Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Kotlin Spring alapú backend rendszer tervezése és megvalósítása

BSc Önálló laboratórium

Készítette
Kovács Bence Zoltán

Konzulens
dr. Ekler Péter

2021. május 7.

Tartalomjegyzék

1. fejezet

Bevezetés

Az alkalmazás amit a félév során fejlesztettem kisebb vállalkozások adminisztrációs feladatait igyekszik enyhíteni, illetve statisztikákat nyújt a vállalkozás anyagi helyzetéről.

Ennek a dokumentumnak a célja az implementáció részleteinek ismertetése. A használt architektúra és technológiák rövid bemutatása után kitérek majd a backend struktúrális felépítésére, és annak tesztelésére is. Végül a kész alkalmazás frontendjét mutatom be képernyőképek és rövid magyarázatok segítségével. Mivel a témámban a backenden van a hangsúly, így a frontendet részletesen nem fogom ismertetni.

Linkek:

Github repository

Swagger REST api

2. fejezet

Backend

2.1. Technológia stack

Az alkalmazás elkészítéséhez használt technológiák
Backend:

- Spring Boot
- Spring Data
- Spring Data REST
- Spring Security, JWT

Frontend:

- React, Material UI
- SWR(stale-while-revalidate)
- RHF(react-hook-form)

2.2. Model

Az üzleti logika meghatározása során igyekeztem egy vállalkozás adminisztriációs feladatait legjobban leíró sémát létrehozni. A tervezés során igyekeztem nem túl bonyolítani az egyes entitásokat, mivel szerettem volna egy általánosítható alapot létrehozni amit később könnyen személyre lehet szabni az adott vállalkozó igényeire.

2.2.1. Category

Kategóriák: A termékek tartozhatnak kategóriákba, hogy megkönnyítsük a keresést esetleg statisztikát készíthessünk arról hogy melyik kategória fogy a legjobban.

- name: Egyedi neve a kategóriának

2.2.2. Product

Termékek: A vállalkozás által kínált termékek, amiket rendelni és eladni lehet.

- name: A termék neve
- available: Rendelkezésre álló termékek száma

- description: Leírás a termékről
- category: Az esetleges kategória amihez tartozik

2.2.3. Orders

Rendelések: A vállalkozás beszerzésének rögzítésére. Minden rendeléshez rendelés elemek tartoznak amik az egyes vásárolt termékeket kötik össze metaadatokkal.

- orderDate: A rendelés dátuma
- orderItems: A rendeléshez tartozó "rendelés elemek"
  - itemIndex: Megjelenítéshez szükséges információ, azt mondja meg hogy hanyadik elemként szeretnénk látni a listában
  - product: A termék amit vásároltunk
  - price: A termék egységára
  - amount: Mennyiség
  - status: A rendelés elem státusza (Pl várjuk hogy megérkezzen)

2.2.4. Customer

Ügyfelek: Egy vállalkozásnak gyakran vannak állandó ügyfelei. Az ő elérhetőségük elmentésével sokat segíthetünk egy vállalkozónak. Statisztikát is tudunk mutatni róla, például hogy mikor vásárolt utoljára vagy hogy mennyire profitáló ügyfélről van szó.

- name: Az ügyfél neve
- phone: Az ügyfél telefonszáma
- email: Az ügyfél emailje
- shippingAddress: Az ügyfél szállítási címe
- billingAddress: Az ügyfél számlázási címe

2.2.5. Sales

Eladások: A vállalkozás eladásainak számontartására. Minden eladáshoz eladott elemek tartoznak amik az egyes eladott termékeket kötik össze metaadatokkal.

- saleDate: Az eladás dátuma
- buyer: A vásárló ügyfél
- soldItems: Az eladáshoz tartozó "eladott elemek"
  - itemIndex: Megjelenítéshez szükséges információ, azt mondja meg hogy hanyadik elemként szeretnénk látni a listában
  - product: A termék amit eladtunk
  - price: A termék egységára
  - amount: Mennyiség

2.3. Statisztika

A statisztikák kialakítása során több problémába is ütköztem. Mivel a bejelentkezés utáni kezdőképernyőnek szántam a statisztikákat így az adatok imperatív feldolgozása nem jöhetett szóba. A statisztikáknak minimális idő alatt be kell tölteni, így kézzel megírt SQL queryk mellett döntöttem. Ennek az előnye, hogy az adatbázis cachelni a tudja az optimális végrehajtási tervet és a megfelelő indexek felvételével hatékonyan tudja számolni. Én releváns információnak a havi bevétel és havi kiadást találtam, illetve hogy melyik termékekből érdemes többet beszerezni, vagy hogy melyik termékek forgalmazása veszteséges. Ennek megjelenítése a statisztikák nézeten látható. [3.2] A másik problémát az "egyediség" okozta: mivel nem egy konkrét entitást kérek le, hanem általános információkat így nem tudtam hasznát venni a Spring Data repository interface-nek és jdbc-t kellett használnom. Alapvetően igyekszem az ilyesmit kerülni, nem kell újra feltalálni a kereket.

2.4. Biztonság

A biztonságos hozzáférést JWT segítségével implementáltam. A JWT azonosítás lényege, hogy a bejelentkezés eredményeként (leegyszerűsítve) egy kulcsot kapunk a felhasználónkhoz. Az Authorization headerbe beletéve ezt a kulcsot, a backend biztonságért felelős komponense el tudja dönteni, hogy van-e jogosultságunk az adott erőforráshoz való hozzáféréshez vagy sem. Ennek megvalósításához *ezt* a cikket használtam fel. Természetesen át kellett alakítanom, mivel az én alkalmazásom kotlinban van írva, de a fő koncepciók megegyeznek.

2.5. Controller

A REST api részletes dokumentációját *itt* találod. Ezen dokumentáció keretein belül csak néhány dologra térek ki.

