

SAPIENTIA ERDÉLYI MAGYAR TUDOMÁNYEGYETEM
MAROSVÁSÁRHELYI KAR,
INFORMATIKA SZAK



SAPIENTIA
ERDÉLYI MAGYAR
TUDOMÁNYEGYETEM

Web felület adószámlák generálására és kezelésére

DIPLOMADOLGOZAT

Témavezetők:
Drd. Csaholczi Szabolcs,
Egyetemi tanársegéd
Dr. Iclănzan David,
Egyetemi docens

Végzős hallgató:
Kőműves Dávid-Márk

2023

UNIVERSITATEA SAPIENTIA DIN CLUJ-NAPOCA
FACULTATEA DE ȘTIINȚE TEHNICE ȘI UMANISTE,
SPECIALIZAREA INFORMATICĂ



UNIVERSITATEA
SAPIENTIA

Platformă web pentru generarea și gestionarea facturilor fiscale

LUCRARE DE DIPLOMĂ

Coordonatori științifici:
Drd. Csaholczi Szabolcs,
Asistent universitar
Dr. Iclănzan David,
Conferențiar universitar

Absolvent:
Kőműves Dávid-Márk

2023

**SAPIENTIA HUNGARIAN UNIVERSITY OF
TRANSYLVANIA
FACULTY OF TECHNICAL AND HUMAN SCIENCES
COMPUTER SCIENCE SPECIALIZATION**



SAPIENTIA
HUNGARIAN UNIVERSITY
OF TRANSYLVANIA

Web platform for generating and managing tax invoices

BACHELOR THESIS

Scientific advisors:
Drd. Csaholczi Szabolcs,
Assistant professor
Dr. Iclănzan David,
Associate professor

Student:
Kőműves Dávid-Márk

2023

Declarație

Subsemnatul/a KÖNÖVES DAVID-MÁRK, absolvent(ă) al/a specializării
INFORMATICA, promoția 2023 cunoscând
prevederile Legii Educației Naționale 1/2011 și a Codului de etică și deontologie profesională a
Universității Sapientia cu privire la furt intelectual declar pe propria răspundere că prezenta lucrare
de licență/proiect de diplomă/disertație se bazează pe activitatea personală, cercetarea/proiectarea
este efectuată de mine, informațiile și datele preluate din literatura de specialitate sunt citate în
mod corespunzător.

Localitatea, COLUNCA

Data: 15.06.2023.

Absolvent

Semnătura.....Könöves David-Márk.....

LUCRARE DE DIPLOMĂ

Îndrumător:
Csaholczi Szabolcs
Coordonator științific:
dr. Iclanzan David

Candidat: **Kőműves Dávid**
Anul absolvirii: 2023

a) Tema lucrării de licență:

Platformă web pentru generarea și gestionarea facturilor fiscale

b) Problemele principale tratate:

- Studiu bibliografic privind limbajele Node.js și Vue.js
- Explorarea bazei de date MongoDB nerelaționale și utilizarea bibliotecilor Mongoose pentru a manipula datele
- Documentarea adecvată a stadiilor de proiectare a aplicațiilor

c) Desene obligatorii:

- Schema bloc a sistemului
- Diagrame de proiectare pentru aplicația software realizată.

d) Softuri obligatorii:

- Aplicația este bazată pe tehnologia Node.js și Vue.js
- Scopul acestei teze este de a genera și stoca facturi fiscale. Aplicația poate fi făcută foarte accesibilă datorită suportului web. Utilizatorii care au un cont pot, prin intermediul unei arhive, să vizualizeze facturile deja generate și statisticile acestora. Acest lucru reduce timpul ocupat de birocrație.
- Datele interfeței sunt stocate într-o bază de date MongoDB, care ar trebui să fie gestionată de un API REST bazat pe Node.js, folosind bibliotecile Express.js sau Mongoose pentru a accesa datele din baza de date - aceasta ar fi partea de "backend" a proiectului. Partea frontend a proiectului ar fi implementată folosind cadrul Vue.js și PrimeVue.

e) Bibliografia recomandată:

- [1] <https://vuejs.org/guide/introduction.html>
- [2] <https://expressjs.com/en/starter/installing.html>
- [3] <https://nodejs.org/en/docs>

f) Termene obligatorii de consultații: săptămânal, preponderent online

g) Locul și durata practicii: Universitatea „Sapientia” din Cluj-Napoca,
Facultatea de Științe Tehnice și Umaniste din Târgu Mureș, (sala 327 B)
Primit tema la data de: 01.10.2022
Termen de predare: 30.06.2023

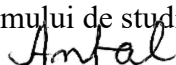
Semnătura Director Departament



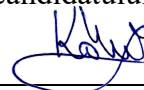
Semnătura coordonatorului



Semnătura responsabilului
programului de studiu



Semnătura candidatului



Kivonat

Napjainkban egyre inkább nő a magánvállalkozók és vállalkozások száma. Normalizálódott a cégvezetés ötlete, már nem tűnik úgy, hogy csak a Steve Jobs vagy Bill Gates kaliberei közé sorolhatók szerepelhetnek egy cég élén, így egyre több fiatal próbálkozik és egyre több új vállalkozás születik. Viszont egy modern vállalkozás optimális működéséhez megfelelő eszközök szükségesek, melyek segíthetnek bizonyos viselkedések megfigyelésében, létrehozni egy sikeres kliens-szolgáltató kommunikációs felületet, segítenek a vállalkozás erőforrásainak a menedzsmentjében, vagy akár lerövidítik a papírmunka által felvett időt.

Dolgozatom célja, hogy a felsoroltak közül találjon az utóbbira egy egyszerű, gyors és vonzó megoldást, pontosabban az adószámlák generálására és tárolására. Annak fényében, hogy egy szolgáltatást vagy terméket áruló vállalkozásnak szüksége van számlakivonatok létrehozására, az általam létrehozott alkalmazás egy garantált megoldás szeretne lenni olyan vállalkozóknak, akiknek szüksége van egy olyan felületre, ami nem csak létrehozza, hanem számon tartja ezen kivonatokat.

Az alkalmazás magas szinten elérhetővé tehető a webes közegének köszönhetően. Létrehoztam egy könnyen átlátható sémát a számlák szerkesztésére és elmentésére. Az alkalmazás helyet hagy olyan felhasználóknak is, akik nem kívánnak bejelentkezni, viszont a fiókkal rendelkező felhasználóknak egy archívummal szolgál, mely nem csak a már létrehozott számlák szemléltetésében segít, hanem ezeknek a statisztikai feltüntetésében.

A felület adatait egy MongoDB adatbázisban tároltam, melynek kezelésére egy Node.js alapú REST API-t építettem, Express.js, illetve Mongoose könyvtárak segítségével - ezt nevezném meg a projekt "backend" részének. A projekt "frontend" részét pedig Vue.js 3 JavaScript keretrendszer segítségével valósítottam meg.

Kulcsszavak: adószámla generálás és kezelés, web felület, REST API, MongoDB, Node és Vue.

Rezumat

În prezent, numărul antreprenorilor și al întreprinderilor private este în creștere. Ideea de a conduce o afacere s-a normalizat, nu mai pare că doar oameni de talia lui Steve Jobs sau Bill Gates pot conduce o companie, astfel, tot mai mulți tineri încearcă și tot mai multe afaceri noi sunt create. Cu toate acestea, funcționarea optimă a unei afaceri moderne necesită instrumente adecvate care să ajute la monitorizarea anumitor comportamente, să creeze o interfață de comunicare client-prestator de succes, să ajute la gestionarea resurselor afacerii sau chiar să scurteze timpul ocupat de birocrație.

Scopul tezei mele este de a găsi o soluție simplă, rapidă și atractivă pentru ultimul dintre acestea, mai exact pentru generarea și stocarea facturilor fiscale. Având în vedere necesitatea ca o afacere care vinde servicii sau produse să genereze extrase de factură, aplicația pe care am creat-o se dorește a fi o soluție garantată pentru antreprenorii care au nevoie de o interfață care nu numai că generează, ci și stochează aceste extrase.

Aplicația este foarte accesibilă datorită platformei sale pe bază web. Am creat o schemă ușor de utilizat pentru editarea și salvarea facturilor. Aplicația lasă loc și pentru utilizatorii care nu doresc să se logheze, dar pentru utilizatorii care au cont, oferă o arhivă care ajută nu numai la vizualizarea facturilor deja create, ci și la evidențierea statistică a acestora.

Am stocat datele din interfață într-o bază de date MongoDB, care este gestionată de un REST API bazat pe Node.js folosind Express.js și Mongoose - aș numi acest lucru backend-ul proiectului. Partea frontend a proiectului a fost implementată folosind cadrul JavaScript Vue.js 3.

Cuvinte cheie: generarea și gestionarea facturilor fiscale, interfață web, REST API, MongoDB, Node și Vue.

Abstract

Today, the number of private entrepreneurs and businesses is growing. The idea of running a business has become more normalised, it no longer seems that only someone of the calibre of Steve Jobs or Bill Gates can run a company, so more and more young people are trying and more new businesses are being born. However, the optimal functioning of a modern business requires the right tools to help monitor certain behaviours, create a successful client-provider communication interface, help manage the resources of the business, or even shorten the time taken up by paperwork.

The aim of my thesis is to find a simple, fast and attractive solution for the latter of these, more specifically for the generation and storage of tax invoices. The application I have created aims to be a guaranteed solution for businesses and entrepreneurs that sell services or products and need to generate invoice statements.

The application can be made highly accessible thanks to its web-based format. I have created an easy-to-use schema for editing and saving invoices. The application also leaves room for users who do not wish to log in, but for users with an account, it provides an archive that helps not only to visualise the already created invoices, but also to highlight them statistically.

The interface data is stored in a MongoDB database, which is managed by a Node.js based REST API using Express.js and Mongoose libraries - I would call this the backend of the project. The frontend part of the project was implemented using Vue.js 3 JavaScript framework.

Keywords: tax invoice generation and management, web interface, REST API, MongoDB, Node and Vue

Tartalomjegyzék

Ábrák jegyzéke	11
Táblázatok jegyzéke	12
1. Bevezető	13
2. Elméleti megalapozás és szakirodalmi tanulmány	14
2.1. Elméleti fogalmak	14
2.1.1. Web közeg	14
2.1.2. Backend és Frontend	14
2.1.3. REST API	15
2.2. Adatbázis	15
2.3. Beépített könyvtárak, keretrendszerek	16
2.3.1. Node.js	16
2.3.2. Express.js	17
2.3.3. Mongoose	17
2.3.4. Vue	17
2.3.5. PrimeVue és stilizálás	19
2.3.6. Chart.js, domtoimage, jsPDF	19
2.4. Felhasznált technológiák, eszközök	19
2.4.1. Microsoft Visual Studio Code	19
2.4.2. GitHub	20
2.4.3. MongoDB Compass	20
2.4.4. Creately	20
2.4.5. Adattitkosítás - jelszó hashelés	20
3. A rendszer specifikációja	22
3.1. Felhasználói követelmények	22
3.2. Rendszerkövetelmények	24
3.2.1. Funkcionális követelmények	24
3.2.2. Nem funkcionális követelmények	24
4. Tervezés	25
4.1. Az adatbázis szerkezete	25
4.1.1. Users tábla	26
4.1.2. Invoices tábla	27
4.2. A backend szerkezete	28

4.2.1.	Mongoose modellek	29
4.2.2.	Express Routerek	30
4.3.	A frontend szerkezete	32
4.3.1.	Fő komponensek	32
4.3.2.	Router modul	33
4.3.3.	“Support” mappa	34
4.3.4.	“Auth” mappa	35
4.3.5.	“Pages” mappa	35
5.	Kivitelezés	37
5.1.	Futtatás	37
5.2.	Felhasználói felület komponensei	38
	Összefoglaló	45
	Köszönetnyilvánítás	46
	Irodalomjegyzék	47

Ábrák jegyzéke

2.1. REST API elméleti működése - szemléltetés	15
2.2. MongoDB dokumentum példa	16
2.3. Egy Express.js GET route deklarációja	17
2.4. Vue példa Options API struktúrára	18
2.5. Projektem felosztása, fejlesztésem során - Visual Studio Code	20
3.1. Használati eset diagram	23
4.1. Táblák és sémák kapcsolata	26
4.2. "Company" Mongoose séma deklarációja	27
4.3. A backend részen használt fájl struktúra	29
4.4. User séma függvényei, lifecycle hook	29
4.5. A frontend részen használt fájl struktúra	32
4.6. App.vue tartalma	33
4.7. Oldalak folyamatábrája (Flow chart)	34
5.1. REST API parancssori futtatása	37
5.2. Felhasználói felület parancssori futtatása	37
5.3. A felület kezdőlapja	38
5.4. Header – első feltüntetés	38
5.5. Header – második feltüntetés	38
5.6. Üzenetdoboz	39
5.7. Bejelentkezési form	39
5.8. Számla készítő oldal	40
5.9. Számlán szereplő ügyfél adatainak módosítása	40
5.10. Számlán szereplő termékek adatainak módosítása	41
5.11. Számlán szereplő termékek adatainak módosítása	41
5.12. Felhasználó adatait megjelenítő kártyák	41
5.13. Céges adatokat módosító menü	42
5.14. Profil oldal alsó fele – ügyfél adatok módosítása	42
5.15. Archivum	43
5.16. Egy számla részletesebb adatainak menüje	43
5.17. Egy felhasználó eladásainak hisztogramja	44

Táblázatok jegyzéke

4.1. Számlakezelésre vonatkozó REST API címek	30
4.2. Felhasználók kezelésére vonatkozó REST API címek	31

1. fejezet

Bevezető

Az egyre gyorsabban fejlődő világunkban egyre növekszik a versengés az újdonsült vállalkozások között. Olyan termékeket vagy módszereket keresnek, amelyek felgyorsítanak vagy minimalizálnak hétköznapi folyamatokat, hogy sokkal hamarabb tudjanak a következő ügyfél szükségleteivel foglalkozni. Az iparjaink egyre komplexebb feladatokat automatizálnak, mert rövidít a termelés folyamatán, úgy, hogy csökken a szükség emberi erőforrásokra.

