XXI. reál- és humántudományi Erdélyi Tudományos Diákköri Konferencia (ETDK)
Kolozsvár, 2018. május 24–27.

Netfood

Ételkiszállítást támogató szoftverrendszer

Szerzők:

Domokos Cristina-Edina
Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar, Informatika szak, III. év
Séra Barna
Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar, Informatika szak, III. év

Témavezetők:

dr. Simon Károly, egyetemi adjunktus
Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar
Kovács Lajos, szoftverfejlesztő, Codespring
Szakács Tas-Béla, szoftverfejlesztő, Codespring

CODESPRING
OUR CODE YOUR SUCCESS

Kivonat

A Netfood projekt célja egy olyan szoftver kifejlesztése, amely segíti az ételkiszállító cégek munkáját, valamint a rendelések követését. Egy olyan kiszállító-orientált rendszer, amely lehetővé teszi, hogy egyszerre több étteremtől is lehessen rendelni, lehetőséget biztosítva egyéni és csoportos rendelésekre is.

A felhasználók egy webes felületen tudnak rendeléseket leadni. A futárokat egy mobil alkalmazás segíti a kiszállításban. Az adminisztrátorok szintén egy webes felületen menedzselhetik az adatokat (menük, éttermek, rendelések). Az említett alkalmazásokat egy központi szerver szolgálja ki adatokkal.

A dolgozat bemutatja a szoftverrendszer felépítését, megvalósítását, valamint az ehhez felhasznált technológiákat, eszközöket és módszereket.

Tartalomjegyzék

Bevezető

A Netfood projekt célja egy kiszállító-orientált rendelésmenedzsment rendszer kifejlesztése, amely lehetővé teszi, hogy a felhasználók egyszerre több étteremtől tudjanak rendelni, és segíti a kiszállítók munkáját, a rendelések követését.

Sok olyan applikáció áll a rendelkezésünkre, amelyek megkönnyítik egy adott városban a helyi éttermektől történő rendelést. Az egyéni rendelések mellett több rendszer támogatja a csoportos rendeléseket, ilyen például a Kolozsváron jól ismert Hipmenu. Ezek a rendszerek általában étterem-orientáltak: egy rendelésen belül csak egy étteremtől igényelhet kiszállítást a felhasználó. Ez a modell jól működik ott, ahol az éttermek külön oldják meg a kiszállítást, de vannak települések (főleg kisebb városok), ahol az éttermek ugyanazzal a kiszállító céggel oldják meg a rendelések eljuttatását a kliensekhez.

Konkrét példaként megemlíthető Székelyudvarhely, ahol az Édes Manna nevű kiszállító céggel működnek együtt a helyi éttermek. Jelenleg minden rendelésnél fel kell hívni a kiszállítócéget, ami nem a legkényelmesebb megoldás, ráadásul előfordul, hogy foglalt a vonal és újra kell próbálkozni, vagy megvárni amíg visszahívnak. Egy kiszállító-orientált rendelésmenedzsment szoftver sokat javíthatna a helyzeten.

A Netfood egy webes felületet biztosít a felhasználók számára, amelyen meg tudják tekinteni az éttermeket, illetve menüket, és rendeléseket tudnak leadni. A felület lehetőséget ad egyéni és csoportos rendelések létrehozására is. Más rendszerekkel ellentétben lehetővé teszi azt, hogy mind az egyéni, mind a csoportos rendelések esetében lehessen a kiszállítócéggel kapcsolatban álló összes étteremtől egyszerre rendelni, nem szükséges külön rendelést leadni minden étteremnek.

A rendszer rendelkezik egy adminisztrációs felülettel is, ahol a kiszállítócég munkatársai követhetik a beérkezett rendeléseket, illetve menedzselhetik az adatokat: új éttermeket, menüket hozhatnak létre, módosíthatják a már létezőket.

A futárokat egy mobil alkalmazás segíti munkájukban. Ennek segítségével megnézhetik a beérkezett rendeléseket, elvállalhatják azokat és minden szükséges információt megkapnak, amelyek elősegítik a sikeres kiszállítást.

A dolgozat bemutaja a Netfood szoftverrendszer felépítését és működését, a fejlesztési folyamatot, valamint a felhasznált technológiákat, fejlesztési eszközöket és módszereket. Az első fejezetben a projekt funkcionalitásairól és architektúrájáról lesz szó. A második fejezet a szerver felépítéséről, fejlesztésének fontosabb lépéseiről és az ehhez felhasznált technológiákról szól. A harmadik fejezet a webes kliens megvalósítását, fejlesztését és technológiai hátterét ismerteti. A negyedik fejezet az Android alkalmazás megvalósítását írja le. Az ötödik fejezetben a felhasznált eszközök és módszerek kerülnek bemutatásra. A hatodik fejezet az Android alkalmazás és

a webes kliens működését szemlélteti példákon keresztül. A dolgozat néhány következtetéssel és továbbfejlesztési lehetőséggel zárul.

A projekt a U-Hub Mentorprogram keretein belül indult, a Codespring koordinálásával. A nyári szakmai gyakorlat alatt a dolgozat szerzői végezték a fejlesztést, majd a csoportos projekt egyetemi tantárgy kezdetekor kibővült a csapat: egy egyetemi félévre csatlakoztak Ferencz Lóránt, Orbán Ákos és Sebestyen Robert-Tiberiu egyetemi hallgatók. A csoportos projekt lezárása után a szerzők folytatták a fejlesztést. Ugyancsak a csoportos projekt kezdetekor csatlakozott mentorainkhoz Hobaj Roland, aki azóta is szerepet vállal a projekt szakmai irányításában. A szerzők ezúton is szeretnének köszönetet mondani mentoraiknak, szakmai irányítóiknak és támogatóiknak, akik hozzájárultak a projekt létrejöttéhez.

1. A Netfood projekt

A Netfood szoftverrendszer kiszállítócégeknek segít az ételrendelések könnyebb lebonyolításában. A rendszer ennek érdekében két kliensalkalmazást biztosít: egy webes és egy Android alkalmazást. A webes rész az adminisztrátoroknak és a megrendelőknek nyújt egy egyszerű és könnyen használható felületet, a mobilalkalmazás a futárok munkáját teszi kényelmesebbé.

1.1. Funkcionalitások

1.1.1. A szerver funkcionalitásai

A szervernek két fő feladata van, az egyik a kliensektől érkező kérések kiszolgálása, a másik az adatbázissal való kommunikáció. Egy webes- és egy mobilos klienssel áll kapcsolatban, amelyekkel a kommunikáció RESTful API segítségével történik. A szerver egy relációs adatbázisban tárolja az adatokat, amikor szükséges lementi, lekérdezi, módosítja vagy törli azokat.

A beérkező kérések alapján kapcsolatba lép az adatbázissal, az adatokat feldolgozza, majd visszaküldi a választ. Hiba esetén a kérésnek megfelelő hibaüzenetet küld. Menedzseli a felhasználókat és biztonsági megoldásokat biztosít, így a szerveren az adatok védve vannak az illetéktelen hozzáférésekkel szemben.

1.1.2. A webes felület funkcionalitásai

A webes felületen keresztül a kliensek és az adminisztrátorok tudnak bejelentkezni. Ez történhet Facebook-on keresztül, vagy a rendszerben regisztrált felhasználóval.

A felhasználók meg tudják tekinteni az összes kategória ételeit, valamint a napi menüket egy adott dátumon. Az általuk kiválasztott ételt egy gombnyomással a kosárhoz adhatják. A kosáron belül lehetőségük nyílik a már kiválasztott fogások eltávolítására, a mennyiségek módosítására, illetve megjegyzések írására. Ugyanakkor kiválaszthatják, hogy milyen címre várják a kiszállítást. A telefonszám szintén megadható, ha ez a regisztráció során nem történt meg. Az Order gomb megnyomásával leadható a rendelés.