2.5.1. Egy bonyolultabb entitás API implementációja

Az api implementálása során megkülönböztettem a gyakran bővülő rendelés és eladás entitást a többitől, mivel huzamosabb használat során a szerver teljesítményének kárára válhatna ha egy GET kérésre több tízezer entitást kéne visszaadni. Erre a standard megoldás az úgynevezett lapozás amit én query paraméterek segítségével valósítottam meg. Paraméterek:

- page: Kért oldalszám
- pageSize: Oldal nagysága
- date: Konkrét dátum amire szűrni szeretnénk
- beforeDate: Megadott dátum előtti eladások
- afterDate: Megadott dátum utáni eladások
- sortParam: Rendezési paraméter
- sortOrder: Rendezés iránya

Példa:
host/sales?page=0pageSize=5beforDate=2021-05-07sortParam=saleDatesortOrder=asc
Ami azt jelenti, hogy az 1. oldalt kérem, a 2021-05-07 előtti eladásokból az eladás dátuma alapján növekvő sorrendben rendezve ötösével.

```kotlin
@CrossOrigin(origins = ["*"])
@RestController
@RequestMapping( ...value: "/api/sales")
class SaleController(
        private val saleRepository: SaleRepository,
        private val productRepository: ProductRepository,
        private val customerRepository: CustomerRepository
) {

    @GetMapping
    fun getPageableSortableFilterableSales(@RequestParam(defaultValue = "7") pageSize: Int,
                                           @RequestParam page: Int?,
                                           @RequestParam date: Date?,
                                           @RequestParam beforeDate: Date?,
                                           @RequestParam afterDate: Date?,
                                           @RequestParam(defaultValue = "saleDate") sortParam: String,
                                           @RequestParam(defaultValue = "desc") sortOrder: String):
            ResponseEntity<Page<Sale>> {
        val sort = Sort.by(Sort.Direction.fromString(sortOrder), sortParam)

        val pageable = if (page == null) { Pageable.unpaged() }
                       else { PageRequest.of(page, pageSize, sort) }

        return ResponseEntity.ok(
                when {
                    date != null -> saleRepository.findSalesBySaleDate(date, pageable)
                    beforeDate != null -> saleRepository.findSalesBySaleDateBefore(beforeDate, pageable)
                    afterDate != null -> saleRepository.findSalesBySaleDateAfter(afterDate, pageable)
                    else -> saleRepository.findAll(pageable)
                }
        )
    }
```

2.1. ábra. Lapozás, rendezés és szűrés implementáció

Eladások illetve rendelések létrehozásakor figyelnem kell a referenciák megfelelő beállítására. Ez gyakorlatilag csak annyit jelent, hogy egy egyszerű forEach használatával át kell adni minden eladott elemnek azt az eladást amihez tartozik.

```kotlin
@PutMapping( ...value: "/{id}")
fun updateSaleById(@RequestBody updatedSale: PutSaleDTO, @PathVariable(value = "id") saleId: Int): ResponseEntity<Sale> {

    val customer: Customer? = updatedSale.customerId?.let { findCustomerOrThrow(it, customerRepository) }
    val sale = Sale(saleId, updatedSale.saleDate, mutableListOf(), customer)
    updatedSale.soldItems.forEach {  it: PutSoldItemDTO
        val product = findProductOrThrow(it.productID, productRepository)
        val orderItem = SoldItem(it.id, it.itemIndex, it.price, it.amount, product, sale)
        sale.soldItems.add(orderItem)
    }
    return ResponseEntity.ok(saleRepository.save(sale))
}

@PostMapping
fun createNewSale(@RequestBody newSale: PostSaleDTO): ResponseEntity<Sale> {
    val customer = newSale.customerId?.let { findCustomerOrThrow(newSale.customerId, customerRepository) }
    val sale = Sale( id: 0, newSale.saleDate, buyer = customer)
    newSale.soldItems.forEach {  it: PostSoldItemDTO
        val product = findProductOrThrow(it.productID, productRepository)
        sale.soldItems.add(SoldItem( id: 0, it.itemIndex, it.price, it.amount, product, sale))
    }
    return ResponseEntity.ok(saleRepository.save(sale))
}
```

2.2. ábra. Eladás entitás megfelelő létrehozása

2.5.2. Egy egyszerűbb entitás API implementációja

Mivel az ügyfelekhez kizárólag CRUD műveletek szükségét láttam, így ennek a controllernek a legegyszerűbb az implementációja. A megfelelő kódszervezés eredményeképp egy ilyen egyszerű entitás változtatása nem okoz problémát.

```
@CrossOrigin(origins = ["*"])
@RestController
@RequestMapping( ...value: "/api/customers")
class CustomerController(private val customerRepository: CustomerRepository) {

    @GetMapping
    fun getAllCustomers(): ResponseEntity<List<Customer>> =
            ResponseEntity.ok().body(customerRepository.findAll())

    @GetMapping( ...value: "/{id}")
    fun getCustomerById(@PathVariable(value = "id") customerId: Int): ResponseEntity<Customer> =
            customerRepository.findById(customerId).map { ResponseEntity.ok(it) }
                    .orElse(ResponseEntity.notFound().build())

    @PostMapping
    fun createCustomer(@RequestBody customer: PostCustomerDTO): ResponseEntity<Customer> =
            ResponseEntity.ok().body(customerRepository.save(Customer.fromDTO(customer)))

    @PutMapping( ...value: "/{id}")
    fun updateCustomerById(@PathVariable(value = "id") customerId: Int,
                           @RequestBody updatedCustomer: PostCustomerDTO): ResponseEntity<Customer> =
            ResponseEntity.ok().body(customerRepository.save(Customer.fromDTO(updatedCustomer, customerId)))

    @DeleteMapping( ...value: "/{id}")
    fun deleteCustomerById(@PathVariable(value = "id") customerId: Int): ResponseEntity<Customer> =
            customerRepository.findById(customerId).map {  it: Customer!
                customerRepository.delete(it)
                ResponseEntity.ok().body(it)
            }.orElse(ResponseEntity.notFound().build())
}
```

2.3. ábra. Az ügyfél erőforrásért felelős implementáció

2.6. Tesztelés

Ahhoz hogy a tesztelés valós képet mutasson az alkalmazás backendjének minőségéről integrációs tesztek mellett döntöttem. A megfelelő konfiguráció beállításához a Spring keretrendszerben profilokat lehet felvenni. Így például biztosítható hogy ne az adatokkal feltöltött adatbázist szemeteljük a tesztadatokkal, hanem egy külön erre szánt adatbázis szolgálja ki a teszteket. A tesztelés lebonyolításához a TestRestTemplate osztályt használtam. Ennek használatával teszteléskor a teljes alkalmazás ugyanúgy felépül mint éles futáskor - azzal a különbséggel hogy az előbb említett test profile gondoskodik róla hogy a tesztadatbázist használja a működés ellenőrzésére. A tesztesetek felvételénél igyekeztem a rest api minden belépési pontjára legalább egy tesztet írni.