Ez a folyamat, úgy gondolom, hogy a világ folytonos haladásra való törekvésének része. Az emberi történelmet megfigyelve megbizonyosodhatunk arról, hogy egyre többet és egyre gyorsabban szeretnénk haladni, akár az ipari forradalomra tekintünk, akár a jelenben kibontakozó mesterséges intelligenciák korszakát. Annak ellenére, hogy a termelés nő, néha ennek gyorsítása negatív hátrányokkal jár, mint például a munkalehetőségek felszívódása.

Ezen gondolatokkal és ezen értékrenddel egyezően állítottam ki célnak azt, hogy egyszerre egy olyan terméket állítsak elő, amely megegyezik az előrehaladás univerzális akaratával, de nem zár ki munkalehetőségeket ennek folyamatában, hanem könnyíti azoknak a feladatát, akiknek ezzel kell foglalkozniuk. Annak tekintetében, hogy a papírmunka egy lassító tényező lehet egy cég munkamenetében, az eladással kapcsolatos dokumentumok létrehozásakor, a dolgozatom az adószámlák és hasonló papírok generálásával és számontartásával szeretne könnyíteni mindenkin, akinek erre szüksége születik. A webes közegnek köszönhetően pedig magas szinten elérhető és rengeteg felhasználót tud kiszolgálni.

Az alkalmazás segíteni törekszik azoknak is, akik először használják a szolgáltatást vendégként, viszont főleg a visszatérő felhasználóknak, akik fiókot hoznak létre. A fiókkal a számlákra felkerülő kliens - és termékadatok is egyaránt eltárolhatóak, illetve számon vannak tartva a már létrehozott számlák is, melyeket újra fel lehet használni. Ezek mellett, az alkalmazás kiemeli a számlán szereplő termékek eladásának történelmét a felhasználó csatlakozása óta.

Ezen eszközök által, úgy érzem, hogy sikeresen hozzájárulhatok számtalan vállalkozás eladási folyamatába, egy olyan környezettel, amit könnyű navigálni és segít az adattárolásban.

2. fejezet

Elméleti megalapozás és szakirodalmi tanulmány

2.1. Elméleti fogalmak

A projekt komplexitásának átlátásához, szükségesnek találom felsorolni egy pár alapvető fogalmat.

2.1.1. Web közeg

Általánosan, egy weboldal elérhetővé tételéhez több elemre van szükség, melyek biztosítják az adatok tárolását, feldolgozását, szolgáltatását, illetve lehetővé teszik a publikus elérhetőségét. Az elérhetőséget biztosító elemek közé tartozik a Domainnév, mely egy egyedi webcím, amely azonosítja és megkülönbözteti a webhelyet a többi weboldaltól (pl. "invoicelab.ro"), a Domainnév-szerver (DNS), amely a domainnév és a weboldal IP címe közötti kapcsolatot teremti meg, illetve egy webtárhely, ami a szoftver futtatását biztosítja a megemlített IP címen.

Ezen fogalmak főleg azért kerülnek megemlítésre, mert a projektem nem foglalja magába. Az általam épített alkalmazás az adatok tárolását, feldolgozását és szolgáltatását biztosítja, viszont nem egy publikusan elérhető eszköz. A web közeg fogalma a dolgozat keretén belül azokra a technológiákra utal, amelyek a webfejlesztés környezetéhez tartoznak. A szoftver teljes összetétele pedig egy adatbázis, egy "backend" rész, mely a szerveroldali tevékenységekkel foglalkozik, illetve egy "frontend" rész, amellyel közvetlen módon kommunikálhat a felhasználó.

2.1.2. Backend és Frontend

A "backend" a webfejlesztésben a projekt azon része, mely adatbázis-kezelő műveletek elvégzésével, az adat - felhasználó kapcsolat biztosításával, illetve háttér logikával foglalkozik. Ez az alkalmazás általában a szerver oldalon fut, hogy biztosítva legyen az adatok gyors feldolgozása.

A "frontend" rész biztosít egy közvetlen felületet a felhasználó és a szoftver szolgáltatásai között. Ez a partíció kliens oldalon fut, a felhasználó böngészőjében.

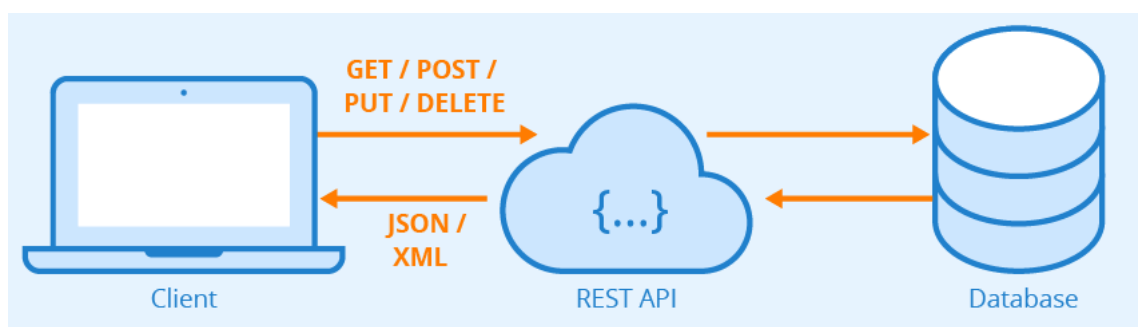
A kettő kapcsolata az úgynevezett “kliens-szerver” architektúra struktúráját követi, miszerint a “frontend” kérést küld a “backend”-nek, ami majd visszaküldi a feldolgozott adatot, hogy megjelenítésre kerüljön [Olu14], leggyakrabban JSON vagy XML alakban.

2.1.3. REST API

Egy REST API (Representational State Transfer Application Programming Interface) egy elterjedt és népszerű architektúráis stílus a webes szolgáltatások tervezésében és fejlesztésében. Lehetővé teszi a kommunikációt és az adatcserét különböző rendszerek között olyan HTTP protokollok segítségével, mint a GET, POST, PUT, DELETE [Sur16]. A REST API az erőforrások központjában áll, amelyeket egyedi URL-eken (Uniform Resource Locator) lehet meghívni. A dolgozat esetében, például a “/invoices” cím a számlák adatainak szolgáltatására van felhasználva.

A REST API az állapot nélkülség elvére épül, miszerint a kliensadatok nincsenek tárolva a lekérdezések között, és minden egyes kérés különálló.

Az általa elküldött adat felismerhető és feldolgozható kell legyen, így a válaszok JSON formátumban vannak szolgáltatva, az adatok strukturált reprezentációja és könnyebb feldolgozása érdekében.



2.1. ábra. REST API elméleti működése - szemléltetés

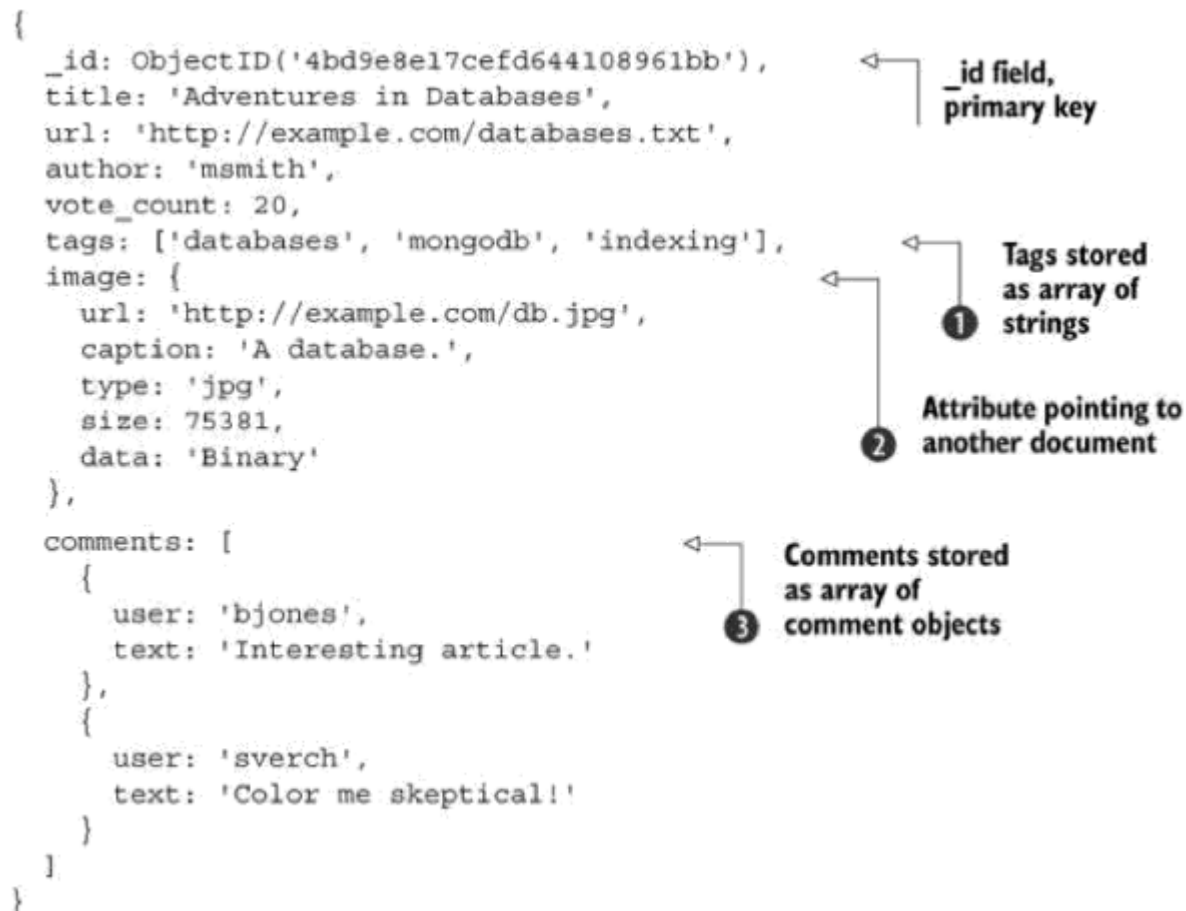
A REST API bevezetése jót jelent adatfeldolgozás szempontjából is. Bármikor skálázható és bővíthető, így az általam épített “backend” program nem csak a meglévő frontend alkalmazás adatokkal való szolgáltatásában segíthet, hanem más harmad-fél szoftverek is felhasználhatják statisztikai felmérésekre vagy esettanulmányokra.

2.2. Adatbázis

A MongoDB egy olyan adatbázis-kezelő rendszer, amelyet webes alkalmazások és internetes infrastruktúra gyors fejlesztésére terveztek. Az adatmodell és a perzisztencia-stratégiák a nagy írási és olvasási sebességre és az automatikus hibaelhárítással való könnyű skálázhatóságra épülnek. Akár egy adatbázis-csomópontra, akár több tucatra van szükség, a MongoDB meglepően jó teljesítményt tud nyújtani [BGBV16].

Az adatbázisok táblákból állnak, melyek dokumentumokat tartalmaznak. Fontos megjegyezni, hogy az adattárolás nem-relációs, ami azt jelenti, hogy nem a megszokott sor és oszlop struktúrát követi, hanem olyan tárolási modellt használ, amely a tárolt adattípus speciális követelményeire van optimalizálva.

Az adatbázis lokálisan lett létrehozva.