A felhasználóknak lehetőségük van csoportos rendelés indítására is, ekkor a cím kiválasztása után, meghívhatják azokat a barátaikat, akikkel együtt szeretnének rendelni. A rendelésbe bekapcsolódó felhasználók megtekinthetik azt, hogy ki milyen ételt választott magának, új felhasználókat hívhatnak be a rendelésbe, és ha végeztek, akkor jelezhetik ezt a többieknek. Amikor minden résztvevő befejezte, a rendelést, akkor az leadhatóvá válik.

A kliensek számára további funkcionalitások is elérhetőek: a már leadott rendeléseket vissza tudják nézni, megtekinthetik a kiszállítócéggel kapcsolatban álló éttermeket, lehetőségük van a

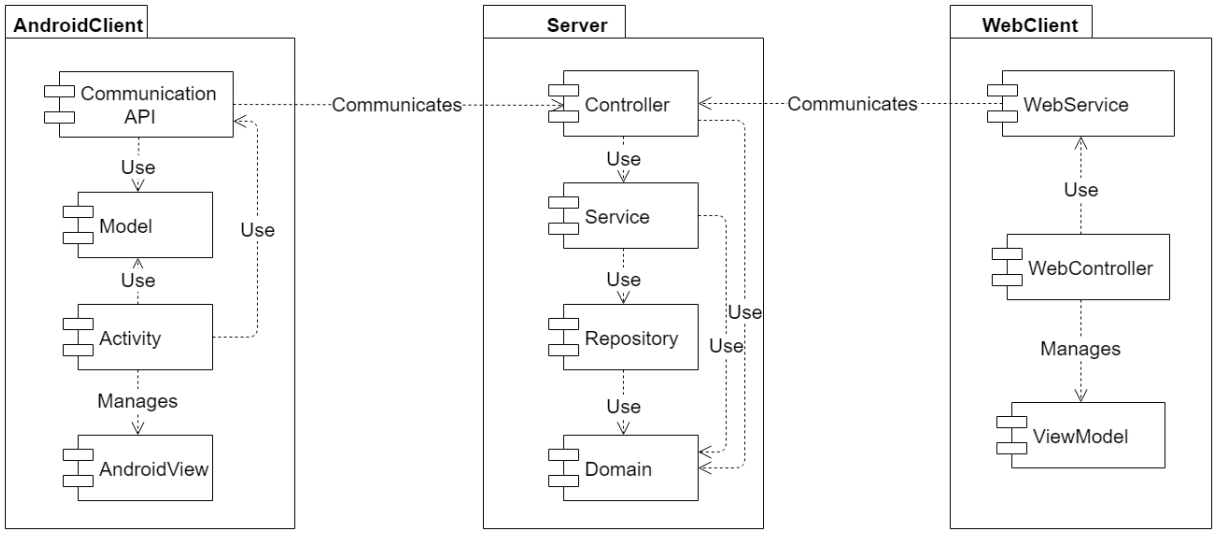
profiljuk szerkesztésére, valamint barátnak is felvehetnek más felhasználókat, akikkel a későbbiekben csoportos rendeléseket indíthatnak.

Az adminisztrátorok megnézhetik a beérkezett rendeléseket az általuk kiválasztott dátumon. Ők tudják menedzselni (hozzáadni, törölni, módosítani) az ételeket, éttermeket, kategóriákat, valamint megvátoztathatják a rendszerben levő felhasználók szerepkörét: pl. felhasználóból lehet futár vagy adminisztrátor.

1.1.3. A mobil alkalmazás funkcionalitásai

A mobil alkalmazás a rendelések kezelésében segít a kiszállítóknak. Futár szerepkörrel rendelkező felhasználók tudnak belépni az alkalmazásba, ahol a sikeres bejelentkezés után meg lehet nézni, hogy arra a napra milyen rendeléseket adtak le a felhasználók. A rendeléseket elfogadva vállalják azok kiszállítását. Mind az egyéni, mind a csoportos rendeléseknek el lehet fogadni csak egy részét is, például amely rész egy bizonyos étteremhez tartozik. A már elfogadott rendeléseket meg lehet tekinteni egy külön listában, és itt megjelennek a kiszállításhoz szükséges információk is. Ezek között megtalálható a megrendelő telefonszáma, amelyet közvetlenül az alkalmazásból is hívhatnak. Ha végeztek egy elvállalt rendelés kiszállításával, akkor ezt be tudják jelölni, visszaigazolást rögzítve a rendszer számára.

1.2. Architektúra



1. ábra. A rendszert három modul alkotja: a szerver, a webes kliens és az Android alkalmazás. A szerver esetében a Controller réteg komponensei fogadják a beérkező kéréseket a két klienstől, és a Service réteg komponenseit felhasználva elvégzik a kért műveleteket, a Repository rétegen keresztül kommunikálva az adatbázissal. Az Android kliens esetében az AndroidView, a Web kliens esetében pedig a ViewModell felelős a felhasználói felületért. Az Android kliens Communication API és a webes kliens WebService rétegei a szerverrel való kommunikációt oldják meg. Az Activity és WebController réteg az adatfeldolgozásért felelős.

Architektúra szempontjából három fő komponensből áll a rendszer: a Java alapú, Spring technológiákra épülő szerverből, az Angular 4 alapú webes, illetve az Android kliensalkalmazásból.

A szerver többrétegű architektúrával rendelkezik. A Repository modul felel az adatbázissal való kommunikációért és minden adatmanipulációval kapcsolatos feladatért. Az adatokat modell osztályok reprezentálják, amelyek a Domain modulban találhatóak. Ezeket a POJO osztályokat a szerver minden komponense felhasználja. A beérkező REST kéréseket a Controller fogadja, és a Service megfelelő részeit felhasználva előállítja a szükséges válaszokat. A Service réteg tartalmazza az üzleti logikát, a Repository réteg segítségével manipulálva a műveletek elvégzéséhez szükséges adatokat.

Az Android kliens az MVC (Model–View–Controller) elvet követi. A Model réteg tartalmazza az adatokat reprezentáló osztályokat. Az Android View réteg nézetei határozzák meg, hogy ezek az adatok milyen formában legyenek megjelenítve a telefon képernyőjén. Az Activity réteg osztályai a vezérlésért felelnek, az alkalmazás a Communication API segítségével küld REST kéréseket a szerver felé, illetve az onnan kapott válaszokat is itt fogadja.

A webes kliens a szerverrel a WebService rétegen keresztül kommunikál, ennek a szolgáltatásait a WebController réteg használja fel, és ez a modul felelős a ViewModell vezérléséért is, amely a felhasználói felület megjelenítését teszi lehetővé.

2. Szerver

A szerver implementálása egy Java keretrendszer, a Spring [1] segítségével valósult meg, amely főként összetettebb vállalati alkalmazások fejlesztésére szolgál. Flexibilis és különböző szolgáltatásokat biztosító modulokat tartalmaz. Ezek közül csak azok vannak beépítve a rendszerbe, amelyek funkcionalitásaira ténylegesen szükség van.

A keretrendszer biztosít egy Inversion of Control (IoC) konténert, amely a komponensek összeillesztését, konfigurálását és kezelését teszi lehetővé. Lehetőséget ad Dependency Injection (DI) minta használatára, amely az IoC egy formája, a komponensek függőségeinek megoldására szolgál, elkülöníti az objektumok létrehozását és ezek használatát.