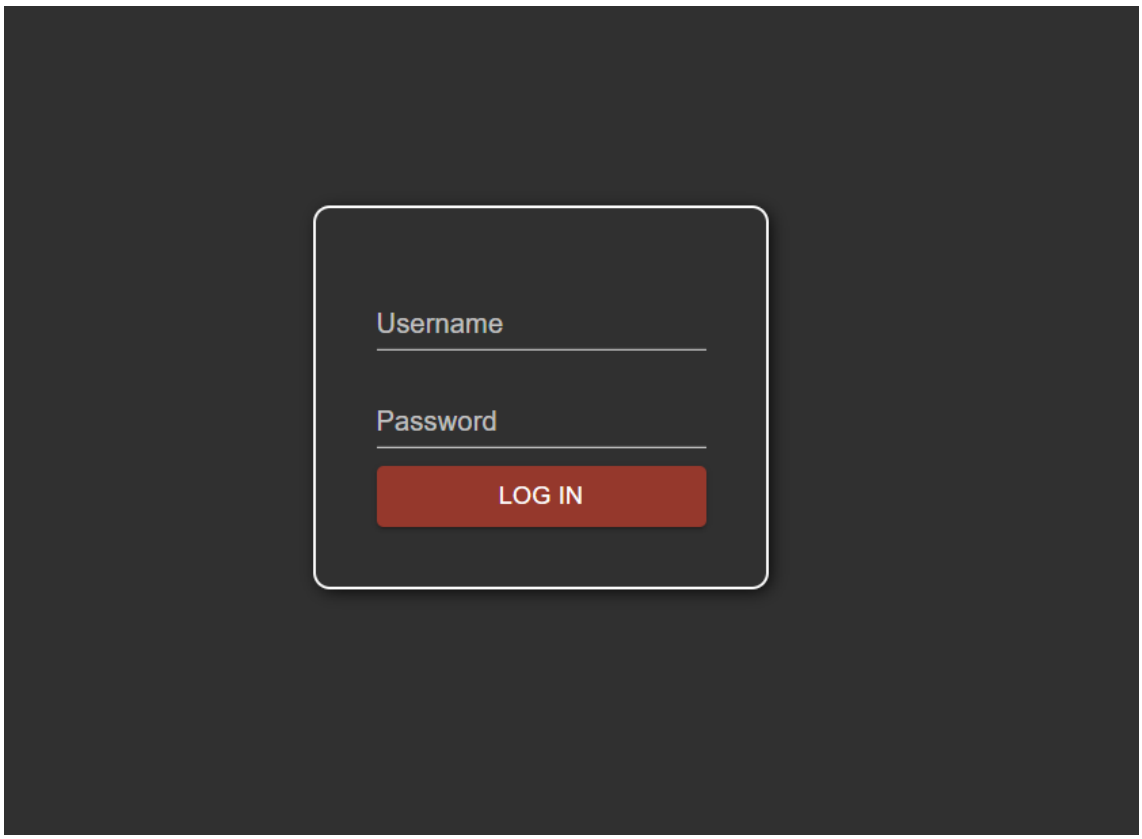
Maga a tesztkód megtalálható *githubon*.

3. fejezet

Frontend

3.1. Bejelentkezés

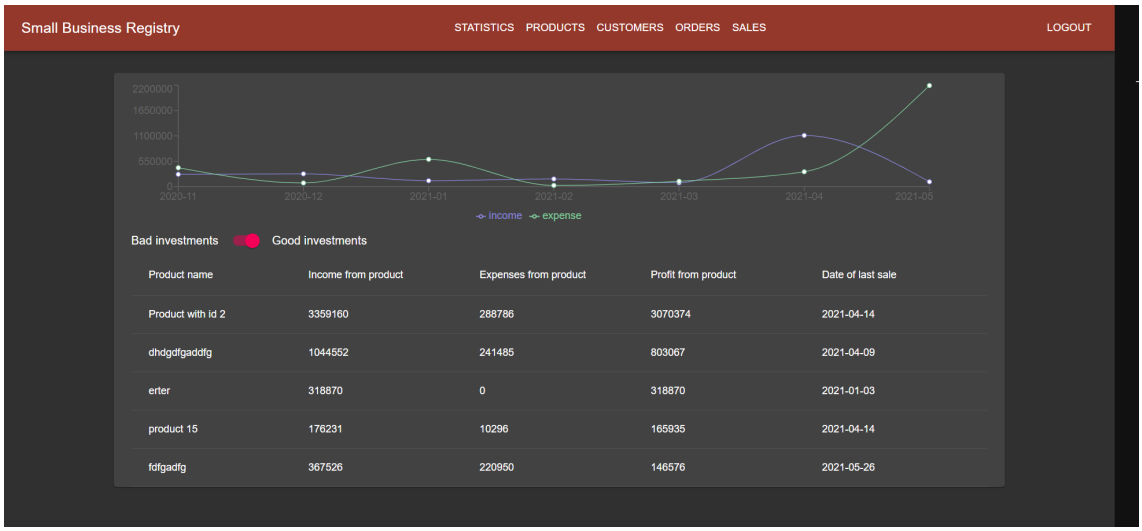
A felhasználónév és jelszó beírása után, a login gombbal lehet bejelentkezni.



