să își creeze un cont și să apese butonul de Place bid în cadrul paginii autovehiculului. Altă facilitare este cea de a oferi un autovehicul spre licitație, utilizatorul fiind nevoit să își creeze de asemenea un cont, și de a încărca informațiile și pozele necesare.

Acest proiect a fost realizat în mai multe etape, încercând diferite soluții și folosind mai multe limbaje de programare:

1. Realizarea unui site folosind React, MaterialUI și CSS
2. Realizarea serverului offline ExpressJs
3. Crearea bazei de date
4. Realizarea diferitelor rute în serverul offline pentru adăugarea funcționalităților
5. Crearea diferitelor component necesare pentru site
6. Testarea funcționalităților

În concluzie, prin respectarea etapelor enumerate mai sus, obiectivele lucrării au fost îndeplinite, iar rezultatul obținut este cel care s-a propus inițial. Deși la început am încercat adăugarea unui cronometru în timp real dar acesta presupunea folosirea unui protocol care necesită cunoștințe mai avansate iar dobândirea acestor cunoștințe ar fi necesitat mult prea mult timp.

Pe viitor cu ajutorul resurselor deja existente în proiect se dorește implementarea mai multor funcționalități și optimizarea site-ului pentru a se face actualizarea mai eficientă. În primul rând, trebuie adăugată funcționalitatea de cronometru în timp real. Un alt lucru care ar trebui adăugat ar fi adăugarea unui chat în care un posibil cumpărător poate să vorbească cu proprietarul autovehiculului.

7.Bibliografie

- [1] Sanchit Aggarwal (2018) “Modern Web-Development using ReactJS” International Journal of Recent Research Aspects , Vol. 5, Issue 1, pp. 133-137
- [2] Adam Boduch (2019) “React Material-UI Cookbook: Build captivating user experiences using React and Material-UI “ Packt Publishing Ltd
- [3] HTML and CSS: Design and Build Websites - Jon Duckett 25 october 2011
- [4] Lucia Rusu "Tehnologii Web Front-end" 2019
- [5] Evan Hahn (2016) “Express in Action: Writing, building, and testing Node.js applications” Simon and Schuster
- [6] Simon Holmes(2013) “Mongoose for Application Development” Packt Publishing Ltd
- [7] History of the Web <https://webfoundation.org/about/vision/history-of-the-web/>
- [8] <https://www.geeksforgeeks.org/difference-between-server-side-scripting-and-client-side-scripting/>

8 Anexe

8.1 Codul folosit pentru realizarea componentei autovehiculului

```
import React from 'react';
import { withRouter } from 'react-router-dom';

import './styles.css'

const Car = (props) => {
  console.log(props.date, "car")
  return (
    <div style={{ width: "350px" }} onClick={() => props.history.push('/car', {
      state: {
        make: props.make,
        year: props.year,
        features: props.features,
        model: props.model,
        mileage: props.mileage,
        name: props.name,
        vin: props.vin,
        date: props.date,
        price: props.price
      }
    })}>

      <div className="card-item" >
        <div className="image"
          style={{
            backgroundImage: `url(http://localhost:3001/image/${props.name}1.jpg)`
          }} />
        <div className="card-information">
          <span className="info">{props.year}</span>
          <span className="info">{props.make}</span>
          <span className="info">{props.model}</span>
        </div>
      </div>
    </div >
  )
}

export default withRouter(Car)
```

8.2 Codul folosit pentru realizarea paginii autovehiculului

```
import React from 'react';

import { Button, Typography, TextField } from '@material-ui/core';

import './styles.css'

export default function CarPage(props) {
  const { year, make, mileage, model, features, name, vin, nr, date, price } = props.location.state.state
  const [inputValue, setInputValue] = React.useState({ newPrice: "" });

  const onChangeHandler = event => {
    setInputValue({ ...inputValue, [event.target.name]: event.target.value });
  }

  const handleSubmit = event => {
    event.preventDefault()
    fetch('http://localhost:3001/cars/auction', { // @todo api call separate filess
```

```

method: 'PUT',
headers: {
  'Accept': 'application/json',
  'Content-type': 'application/json'
},
body: JSON.stringify({
  newPrice: inputValue.newPrice,
  name: name,
  date: date
})
}).then(res => res.json()).then(res => {
  console.log(res)

})
}
console.log(date, 'data');
return (
  <div>
    <div className="images">
      <div className="image"
        style={{
          backgroundImage: `url(http://localhost:3001/image/${name}1.jpg`
        }} />
      <div className="image"
        style={{
          backgroundImage: `url(http://localhost:3001/image/${name}2.jpg`
        }} />
    </div>
    <div>
      <dl >
        <div classname="column">
          <div className="details">
            <dt>
              Seller
            </dt>
            <dd>
              { name }
            </dd>
          </div>
          <div className="details">
            <dt>
              Make
            </dt>
            <dd>
              { make }
            </dd>
          </div>
        </div>
        <div classname="column">
          <div className="details">
            <dt>
              Model
            </dt>
            <dd>
              { model }
            </dd>
          </div>
          <div className="details">
            <dt>
              Mileage