2.1. Spring Boot

A Spring Boot [2] lehetővé teszi, hogy önálló Spring alapú alkalmazásokat lehessen létrehozni, amelyek kevés konfigurációt igényelnek. Az alkalmazás létrehozásakor a kezdeti beállításokat nem bízza a fejlesztőre, hanem hasonló alkalmazásokra jellemző konfigurációt vesz alapértelmezettnek, felgyorsítva a kezdeti fejlesztést. A beállítások bármikor könnyen módosíthatóak. A Spring Boot egy beépített webszerverrel is szolgál, így nem szükséges külön telepíteni az alkalmazást. A projekt esetében Tomcat webszerver volt használva.

```
@SpringBootApplication
public class NetfoodApplication {

        public static void main(String[] args) {
                SpringApplication.run(NetfoodApplication.class, args);
        }
}
```

1. kódrészlet. A main metódus a SpringApplication.run függvényét meghívva elindítja a webszervert és elérhetővé teszi lokálisan.

2.2. Maven build-rendszer

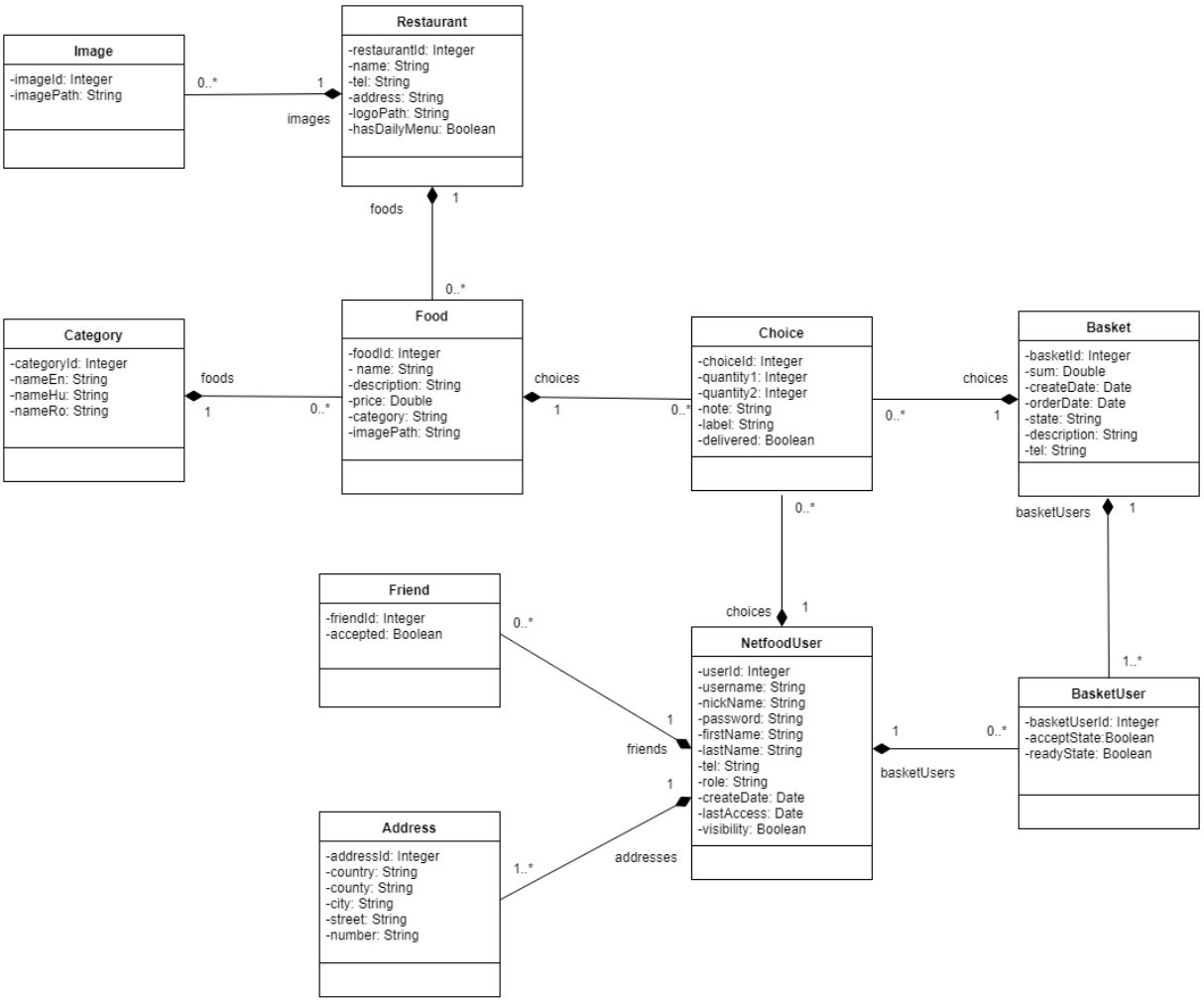
A szerver fejlesztése során használt build rendszer az Apache Maven, ez egy automatizált build, függőség- és projektmenedzsment eszköz, amely elsősorban Javaban írt alkalmazások fejlesztése során használatos. A Project Object Model (POM) koncepcióra épül, egy XML konfigurációs állományban határozza meg a fordítási lépéseket, a verziószámot, a függőségeket és a projekt szerkezetét.

2.3. Adatmodell

A rendszeren belül az entitások Java Bean osztályok segítségével vannak reprezentálva. Ezek rendelkeznek privát adattagokkal, ezekhez tartozó publikus getter és setter metódusokkal, illetve publikus paraméter nélküli konstruktorral. A projekt esetében használt entitások az *edu.codespring.netfood.domain* csomagban találhatóak.

Ezek az osztályok az *@Entity* és *@Table* JPA annotációkkal vannak ellátva, minden entitás egy táblának felel meg az adatbázisból.

A fontosabb entitások egyike a *Food* osztály, amelynek egy példánya egy ételt reprezentál a hozzá tartozó specifikus információkkal együtt. Ahhoz, hogy az adott ételhez kategória is legyen hozzárendelve (napi menü, leves stb.) a *Food* osztályt aggregáció kapcsolja össze a *Category* osztállyal (n:1-hez viszony). Hasonlóan, annak érdekében, hogy egy rendelés esetében az adott étel hozzáadható legyen egy kosárhoz, 1:n-hez kapcsolatban van a *Choice* entitással, amely tartalmazza a kiválasztott fogás esetében a kért mennyiséget, valamint a hozzá kapcsolódó megjegyzéseket. A *Choice* n:1 kapcsolatban áll a *Basket* osztállyal.



2. ábra. Osztálydiagram, amely szemlélteti a Netfood modellosztályait, azok adattagjait és a kapcsolataikat egymással.

Egy másik fontos entitás a *NetfoodUser*, amelynek egy instanciája egy felhasználó adatait tartalmazza (név, email, jelszó stb.). Ahhoz, hogy egy rendelés kiszállítható legyen, szükség van a felhasználók címére, ezt az *Address* osztály reprezentálja, amely n:1-hez viszonyban van a *NetfoodUser* osztállyal.

2.4. Adathozzáférési réteg

Az adathozzáférési réteg feladata, hogy kapcsolatot létesítsen az adatbázissal, illetve lehetővé tegye az adatokkal elvégzendő műveleteket (lekérdezés, módosítás, beszúrás). A Netfood projekt esetében az adatok tárolása egy MySQL adatbázisban történik.

Az adatok perszisztálására a szerver a Hibernate Object-Relational Mapping (ORM) keretrendszert használja, amely fölött absztrakciós szintet képez a Spring Data JPA. A modul használatával nem szükséges a Data Acces Object (DAO) osztályokat implementálni, elegendő az entitásokat ellátni a megfelelő JPA annotációkkal, és megírni a megfelelő Repository interfészeket a JpaRepository interfészből származtatva. Ezekben lehet metódusokat deklarálni, amelyek nevei adott konvencióknak kell megfeleljenek és a rendszer a metódusnevek alapján generálja a megfelelő lekérdezéseket. Bonyolultabb lekérdezések esetében a *@Query* annotáció után megadható a konkrét JPQL utasítás (2. kódrészlet).