3.1. ábra. Login képernyő

3.2. Statisztikák

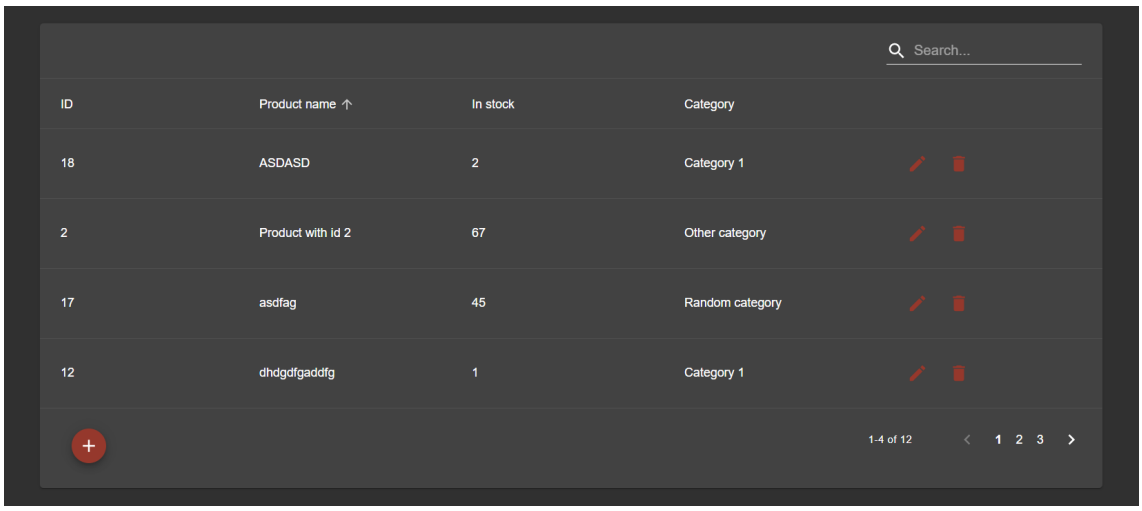
Bejelentkezés után a kezdőképernyőn a vállalkozás statisztikáit láthatjuk. A grafikon vízszintes tengelyén idő, függőleges tengelyén pedig a bevétel/kiadás értéke látható.



3.2. ábra. Statisztikák képernyő

3.3. Termékek

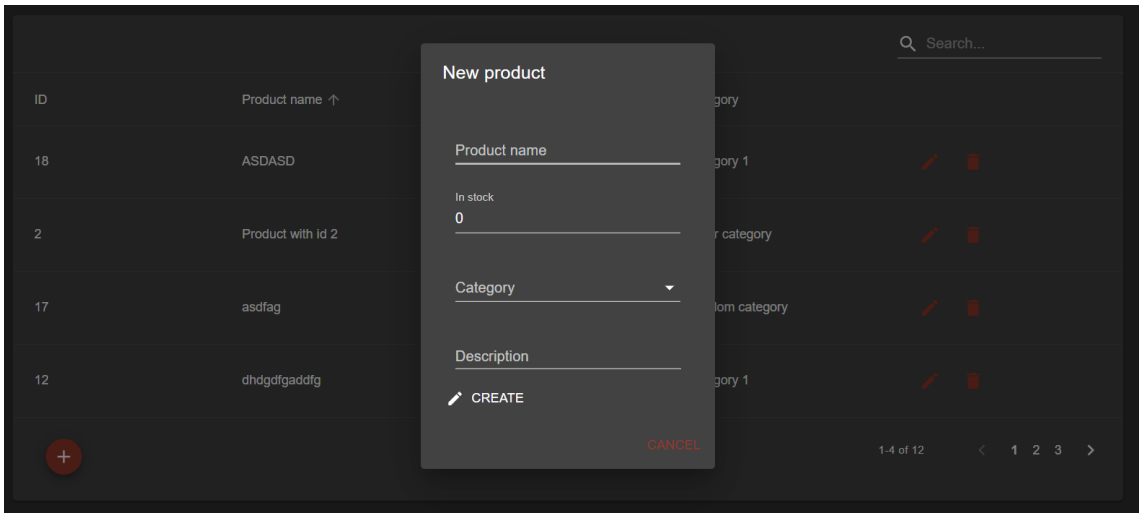
A termékek képernyőn tudja a felhasználó ellenőrizni a termékeinek adatait. Az egyes oszlopok könnyen lecserélhetőek, így egy valós alkalmazásnál ez nem jelentene gondot.



3.3. ábra. Termékek képernyő

3.3.1. Új termék létrehozása

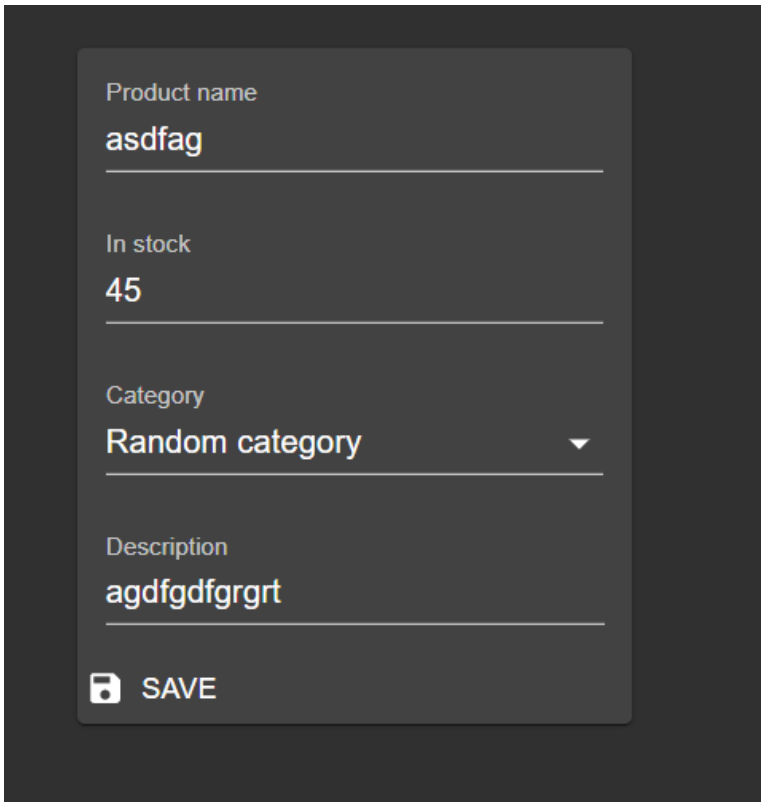
A + gombra kattintva megjelenik egy felugró ablak. Itt az adatok megadása után a create gombra kattintva tud a felhasználó új elemet létrehozni. A kategória kiválasztásánál az előre definiált kategóriák közül lehet válogatni automatikus kitöltés segítségével.