```

```

        </dt>
        <dd>
            { mileage }
        </dd>
    </div>
</div>
<div className="column">
    <div className="details">
        <dt>
            Vin
        </dt>
        <dd>
            { vin }
        </dd>
    </div>
    <div className="details">
        <dt>
            Price:
        </dt>
        <dd>
            { price }
        </dd>
    </div></div>

</dl>
</div>
<div className="bid">
    <Typography>This auction ends on: { date}</Typography>
    <TextField
        name="newPrice"
        autoFocus
        variant="outlined"
        size="small"
        onChange={onChangeHandler}></TextField>
    <Button onClick={handleSubmit}> Place bid</Button>
</div>
</div>
)
}

```

8.3 Codul folosit pentru realizarea paginii principale

```

import React from 'react';

import { TextField } from "@material-ui/core"

import Car from './Car';
import CarPage from './CarPage'

import './styles.css'

class HomePage extends React.Component {
    constructor(props) {
        super(props)
        this.state = {
            users: [],
            defaultUsers: []
        }
    }

    componentDidMount() {
        fetch('http://localhost:3001/cars').then((res) => res.json()).then(res => { console.log(res, "res");
        this.setState({ users: res, defaultUsers: res }) }
        )
    }
}

```

```

    }

    updateInput = (event) => {
      const filteredMake = this.state.defaultUsers.filter(cars => {
        return cars.make.toLowerCase().includes(event.target.value.toLowerCase())
      })

      const filteredModel = this.state.defaultUsers.filter(cars => {
        return cars.model.toLowerCase().includes(event.target.value.toLowerCase())
      })

      const unfilteredList = [...filteredMake, ...filteredModel]
      const filteredList = [...new Set(unfilteredList)];
      this.setState({
        users: filteredList
      })
    }
  }

  render() {
    return (
      <div>

        <div className="search">
          <TextField id="standard-start-adornment" variant="outlined" size="small" placeholder="Search for
cars" onChange={(event) => this.updateInput(event)} />
        </div>
        <div className="preview" >
          {this.state.users.map(({ id, ...props }) =>
            <Car id={this.state.users.id} {...props} />
          )}
        </div>
      </div>
    )
  }
}

export default HomePage;

```

8.4 Codul folosit pentru realizarea paginii barei de de navigație

```

import React, { useState } from "react";
import AppBar from "@material-ui/core/AppBar"
import { Button, Hidden, Drawer, Toolbar, Divider, List, ListItem } from '@material-ui/core';
import { makeStyles, useTheme } from '@material-ui/core/styles';
import MenuItem from '@material-ui/icons/Menu';
import IconButton from '@material-ui/core/IconButton';
import TextField from '@material-ui/core/TextField';
import { Menu, MenuItem, Grow } from "@material-ui/core";
import AccountBoxIcon from '@material-ui/icons/AccountBox';

import SignIn from '../SignIn'

import "../styles.css";
import SettingsPage from "../SettingsPage";

const drawerWidth = 240;
const useStyles = makeStyles((theme) => ({
  root: {
    flexGrow: 1,
  },
  drawer: {
    [theme.breakpoints.up('sm')]: {

```

```

        width: drawerWidth,
        flexShrink: 0,
      },
    },
    menuButton: {
      marginRight: theme.spacing(2),
      justifyContent: "start",
      [theme.breakpoints.up('sm')]: {
        display: 'none',
      },
    },
    navButton: {
      [theme.breakpoints.up('sm')]: {
        flexGrow: 1,
        marginLeft: theme.spacing(3),
        textTransform: "none",
      }
    },
    navSearch: {
      [theme.breakpoints.up('sm')]: {
        marginRight: theme.spacing(3),
        width: "300px"
      }
    },
    signUp: {
      '&:hover': {
        background: "#ffcc00"
      },
      background: "#e6b800",
      width: "100px",
      borderRadius: "10px"
    },
    sellButton: {
      '&:hover': {
        background: "#ffcc00"
      },
      background: "#e6b800",
      borderRadius: "50px"
    },

    // necessary for content to be below app bar
    toolbar: theme.mixins.toolbar,
    drawerPaper: {
      width: drawerWidth,
    },
    content: {
      flexGrow: 1,
      padding: theme.spacing(3),
    },
  },
));

function Nav(props) {
  const { window } = props;
  const classes = useStyles();
  const theme = useTheme();
  //const user = localStorage.getItem('user');

  const [user, setUser] = React.useState(localStorage.getItem('user'))