2.2. ábra. MongoDB dokumentum példa

2.3. Beépített könyvtárak, keretrendszerek

2.3.1. Node.js

Node.js egy nyílt forráskódú, platformfüggetlen, aszinkron, eseményvezérelt JavaScript futtatókörnyezet, amely lehetővé tesz skálázható hálózati alkalmazások építését. Különleges futási módszere van a Node.js alkalmazásoknak, mivel elkerüli az I/O blokkolást és mikor nincs feladat, alszik. Ez ellentétben áll a manapság elterjedtebb párhuzamossági modellel, amelyben az operációs rendszer szálait használják. A szálalapú hálózatépítés viszonylag kevésbé hatékony és nehezen használható. Továbbá, a Node.js felhasználói nem kell aggódjanak a folyamat holtponthoz lezárása miatt, mivel nincsenek zárok. A Node.js-ben szinte egyetlen függvény sem végez közvetlenül I/O-t, így a folyamat soha nem blokkol. Így válik egyszerűvé a skálázható rendszerek fejlesztése. A HTTP első osztályú polgár a Node.js-ben, ezt a streaming és az alacsony késleltetés szem előtt tartásával tervezték. Ezek által, a Node.js egy jó választás lehet webes könyvtárak vagy keretrendszerek alapjául [Fc].

Emellett, A Node beépített csomag kezelőjének, az npm-nek segítségével számos modul telepíthető bármilyen alkalmazáshoz [MDK15]. Ez által telepítettem a “backend” és “frontend” könyvtárait egyaránt.

Megemlíteném még a Nodemon könyvtárat, mely kissé felgyorsítja a Node alkalmazások tesztelését, mivel minden elmentett változásnál újraindítja az alkalmazást, illetve a dotenv könyvtárat, mely segít a környezeti változók (environmental variables) sikeres használatában.

2.3.2. Express.js

Az Express.js egy ingyenes, nyílt forráskódú backend-keretrendszer a Node.js számára, amely a Node http alap modulján és a Connect komponensek összetételén alapszik [Mar18]. Enterprise szinten is rendszeresen használatba kerül megbízhatóságának köszönhetően. Gyakran van API létrehozásra használva, tekintve, hogy “session”, “routing” és hibakezelésről gondoskodik. Számomra a “routing” aspektus volt a legfontosabb, hogy létrehozassak olyan címeket, melyek megfelelőképpen szolgálják az adatot a “frontend”-nek.

Az Express.js által használt “routing” eszközök konkrét címeket deklarálnak az adatlekérésre vagy rendezésre és képesek adatot vissza is küldeni. Hozzáférésük van a “req” (request) és “res” (response) változókhoz, melyek paramétereket is tartalmazhatnak meghíváskor és értékeket küldhetnek vissza. A paraméterek a “frontend” által meghívott URL-ben találhatóak. Illetve, felhasználhatóak “middleware” függvények, hogy rövidítsék a függvény által felvett időt.

```
router.get('/:id', getUserById, (req, res) =>{
```

2.3. ábra. Egy Express.js GET route deklarációja

2.3.3. Mongoose

A Mongoose egy teljesen kifejlesztett objektum relációs leképező (ORM) könyvtár a Node.js és a MongoDB számára. Az ORM használatának számos előnye van, melyek messze túlmutatnak a kódszervezés vagy a fejlesztés egyszerűségén [MA14].

A projekt számára óriási szerepet játszik, mivel megteremti az Express REST API és a MongoDB közötti kapcsolatot. A Mongoose könyvtár képes létrehozni különböző sémákat, mely a tárolandó adatok struktúrájának felelnek meg. Minden ilyen sémára pedig külön táblát fog létrehozni az adatbázisunkban. Ráadásul, beépített függvényekkel szolgál, hogy ezekben a táblákban keresni tudjunk bármelyik jellemzőjük által, illetve lehetőséget ad arra, hogy a sémákhoz is kapcsolhassunk egyedi függvényeket.

2.3.4. Vue

A Vue egy JavaScript keretrendszer felhasználói felületek építésére. A szabványos HTML, CSS és JavaScript alapokra épül, és egy deklaratív és komponens alapú programozási modellt biztosít, amely segít a felhasználói felületek hatékony fejlesztésében, legyenek azok egyszerűek vagy összetettek. A Vue legfontosabb jellemzői közé tartoznak

a deklaratív renderelés, mely egy sablon szintaxissal bővíti a szabványos HTML-t, ezáltal lehetővé téve a HTML-kimenet deklaratív leírását a változók állapota alapján, illetve a reaktivitás, ami lehetővé teszi, hogy a keretrendszer automatikusan kövesse a JavaScript állapotváltozásokat, és hatékonyan frissítse a DOM-ot, amikor a változások megtörténnek [You].

A keretrendszer lehetővé tesz két kód strukturálási lehetőséget: “Composition API” és “Options API”. A kettő között az a különbség, hogy amíg az egyik törekszik mindent egy helyen elrendezni, a másik kategóriákra osztva segíti a felhasználót megépíteni alkalmazását. A projektem megépítése során az “Options API” struktúrát választottam az átláthatóság érdekében.

```
<script>
export default {
  // Properties returned from data() become reactive state
  // and will be exposed on `this`.
  data() {
    return {
      count: 0
    }
  },

  // Methods are functions that mutate state and trigger updates.
  // They can be bound as event handlers in templates.
  methods: {
    increment() {
      this.count++
    }
  },

  // Lifecycle hooks are called at different stages
  // of a component's lifecycle.
  // This function will be called when the component is mounted.
  mounted() {
    console.log(`The initial count is ${this.count}.`)
  }
}
</script>

<template>
  <button @click="increment">Count is: {{ count }}</button>
</template>
```

2.4. ábra. Vue példa Options API struktúrára

A keretrendszer helyet ad, hogy a HTML elemek fejléceibe szúrjunk feltételeket vagy ciklus vezérlőket - ezt a folyamatot feltételes vagy listarenderelésnek nevezzük. Ez azt jelenti, hogy lehetséges közvetett módon összekötni egy változó állapotát bármilyen elemmel és mivel jellemző a reaktivitás, automatán alkalmazkodik az oldal minden változáshoz.

Vue biztosít élethciklus horgokat (lifecycle hook), melyek segítenek a futtatási, renderelési és a DOM által véghezvitt élethciklusokat követni és ezek által kódot futtatni.

Emellett, esély van felhasználni a “watch” függvényeket, melyek, amint nevük sugallja, követik egy változó vagy változókból álló tömbök változásait és azok hatására futtatnak kódot.

A projektem tervezése alatt a fentiekben megemlített funkciók kerültek leginkább használatra.

2.3.5. PrimeVue és stilizálás

A PrimeVue egy népszerű, nyílt forráskódú UI komponens könyvtár a Vue.js számára. Testreszabható és újrafelhasználható komponensek gazdag készletét kínálja, melyek modern és esztétikus webalkalmazások interfészének megépítésében segíthetnek.

Az “npm” csomagkezelő segítségével könnyű telepíteni és gyorsan a projekthez köthető a masszív könyvtár, majd a Vue komponensekbe bármikor importálhatóak a modellek.

Projektem fejlesztése során felhasználtam egy pár ilyen modellt az adatok tisztább megjelenítésének érdekében, illetve a könnyen implementálható ikonkönyvtárt, PrimeIcons, viszont nagyobb részben van jelen az egyszerű CSS által történő stilizálás.

2.3.6. Chart.js, domtoimage, jsPDF

A “Chart.js” egy nyílt forráskódú JavaScript könyvtár, amely lehetővé tesz interaktív és vizuálisan esztétikus grafikonok és diagramok létrehozását weboldalakon. A PrimeVue által felhasznált hisztogram sablon működéséhez szükséges.

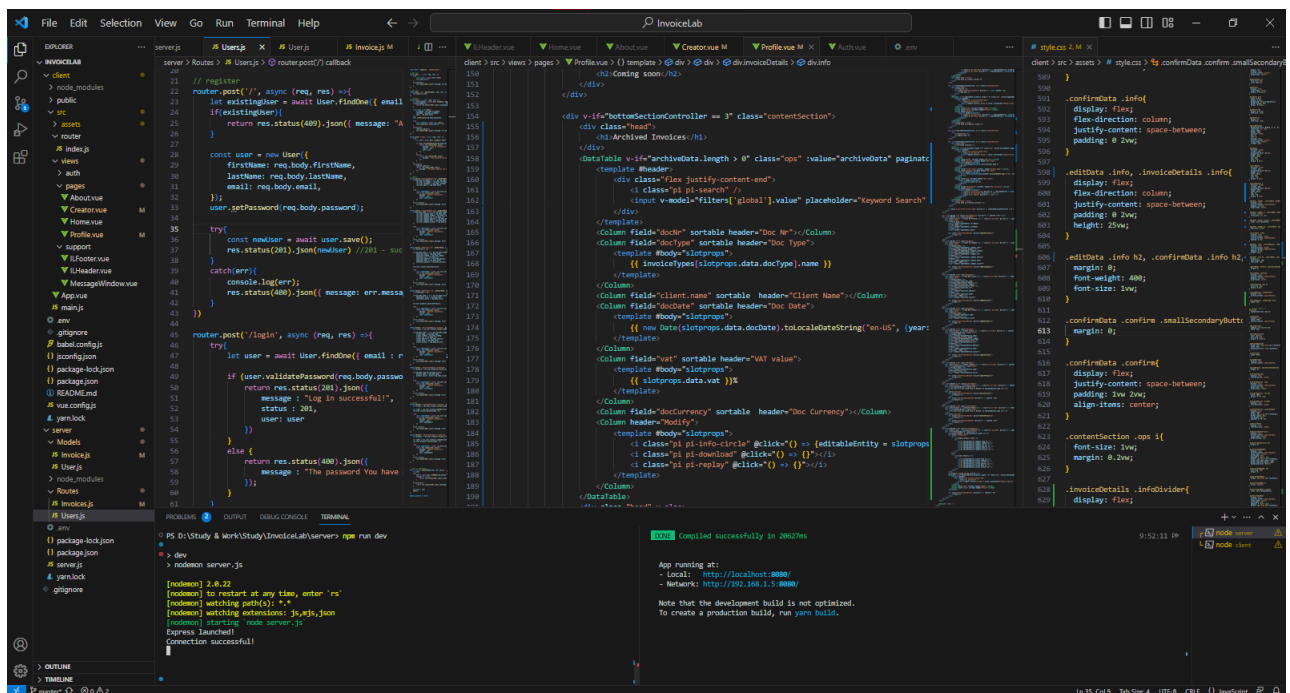
A “domtoimage” könyvtár tetszőleges DOM csomópontokat képes átalakítani képeké és szorosan együttműködik a “jsPDF” könyvtárral, amely lehetővé teszi PDF dokumentumok generálását a böngészőben. A “domtoimage” könyvtár képet készít az elmentendő számláról, majd PDF formátumra helyezi, hogy a dokumentum elmentésére kerülhessen sor.

2.4. Felhasznált technológiák, eszközök

2.4.1. Microsoft Visual Studio Code

A Visual Studio Code egy egyszerű, de nagy teljesítményű forráskód-szerkesztő, amely asztali eszközökön fut, és Windows, macOS és Linux operációs rendszereken érhető el. Beépített támogatással rendelkezik a JavaScript, a TypeScript és a Node.js számára, és más nyelvekhez (például C++, C#, Java, Python, PHP, Go, .NET) készült bővítmények gazdag ökoszisztémájával rendelkezik. [\[Mic\]](#)

Az editor magas szintű testreszabhatóságának köszönhetően egyszerre figyelhettem a szoftverem által minden megtett lépést, illetve letisztult és gyors módon férhettem hozzá bármelyik fő részhez vagy komponenshez.



2.5. ábra. Projektem felosztása, fejlesztésem során - Visual Studio Code

2.4.2. GitHub

A GitHub [Git] egy web alapú platform, amely Git alapú verziókezelési és együttműködési lehetőségeket biztosít a szoftverfejlesztési projektek számára. Git-tárhelyek (repositories) tárolására szolgál, amelyek a forráskódban bekövetkezett változások nyomon követésére és a közös fejlesztési munkafolyamatok kezelésére szolgálnak.

2.4.3. MongoDB Compass

A Compass egy ingyenes, interaktív eszköz a MongoDB-adatok lekérdezésére, optimalizálására és elemzésére. Projektem fejlesztése alatt az adatok adatbázisban való megtekintésére és ellenőrzésére használtam.

2.4.4. Creately

A Creately egy ingyen használható diagram készítő weboldal, mely magas szintű HR tervezéstől, egyszerű eset diagramokig, minden alapon lefedheti a adatvizualizálás kérdését. A dolgozatomban megírása alatt felhasználtam szolgáltatásait, hogy flexibilis módon képviseljem az alkalmazásom képességeit és az adataim közötti kapcsolatokat.