3.4. ábra. Új termék képernyő

3.3.2. Termék módosítása

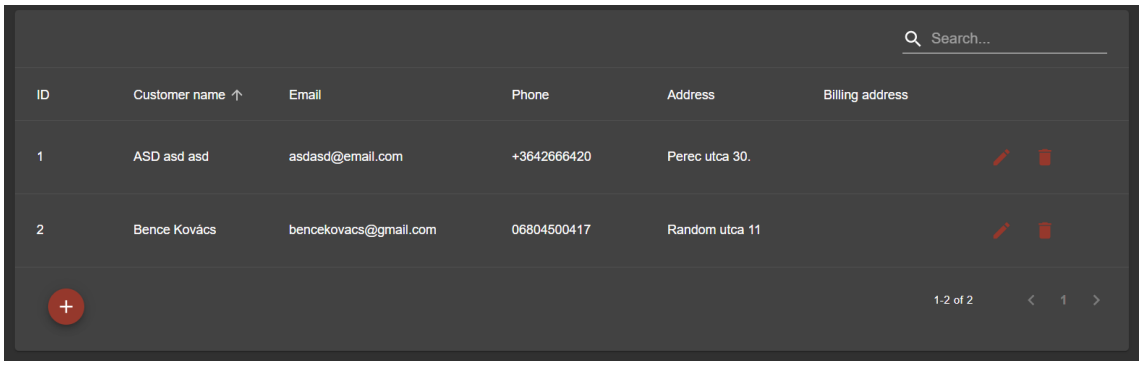
Meglévő termékek módosításához a módosítandó termék szerkesztő ikonjára kattintva juthatunk el. Ugyanaz a szerkesztő mint új létrehozásánál, annyi különbséggel hogy már ki vannak töltve az értékek.



3.5. ábra. Termék módosítása képernyő

3.4. Ügyfelek

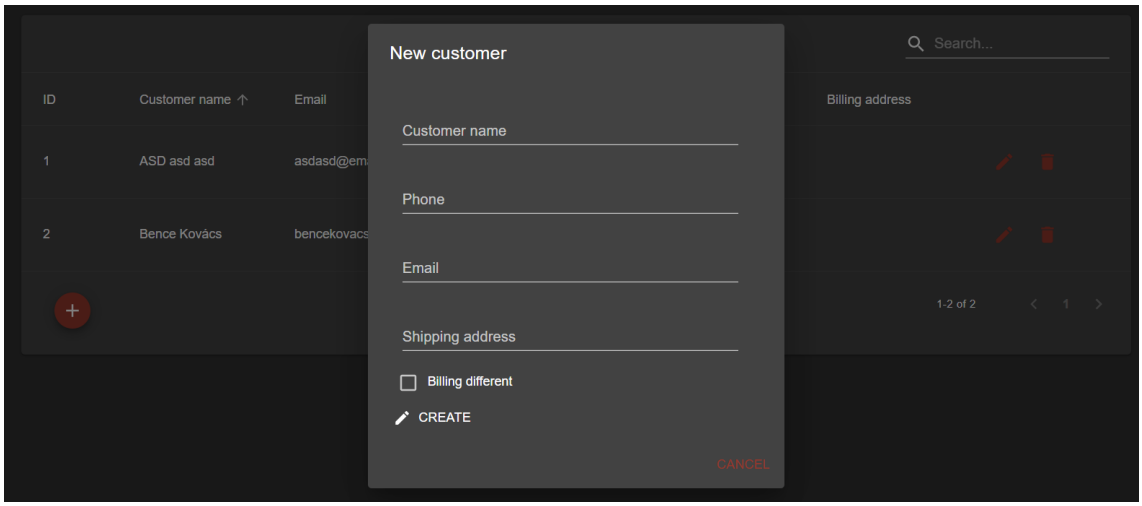
Az ügyfelek képernyőn a vállalkozásunk gyakori ügyfeleinek elérhetőségeit tudjuk számon tartani.



3.6. ábra. Ügyfelek képernyő

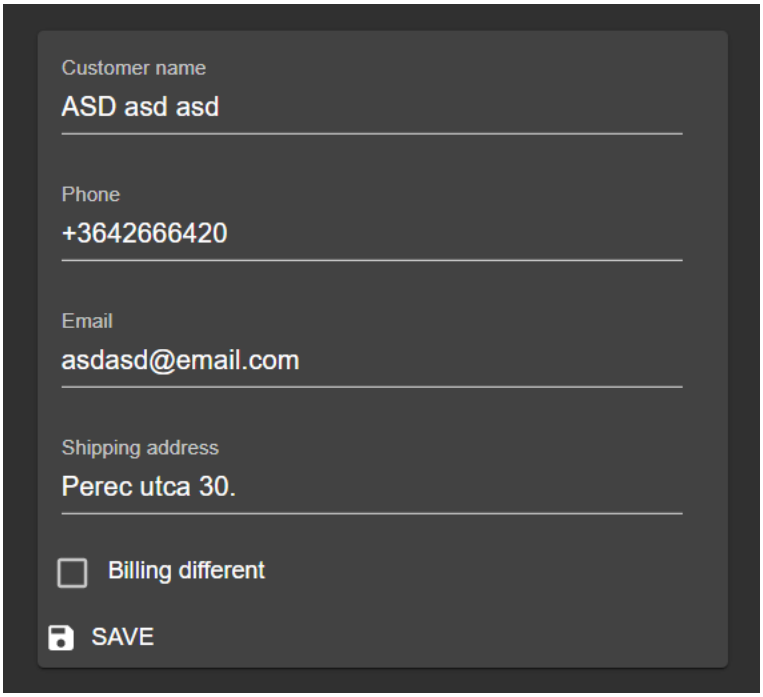
3.4.1. Új ügyfél létrehozása

Hasonlóan a termékekhez, itt is a + gomb segítségével tudunk új ügyfelet létrehozni. Ha eltér a számlázási cím, azt meg lehet adni a megfelelő checkbox bepipálásával.