```

```

const [mobileOpen, setMobileOpen] = React.useState(false);
const [openSignIn, setSignIn] = useState(false)

const handleDrawerToggle = () => {
  setMobileOpen(!mobileOpen);
};

const [anchorEl, setAnchorEl] = React.useState(null);

const handleClick = (event) => {
  setAnchorEl(event.currentTarget);
};

const handleClose = () => {
  setAnchorEl(null);
};

const drawer = (
  <div>
    <div className={classes.toolbar} />
    <Divider />
    <List >
      <ListItem><Button  disableElevation="true"  disableFocusRipple="true"  disableRipple="true"
fullWidth className="nav">Auctions</Button></ListItem>
      <ListItem><Button  disableElevation="true"  disableFocusRipple="true"  disableRipple="true"
fullWidth className="nav">Sell a Car</Button></ListItem>
      <ListItem><Button  disableElevation="true"  disableFocusRipple="true"  disableRipple="true"
fullWidth className="nav">About Us</Button></ListItem>
      <ListItem><Button  disableElevation="true"  disableFocusRipple="true"  disableRipple="true"
fullWidth className="nav">Daily Mail</Button></ListItem>
      <ListItem className="nav">
        {
          user ? <div> <Button
            onClick={handleClick}>
              <AccountBoxIcon fontSize='large'></AccountBoxIcon>
            </Button>
            <Menu
              id="simple-menu"
              anchorEl={anchorEl}
              keepMounted
              open={Boolean(anchorEl)}
              onClose={handleClose}
              className="dropdown"
            >
              <MenuItem onClick={handleClose} className="dropdown-btn" >
                <Button href="/settings">
                  Settings
                </Button>
              </MenuItem>
              <MenuItem
                onClick={handleClose}
                className="dropdown-btn">My
Listings</MenuItem>
              <MenuItem
                onClick={() => { localStorage.removeItem('user'); setUser(null) }}
                className="dropdown-btn" id="sign-out" >Sign Out</MenuItem>
            </Menu>
          </div>
        }
      </ListItem>
    </List>
  </div>
)

```

```

      : <Button
        disableElevation="true"
        disableFocusRipple="true"
        disableRipple="true" className="nav"
        onClick={() => setSignIn(true)} >Sign In</Button>
    }
  </ListItem>
</List>
<List>

  </List>
</div>
);

const container = window !== undefined ? () => window().document.body : undefined;

return (
  <div className={classes.root} >
    <AppBar position="sticky" color="default" className={classes.appBar}>
      <Toolbar >
        <Hidden xsDown >
          <div className={classes.navButton}>
            <Button disableElevation="true" disableFocusRipple="true" disableRipple="true"
>Auctions</Button>
            <Button disableElevation="true" disableFocusRipple="true" disableRipple="true"
className={classes.sellButton}>Sell a Car</Button>
            <Button disableElevation="true" disableFocusRipple="true" disableRipple="true" >About
Us</Button>
            <Button disableElevation="true" disableFocusRipple="true" disableRipple="true" >Daily
Mail</Button>
          </div>
          <div className={classes.navSearch}>

          </div>
          <div>
            {user ? <div> <Button onClick={handleClick}>
              <AccountBoxIcon fontSize='large'></AccountBoxIcon>
            </Button>
              <Menu id="simple-menu" anchorEl={anchorEl} keepMounted open={Boolean(anchorEl)}
onClose={handleClose} className="dropdown" >
                <MenuItem onClick={handleClose} className="dropdown-btn" >
                  <Button href="/settings"> Settings </Button>
                </MenuItem>
                <MenuItem
                  onClick={handleClose}
                  className="dropdown-btn">My
Listings</MenuItem>
                <MenuItem
                  onClick={() => { localStorage.removeItem('user'); setUser(null) }}
                  className="dropdown-btn" id="sign-out" >Sign Out</MenuItem>
              </Menu>
            </div>
            : <Button disableElevation="true" disableFocusRipple="true" disableRipple="true"
className="nav" onClick={() => setSignIn(true)} >Sign In</Button>
          }
          <SignIn openSignIn={openSignIn} setSignIn={setSignIn} ></SignIn>
        </Hidden>
        <IconButton
          color="inherit"
          aria-label="open
drawer"
          edge="start"
onClick={handleDrawerToggle} className={classes.menuButton} >
          <MenuIcon />
        </IconButton>
      </Toolbar>
    </AppBar>
    <nav className={classes.drawer}>

```



```

<Hidden smUp implementation="css">
  <Drawer
    container={container}
    variant="temporary"
    anchor={theme.direction === 'rtl' ? 'right' : 'left'}
    open={mobileOpen}
    onClose={handleDrawerToggle}
    classes={{
      paper: classes.drawerPaper,
    }}
    ModalProps={{
      keepMounted: true, // Better open performance on mobile.
    }}
  >
    {drawer}
  </Drawer>
</Hidden>
</nav>
</div >
)
}