A Netfood repository interfészei a *edu.codespring.netfood.repository* csomagban találhatóak.

```
public interface RestaurantRepository extends
JpaRepository<Restaurant, Integer> {

        Restaurant findByName(String name);

        @Query("select r from Restaurant r where r.hasDailyMenu = true
    and not exists ( select f2 from Food f2 where f2.addDate = :date
    and f2.restaurant.restaurantId = r.restaurantId and
    f2.category.nameEn = 'Daily Menu')" )
        List<Restaurant> getRestaurantsWithoutMenu(@Param("date") Date date);

}
```

2. kódrészlet. A RestaurantRepository interfészen belül a metódusok megfelelő névvel való deklarálása elegendő, hogy a rendszer a megfelelő lekérdezéseket generálja és végrehajtsa. Bonyolultabb lekérdezés esetében a @Query annotáció után megadható az a JPQL kód, amely végre lesz hajtva a metódus meghívásakor.

2.5. Szolgáltatási réteg

A szolgáltatási réteg az alkalmazás üzleti logikájáért felelős és kapcsolatban áll az adathozzáférési réteggel. A szerverhez beérkező kéréseket a controllerek ide továbbítják, és itt megtörténik az adatok megfelelő feldolgozása. Meghívják az adathozzáférési réteg szolgáltatásait és a felépített választ visszaküldik a hívó rétegnek.

A szolgáltatási réteghez tartozó osztályok a *@Service* annotációval vannak ellátva. A Netfood projekt esetében az *edu.codespring.netfood.service* csomagban találhatóak.

2.6. RESTful API

A REST (Representational State Transfer) a kliens és a szerver közötti kommunikációra tervezett architektúra modell, tipikusan HTTP protokoll alapján történő kommunikációt tesz lehetővé. Az adatok és a velük végzett műveletek erőforrásokként vannak kezelve. Ezeket az erőforrásokat a következő négy művelet segítségével lehet manipulálni:

- POST: erőforrás létrehozása
- DELETE: erőforrás törlése
- GET: erőforrás állapotának lekérdezése
- PUT: erőforrás állapotának megváltoztatása

Az adatátvitel különböző formátumokban történhet, mint például XML, JSON, HTML, PDF stb. A Netfood projekt esetében ez a formátum a JSON.

2.6.1. Spring Web MVC

A Spring Web MVC keretrendszer segít MVC (Model-View-Controller) elvet követő webalkalmazások és RESTful webszolgáltatások fejlesztésében.

A keretrendszer egy DispatcherServlet köré épül, ez fogadja a beérkező kéréseket és továbbküldi a megfelelő *handler*-nek. A *handler* osztályok a *@RestController* és *@RequestMapping* annotációkkal vannak ellátva (3. kódrészlet).

Az erőforrásokhoz *controller* oszályok vannak társítva. A *@RequestMapping* annotáció adja meg, hogy a beérkező kérés melyik vezérlőhöz továbbítódjon. A *controller* osztályon belüli metódusok a *handler*-ek, megkapkják a kérést és továbbítják azt a szolgáltatási réteg felé.

```java
@RestController
@RequestMapping("/api/foods")
public class FoodController {

        @Autowired
        private FoodService foodService;

        @GetMapping("/{id}")
        public Food getFood(@PathVariable Integer id) {
                return foodService.getFood(id);
        }
}
```

3. kódrészlet. A *FoodController* osztály *@RestController* és a *@RequestMapping* annotációkkal ellátva képes az "/api/foods" előtaggal kezdődő kérések fogadására, a megfelelő handler pedig továbbítja a szolgáltatási réteg felé ezeket a kéréseket.

2.7. Spring Security

A Spring Security keretrendszer az alkalmazás biztonságáért felelős, elsősorban a felhasználók azonosítását és a jogosultságok kezelését valósítja meg. A keretrendszer könnyen kiegészíthető, ezért nagyon egyszerű beépíteni egy token alapú autentikációt és autorizációt. A Netfood szerver esetében ez JWT (JSON Web Token) *token*-ek segítségével történik.

A JWT egy nyílt szabvány, amely meghatározza a kommunikáló felek közti információ biztonságos továbbítását JSON objektumok segítségével. A *token* három karakterláncból áll, ezek ponttal vannak elválasztva egymástól. Az első rész a fejléc, itt a *token* típusa és a használt titkosítási algoritmus található. A második rész a hordozott információ, a harmadik rész az aláírás.

Az autentikáció során, ha a felhasználó sikeresen azonosította magát, akkor a szerver a megfelelő adatokkal generál egy JWT-t és azt visszaküldi a kliensnek. A kliens elmenti a *token*-t, és ha bármikor egy erőforrásra van szüksége a szervertől, az autorizációs *header*-ben küldi a kéréssel együtt, a következő alakban: **Authorization: Bearer <token>**. A szerver ellenőrzi az adott *token* érvényességét, és ha az megfelel, akkor engedélyezi az adott erőforrás elérését. Ez a mechanizmus lényegesen lecsökkenti az adatbázis lekérdezések számát.

2.8. Mail service

Sikeres regisztráció esetében a rendszer egy e-mailt küld a megadott címre, hogy az újonnan regisztrált személy aktiválhassa a felhasználóját. Ugyanúgy egy e-mailt küld a rendszer abban az esetben is, ha egy felhasználó az 'Elfelejtett jelszó' opciót használja. Az e-mail mindkét esetben tartalmaz egy linket, amelynek segítségével a felhasználó azonosítani tudja magát, hogy elvégezhesse a szükséges műveleteket. Az e-mail küldés a Spring keretrendszer által biztosított *JavaMailSender* interfész segítségével van megvalósítva.

Az e-mailekben elküldött link tartalmazza az URL címet, amelyre irányítja majd a felhasználót, valamint egy JWT-t. A *token* segít azonosítani a felhasználót. Generálása az e-mail cím alapján történik, beépítve a felhasználhatósági időkorlátot is.

2.9. Liquibase

A Liquibase egy adatbázis verziókövető eszköz, amely a fejlesztés során az adatbázis állapotváltozásait menedzselte. Minden változást egy *changelog* fájlban tárol, amely lehet XML, YAML, JSON vagy SQL formátumú. A Netfood esetében XML alapú *changelog* fájl van használva.

A Liquibase lehetővé teszi az adatbázis frissítésének az automatizálását. A konfigurációs állományban megadott adatbázishoz csatlakozik, és ha vannak új változtatások, akkor az azoknak megfelő műveleteket végrehajtja. Ezek a változások *changeset*-ekbe vannak szervezve, lehetnek új tábla létrehozására, új adat beszúrására, új oszlop hozzáadására, törlésére stb. vonatkozó műveletek.

3. Web kliens

3.1. Felhasznált technológiák

3.1.1. Angular 4

Az Angular [3] egy *frontend* JavaScript keretrendszer, amely *single page application* (SPA) típusú alkalmazások fejlesztését teszi lehetővé. Komponens alapú architektúrát biztosít, támogatja az MVC tervezési mintát. A nézetek Angular direktívákkal felruházott HTML oldalak. Ezek a sablonok a dinamikus oldalak fejlesztését segítik.

A keretrendszer adatkötést (*data binding*) biztosít, összeköti az adatmodellt a nézettel. Ez lehet *one-way data binding*, amikor például az adatmodell kerül kiíratásra a *view*-ra és ha a modellben vátozás történik az látszik a nézeten. Lehetőség van *two-way data binding*-ra is, ekkor ha a nézetben történt változás, akkor az látható a modellben, illetve ez fordítva is érvényes.