2.4.5. Adattitkosítás - jelszó hashelés

A jelszavak helyes tárolása azt igényli, hogy bármilyen adatszivárgás vagy támadás esetén, ne lehessen közvetlenül hozzáférni a felhasználók fiókjaihoz. Ehhez pedig ajánlott

a felhasználók által beírt jelszó szóvá és egy hatékony hashelési algoritmus használata ennek tárolása előtt.

Ennek megvalósítására a “crypto” könyvtárat használtam fel, amely a Node.js egy beépített modulja. Kriptográfiai függvényeket biztosít, beleértve a titkosítást, a visszafejtést, a hashelést és a biztonságos véletlenszámok generálását. Felhasználható adatbiztonságra, digitális aláírásra, hashelésre és más célokra.

A felhasználók jelszavai szóva lettek egy random generált 16 bites sóval a “randomBytes” függvény segítségével, majd hashelve lettek a “pbkdf2Sync” függvény segítségével, SHA512 algoritmussal.

3. fejezet

A rendszer specifikációja

3.1. Felhasználói követelmények

Egy barátságos és célratörő oldal üdvözl bennünket a webhely megnyitásakor. A “frontenden” épített felhasználói felület olyan célokkal volt megtervezve, hogy akár a hozzá nem értő személyek is tisztán navigálhassanak az oldalak között. Az adatok tisztán el vannak különítve és a felület törekszik minimális mennyiségű információt megjeleníteni, egyben a testreszabhatóságra igyekezve. A 3.1 ábrán található a használati eset diagram (Use Case Diagram).

A diagram megjeleníti számunkra, hogy a szoftver mégis mire képes, mit csinál és a felhasználó mit tehet státusza alapján. A bal oldalon látható két alak ábrázolja az oldal látogatóinak két elkülönített kategóriáját, a vendég, illetve a bejelentkezett felhasználó, ők a felhasználói folyamatnak a két főszereplője. A jobb oldalon a REST API mellékszereplőként játszik.

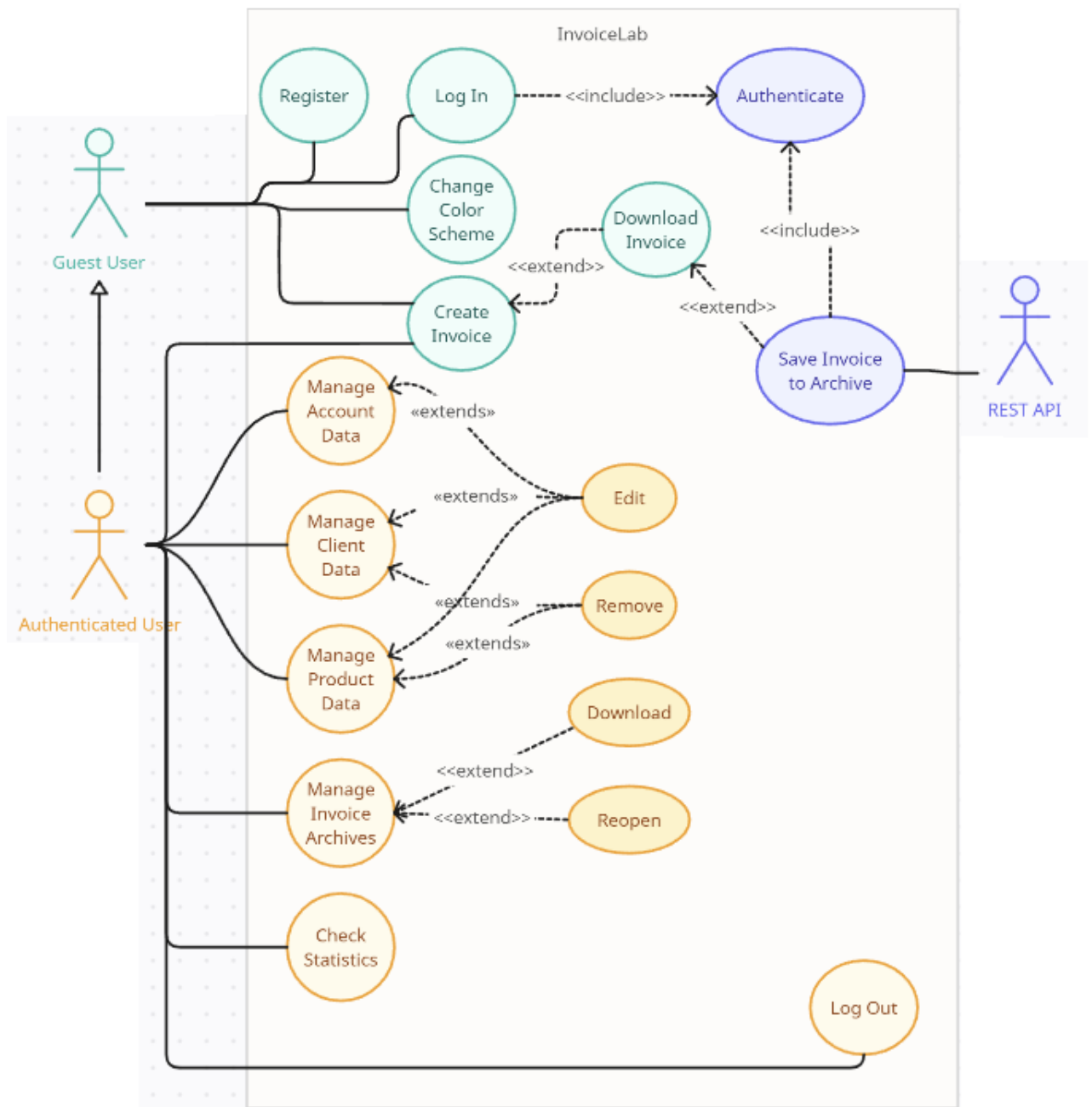
A diagramon négy fajta kapcsolódás vehető észre az elemek között:

- Általános: A folytonos, nyílmentes összekötés az egyszerű lehetőségeket jelképezi.
- Öröklődő: A folytonos, nyíllal ábrázolt összekötés öröklődést jelképez. Ez esetben a bejelentkezett felhasználó a vendégnek egy örököltje, mert a bejelentkezett felhasználó képes arra, amire a vendég, de ez fordítva nem igaz.
- Extend: A szaggatott “«extend»” címkés összekötések olyan eseményekre vonatkoznak, amelyek csak bizonyos feltételek között vagy a felhasználó választására következnek be.

Mindzek mellett, észrevehető a színekkel való megkülönböztetés:

- Zölddel jelöltem a vendég használati eseteit.
- Sárgával jelöltem a bejelentkezett felhasználó használati eseteit.
- Lilával jelöltem a “backend” komponens által szolgált eseteket.

Megemlíteném, hogy a “backend” adatokat szolgál minden lekéréshez és szerkesztéshez. Azért lett csak a felhasználó hitelesítése és az archívumba való mentés feltüntetve, hogy tisztává váljon, hogy csak abban az esetben menti el a később számonkérendő számlákat, hogyha már be van jelentkezve a felhasználó. Emelett, minden adatszolgáltatást feltüntetése zavaros ábrát eredményezett volna.



3.1. ábra. Használati eset diagram

3.2. Rendszerkövetelmények

A következőkben a rendszer képességeiről, szolgáltatásairól és működési feltételeiről szeretnék beszélni.

3.2.1. Funkcionális követelmények

A rendszer szolgáltatásai és képességei:

1. Bármilyen felhasználó létrehozhat egy számlát és lementheti azt.
2. A bejelentkezett felhasználó hozzáférhetőséget nyer a következőkhöz: kliens -, termék- és számlainformáció tárolása. Emellett, a saját kereskedelmi adatai is mentve vannak.
3. Az eltárolt kliens - és termékadatokat módosítani lehet, illetve bármikor hozzáférhet a számla készítésekor, hogy elhelyezze ezeket a számlán.
4. A bejelentkezett felhasználó újra letöltheti vagy újra megnyithatja a már elmentett számlákat.
5. Kulcsszó alapú keresés és kategória szerinti rendezés az összes adattábla esetén.
6. Statisztika generálás.
7. Színskála változtatása

3.2.2. Nem funkcionális követelmények

A szoftver korlátozásai a **lokális** futtatás esetében:

- Ajánlott és tesztelt operációs rendszer: Windows 10.
- npm csomag kezelő (2.3.1 fejezet) a node csomagok telepítéséhez és a futtatáshoz.
- MongoDB jelenléte a rendszerben - ha nem fontosak az adatok, akkor nem kell adatbázist importálni, mert a sémákat a Mongoose (2.3.3 fejezet) könyvtár hozza létre a “backend” futása alatt.
- Egy környezeti változót (environmental variables) tartozó “.env” fájl a címek sikeres elérésének érdekében.
- CPU és GPU szempontjából elegendő minimális (pl. Intel i3) felszerelést használni, viszont nagyobb adatmennyiségek lekérése érdekében ajánlott nem korlátozni.
- RAM szempontjából, 4GB garantálhat optimális működést.
- Tárhely: Az alkalmazásnak 300MB tárhely szükséges, viszont az adatbázis az adatmennyiségtől függ - a minimális ajánlott 1MB, ha egy pár adat tesztelésére vagyunk kíváncsiak.

4. fejezet

Tervezés

4.1. Az adatbázis szerkezete

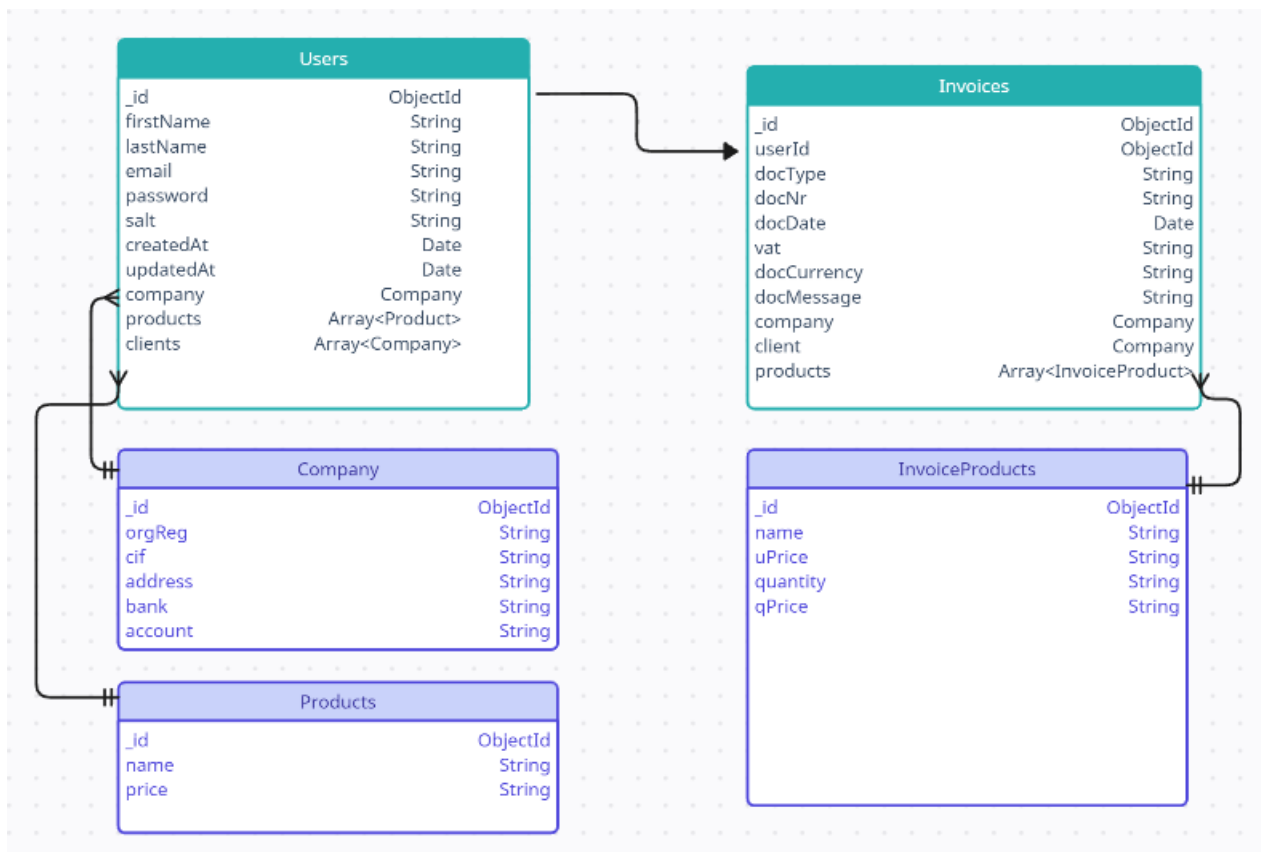
A tervezési folyamat nem indulhatott volna másként, mint annak az adatnak a meghatározásával, amit tárolnia kell az alkalmazásnak. A nem-relációs adatbázis természetével ellentétben, előre létrehoztam sémákat, hogy ismertessem magam a később tárolandó adatok típusaival.