```

```
export default Nav
```

8.5 Codul folosit pentru realizarea paginii setării metodei de plată

```
import React from 'react';
```

```
import { Typography, Divider, TextField, Button } from '@material-ui/core';
```

```
import DriveEtaIcon from '@material-ui/icons/DriveEta';
```

```
import './styles.css';
```

```
export default function SettingsPage() {
```

```
  const [inputValue, setInputValue] = React.useState({ name: "", zip: "", cardNumber: "", date: "", cvc: "",
  phone: "" })
```

```
  const handleSubmit = event => {
```

```
    event.preventDefault()
```

```
    fetch('http://localhost:3001/users/settings', { //@todo api call separate filess
```

```
      method: 'PUT',
```

```
      headers: {
```

```
        'Accept': 'application/json',
```

```
        'Content-type': 'application/json'
```

```
      },
```

```
      body: JSON.stringify({
```

```
        email: localStorage.getItem('user'),
```

```
        name: inputValue.name,
```

```
        zip: inputValue.zip,
```

```
        cardNumber: inputValue.cardNumber,
```

```
        date: inputValue.date,
```

```
        cvc: inputValue.cvc,
```

```
        phone: inputValue.phone,
```

```
      })
```

```
    }).then(res => res.json()).then(res => {
```

```
      console.log(res)
```

```
    })
```

```
  }
```

```
  const onChangeHandler = event => {
```

```

    setInputValue({ ...inputValue, [event.target.name]: event.target.value });
  }
  return (
    <div className="settings">
      <div>
        <Typography variant="h4" align="center">Payment info for bidding</Typography>
        <Divider></Divider>
      </div>
      <div className="details">
        <DriveEtaIcon fontSize="large"></DriveEtaIcon>
        <Typography variant="h5" id="register">Register to bid</Typography>

        <Typography>We require a valid credit card on file before you can bid. Winning bidders pay a 4.5%
        buyer's fee to Cars & Bids on top of the winning bid amount. The minimum buyer's fee is $225, and the
        maximum is $4,500.

```

```

        </Typography>
        <br />
        <Typography> Bids are binding, so please bid wisely!</Typography>
      </div>
      <div className="card">
        <Typography variant="h6">Credit Card Information</Typography>
        <br />
        <div className='card-info'>

          <Typography>Name on card</Typography>
          <TextField
            name="name"
            autoFocus
            variant="outlined"
            size="small"
            fullWidth
            onChange={onChangeHandler}
          ></TextField>
        </div>
        <div className='card-info'>

          <Typography>Zip or Postal Code</Typography>
          <TextField
            name="zip"
            autoFocus
            variant="outlined"
            size="small"
            onChange={onChangeHandler}
          ></TextField>
        </div>
        <div className='card-info'>

          <Typography>Credit Card Number</Typography>
          <TextField
            name="cardNumber"
            autoFocus
            variant="outlined"
            size="small"
            onChange={onChangeHandler}
          ></TextField>
        </div>
        <div className='card-info'>

          <Typography>Expiration</Typography>
          <TextField
            type="date"

```

```

        name="expirationDate"
        autoFocus
        variant="outlined"
        size="small"
        onChange={onChangeHandler}
      ></TextField>
    </div>
    <div className='card-info'>

      <Typography>CVC (3 or 4 digit code)</Typography>
      <TextField
        name="cvc"
        autoFocus
        variant="outlined"
        size="small"
        onChange={onChangeHandler}
      ></TextField>
    </div>
  </div>
  <div className="contact" variant="h6">
    <Typography>Contact Information</Typography>
    <br />
    <div>
      <Typography>Phone Number (in the event you win an auction)</Typography>
      <TextField
        name="phone"
        autoFocus
        variant='outlined'
        size="small"
        onChange={onChangeHandler}
      ></TextField>
    </div>
  </div>
  <div className="btn">

    <Button
      onClick={handleSubmit}
    >Register to bid</Button>
  </div>
</div>
)
}

```

8.6 Codul folosit pentru realizarea modalului de logare

```

import React, { useEffect } from 'react'
import { Dialog, DialogTitle, DialogContent, makeStyles, Typography, TextField, Button, IconButton, Link }
from '@material-ui/core';
//import Controls from "../controls/Controls";
import CloseIcon from '@material-ui/icons/Close';
import { findByLabelText } from '@testing-library/dom';
import FormControl from '@material-ui/core/FormControl';

import SignUpPage from '../SignUpPage'

const useStyles = makeStyles(theme => ({
  dialogWrapper: {
    padding: theme.spacing(2),
    position: 'absolute',
    top: theme.spacing(4),
    borderRadius: "5px"
  },
  dialogTitle: {