3.1.2. TypeScript

A Netfood projekt webes kliensének fejlesztése TypeScript segítségével történt, amely a Microsoft által fejlesztett objektumorientált script nyelv.

Lehetőséget ad az osztályok és típusok használatára, valamint támogatja az öröklődést. Ezek biztosításával szigorúbban meghatározhatóak az adatmodellek, átláthatóbbá válnak, valamint a hivatkozások biztonságosabbak lesznek, így a típus inkompatibilitás elkerülhető.

A TypeScript egy böngésző-független nyelv, a forráskódból egy fordító JavaScript kódot generál.

3.1.3. Ng-bootstrap

A ng-bootstrap egy nyílt forráskódú gyűjtemény, amely a Bootstrap csomag népszerű felhasználói felület elemeit Angular komponensekkel valósítja meg. A Bootstrap keretrendszerrel ellentétben az ng-bootsrap nem használ jQuery-t vagy más JavaScript könyvtárakat.

A Netfood esetében szükség volt olyan UI elemekre is, amelyekkel az ng-bootstrap nem rendelkezik, ezért a Bootstrap is használva van a webes felület felépítésében.

3.1.4. Bootstrap

A Bootstrap egy *frontend* keretrendszer, amely reszponzív, *mobile-first* [4] webes felületek fejlesztését teszi lehetővé. Egy előre megírt eszközkészletet kínál, amelyeket HTML struktúrák, CSS tulajdonságok, JavaScript bővítmények alkotnak. Ezeknek a komponenseknek a felhasználásával a Netfood projekt webes felülete rendelkezik egy alapstílussal, illetve egy egységes

beállítással a különböző böngészők renderelő egységei számára. Ugyanakkor optimális megjelenést biztosít, a felület alkalmazkodik az eszközhöz, vagyis ugyanaz a tartalom más és más megjelenítéssel lesz látható két különböző méretű képernyő esetében. Ennek köszönhetően mobilon is kényelmesen böngészhető a Netfood webes felülete.

3.1.5. Ngx-translate

Az ngx-translate egy nemzetköziesítésre használt könyvtár az Angular számára. Lehetővé teszi a *frontend* többnyelvűsítését, valamint a gyors váltást a nyelvek között.

A könyvtár egy *TranslateService* nevű komponenst tartalmaz, amely különböző metódusok használatát teszi lehetővé, amelyek révén beállítható, hogy mi legyen az alapértelmezett nyelv (*setDefaultLang*), lekérdezhető, hogy milyen nyelv van használatban (*currentLang*), megadható, hogy milyen nyelvet használjon a továbbiakban (*use*) az alkalmazás.

A fordítani kívánt szöveget **kulcs : fordítás** formájában kell megadni az adott nyelvnek megfelelő .json fájlban. A *TranslateService* (4. kódrészlet), *Pipe* (cső, 5. kódrészlet) vagy *Directive* (direktíva, 6. kódrészlet) segítségével lehet betölteni a kulcsszavak által jelölt fordításokat.

Nyelvváltáskor betölti a megfelelő JSON fájlt és lecseléri az összes többnyelvű címkét az oldalon. A TypeScript kódban történő fordítások esetében ugyanazt a JSON fájlt használja fel.

```
translateService.get('food').subscribe((res: string) => {
    console.log(res);
});
```

4. kódrészlet. TypeScript kódból translateService get metódusának segítségével a 'food' kulcsszóhoz tartozó fordítást adja vissza, majd azt kiírja a konzolra.

```
<div>{{ 'food' | translate }}</div>
```

5. kódrészlet. *Pipe* használatával történő fordítás: 'food' kulcsérték által tárolt fordítást írja ki a kiválasztott nyelv .json állományából a weboldalra.

```
<div [translate]="'food'" ></div>
```

6. kódrészlet. *Directive* használatával történő fordítás: 'food' kulcsérték által tárolt fordítást írja ki a kiválasztott nyelv .json állományából a weboldalra.

3.1.6. Google Maps API

A kiszállítócéggel kapcsolatban álló éttermek megtekinthetőek a felhasználók által, helyzetük Google térképen is látható. Erre a Google Maps API van használva a projektben, amelyet

egy API Key segítségével kell betölteni, valamint szükség van a *AgmCoreModule* nevű *angular-google-maps* modulra, hogy a térkép megjeleníthető legyen.

Ahhoz, hogy ezeken a térképeken az éttermek helyzete megjelenjen, szükség van ezeknek a koordinátáira. Mivel az éttermek címe adott, elegendő a kiválasztott vendéglő címét a *'https://maps.googleapis.com/maps/api/geocode/json?address='* linkhez csatolni és egy **GET** kérést küldeni erre a címre. A kapott válasz tartalmazza az étterem koordinátáit.

3.2. Kommunikáció

A webes kliens az Angular HttpClient modulját felhasználva kommunikál a szerverrel. A HttpClient [5] egy egyszerűsített HTTP API-t kínál, amely a XMLHttpRequest API-n alapszik. Képes különböző típusú HTTP kérések végrehajtására a megfelelő metódusokat felhasználva: *get()*, *post()*, *put()*, *delete()* stb. Minden kérés küldésénél fel kell iratkozni a *subscribe()* metódussal a válasz fogadására, ezt követően lesz felépítve és elküldve a kérés a szervernek.

A különböző formátumú válaszok mellett a HttpClient képes a kérések végrehajtása közben fellépő hibákat is kezelni. *Interceptor*-ok használatát is lehetővé teszi, amelyekkel a webes felületről érkező HTTP kérések átalakíthatóak, mielőtt azok ténylegesen továbbítva lesznek a szervernek. Az *interceptor*-ok a backendtől érkező válaszok esetében is alkalmazhatóak, a beérkező választ megváltoztathatják, csak ezután továbbítva azokat a webes kliensnek.

Az alkalmazás esetében vannak olyan szolgáltatások, amelyeket csak különböző szerepkörrel rendelkező felhasználók érhetnek el. Ehhez a bejelentkezéskor kapott JWT *token*-t minden HTTP kérés fejlécében el kell helyezni, így a szerver azonosítani tudja azt a felhasználót, aki az adott szolgáltatáshoz hozzá akar férni, és visszaküldi a megfelelő választ, amelyet a kliens fel-

```
import { HttpClient } from '@angular/common/http';

@Injectable()
export class FoodService {
  constructor(private http: HttpClient) { }

  getDailyMenus(date) {
      return this.http.get('/api/foods/dailyMenu/' + date)
        .catch((error: any) =>
        Observable.throw('Server error(getDailyMenus)'));
  }
}
```

7. kódrészlet. Egy GET kérés lesz küldve a szerver felé, amely a napi menüket szeretné megkapni a paraméterként megadott dátumra. Ha hiba lép fel, azt elkapja a *catch*-ben és egy *Observable*-t dob a megfelelő üzenettel, hogy az őt meghívó metódus hiba esetén a megfelelő műveleteket végre tudja hajtani.

```
getCurrentDatesMenu(date) {
    this.getDailyMenus(date).subscribe(resp => {
      this.dailyMenus = resp;
    },
      error => {
        this.notifications.showErrorMessage("Can't load the daily menus
        for this " + date + " date!");
      })
}
```

8. kódrészlet. A 7. kódrészletbeli metódust meghívja és feliratkozik rá, így el lesz küldve a HTTP kérés a szervernek. Hiba esetén az *error* ágba lép be és megjelenít egy hibaüzenetet a felhasználónak, ellenkező esetben a választ átadja a *dailyMenus* változónak.

dolgoz. Ha a felhasználó nem igényelheti azt a szolgátatást, akkor egy hibaüzenet lesz a számára megjelenítve.