A MongoDB és Mongoose összeműködése lehetőséget adott egy dinamikus és könnyen módosítható adatbázis struktúra tervezésére. Két típusú adattípust különíthetünk el:

- Táblák: Itt említhető a megszokott adattárolás, ami a relációs adatbázisokban is normalizált. Különböző dokumentumok egy-egy sorba rendezve vannak eltárolva, azzal a különbséggel, hogy csak azok az oszlopok vannak értéktárolásra kötelezve, amelyeket kötelezőnek jegyeztem meg kitölteni.
- Sémák: A Mongoose könyvtár teszi lehetővé, hogy sémákkal dolgozhassunk. Ezesetben, a sémára úgy kell gondolni, mint egy új adattípusra vagy osztályra.

A 4.1-es ábra bemutatja, hogy valójában milyen különleges struktúrát lehetséges létrehozni a táblák és sémák megkülönböztetésével. A zöldben jelképezett táblák adatokat tárolnak, míg a lilával képviselt sémák adattípusokat határoznak meg a tárolandó dokumentumok egy-egy attribútumának. A sémák csak a kódban jelennek meg.

Megjegyezném, hogy a “_id” attribútumok automatán vannak generálva, annak érdekében, hogy mindegyik dokumentum vagy példányosított séma sajátos azonosítót nyerjen - ezt a típust a MongoDB “ObjectId”-ként említi.



4.1. ábra. Táblák és sémák kapcsolata

A következőkben, elemezni szeretném a két táblám adatait az átláthatóság érdekében.

4.1.1. Users tábla

Minden felhasználó egy dokumentumot jelent a “Users” táblában. Tárolva vannak a személyes adatok, mint a családnév, keresztnév, email és a jelszó, valamint a só, amivel a jelszó szóva lett a hashelési folyamatban. Emellett, két dátum jelenik meg: a létrehozás, illetve a legfrissebb módosítás időpontja.

A maradandó pár adat válik ezen a ponton a legfontosabbá:

- **company:** A “company” adattípus valójában egy objektum, amely egy Mongoose séma által lett létrehozva. Ez tárolja a felhasználó kereskedelmi (avagy céges) adatait, amelyek jogilag fel kell kerülnenek a számlára. A 4.2 ábra szemlélteti a séma struktúráját.
- **products:** A “products” attribútum egy “product” típusú tömböt tárol. A felhasználónak termékei vannak ebben a tömbben eltárolva, melyeket hozzáadhat egy számlához, ennek generálásakor. A projektem esetében csak egy egyszerű megnevezésre és egy árra van szükség. A pénznem kérdése pedig a számlák készítésekor dől el.
- **clients:** A “products” attribútumhoz hasonlóan, a “clients” jellemző is egy tömb szerepét tölti be, amely a felhasználó ügyfeleinek tárolására szolgál. Típusa pedig

azért “company”, mert virtuálisan ugyanolyan adatokat van szükség elmenteni az ügyfelekkel kapcsolatban, mint a felhasználó cégi adatai. 4.2 ábra szemlélteti a “company” sémát.

Fontosnak tartom újra megjegyezni a nem-relációs MongoDB adattárolását. Több adattípus esetén, nincs szükség több tábla létrehozására. Egyszerűen tömböket és objektumokat hozhatunk létre, mint egy osztály esetén. Rengeteget segít az átláthatóságon, a használhatóságon, illetve az adatok lekérésének folyamatában is, tekintve, hogy nincs szükség különböző táblák szűrésére, hogy megkapjuk a felhasználóhoz tartozó adatokat - pontosan azért, mert a felhasználó adatai mind egy helyen vannak tárolva.

```
const companySchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
    default: "Company Name"
  },
  orgReg: {
    type: String,
    required: true,
    default: "Company Registration Number"
  },
  cif: {
    type: String,
    required: true,
    default: "Customer Identification File"
  },
  address: {
    type: String,
    required: true,
    default: "Address"
  },
  bank: {
    type: String,
    required: true,
    default: "Bank"
  },
  account: {
    type: String,
    required: true,
    default: "Account - IBAN"
  },
});
```

4.2. ábra. “Company” Mongoose séma deklarációja

4.1.2. Invoices tábla

Az “Invoices” tábla jelenti a projektem esetén az archívumot. Ebben a táblában van eltárolva az összes már generált és elmentett számla. Ebben az esetben nehéz lett volna alkalmazni a korábban (4.1.1 alatt) említett adatlekérést, miszerint minden egy helyre kerül. Ha a Users osztály egyik attribútumaként mentenénk el az összes hozzá tartozó számla adatát, nagy valószínűséggel rossz befolyást gyakorolna az adatlekérés gyorsaságára, mert az “Invoices” tábla a következőkből áll:

- `userId`: A felhasználó azonosítója, akihez tartozik a számla.
- `docType`: A számla típusa.
- `docNr`: A számla sorszáma - az archívum utolsó sorszámaához tesz hozzá egyet minden új számla esetén.
- `docDate`: A számla dátuma.
- `vat`: A számlán szereplő ÁFA értéke.
- `docCurrency`: A számlán szereplő pénznem.
- `docMessage`: A számlán szereplő kisbetűs utóirat.
- `company`: A felhasználó céges adatai a számla kiállításakor - objektum.
- `client`: A számlán szereplő ügyfél adatai - objektum.
- `products`: A számlán szereplő termékeket tároló tömb.

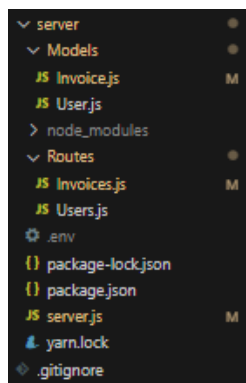
Tisztán észrevehető, hogy ebben az esetben átalakítottam az eddigi termékekre vonatkozó sémát, azzal az elnevezéssel, hogy “InvoiceProduct”. Ez annak érdekében történt, hogy lehetséges legyen nem csak a termék árának az elmentése, hanem a mennyiség, illetve a mennyiségre vonatkozó ár tárolása is (`uPrice = unitPrice`, `quantity`, `qPrice = quantityPrice`, ebben a sorrendben).

Az adatmennyiség megtekintésekor az is szembetűnő, hogy minden számlán szereplő minden termék adatainak a lekérése igen sok adatot kérne le egyszerre, ha csak a felhasználó adataira lenne szükségünk.

4.2. A backend szerkezete

Fontossá vált a projekt tervezésének ideje alatt az, hogy milyen fájl struktúrát követ a “backend” rész a tiszta osztályok közötti megkülönböztetés érdekében - ezt mutatja be a 4.3 ábra.

A kiinduló és a REST API futásához legfontosabb fájl a “server.js”. Ez a fájl teremti meg a kapcsolatot az Express segítségével megépített “route” osztályok és a Mongoose segítségével megépített modellek, illetve az alap Node szerver között. Megemlítésre hoznám az ebben importált “cors” modult (Cross Origin Resource Sharing), melynek deklarálása lehetővé teszi a nem-biztonságos webhelyekkel való kommunikációt, viszont szükséges a lokális hálózaton való teszteléshez, mert a lokális hálózat nem rendelkezik SSL tanúsítvánnyal.



4.3. ábra. A backend részen használt fájl struktúra

4.2.1. Mongoose modellek

A 4.3 ábra megfigyelése alapján, ráeszmélünk arra, hogy a “Models” mappa csak két fájlt tartalmaz. Ez azért van, mert a “company”, illetve a terméktárolásra felhasznált sémák ugyancsak ebben a két fájlban vannak deklarálva. Úgy képzelhető el, mint ahogy 4.1 ábrán van feltüntetve. A “company” séma a “User.js” tartalmában található, a “product” és a “User” sémákkal együtt, míg az “Invoice.js” a számlákra vonatkozó “Invoice” és a számlákon tárolt termékekre vonatkozó “InvoiceProduct” sémát tartalmazza, majd importálja a “company” sémát a “User.js”-ből. A modellek tartalmi megegyeznek a 4.1 fejezet alatt leírtakkal.

A “User” modellhez viszont két függvény és egy életciklus horog (lifecycle hook) tartozik, melyek a 4.4 ábrán találhatóak. Ezek a függvények minden egyes “User” dokumentumra példányosodnak, ami engedélyezi számukra a dokumentum attribútumainak elérését a “this” előszóval.

```

userSchema.methods.setPassword = function(password) {
  this.salt = crypto.randomBytes(16).toString('hex');
  this.password = crypto.pbkdf2Sync(password, this.salt, 1000, 64, `sha512`).toString('hex');
};

userSchema.methods.validatePassword = function(password) {
  let hash = crypto.pbkdf2Sync(password, this.salt, 1000, 64, `sha512`).toString('hex');
  return this.password === hash;
};

userSchema.pre('save', function(next) {
  this.updatedAt = Date.now();
  next();
})

```

4.4. ábra. User séma függvényei, lifecycle hook

A függvények működését a következőkben írtam le:

- setPassword (password) : A paraméterben átadott “password” változót, illetve a crypto könyvtárral generált sót bevezeti a hashelő függvénybe és elmenti a dokumentum “password” mezőjébe a “this” előszó segítségével.

- `validatePassword (password)` : A bejelentkezés alatt történő validálási függvényt okosnak tartottam ugyancsak a dokumentumokra példányosítani a változók könnyű és szinte automata eléréséért és összehasonlításáért. Visszatérít egy igaz / hamis értéket.
- “save” folyamat horog: Minden “User” típusú dokumentum mentése előtti pontban fut le a függvény és a legutóbb módosított dátum (“updatedAt”) frissítésével foglalkozik. A “next()” függvény meghívásakor továbbugrik az alkalmazás, hogy más horgokat keressen vagy befejezze a szálat.

4.2.2. Express Routerek

A REST API fogalom alapjai rejlenek a “router” komponensek helyes megírásában. Ezen modulok által köthetjük össze a felhasználói felületet (frontend) az adatainkkal. Az ezekre való hivatkozás fogja meghatározni, hogy mi fog történni a tárolt információval: megvalogatott CRUD műveletek folyamatait biztosítják (Create, Read, Update, Delete).

A komponensek a “Routes” folderben találhatóak, a két fő tábla műveleteit elvégző “User.js”, illetve “Invoices.js”.

Mindkét komponensre jellemző, hogy a fájl elején importálja a Mongoose által létrehozott sémákat az adatok elérésének érdekében, illetve az, hogy egy-egy “middleware” függvényt használnak a keresési folyamat gyorsítására.

Az Express route-ok esetében, a “middleware function” a route-ban létező kód lefutása előtt hívódik meg és eléri ugyanazokat a paramétereket, mint a szülő komponens. Az alkalmazás a “getInvoiceById” és “getUserById” függvényeket használja, melyeknek ugyanaz a szerepe: egy CRUD műveletre keresett elem visszatérítése, hogy a kód bázist lehessen rövidíteni és újrahasznosítani.

Az “Invoices.js” címeit a 4.1 táblázatban figyelhetjük meg. A táblázat bal oldali oszlopában a lekérések HTTP típusai vannak feltüntetve, a középső oszlop a címeiket tartalmazza, a jobb oldaliban pedig a címek meghívásának következményei vannak felsorolva. A címekben használt kettősponttal megelőzött változók valójában URL-en átadott paraméterek.

Típus	Cím	Eredmény
GET	/invoices	Összes számla
GET	/invoices/:id	Egy számla, megadott azonosítóra
GET	/invoices/users/:id	Az azonosítóval rendelkező felhasználó számlái
POST	/invoices	Egy új számla feltöltése

4.1. táblázat. Számlakezelésre vonatkozó REST API címek

Az összes számlának a lekérése jelen időpontban nincs implementálva az alkalmazásban, viszont a továbbfejlesztési lehetőségeket, úgy gondolom, hogy különböző kategóriájú adatfeldolgozás fele irányíthatja. A felhasználóra vonatkozó számlák lekérése az archívum betöltéséhez szükséges, illetve a POST kérés az archívum egyenkénti új számlákkal való feltöltésére szolgál.

A “Users.js” címeit a 4.2 táblázatban figyelhetjük meg. Ebben az esetben, a lekérések száma és komplexitása nőtt, mivel az alkalmazás több adatot kell elérjen és módosítson a felhasználói táblából.