```

```

      textAlign: 'center',
      marginBottom: "20px"
    },
    accountBtn: {
      '&:hover': {
        background: "#ffcc00"
      },
      background: "#e6b800",
      width: "200px",
      borderRadius: "10px",
      marginTop: "30px",
      marginBottom: "30px",
      marginLeft: "auto",
      marginRight: "auto",
      textAlign: "center"
    },
    closeBtn: {
      display: "flex",
      justifyContent: "flex-end"
    },
    signUpBtn: {
      color: "#e6b800",
      display: "inline-block",
      marginLeft: "5px"
    }
  }
}))

```

```
export default function SignIn(props) {
```

```

  const { openSignIn, setSignIn } = props;
  const classes = useStyles();
  const [inputValue, setInputValue] = React.useState({ email: "", password: "" })
  const [error, setError] = React.useState({ emailStatus: false, passwordStatus: false, errorMessage: "" });

```

```

  const onChangeHandler = event => {
    setInputValue({ ...inputValue, [event.target.name]: event.target.value });
  }

```

```

  useEffect(() => {
    setError({
      emailStatus: false,
      passwordStatus: false,
      errorMessage: ""
    })
  }, [inputValue]); // Only re-run the effect if count changes

```

```

  const handleSubmit = event => {
    event.preventDefault()
    fetch('http://localhost:3001/users/login', {
      method: 'POST',
      headers: {
        'Accept': 'application/json',
        'Content-type': 'application/json'
      },
      body: JSON.stringify({
        email: inputValue.email,
        password: inputValue.password,
      })
    }).then(res => res.json()).then(res => {
      console.log(res.isAuthenticated)
      if (res.isAuthenticated === true) {
        localStorage.setItem('user', inputValue.email)
      }
    })
  }

```

```

        window.location.reload();
    } else {
        setError({
            emailStatus: res.emailError,
            passwordStatus: res.passwordError,
            errorMessage: res.message
        })
    }
})
}
}

return (
    <div className={classes.root}>
        <form onSubmit={handleSubmit}>
            <Dialog open={openSignIn} maxWidth="lg" classes={{ paper: classes.dialogWrapper }} >
                <div className={classes.closeBtn}>
                    <IconButton>
                        <CloseIcon onClick={() => { setSignIn(false) }} />
                    </IconButton>
                </div>
                <DialogTitle>
                    <div className={classes.dialogTitle}>
                        <Typography variant="h5">
                            Sign in
                        </Typography>
                        <Typography>
                            Need to create an account?
                            <Link className={classes.signUpBtn} color="inherit" href="/signup">Sign up here</Link>
                        </Typography>
                    </div>
                </DialogTitle>
                <DialogContent style={{ overflow: "hidden" }}>
                    <div className={classes.formGroup}>
                        <Typography>
                            >
                            Email address
                        </Typography>
                        <TextField name="email" required autoFocus variant="outlined"
                            fullWidth size="small" onChange={onChangeHandler} error={error.emailStatus}
                            helperText={error.errorMessage.includes("email") ? error.errorMessage : " "}
                        >
                    </TextField>
                </div>
                <div className={classes.formGroup}>
                    <Typography>
                        Password
                    </Typography>
                    <TextField
                        name="password" required autoFocus variant="outlined"
                        fullWidth type="password" size="small" onChange={onChangeHandler}
                        helperText={error.errorMessage.includes("username") ? error.errorMessage : " "}
                    >
                </TextField>
                </div>
                <div className={classes.accountBtn}>
                    <Button
                        disableElevation="true"
                        disableFocusRipple="true"
                        disableRipple="true"
                        fullWidth
                        type="submit"
                        onClick={handleSubmit}

```

```

        >Sign In</Button>
      </div>
    </DialogContent>
  </Dialog>
</form>
</div>
)
}