Ennek érdekében minden HttpClient segítségével küldött kérést egy *interceptor* feldolgoz és a fejléchez hozzáadja a *token*-t, majd továbbítja a szervernek.

3.3. Biztonság

A webes felület két felhasználói szerepkört támogat: kliens és adminisztrátor. A különböző felhasználók különböző felületeken tudják elérni azokat a szolgáltatásokat, amelyekre jogosultak. Annak érdekében, hogy mindenki csak azokat a funkcionalitásokat érhesse el, amelyekhez

```
@Injectable()
export class CanActivateAdminAuthGuard implements CanActivate {

  constructor(private router: Router,
              private authService: AuthenticationService) {}

  canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot) {
        if (this.authService.isAdmin() && this.authService.isLoggedIn()) {
            return true;
        }
        this.router.navigate(['/']);
        return false;
    }
}
```

9. kódrészlet. A *CanActivateAdminAutGuard* implementálja a *CanActivate* interfészt. Rendelkezik egy konstruktorral, illetve egy *canActivate* metódussal, amelyben ellenőrzi, hogy az a felhasználó aki el szeretné érni azt a komponenst, amelynek útvonalát ez az osztály védi, adminisztrátor szerepkörrel rendelkezik-e és be van-e jelentkezve. Ha a feltételek nem teljesülnek, akkor a '/' útvonallal ellátott oldalra lesz átirányítva a felhasználó.

joga van, a különböző felületek elérési útvonalát le kell védeni (*route guards*), valamint egyes felületeket elérhetővé kell tenni a vendég felhasználók számára is.

Az útvonalvédő csak akkor engedi betölteni az elérni kívánt nézetet, ha bizonyos feltételek teljesülnek (pl. be van jelenkezve a felhasználó, adminisztrátor szerepköre van stb.), ellenkező esetben átirányítja a felhasználót a szerepkörének megfelelő alapértelmezett oldalra. Egy ilyen *route guard* osztálynak implementálnia kell a *CanActivate* interfészt, valamint a *canActivate* metódust, amelyben megtörténik a megfelelő feltételek ellenőrzése, és ha *true* a visszatérítési érték, akkor az útvonal elérhetővé válik.

A 9. kódrészletben definiált *CanActivateAdminAutGuard* osztály *canActivate* metódusa ellenőrzi, hogy a felhasználó be van-e jelentkezve, illetve, hogy adminisztrátor szerepkörrel rendelkezik-e. Ha ezek a feltételek teljesülnek, akkor elérhetővé teszi a kiválasztott útvonalat, ellenkező esetben a '/' útvonallal ellátott komponenst tölti be.

Az útvonalvédő osztályokat először hozzá kell rendelni azokhoz az útvonalakhoz, amelyeket védeniük kell. A 10. kódrészletben látható, hogy a 9. kódrészletben implementált *CanActivateAdminAutGuard* az 'addFood', valamint az 'addRestaurant' útvonallal elérhető komponensekhez van hozzárendelve. Egyedül a 'restaurants' útvonalhoz nincs semmi hozzárendelve, ezért a *ShowRestaurantsComponent* elérhető bármilyen szerepkörrel rendelkező felhasználónak, így ezt a vendég felhasználók is megtekinthetik.

```
const appRoutes: Routes = [

  { path: 'restaurants', component: ShowRestaurantsComponent },

  { path: 'addFood',
  component: AddFoodComponent, canActivate: [CanActivateAdminAuthGuard] },

  { path: 'addRestaurant',
  component: AddRestaurantComponent, canActivate: [CanActivateAdminAuthGuard] }

  ];
```

10. kódrészlet. A webes felület komponensei, illetve a hozzájuk rendelt elérési útvonalak láthatóak. A 9. kódrészletben implementált *CanActivateAdminAutGuard* osztály hozzá van rendelve két útvonalhoz is, így azokhoz tartozó komponensek csak bejelentkezett adminisztrátorok számára elérhetőek. Egyedül a *ShowRestaurantsComponent* komponenshez nincs társítva egyetlen útvonalvédő osztály sem, ezért ez elérhető bármilyen szerepkör számára.

4. Android kliens

A Netfood esetében az étekiszállítást elvégző futárok számára egy Android alkalmazást biztosít a rendszer a rendelések kezelésére.

Az Android [6] egy Linux alapú operációs rendszer, amely főként okostelefonokon vagy táblagépeken használatos, legújabb veziója a *8.1 Oreo*. Az Android platformot egy többrétegű architektúra jellemzi. A legalsó réteg egy Linux kernel, a második réteg a szolgáltatások és programkönyvtárak, ezek mind *C*-ben vagy *C++*-ban vannak implementálva. A következő réteg az Android futtatókörnyezet, az 5.0 verziót megelőzően a *Dalvik* virtuális gép volt az alapja, ezt a verziót követően az *Android Runtime* (ART), amelynek fontosabb jelemzői az *ahead-of-time* (AOT) és *just-in-time (JIT)* kompilálás, valamint a *garbage collector* optimalizálása. A legfelső réteget a rendszeralkalmazások (e-mail, SMS küldés, internetes böngészés stb.) alkotják, ez alatt található a Java API, amelynek segítségével megvalósulhat az új Android alkalmazások fejlesztése.

Android alkalmazások fejlesztéséhez szükséges az *Android Software Development Kit* (Android SDK). Az SDK tartalmaz olyan eszközöket, amelyek szükségesek a fejlesztési folyamat során, mint például: *debugger*, könyvtárak, emulátor, dokumentáció, példa kód, oktatóanyagok stb. Az elsődleges programozási nyelv a Java. A nézeteket XML állományok segítségével határozzuk meg. Felépíthetőek Java kódból is, viszont ez nem ajánlott a nézet és az üzleti logika keveredése miatt.

4.1. Gradle build-rendszer

A Netfood mobil kliens esetében a Gradle [7] *build* rendszer van használva. A Gradle egy *build* és függőségkezelő rendszer, a *build* folyamat leírása egy **build.gradle** szkript állományban történik, ennek megírására egy Groovy alapú DSL (Domain Specific Language) használható. A Groovy egy dinamikus nyelv, szkriptek írására használható Java platformon.

4.2. Retrofit

Az Android kliens és a szerver közötti kommunikáció REST (2.6 fejezet) kéréseken keresztül történik, Retrofit [8] keretrendszer segítségével. A Retrofit egy REST kliens, amely megkönnyíti a kérések felépítését, küldését, fogadását és feldolgozását, továbbá az objektumok szerializálását és deszerializálását is megoldja.

A Retrofit használatához három eszközre van szükség. Az első egy interfész (11. kódrészlet), amelyben a REST kéréseknek megfelelő metódusokat kell definiálni annotációk (*@GET, @POST, @PUT, @DELETE*) segítségével.

```
public interface BasketService {
    @GET("api/baskets/new/{date}")
    Call<List<Basket>> findNewBasketsByDate(@Path("date") String date,
    @Header("Authorization") String token);
}
```

11. kódrészlet. A mobil kliens a szervertől egy dátummal paraméterezett GET kérés segítségével kapja meg az új rendeléseket.

A második egy Retrofit osztály, amelynek segítségével létrehozható egy előzőleg definiált interfész (például a 11. kódrészletben levő BasketService interfész) egy implementációja. A projekt esetében a kódismétlés elkerülésének érdekében a *ServiceGenerator* osztály végzi ezt a feladatot. A *createService()* metódusának paraméterként megadva a megfelelő interfészt, létrehozza annak egy implementációját, ezáltal az interfészben deklarált metódusok hívhatóak lesznek.