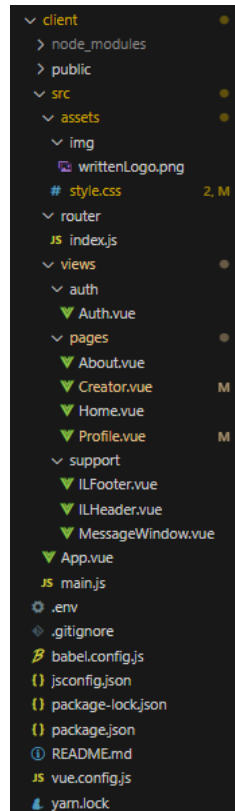
Érdekesnek találom megjegyezni, hogy egy felhasználó bizonyos ügyfelének vagy termékének módosításához két különböző paramétert kellett átadni a címnek: a felhasználó azonosítója, illetve az ügyfél vagy termék azonosítója. Ez azért történik, mert minden felhasználónak minden ügyfele és minden terméke sajátos automatán generált MongoDB azonosítóval rendelkezik. Ezek az azonosítók pedig elérhetőek a “frontend” táblázataiban, így a pontos hívás könnyen leprogramozható.

Típus	Cím	Eredmény
GET	/users	Összes felhasználó
GET	/users/:id	Egy felhasználó, megadott azonosítóra
POST	/users	Egy új felhasználó feltöltése
POST	/users/login	Egy felhasználó adatainak a validálása
PATCH	/users/personalUpdate/:id	Felhasználó személyes adatainak a frissítése
PATCH	/users/companyUpdate/:id	Felhasználó céges adatainak a frissítése
PATCH	/users/addClient/:id	Új ügyfél hozzáadása az azonosítóval rendelkező felhasználóhoz
PATCH	/users/addProduct/:id	Új termék hozzáadása az azonosítóval rendelkező felhasználóhoz
PATCH	/users/:id/clientUpdate/:clientId	“id” Azonosítóval rendelkező felhasználó “clientId” azonosítóval rendelkező ügyfél adatainak frissítése
PATCH	/users/:id/productUpdate/:productId	“id” Azonosítóval rendelkező felhasználó “productId” azonosítóval rendelkező termék adatainak frissítése
DELETE	/users/:id/deleteClient/:clientId	“id” Azonosítóval rendelkező felhasználó “clientId” azonosítóval rendelkező ügyfél törlése
DELETE	/users/:id/deleteProduct/:productId	“id” Azonosítóval rendelkező felhasználó “productId” azonosítóval rendelkező termék törlése

4.2. táblázat. Felhasználók kezelésére vonatkozó REST API címek

4.3. A frontend szerkezete

A 4.5 ábrán megfigyelhető struktúra egy átlagos Vue által generált mappa gyűjtemény, egy pár módosítással. Megfigyelhetjük, hogy jelen van egy “assets” mappa, mely képeket és a .css stilizálási fájlt tartalmazza, illetve a “views” mappát, melyben az oldalakat irányító “ViewModel”-ek vannak elhelyezve. Ezeknek emelt száma miatt elkülönítettem kisebb mappákba, illetve létrehoztam egy Vue “Router” komponenst, mely a “ViewModel”-ek közötti navigációban segít.



4.5. ábra. A frontend részen használt fájl struktúra

4.3.1. Fő komponensek

A két alapvető komponenst a következőképpen lehet összefoglalni:

- main.js: A projekt gyökerében deklarálja a Vue csomagok használatát. Importálja a “router” modult, illetve a stilizálást illető .css fájlt és a PrimeVue csomagokat.
- App.vue: A projekt alapvető “ViewModel”-je, mely a 4.6 ábrán megfigyelhető. Az alkalmazás egy fő komponenst körülvevő sablont határoz meg. A mi esetünkben, ez a sablon a külön komponensként meghatározott header és footer elemekből áll; a fő komponens a “router-view” beépített komponens által lesz betöltve - ez a “router” modul jelenlegi oldalát fogja feltüntetni.

- “save” folyamat horog: Minden “User” típusú dokumentum mentése előtti pontban fut le a függvény és a legutóbb módosított dátum (“updatedAt”) frissítésével foglalkozik. A “next()” függvény meghívásakor továbbugrik az alkalmazás, hogy más horgokat keressen vagy befejezze a szálát.

```

<template>
  <ILHeader/>
  <router-view />
  <ILFooter/>
</template>

<script>
import ILHeader from "../views/support/ILHeader.vue"
import ILFooter from "../views/support/ILFooter.vue"

export default{
  name: "App",
  components: {
    ILHeader,
    ILFooter
  }
}
</script>

```

4.6. ábra. App.vue tartalma

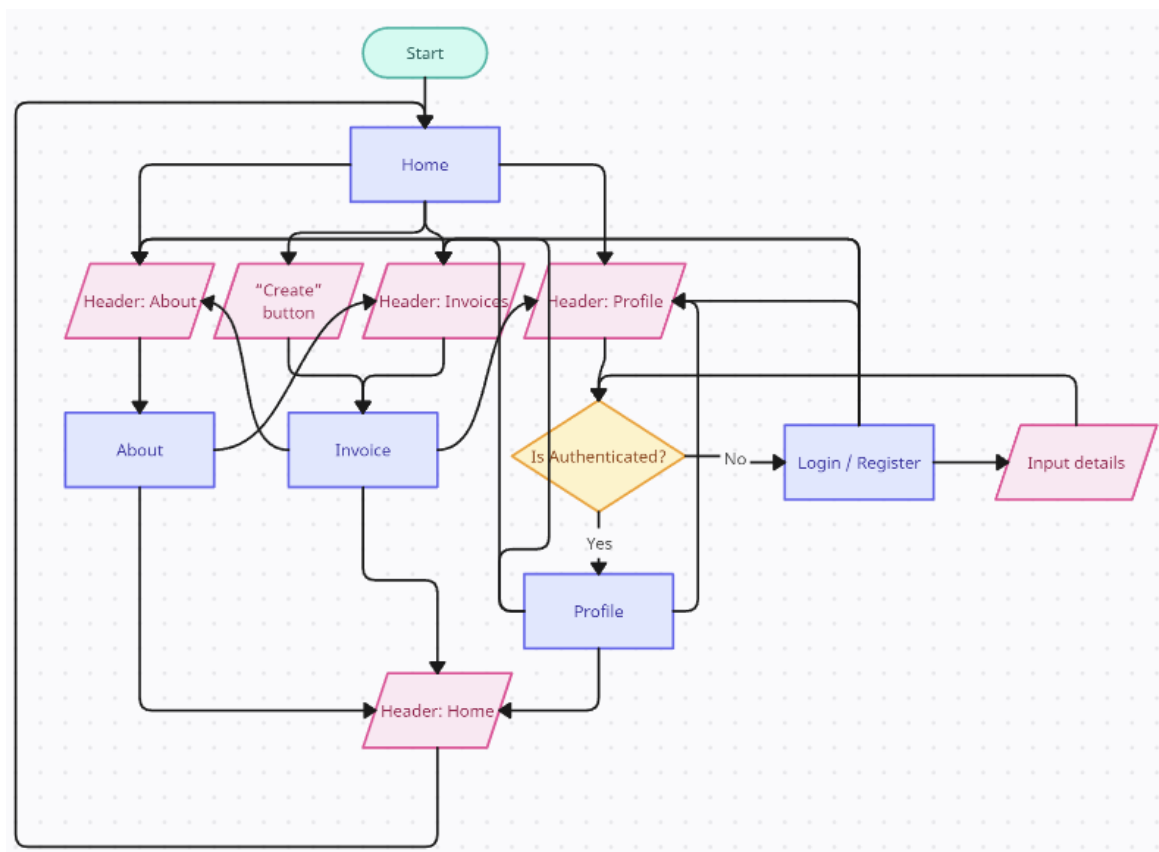
4.3.2. Router modul

A “router” mappában található “index.js” a felhasználói felületen történő navigációban óriási szerepet játszik, mivel az összes elérendő oldal között megteremti a kapcsolatot. A fájlban lévő kód elment egy objektumot mindegyik oldalról, mely az oldal nevét, fájlbeli pozícióját és a böngészőben megjelenő URL-t tartalmazza, egy tömbbe. Ezt a tömböt pedig átadja a “createRouter(...)” beépített függvénynek, amely az egész alkalmazásnak elérhetővé teszi ezt az oldalak közötti kommunikációs módszert. A 4.7 ábra tisztán feltünteti az oldalak közötti kommunikáció folyamatát és feltételeit.

Az ábrán három fő modult különböztetünk meg:

- Oldalak vagy események, melyek lila téglalapokkal vannak feltüntetve;
- Döntések vagy programlogika, melyek sárga rombuszokkal vannak feltüntetve;
- Felhasználói input vagy gombnyomás, melyek rózsaszín paralelogrammával vannak feltüntetve.

Megjegyezném, hogy a “header”-ben található linkek kattintását is gombnyomásként tekintettem, így ezeket úgy jelöltem, hogy “Header: <Oldal neve>”. Észrevehető, hogy az oldalak magas szinten elérhetőek bármelyik oldalról.



4.7. ábra. Oldalak folyamatábrája (Flow chart)

A következőkben, a “views” mappa struktúráját követve szeretném kategóriákra osztva elemezni a felületen használt “ViewModel”-eket. Emellett, ha jelen vannak a 2.3.4 fejezet alatt leírt “Options API” kategóriái, meg szeretném említeni a fontosabb függvények szerepét.

4.3.3. “Support” mappa

A “Support” mappa a következő három elemet tartalmazza:

- ILFooter.vue: Egy egyszerű sort tartalmaz, mely mindig a fő sablon alá kerül, hogy esztétikus módon keretbe zárja a megjelenített oldalt.
- ILHeader.vue: Ez a komponens egy navigációs sort tartalmaz, mely mindig az első dolog, ami üdvözl az oldalon. Tartalmazza a 4.3.2 fejezetben leírt “router” modul elemeit, illetve egy kijelentkezési gombot, amely csak akkor jelenik meg, ha a felhasználó be van jelentkezve. Ezek mellett, jelen van egy “téma” változtató lenyíló menü, amely megengedi a felhasználónak, hogy kicserélje a weboldal színeit - ez hatással lesz mindenre az felületen.
 - A “setup” kategória: Ellenőrzi, hogyha létezik bejelentkezett felhasználó, illetve, elmentett téma, majd ezek szerint modellezi a header megjelenését.
 - A “methods” függvényei:

- * `logout ()` : Kijelentkezteti a felhasználót.
- * `changeTheme (version)` : Kicseréli a témát a stíluslapban meghatározott “primaryColor” kicserélésével. Ezt az adatot elmenti a böngésző helyi tárhelyébe, hogy a következő betöltésnél is ugyanaz a téma jelenjen meg.
- `MessageWindow.vue`: Egy üzenődoboz komponens, amely csak akkor aktivál, hogyha létezik egy felhasználónak megjelenítendő üzenet. Ez minden oldalon importálva van, hogy bármikor tisztán kommunikálhasson az alkalmazás a felhasználóval
- Felhasználói input vagy gombnyomás, melyek rózsaszín paralelogrammával vannak feltüntetve.

4.3.4. “Auth” mappa

Az “Auth” mappa kivételesen az “Auth.vue” komponenszt tartalmazza.

- `Auth.vue`: Amint neve sugallja, a bejelentkezési adatok hitelesítésének feladatával foglalkozik, illetve az új felhasználók regisztrálásával.
 - A “methods” függvényei:
 - * `switchEventBoxes ()` : Kicseréli a login formot a regisztrációs formra.
 - * `validatedRegistration ()` : Ellenőrzi a szükséges adatok jelenlétét, illetve reguláris kifejezéses ellenőrzéssel biztosítja a megadott email cím valódiságát. Ha minden rendben van, elküldi a regisztráló adatait a megfelelő REST API címre.
 - * `validatedLogin ()` : Validálja a bejelentkező adatait egy REST API hívással. Ha sikeres a bejelentkezés, elmenti a felhasználó dokumentumjának szükséges adatait a böngésző helyi tárolójába, hogy elérhetőek legyenek mindegyik oldalon.

4.3.5. “Pages” mappa

A “Pages” mappa a következő négy komponenszt tartalmazza:

- `Home.vue`: Egy rövid köszöntő szöveget, egy CTA (call to action) gombot és a kiválasztott témának megfelelő hátteret tartalmazza.
- `About.vue`: A dolgozat kivonatának angol változatát tartalmazza.
- `Creator.vue`: Az alkalmazás lényege rejlik ebben a komponensben. Az oldal két fő részre van felosztva: a bal felén adatok és “input” mezők, a jobb felén egy élő előnézete a készülődésben lévő számlának. Az oldal bal felén lenyíló dobozok segítségével sikerült elrendezni az adatokat - a felhasználó dönthet, hogy egyszerre csak egyet nyit meg, vagy megtekinti az összeset. A `PrimeVue` sablonoknak köszönhetően, egyszerű a dátumok kiválasztása, illetve ki lehet választani, hogy milyen fajta adatok jelennek meg a számlán, úgy a felhasználó saját adatai esetében, mint az ügyfele

esetében. Emellett, keresni lehet a felhasználó ügyfelei között, hogy adatait hozzáadhassuk a számlához, illetve megjeleníthetünk és szerkeszthetünk már elmentett termékeket, vagy akár teljesen újakat. A mentés gomb lenyomásával pdf formátumban lesz letöltve a kialakított dokumentum.