```

8.7 Codul folosit pentru realizarea paginii de înregistrare

```
import React, { useEffect } from 'react'
```

```
import { Dialog, DialogTitle, DialogContent, makeStyles, Typography, TextField, Button, IconButton,
Container, Avatar } from '@material-ui/core';
import AccountCircleIcon from '@material-ui/icons/AccountCircle';
//import Controls from "../controls/Controls";
import CloseIcon from '@material-ui/icons/Close';
```

```
import SignIn from '../SignIn'
```

```
import './styles.css'
```

```
const useStyles = makeStyles(theme => ({
}))
```

```
export default function SignUpPage() {
```

```
  const classes = useStyles();
  const [inputValue, setInputValue] = React.useState({ email: "", password: "", username: "" })
  const [error, setError] = React.useState({ emailStatus: false, userNameStatus: false, errorMessage: "" });
```

```
  const onChangeHandler = event => {
    setInputValue({ ...inputValue, [event.target.name]: event.target.value });
  }
```

```
  useEffect(() => {
    setError({
      emailStatus: false,
      userNameStatus: false,
      errorMessage: ""
    })
  }, [inputValue]);
```

```
  const handleSubmit = event => {
    event.preventDefault()
    fetch('http://localhost:3001/users/register', {
      method: 'POST',
      headers: {
        'Accept': 'application/json',
        'Content-type': 'application/json'
      },
      body: JSON.stringify({
        email: inputValue.email,
        password: inputValue.password,
        username: inputValue.username
      })
    }).then(res => res.json()).then(res => {
      console.log(res)
    })
  }
```

```

setError({
  emailStatus: res.emailError,
  userNameStatus: res.userNameError,
  errorMessage: res.message
})
console.log(error.emailError)
})
}

return (
  <Container>
    <div className="paper">
      <Avatar className="avatar">
        <AccountCircleIcon />
      </Avatar>
      <Typography><h2>Sign Up</h2></Typography>
      <div>
        <div className="form">
          <Typography>
            >
            Email address
          </Typography>
          <TextField
            name="email"
            required
            autoFocus
            variant="outlined"
            fullWidth
            size="small"
            onChange={onChangeHandler}
            error={error.emailStatus}
            helperText={error.errorMessage.includes("email") ? error.errorMessage : " "}
          >
          </TextField>
        </div>
        <div className="form">
          <Typography>
            Password
          </Typography>
          <TextField
            name="password"
            required
            autoFocus
            variant="outlined"
            type="password"
            size="small"
            onChange={onChangeHandler}
            helperText={error.errorMessage.includes("username") ? error.errorMessage : " "}
          >
          </TextField>
        </div>
        <div className="form">
          <Typography>
            Create a Username (public name that others see)
          </Typography>
          <TextField
            name="username"
            required
            autoFocus
            variant="outlined"
            fullWidth
            size="small"

```

```

      onChange={ onChangeHandler }
      error={ error.userNameStatus }
      helperText={ error.errorMessage.includes("username") ? error.errorMessage : " " }
    >
    </TextField>
  </div>
</div>
<div className="btn">
  <Button
    disableElevation="true"
    disableFocusRipple="true"
    disableRipple="true"
    fullWidth
    type="submit"
    onClick={ handleSubmit }
  >Create account</Button>
</div>
</div>
</Container>
)
}

```

8.8 Codul folosit pentru realizarea paginii de crearea a licitațiilor

```

import { Typography, Divider, TextField, Button } from '@material-ui/core';
import React from 'react';

import './styles.css'

export default function SubmitPage() {
  const [inputValue, setInputValue] = React.useState({ name: "", number: "", year: "", make: "", model: "", vin:
  "", mileage: "", features: "", nr: "", price: " " })
  const [images, setImages] = React.useState([])

  // const fileSelectedHandler = (e) => {
  //   setImages([...e.target.files])
  //   console.log(images, "imagini")
  // }

  const onChangeHandler = event => {
    setInputValue({ ...inputValue, [event.target.name]: event.target.value });
  }

  const handleSubmit = event => {
    const formData = new FormData();
    formData.append('name', inputValue.name);
    formData.append('number', inputValue.number);
    formData.append('year', inputValue.year);
    formData.append('make', inputValue.make);
    formData.append('model', inputValue.model);
    formData.append('vin', inputValue.vin);
    formData.append('mileage', inputValue.mileage);
    formData.append('features', inputValue.features);
    formData.append('nr', images.length)
    formData.append('price', inputValue.price)
    for (let i = 0; i < images.length; i++) {
      formData.append('images', images[i]);
    }

    fetch('http://localhost:3001/cars/submit', { //@todo api call separate filess
      method: 'POST',

```



```

        body: formData
      })
      .then(res => { console.log(res) })
    }
  }

  return (
    <div className="submit">
      <div>
        <Typography variant="h3">Tell us about your car</Typography>
        <Divider></Divider>
      </div>
      <div className="personal-info">
        <Typography variant="h4">Your info</Typography>
        <Typography>Your Name</Typography>
        <TextField
          name="name"
          autoFocus
          variant="outlined"
          size="small"
          onChange={onChangeHandler}
        ></TextField>
        <Typography>Contact Phone Number</Typography>
        <TextField
          name="number"
          autoFocus
          variant="outlined"
          size="small"
          onChange={onChangeHandler}
        ></TextField>
      </div>
      <div className="car-details">
        <Typography>Year</Typography>
        <TextField
          name="year"
          autoFocus
          variant="outlined"
          size="small"
          onChange={onChangeHandler}
        ></TextField>
        <Typography>Make</Typography>
        <TextField
          name="make"
          autoFocus
          variant="outlined"
          size="small"
          onChange={onChangeHandler}
        ></TextField>
        <Typography>Model</Typography>
        <TextField
          name="model"
          autoFocus
          variant="outlined"
          size="small"
          onChange={onChangeHandler}
        ></TextField>
        <Typography>VIN</Typography>
        <TextField
          name="vin"
          autoFocus
          variant="outlined"
          size="small"