A harmadik részt a POJO osztályok képviselik, ezek képezik az adatmodellt, amely hasonló a szerver adatmodelljéhez (2.3 fejezet).

```
public class ServiceGenerator {
    private static final String API_BASE_URL = "http://10.0.2.2:8080/";
    private static OkHttpClient.Builder httpClient = new OkHttpClient.Builder();
    private static Retrofit.Builder builder = new Retrofit.Builder()
                    .baseUrl(API_BASE_URL)
            .addConverterFactory(GsonConverterFactory.create());
    private static Retrofit retrofit = builder.build();
    private static HttpLoggingInterceptor logging =
            new HttpLoggingInterceptor()
                    .setLevel(HttpLoggingInterceptor.Level.BODY);

    public static <S> S createService(Class<S> serviceClass) {
        if (!httpClient.interceptors().contains(logging)) {
                httpClient.addInterceptor(logging);
                builder.client(httpClient.build());
                retrofit = builder.build();
        }
        return retrofit.create(serviceClass);
    }
}
```

12. kódrészlet. Ebben az osztályban vannak a megfelelő beállítások, és a *createService()* metódus visszatéríti a kívánt interfész implementációkat.

```
...
BasketService basketService = ServiceGenerator.
createService(BasketService.class);
Call<List<Basket>> call = basketService
.findNewBasketsByDate(new Date(),userDetails.getToken());
...
```

13. kódrészlet. A mobil kliens a *BasketService* interfészben deklarált *findNewBasketsByDate()* metódus meghívásával kapja meg az új rendelések listáját a szervertől.

4.3. Felhasználói felület

A felhasználói felület alapját az Android *Activity* osztályai képezik, amelyek *controller* szerepet töltenek be az alkalmazáson belül. Minden *Activity*-hez tartozik egy a nézetet leíró XML állomány, itt vannak meghatározva a megjelenő elemek és az elrendezésükre vonatkozó információk. Az elemekhez rendelhető egy azonosító, ez által lehet elérni őket az *Activity* oszályon belül, és így lehet befolyásolni a viselkedésüket. Egy *activity*-hez több *fragment* tartozhat, ezek általában egyetlen funkcionalitást tartalmaznak és életciklusuk függ az őket tartlamazó *activity* osztály életciklusától. A különböző *activity*-k közötti váltás egy *Intent* osztály segítségével valósul meg.

A projekt mobil kliensének esetében három fő *activity* van. Az első a bejelentkezési felület. Sikeres bejelentkezés után az alkalmazás átvált a második fő *activity*-re, itt lehet rendezni

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ...
        TextView forgotTextView = (TextView) findViewById(R.id.forgotTextView);
        forgotTextView.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent forgotPasswordIntent = new Intent(getApplicationContext(),
                                ForgotPasswordActivity.class);
                startActivity(forgotPasswordIntent);
            }
        });
    }
}
```

14. kódrészlet. A *TextView*-t a hozzá rendelt azonosító alapján lehet elérni. Egy *listener* rendelhető az elemhez és kattintás hatására az onClick() metódus végrehajtódik. Az *Intent* és a *startActivity()* metódus segítségével vált át egy másik *activity*-re.

a rendeléseket. Ezen a felületen több *fragment* is található, a különböző funkcionalitásoknak megfelelően. A harmadik *activity*-t elfelejtett jelszó esetén lehet használni, egy érvényes felhasználóhoz tartozó e-mail megadása után a *mail service* (2.8 fejezet) erre a címre küld egy linket, amelynek segítségével új jelszó adható meg.

4.4. Nemzetköziesítés

A mobil kliens felülete is több nyelven elérhető, hasonlóan a webes klienshez. Az *Android* platform támogatja az alkalmazás nemzetköziesítését, ennek köszönhetően könnyen többnyelvűsíthető a felület. A felhasználói felületen megjelenítendő szöveget, a *res/values/strings.xml* állományban (15. kódrészlet) kell megadni erőforrásként.

```
<string name="app_title">Netfood</string>
```

15. kódrészlet. Minden erőforrásnak van egy azonosítója, és tartozik hozzá egy a felhasználói felületen megjelenítendő szöveg.

Amikor meg kell jeleníteni egy szöveget, akkor az *activity*-hez tartozó XML állományban lehet hivatkozni arra, az azonosítója által, a 16. kódrészletben látható módon. A többnyelvűsítéshez a *res* könyvtáron belül a kívánt nyelvnek megfelelően létre kell hozni egy *values-***xy***/strings.xml* (**xy** a nyelvnek megfelelő ISO standard rövidítés [9]) könyvtárszerkezetet. Például, a magyar nyelvnek megfelelő szerkezet a következő: **values-hu/strings.xml**. Ebben az XML állományban meg kell adni az összes erőforrásnak az értékét, a kívánt nyelven, az azonosító alapján. Az alkalmazás a készülék alapértelmezett nyelvének megfelelő állományból veszi az erőforrások értékét. Ha itt nem találja meg az adott fordítást, akkor az alkalmazás által alapértelmezettnek tekintett *values/strings.xml* állományban keresi.