3.7. ábra. Új ügyfél képernyő

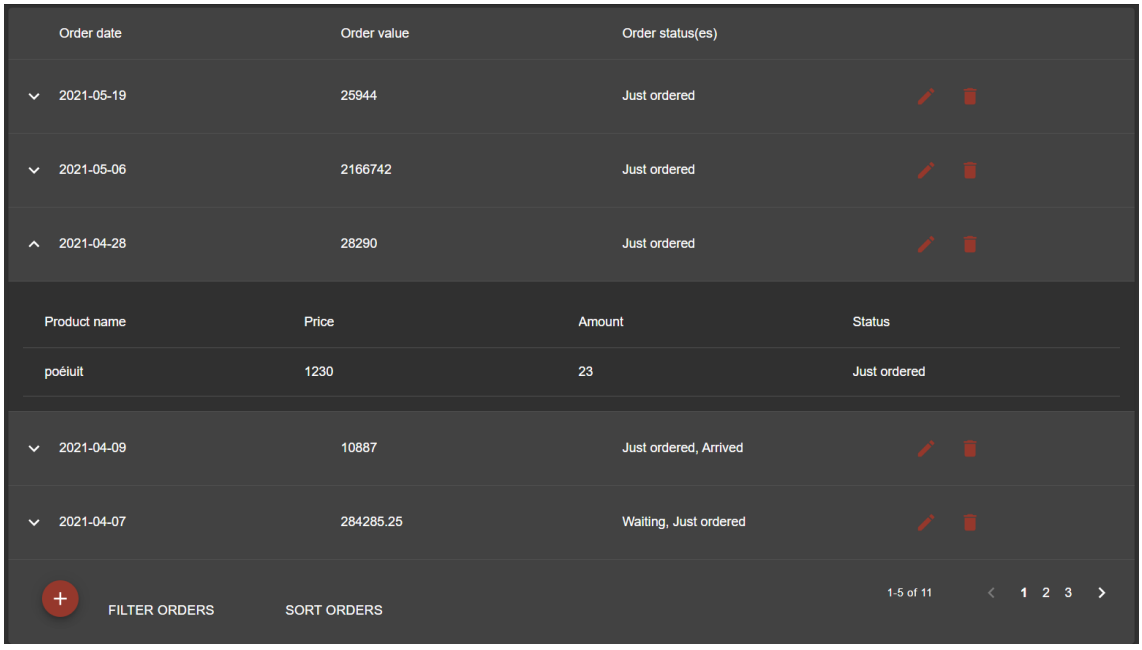
3.4.2. Ügyfél módosítása



3.8. ábra. Ügyfél módosítása képernyő

3.5. Rendelések

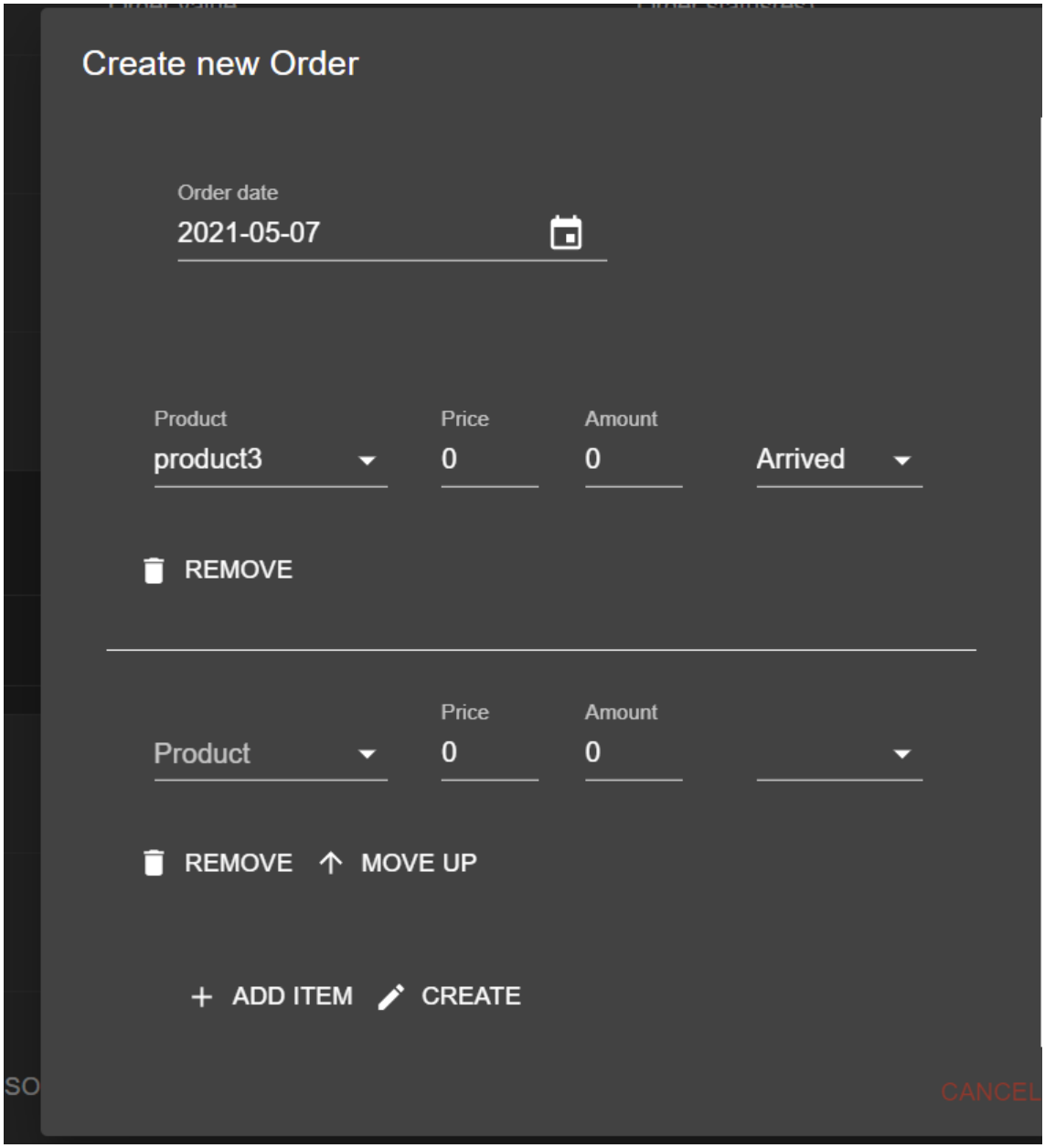
A rendelések képernyőn tarthatjuk karban a beszerzési adatokat. Mikor mit rendeltünk, mennyit mennyiért. A statisztikák ez alapján készítik el a kiadási adatokat. Az alul található gombok segítségével tudunk szűrőket és rendezőket beállítani ha egy konkrét rendelésre vagyunk kíváncsiak.