```

```

      onChange={onChangeHandler}
    ></TextField>
    <Typography>Mileage(km)</Typography>
    <TextField
      name="mileage"
      autoFocus
      variant="outlined"
      size="small"
      onChange={onChangeHandler}
    ></TextField>
    <Typography>Noteworthy Options/Features</Typography>
    <TextField
      name="features"
      autoFocus
      variant="outlined"
      size="small"
      onChange={onChangeHandler}
    ></TextField>
  </div>

  <div className="price">
    <Typography variant="h4">Starting Proce</Typography>
    <TextField
      name="price"
      autoFocus
      variant="outlined"
      size="small"
      onChange={onChangeHandler}
    ></TextField>
  </div>

  <div className="photos">
    <Typography variant="h4">Photos</Typography>
    <Typography>Please upload at least 4 photos of the exterior and the interior of the car</Typography>
    <form encType="multipart/form-data" >
      <input
        accept=".jpg"
        id="file"
        multiple
        type="file"
        onChange={(event) => { const files = event.target.files; setImages(files) }}
      />
    </form>
    <label htmlFor="contained-button-file">
      <Button variant="contained" color="primary" component="span">
        Upload
      </Button>
    </label>
  </div>
  <div>
    <Button onClick={handleSubmit}>Submit</Button>
  </div>
</div>
)
}

```

8.9 Codul folosit pentru realizarea afişarea întregii aplicaţii

```

import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
import { BrowserRouter } from 'react-router-dom'

```

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

8.10 Codul folosit pentru rutarea paginilor

```
import React from "react";
import NavBar from './components/NavBar'
import { Route, Switch } from 'react-router-dom'

import SignUpPage from "./components/SignUpPage";
import SettingsPage from "./components/SettingsPage";
import SubmitPage from "./components/SubmitPage";
import HomePage from "./components/HomePage";
import CarPage from './components/CarPage';

function App() {
  return (
    <div>
      <NavBar />
      <Switch>
        <Route exact path='/signup' component={SignUpPage} />
        <Route exact path='/settings' component={SettingsPage} />
        <Route exact path='/submit' component={SubmitPage} />
        <Route exact path="/" component={HomePage} />
        <Route exact path='/car' component={CarPage} />
      </Switch>
    </div>
  )
}

export default App;
```

8.11 Schema pentru autovehicule

```
const CarsSchema = mongoose.Schema({
  name: { type: String, required: true },
  number: { type: String, required: true },
  year: { type: String, required: true },
  make: { type: String, required: true },
  model: { type: String, required: true },
  vin: { type: String, required: true },
  mileage: { type: String, required: true },
  features: { type: String, required: true },
  nr: { type: Number },
  date: { type: Date },
  price: { type: String, required: true }
})

module.exports = mongoose.model('Cars', CarsSchema);
```

8.12 Schema pentru utilizatori

```
const mongoose = require('mongoose');
```

```

const UsersSchema = mongoose.Schema({
  email: { type: String, required: true },
  password: { type: String, required: true },
  username: { type: String, required: true },
  name: { type: String, required: true },
  zip: { type: String, required: true },
  cardNumber: { type: String, required: true },
  date: { type: Date, required: true },
  cvc: { type: String, required: true },
  phone: { type: String, required: true },
})

module.exports = mongoose.model('Users', UsersSchema);

```

8.13 Codul pentru crearea rutelor specifice autovehiculelor

```

const express = require('express');
const bcrypt = require('bcryptjs');
const Cars = require('../models/Cars');
const path = require('path')

var multer = require('multer')

var storage = multer.diskStorage({
  destination: function (req, file, cb) {
    cb(null, 'uploads')
  },
  filename: function (req, file, cb) {
    console.log(file.files)

    for (let x = 1; x <= req.body.nr; x++) {
      if (typeof (file) === "object") {
        cb(null, req.body.name + x + '.jpg')
      }
    }
  }
})

var upload = multer({ storage: storage })

const router = express.Router();

router.get('/', async (req, res) => {
  try {
    const cars = await Cars.find();
    res.json(cars);
  } catch (err) {
    res.json({ message: err });
  }
});

router.post('/submit', upload.array('images', 4), async (req, res) => {
  date = new Date();
  date.setDate(date.getDate() + 7)
  const car = new Cars({
    name: req.body.name,
    number: req.body.number,
    year: req.body.year,
    make: req.body.make,
    model: req.body.model,

```

```

    vin: req.body.vin,
    mileage: req.body.mileage,
    features: req.body.features,
    nr: req.body.nr,
    date: date + 7,
    price: req.body.price
  });

  try {
    await car.save();
    res.status(200).json({ message: "ok", statusCode: 200 })
  } catch (err) {
    res.status(400).send(err)
  }
}

router.get('/', async (req, res) => {
  try {
    const cars = await Cars.find();
    res.json(cars);
  } catch (err) {
    res.json({ message: err });
  }
});

router.put("/auction", async (req, res) => {
  var d = new Date(req.body.date); d.setMinutes(d.getMinutes() + 30)
  Cars.findOneAndUpdate({ name: req.body.name },
    {
      $set: {
        price: req.body.newPrice,
        date: d
      }
    },
    { new: true },
    (err, doc) => {
      if (err) {
        console.log("Something wrong when updating data!");
      }

      console.log(doc);
    }
  );
});

module.exports = router;