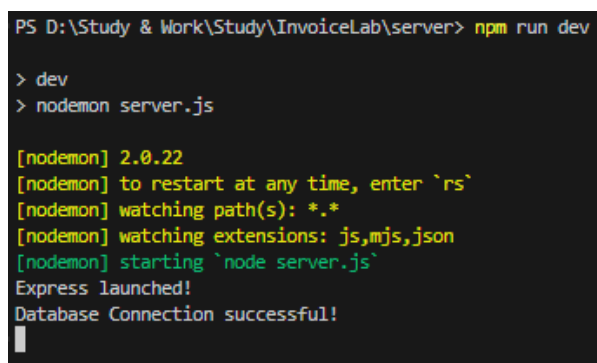
- A “setup” kategória: Inicializálja a számla generálásához szükséges adatokat
 - ha van bejelentkezett felhasználó, akkor vannak számlára írandó adatok, ha pedig nem, akkor egyfajta “default” objektum lesz meghatározva.
- A “methods” függvényei:
 - * saveProcess () : Megemlíti ez a függvény, mivel végigvezet egy hosszas ellenőrzést, hogy megbizonyosodjon az alkalmazás arról, hogy a megfelelő adatok rajta vannak a számlán. Ezután, létrehoz egy objektumot a számlán szereplő adatokkal és egy POST kéréssel a REST API-val törekszik kommunikálni. Ha ez a folyamat sikeres volt, akkor meghívja a “downloadProcess” függvényt.
 - * downloadProcess () : Létrehoz egy képet a készenlétben álló számláról, majd a megfelelő rezolúciós állításokkal elmenti pdf formátumban.
- Profile.vue: A bejelentkezett felhasználók ezen az oldalon örveendhetnek a számos testreszabhatósági lehetőségnek. A felhasználó személyes és céges adatai kártyákon vannak szemléltetve, amelyeket pár kattintással szerkeszthető. Ugyanez jellemző ügyfeleinek, illetve termékeinek adataira, melyekre mind CRUD műveleteket végző függvények lettek létrehozva, viszont az archívum és a statisztika társaságában, ezek az adatok az oldal alsó felén lévő ablakban szerepelnek. Megemlíti, hogy csak az archívumra való kattintás pontjában vannak lekérve a bejelentkezett felhasználóhoz tartozó számlaadatok, hogy segítsen a teljesítményen.

5. fejezet

Kivitelezés

5.1. Futtatás

A projekt lokális hálózaton való teszteléséhez két parancssori ablakot kell megnyitnunk a projekt fő könyvtárában. Az első terminált a “server” mappába irányítjuk, majd az “npm run dev” parancsot futtatjuk. A második terminált a “client” mappába irányítjuk, majd futtatjuk az “npm run serve” parancsot. Az 5.1, illetve 5.2 ábrákon látható a terminál által adott válasz. Az 5.1 ábrán a REST API fut le, míg az 5.2 ábrán a felhasználói felületünk. A megemlített konzolok segítenek a debugolásban.

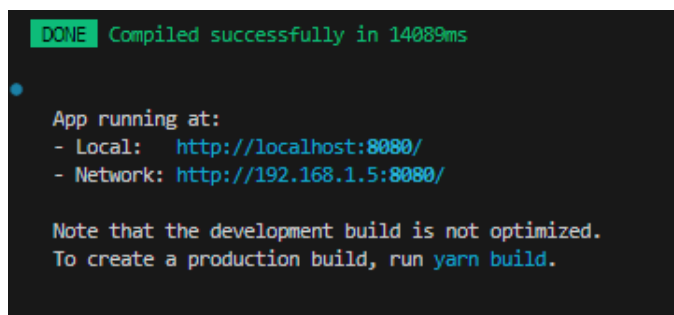


```
PS D:\Study & Work\Study\InvoiceLab\server> npm run dev

> dev
> nodemon server.js

[nodemon] 2.0.22
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node server.js`
Express launched!
Database Connection successful!
```

5.1. ábra. REST API parancssori futtatása



```
DONE Compiled successfully in 14089ms

App running at:
- Local: http://localhost:8080/
- Network: http://192.168.1.5:8080/

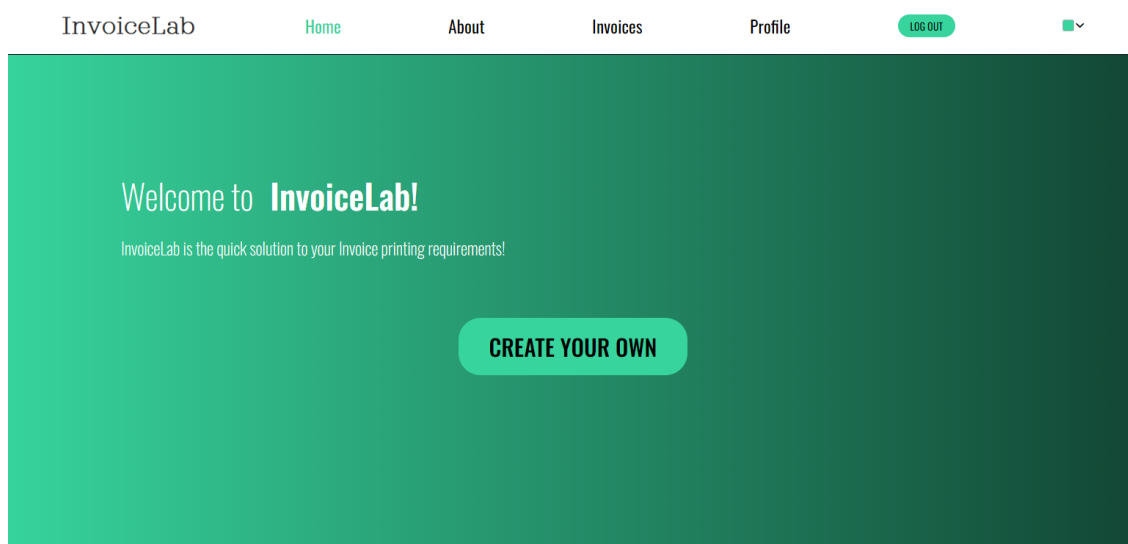
Note that the development build is not optimized.
To create a production build, run yarn build.
```

5.2. ábra. Felhasználói felület parancssori futtatása

5.2. Felhasználói felület komponensei

A felhasználói felület első elképzelésekhez hű alakot vett fel. A következőkben olyan sorrendben szeretném bemutatni a felületen található komponenseket, mint ahogy a 4.3.3 - 4.3.5 fejezetekben van leírva.

Az 5.3 ábrán találjuk a kezdőlapot, amely egy rövid szöveggel köszönt, illetve egy CTA gombbal, ami a számlakészítő felületre vezet. Emellett, látható teljes szélességben a “header”, amely a navigációs linkeket tartalmazza.

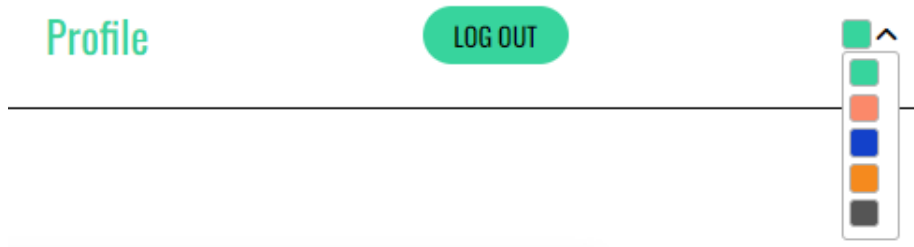


5.3. ábra. A felület kezdőlapja

Az 5.4, illetve 5.5 ábrákon megfigyelhetjük a “header” változásait, melyek a bejelentkezésre, illetve a témaválasztásra vonatkoznak.

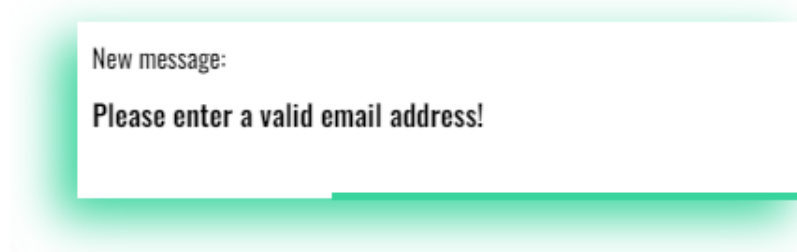


5.4. ábra. Header – első feltüntetés



5.5. ábra. Header – második feltüntetés

Az 5.6 ábrán a “MessageWindow” komponens egy példája látható. Az oldal jobb felső sarkából becsúszik az oldalra, mikor a felhasználóval szeretne kommunikálni, illetve az ablak alján található csík az üzenet hátramaradt időmennyiségét mutatja. Az eddigi teszteléseim alapján úgy döntöttem, hogy 6 másodperc a megfelelő időtartam egy ilyen kaliberű üzenet megjelenítésére.



5.6. ábra. Üzenetdoboz

Továbbiakban, az autentikációs oldal alakját figyelhetjük meg az 5.7 ábrán. Ezen az oldalon válthatunk a regisztrációs, illetve a bejelentkezési formok között.

here'."/>

5.7. ábra. Bejelentkezési form

Az 5.8 ábrán a számla generáló oldalt tekinthetjük meg. Az oldal két részre van osztva, hogy könnyen átlátható és állítható legyen a számla tartalma. A bal oldalon található lenyíló dobozok különböző információkat tartalmaznak, melyek változtatásánál a jobb oldalon bemutatott előnézet élőben, minden gombnyomás nyomán változik, kiszámolja a végső árakat, illetve az ezekhez tartozó ÁFÁ-t.

Az 5.9, illetve az 5.10 ábrákon a számlán szereplő ügyfél, illetve a számlán szereplő termékeket illető adatok vannak megjelenítve.

Az ügyfél adatait illető menüben előre kiválaszthatjuk az ügyfelet a már elmentettek közül – az adatai automatán kitöltődnek, hogy időt nyerjen a felhasználó.

Create your invoice

Invoice Information

Document Type

FISCAL INVOICE

Document Number

80066

Document Date

05/10/2023

VAT (%)

19

Document Currency

RON

Company Information and Logo

Client Information

Products

Add Message

Reset

Save

FISCAL INVOICE

80066

DATE: 5/10/2023

VAT: 19%

TOTAL:

599.76 RON

BENEFICIARY

KJ Specs SRL

Company Registration Number:

41075754

Customer Identification File:

J26/1059/2019

Address:

Str. Lucei nr. 4

Bank:

OTP

Account - IBAN(RO):

RO49AAAA1831007593840000

CLIENT

KICO 2008 SRL

Company Registration Number:

71095554

Customer Identification File:

J17/5512/2008

Address:

Str. Oltului 21, Târgu Mureș

Bank:

Banca Transilvania

Account - IBAN(RO):

RO49CQEA9N2105751342542

Number	Name	Unit Price	Quantity	Quantity Price
1	Curea trapezoidală 2-10x22 SET 2	350	1	350
2	Inel cilindric C2192	10.5	2	21
3	Inel cilindric 32x50	4	2	8
4	MANIPERĂ	125	1	125

SUBTOTAL:

504.00 RON

VAT:

95.76 RON

TOTAL:

599.76 RON

Signature:

KJ Specs SRL

5.8. ábra. Számla készítő oldal

Client Information

Choose a Client

KICO 2008 SRL

Client Name

KICO 2008 SRL

Company Registration Number

71095554

Customer Identification File

J17/5512/2008

Address

Str. Oltului 21, Târgu Mureș

Bank

Banca Transilvania

Account - IBAN

RON

RO49CQEA9N21057513

5.9. ábra. Számlán szereplő ügyfél adatainak módosítása

A termékek listája előtti menü bármelyik elemére való kattintáskor pedig egyszerűen hozzászúrhatjuk az elmentett termékek közül bármelyiket a számlára. Az 5.11 ábra bemutatja a keresési funkciót a menü lenyitásakor. Ez a funkció az ügyfeleket illető menüben is működik.