3.9. ábra. Rendelések képernyő

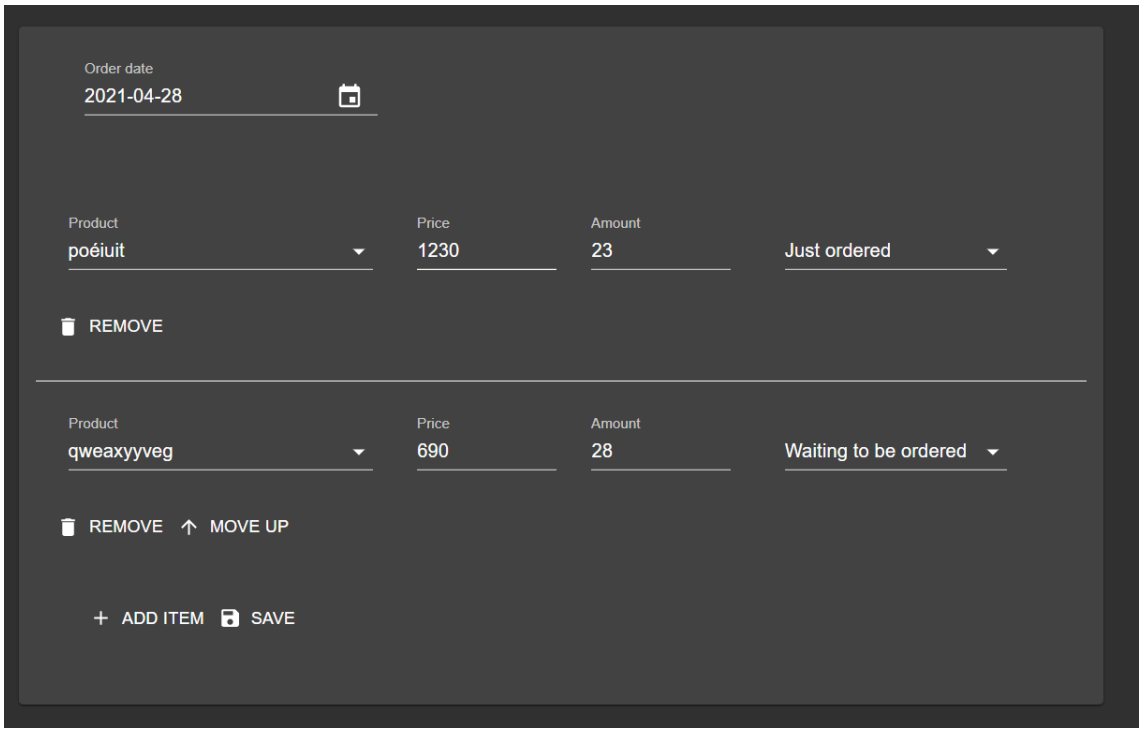
3.5.1. Új rendelés létrehozása

Új rendelés létrehozásakor az egyes vásárolt termékeket adjuk meg. A termékek kiválasztása automatikus kitöltéssel működik meglévő termékek közül.