```

8.14 Codul pentru crearea rutelor specifice utilizatorilor

```

const express = require('express');
const bcrypt = require('bcryptjs');
const Users = require('../models/Users');
const { registerValidation } = require('../models/Validation')

const router = express.Router();

router.get('/', async (req, res) => {
  try {
    const users = await Users.find();
    res.json(users);
  }
}

```

```

    } catch (err) {
      res.json({ message: err });
    }
  });

router.post('/register', async (req, res) => {

  const salt = await bcrypt.genSalt(10);
  const hashedPassword = await bcrypt.hash(req.body.password, salt);

  const emailExist = await Users.findOne({ email: req.body.email });
  if (emailExist) return res.status(400).json({ message: "email already exists", statusCode: 400, emailError: true
  })

  const usernameExist = await Users.findOne({ username: req.body.username });
  if (usernameExist) return res.status(400).json({ message: 'username already exists', statusCode: 400,
  userNameError: true })

  const user = new Users({
    email: req.body.email,
    password: hashedPassword,
    username: req.body.username
  });

  try {
    await user.save();
    res.status(200).json({ message: "ok", statusCode: 200 })
  } catch (err) {
    res.status(400).send(err)
  }
});

router.post('/login', async (req, res) => {
  console.log(req.body)

  const user = await Users.findOne({ email: req.body.email });
  if (!user) return res.status(400).send({ message: "email not found", statusCode: 400, emailError: true })
  //todo    object with message and statusCode

  const validPass = await bcrypt.compare(req.body.password, user.password);
  if (!validPass) return res.status(400).send({ message: "wrong password", statusCode: 400, passwordError:
  true })

  res.status(200).send({ isAuthenticated: true })
})

router.put('/settings', async (req, res) => {
  console.log(req.body);

  Users.findOneAndUpdate({ email: req.body.email },
    {
      $set: {
        name: req.body.name,
        zip: req.body.zip,
        cardNumber: req.body.cardNumber,
        date: req.body.date,
        cvc: req.body.cvc,
        phone: req.body.phone,
      }
    },
    { new: true },

```

```

(err, doc) => {
  if (err) {
    console.log("Something wrong when updating data!");
  }

  console.log(doc);
});
})

```

```
module.exports = router;
```

8.15 Codul pentru crearea serverului offline

```

const express = require('express');
const app = express();
const bodyParser = require('body-parser');
const cors = require('cors');
var multer = require('multer');
var fs = require('fs');
const path = require('path')

require('./initDB')();

app.use(cors());

const UsersRoute = require('./routes/users');
const CarsRoute = require('./routes/cars');

app.use('/users', UsersRoute);

app.use('/cars', CarsRoute);

app.use("/image", express.static(path.join(__dirname, "uploads")));

app.listen(3001);

```

8.16 Codul pentru conectarea bazei de date la serverul offline

```

const mongoose = require('mongoose');

module.exports = () => {
  mongoose.connect('mongodb://localhost/licenta', {
    useNewUrlParser: true,
    useUnifiedTopology: true
  })
  .then(() => {
    console.log("Mongodb connected...")
  });
}

```

9.1 Cv-ul autorului

Robert Szucs

Data nașterii: 06/09/1996 | Cetățenie: română | Gen Masculin | (+40) 748154429

szucs.robert96@gmail.com

str. Fagetului, nr. 18, 450121, Zalau, România

Despre mine: I am a persevering person, constantly looking for activities that would enhance my work qualities and my personality. I am focused on learning more about web development, as my previous experience such as completing tutorials on "Pluralsight" and "w3schools", made me interested in JavaScript, HTML and CSS. I am sure that my tenacious and precise manner of conduct is matching accordingly with this internship.

● EDUCAȚIE ȘI FORMARE PROFESIONALĂ

01/10/2015 – ÎN CURS

STUDENT – Facultatea de Electronică, Telecomunicații și Tehnologia Informației

● COMPETENȚE LINGVISTICE

Limbă(i) maternă(e): ROMÂNĂ

Altă limbă(Alte limbi):

	COMPREHENSIUNE		VORBIT		SCRIS
	Comprehensiune orală	Citit	Exprimare scrisă	Conversație	
ENGLEZĂ	B1	B1	A2	A2	B1

Niveluri: A1 și A2 Utilizator de bază B1 și B2 Utilizator independent C1 și C2 Utilizator experimentat

● COMPETENȚE DIGITALE

Microsoft Office Navigație Internet Social Media HTML/CSS/JavaScript - mediu level
Cunoștiințe de baza în Javascript, React și Redux