40

Products

Add a Product
Add Product

Products
Add Product

Nr	Name	Unit Price	Quantity	Quantity Price
1	Curea trapezoidală Z.	350	1	350
2	Inel clemă E2192	10.5	2	21
3	Inel prindere 32x50	4	2	8
4	MANOPERĂ	125	1	125

5.10. ábra. Számlán szereplő termékek adatainak módosítása

Products

Add a Product
Add Product

Products
Add Product

Nr	Name	Unit Price	Quantity	Quantity Price
1	Curea trapezoidală Z.	350	1	350
2	Inel clemă E2192	10.5	2	21
3	Inel prindere 32x50	4	2	8
4	MANOPERĂ	125	1	125

5.11. ábra. Számlán szereplő termékek adatainak módosítása

A felhasználó profiljával kapcsolatos adatokat a “Profile” oldal tartalmazza. Az oldalt vízszintesen kettőbe szeretném osztani. Az 5.12 ábrán találhatjuk a felső felét, mely a felhasználó személyes fiókjához, illetve a számlán megjelenő céges adatait tekinthetjük meg két kártya alakjában. Az 5.13 ábra bemutat egy beugró menüt, amely az “Edit” gomb megnyomásakor jelenik meg.

My Profile

János Kiss

First Name: János

Last Name: Kiss

E-mail address: kissjanos@yahoo.com

Edit

My Company

KJ Specs SRL

Company Registration Number: 41075754

Customer Identification File: J26/1059/2019

Address: Str. Lavandei nr. 4

Bank: OTP

Account - IBAN: RO49AAAA1B31007593840000

Edit

5.12. ábra. Felhasználó adatait megjelenítő kártyák

41

Edit My Company

Company Name:

KI Specs SRL

Company Registration Number:

41075754

Customer Identification File:

J26/1059/2019

Address:

Str. Lavandei nr. 4

Bank:

OTP

Account - IBAN:

RO49AAAA1B31007593840000

Save

5.13. ábra. Céges adatokat módosító menü

Az 5.14 ábra feltünteti a profiloldal alsó felét, amely egy négy menüpontból álló ablak. Ebben a részben szerkeszthetjük az elmentett ügyfél - és termékadatokat, megtekinthetjük az archívumunkat, illetve a statisztikát tartalmazó oldalt. Az ábrán láthatjuk az ügyfeleket tartalmazó táblázatot. Kereshetünk az elemek között, rendezhetjük őket a táblázatban található fejlécekkel, kiválaszthatjuk a lenyíló menüből, hogy hányat szeretnénk egyszerre megjeleníteni, illetve módosíthatjuk vagy törölhetjük őket a sorok végén található ikonokkal. A termékek esetén, ugyanez a formátum lett felhasználva.

Clients











Products

Statistics

Archives

Add new

Keyword Search

Name ↑↓	Company Registration Number ↑↓	Customer Identification File ↑↓	Address ↑↓	Bank ↑↓	Account ↑↓	Modify
Hidro Tech SRL	51085854	J24/5523/2006	Str. Someșului 34, Târgu Mureș	Banca Transilvania	RO4988862N21007593845555	 
KICO 2008 SRL	71095554	J17/5512/2008	Str. Oltului 21, Târgu Mureș	Banca Transilvania	RO49CQEA9N21057513842542	 
Strong Fiber SRL	95091453	J57/5545/2012	Str. Mărgareții 101, București	CEC Bank	RO12BHAM1K35057413652325	 
MiniTech SRL	85085153	J12/4410/2017	Str. Vasile Lupu 43, Cluj-Napoca	BRD	RO69JHZSP3M046953121374	 
CONST SOLUTIONS SRL	22386134	J54/5568/2011	Strada Bujorului 13, Târgu Mureș	Banca Comercială Română BCR	RO55KZLEH26246952135852	 

<<

<

1

>

>>

5

5.14. ábra. Profil oldal alsó fele – ügyfél adatok módosítása

Az 5.15 ábra az archívumban létrehozott táblázatot mutatja be. Hasonlít a struktúra az 5.14 ábrában bemutatott táblázatokra, viszont a módosító ikonok ebben az esetben megengedik, hogy újra letölthessük, újra megnyithassuk vagy megtekintsük a számla adatait. A bal oldali információs ikonra kattintva, az 5.16 ábrán feltüntetett menü jelenik meg amely a számlán használt céges adatok, a számlán megjelenő ügyfél adatait, illetve a számlán megjelenő termékek listáját tartalmazza.

Archived Invoices						
<input type="text" value="Keyword Search"/>						
Doc Nr ↑↓	Doc Type ↑↓	Client Name ↑↓	Doc Date ↑↓	VAT value ↑↓	Doc Currency ↑↓	Modify
1	FISCAL INVOICE	Hidro Tech SRL	January 24, 2023 at 12:00:00 AM	19%	RON	ⓘ ⬇ ⬆
2	FISCAL INVOICE	KIGO 2008 SRL	February 10, 2023 at 12:00:00 AM	19%	RON	ⓘ ⬇ ⬆
3	FISCAL INVOICE	MiniTech SRL	February 14, 2023 at 12:00:00 AM	19%	RON	ⓘ ⬇ ⬆
4	FISCAL INVOICE	CONST SOLUTIONS SRL	March 7, 2023 at 12:00:00 AM	19%	RON	ⓘ ⬇ ⬆
5	FISCAL INVOICE	Strong Fiber SRL	April 29, 2023 at 12:00:00 AM	19%	RON	ⓘ ⬇ ⬆
<< < 1 > >> 5 ▼						

5.15. ábra. Archívum

Invoice Details

Invoice Number: 5

Company details used:

- KJ Specs SRL
- 41075754
- J26/1059/2019
- Str. Lavandei nr. 4
- OTP
- RO49AAAA1B31007593840000

Client details:

- Strong Fiber SRL
- 95091453
- J57/5545/2012
- Str. Măgareanului 101, București
- CEC Bank
- RO12BHAH1K35057413652325

Products listed on the invoice:

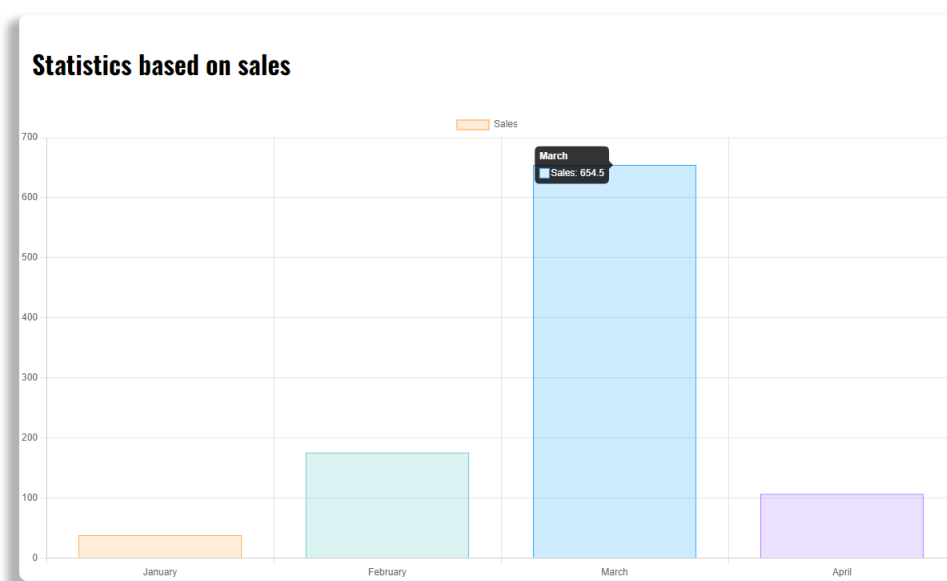
Name ↑↓	Price ↑↓
Inel prindere 55x50	
Inel prindere 32x50	

<< < 1 > >>

Done

5.16. ábra. Egy számla részletesebb adatainak menüje

Az 5.17 ábra a felhasználó eladásainak hisztogramját mutatja be. Az ablakra való kattintáskor az alkalmazás lekéri a felhasználó eddigi számláinak adatait, majd hónapokra osztva összeszámolja az összes abban a hónapban eladott tárgyak értékét.



5.17. ábra. Egy felhasználó eladásainak hisztogramja

Összefoglaló

A diplomadolgozatom célja a papírmunka által felvett idő lerövidítése volt bármilyen felhasználónak, akinek szüksége van ilyen dokumentumokat előállítására. Dolgozatomban megismerhettük a webbel kapcsolatos alapfogalmakat, a felhasznált keretrendszerek specifikációit, a megírt függvények és a felhasználói felület alapjait és később képességeit.

A projekt megvalósítása, úgy vélem, hogy egy performáns, vizuálisan esztétikus és könnyen használható felületet eredményezett, mely talán egy teljes terméket nem minden kategóriából jellemez, de nagyon közel áll ahhoz. Mindent összevetve, fontosnak találok a szoftver helyesnek vélt tárolását, így a következő GitHub tárhelyen lett első percétől megosztva, ahol visszakövethető minden nagyobb frissítés a **master** ágon:

<https://github.com/Muvesgit/InvoiceLab.git>

Számomra, a dolgozat előállításának folyamata egy önmegismerő folyamatot is elindított, mivel arra kényszerültem, hogy aktívan figyeljek a projekt struktúrai pontosságára, a kódbázis standardjait folyton emeljem, illetve, hogy az optimális működés megmaradjon a projekt hosszú távú fejlesztése alatt, úgy, hogy ne hagyjam figyelmen kívül az eredeti célt. Úgy gondolom és érzem, hogy rengeteg tapasztalattal gazdagodtam és hasznossá fognak válni a jövőbeli web alapú projektjeim számára.

Továbbfejlesztési lehetőségek

Véleményem szerint, sosincs egy projekt véglegesen befejezve. A tökéletesség hajszolása a folytonos fejlődés akaratával egyezik meg. Úgy gondolom érdemes foglalkoznunk projektjeinkkel, hogy a lehető leghasznosabb változataikká alakíthassuk.

Először is, a weboldal mobil felületen való használata egy masszív haladást fog jelenteni a projektnek. Az ilyen kategóriájú adatok bárholnan való elérése fontossá válik az egyre gyorsuló világunkban.

Másodszorban, ha a publikálás válik prioritássá, a REST API kommunikációjának a korlátozása lesz a következő lépés. A projekt ideje alatti tesztelés érdekében nem lett ez kialakítva, viszont a JWT vagy hasonló token alapú technológia egy fontos biztonsági réteget fog kialakítani az alkalmazásnak.

Kiegészítésként, érdemesnek találok a statisztikai rész gazdag kibővítését, annak érdekében, hogy az oldal felhasználói tisztábban átláthassák a vállalkozásukkal kapcsolatos adataikat.

Köszönetnyilvánítás

Ezúton szeretnék köszönetet nyilvánítani témavezető tanáromnak, Csaholczi Szabolcsnak, a rendszeres konzultációkért, az alattuk elhangzott elengedhetetlenül hasznos tanácsokért, illetve a folytonos támogatásért és biztatásért.

Külön köszönetet szeretnék nyilvánítani dr. Iclănzan David tanár úrnak a támogatásért és a magas szintű szakmai tanácsadásért, amely segített a projektemnek a legjobb változata fele haladni.

Hálás köszönettel tartozom mindazoknak, akik a három egyetemi évem alatt támogatták választásaim és segítettek abban, hogy sikeresen haladjak előre és az élet minden értékes oldalát megismerjem és kultiváljam. Köszönöm a családomnak, a barátaimnak, a kollégáimnak, és nem utolsósorban, a páromnak, hogy főleg az utolsó év hozta megpróbáltatások során, mikor kilátástalanság jellemezte legjobb esetben is a helyzetem, hittek képességeimben és biztattak a törtetésre. Elmondhatom, hogy fontos és életrevaló tanulságokkal folytathatom a szakmában és az életben való tevékenységem és abban reménykedem, hogy visszafizethetem a hozzám közel állóknak az értékes tanácsaikat és erőfeszítéseiket.

Irodalomjegyzék

- [BGBV16] Kyle Banker, Douglas Garrett, Peter Bakkum, and Shaun Verch. *MongoDB in action: covers MongoDB version 3.0*. Simon and Schuster, 2016.
- [Fc] OpenJS Foundation and Node.js contributors. Node.js®. URL: <https://nodejs.org/en/about>.
- [Git] Inc. GitHub. Github. URL: <https://github.com/about>.
- [Mar18] Azat Mardan. *Practical Node.js: Building Real-World Scalable Web Apps*. Mardan and Corrigan, 2018.
- [MDK15] Satheesh Mithun, Bruno Joseph D'mello, and Jason Krol. *Web development with MongoDB and NodeJs*. Packt Publishing Ltd, 2015.
- [Mic] Microsoft. Visual studio code. URL: <https://code.visualstudio.com/docs>.
- [Olu14] Haroon Shakirat Oluwatosin. Client-server model. *IOSR-JCE*, 16(1):67–71, 2014.
- [Sur16] Vijay Surwase. Rest api modeling languages - a developer's perspective. *IJS-TE*, 2(10), 2016.
- [You] Evan You. Vue. URL: <https://vuejs.org/guide/introduction.html#what-is-vue>.