3.10. ábra. Új rendelés képernyő

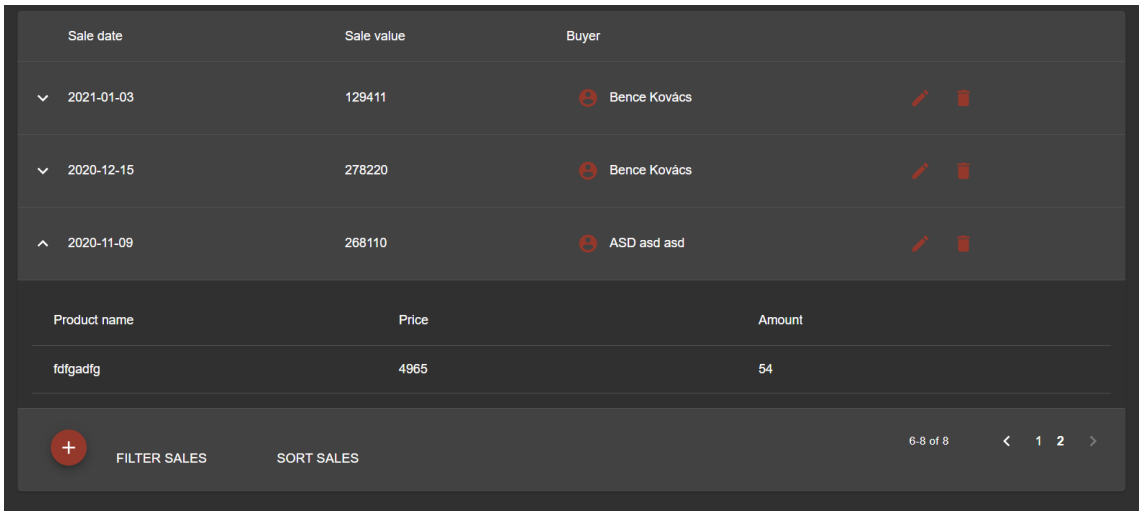
3.5.2. Rendelés módosítása



3.11. ábra. Rendelés módosítása képernyő

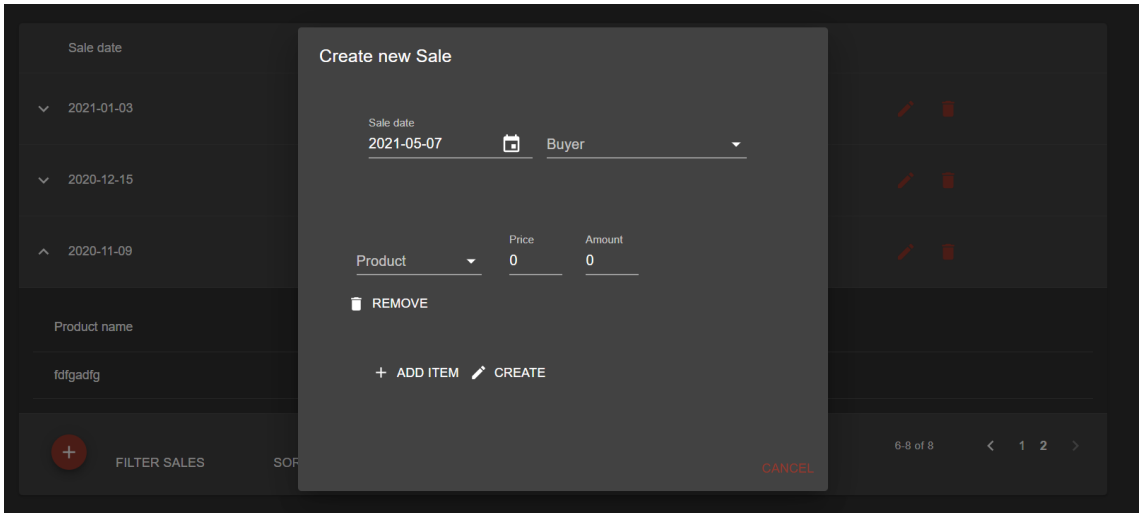
3.6. Eladások

Az eladások képernyőn láthatjuk értelemszerűen az eladásainkat. A statisztikák ez alapján számolja a bevételeinket. Ha megadjuk melyik ügyfélnek adtunk el akkor az ügyfél neve melletti gombra kattintva könnyen megtalálhatjuk az ügyfél elérhetőségeit



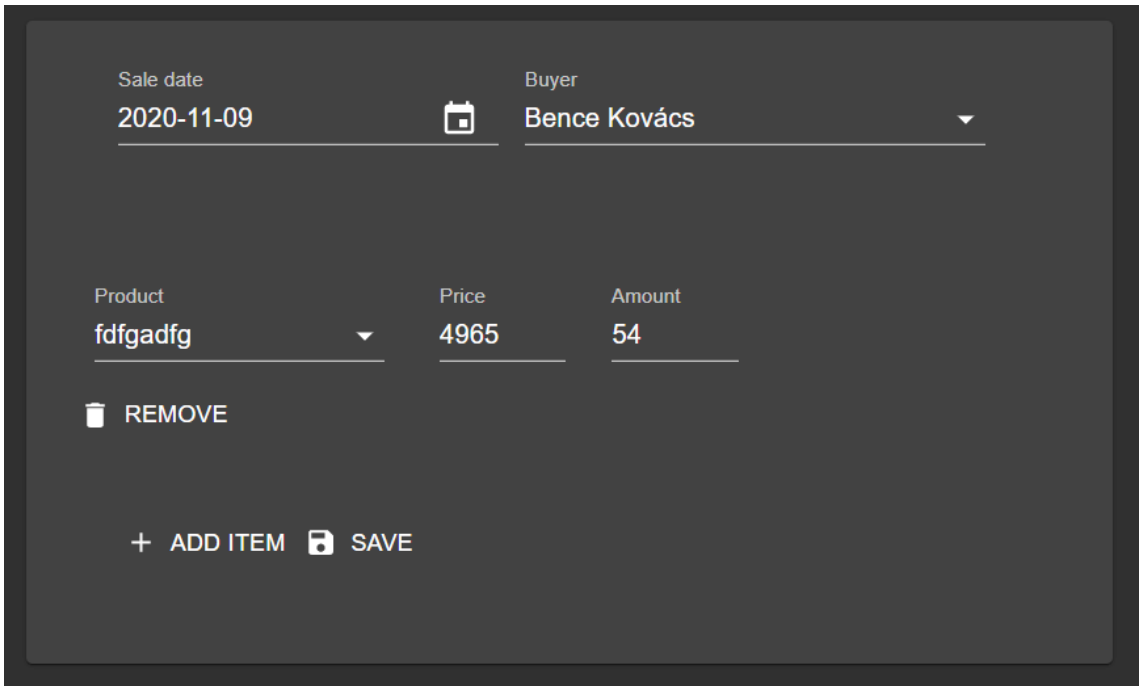
3.12. ábra. Eladások képernyő

3.6.1. Új eladás létrehozása



3.13. ábra. Új eladás létrehozása képernyő

3.6.2. Eladás módosítása



3.14. ábra. Eladás módosítása képernyő

4. fejezet

Összefoglalás és kitekintés

4.1. Összefoglalás

A kész alkalmazás véleményem szerint egy jó kiinduló alap lehet egy ügyfélkörrel rendelkező kisebb vállalkozásnak. Éles működésben valószínűleg szükség lenne személyreszabásra, hiszen minden vállalkozás különböző attribútumokat szeretne látni termékeken, eladásokon vagy akár az ügyfelein, de a fejlesztés során igyekeztem maximálisan odafigyelni a bővíthetőség szem előtt tartására.

4.2. Kitekintés

Az alkalmazás továbbfejlesztéséhez érdemes lenne akár egy konfigurációs fájlból beolvashatóvá tenni az egyes entitások felépítését. Például a frontend felépítése során az ügyfelek és a termékek megjelenítése nagyon hasonló, a szerkesztés minden entitásnál ugyanúgy működik. Így a megfelelő adatok megadásával könnyen példányosítható lehetne további fejlesztés nélkül az említett személyreszabás. Ezentúl bővíteni a projektet természetesen egy egyszerű webshop megvalósításával lehetne. Ezzel automatizálhatnánk az eladások egy részét és elősegíthetnénk a vállalkozás további fejlődését.