```
<TextView
        android:text="@string/app_title"
/>
```

16. kódrészlet. A címke szövegébe behelyettesíti a megfelelő azonosítóval rendelkező erőforrás értékét.

5. Eszközök és módszerek

A Netfood fejlesztése során a Scrum agilis szoftverfejlesztési módszert alkalmazta a csapat, amelynek megfelelően inkrementális és iteratív fejlesztés valósult meg, amely két hetes sprintekben zajlott. Minden iterációs szakasz tervezéssel kezdődött (sprint planning), amely során kiválasztásra kerültek a backlog-ból az adott futamban megvalósítandó feladatok. Minden sprint végén egy demo keretein belül voltak bemutatva az újonnan bekerült funkcionalitások. Ezután következett egy *retrospective* megbeszélés, amikor a csapattagok elmondhatták az előző futamról alkotott véleményüket, javaslataikat, illetve elvárásaikat a következő futamokkal kapcsolatban.

A feladatok számontartására, azok nyomon követésére, valamint annak nyilvántartására, hogy ki mivel foglalkozik, projektmenedzsment eszközként a GitLab szolgált. Továbbá a forráskód tárolásáról is ez a rendszer gondoskodott, illetve a folytonos integráció megvalósításában is segítséget nyújtott.

Verziókövetőként Git-et használt a csapat. Minden funkcionalitás fejlesztése külön ágon történt. Ha egy funkcionalitás elkészült és jóvá lett hagyva, akkor az ág, amelyen a fejlesztése történt lezárult, és a változtatások bekerültek egy stabilnak tekintett ágba.

A fejlesztés során több fejlesztői környezet volt használva: a szerver fejlesztése STS-ben (Spring Tool Suite), az Android kliensé Android Studio-ban, a webes felületé Visual Studio Code-ban történt.

A Maven build rendszer felelős a szerver build folyamatának lefuttatásáért és a függőségkezelésért. Az Android kliens estében mindezt a Gradle végzi el, a webes kliens fejlesztéséhez használt eszközök pedig az npm (Node Package Manager) csomagkezelő rendszerrel vannak felügyelve.

6. A Netfood működése

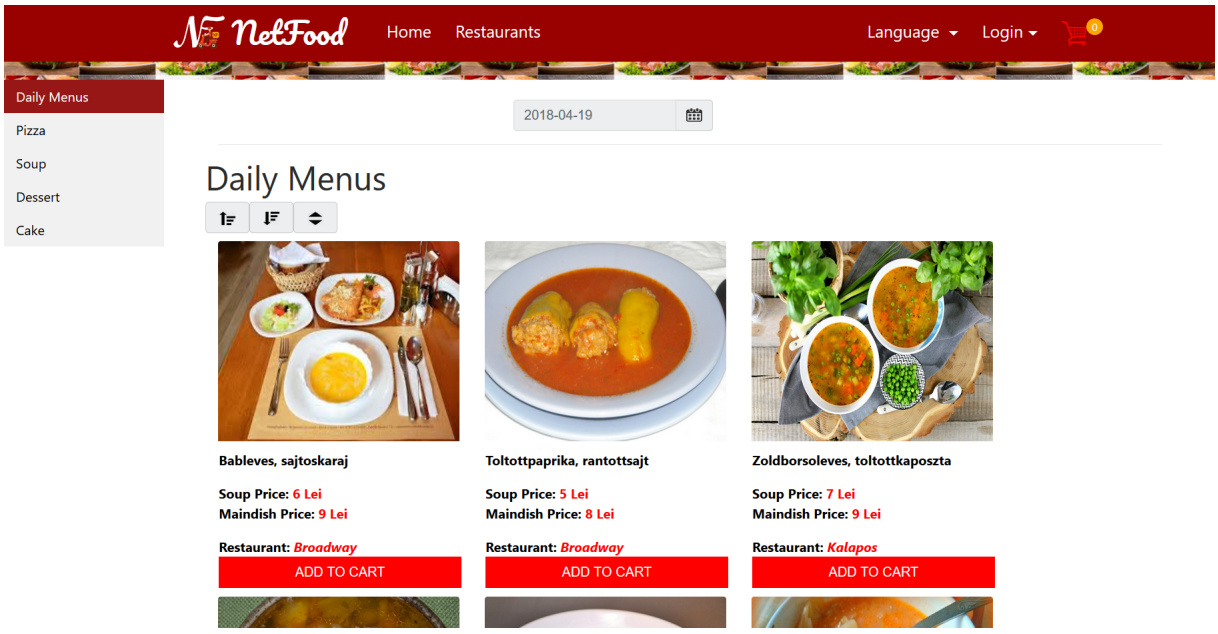
A fejezet a Netfood két kliensfelületének működését szemlélteti.

6.1. A webes felület használata

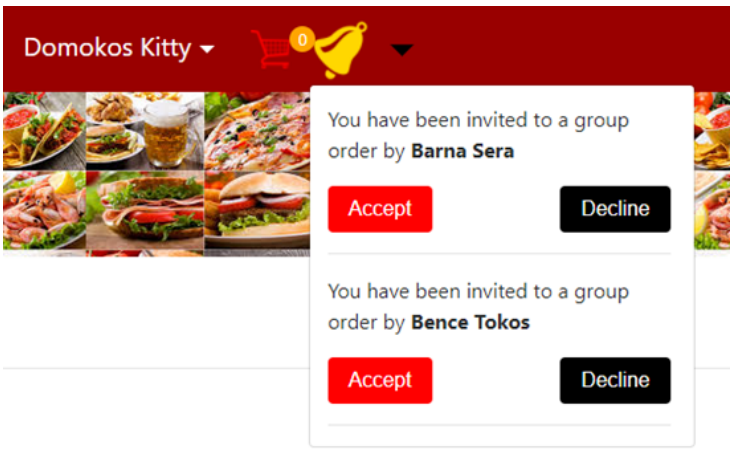
A webes felületen minden vendégfelhasználó megnézheti a megrendelhető ételeket (3. ábra), a kiszállító céggel kapcsolatban álló éttermeket. A felhasználó kiválaszthatja, hogy milyen nyelven szeretné használni az alkalmazást. Jelenleg három nyelv közül választhat: angol, magyar és román. A felhasználóknak lehetőségük van a Login menüpont kiválasztásával bejelentkezni, a rendszerben regisztrált felhasználóval vagy Facebook-on keresztül. Ez alatt a menüpont alatt kérhetnek jelszóemlékeztetőt is, illetve a regisztrációs oldalra is átnavigálhatnak.

Egy bejelentkezett felhasználó megnézheti, illetve szerkesztheti az adatait a profilja alatt, megtekintheti a már leadott rendeléseit, menedzselheti a barátait. A felhasználók között kereshet név, illetve felhasználónév alapján, jelöléseket küldhet nekik, illetve elfogadhatja a hozzá érkezett jelöléseket és kilistázhatja saját barátait. Rendelést is indíthat, kiválaszthatja a kívánt ételeket és azokat beteheti a kosarába, valamint csoportos rendelést is kezdeményezhet.

Csoportos rendelés (5. ábra) esetében egy oldalsó sávban megjelennek a felhasználó barátai és azok közül kiválaszthatja, hogy kiket szeretne meghívni. Minden személy, akit meghívtak egy csoportos rendelésbe, kap egy értesítést (4.ábra). A meghívót elfogadhatják vagy elutasíthatják. Elfogadás esetén résztvevői lesznek a rendelésnek. Ezt követően lehetőségük lesz kilépni, illetve további felhasználókat hozzáadni a rendeléshez, és jelezhetik a többieknek, ha végeztek a



3. ábra. Az ételek megtekinthetőek kategóriák szerint a főoldalon. A napi menük láthatóak, amelyek 2018.04.19-i dátumon voltak rendelhetőek.



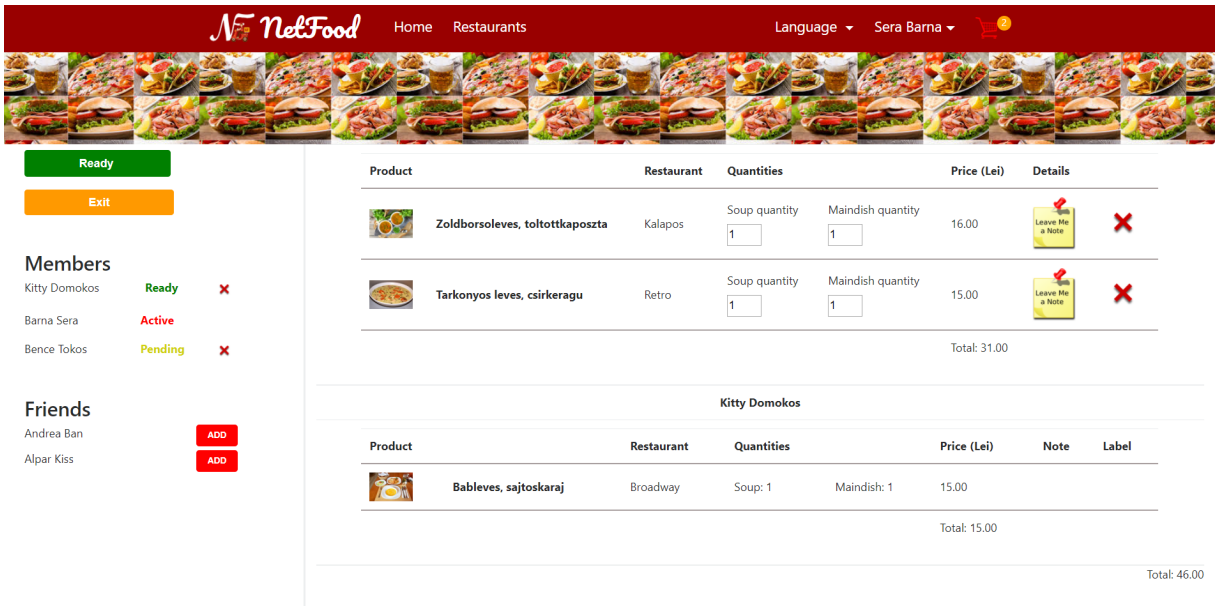
4. ábra. Két csoportos rendelésre kapott meghívót a felhasználó, amelyeket a csengő ikon megnyomásával tud megnézni.

rendeléssel.

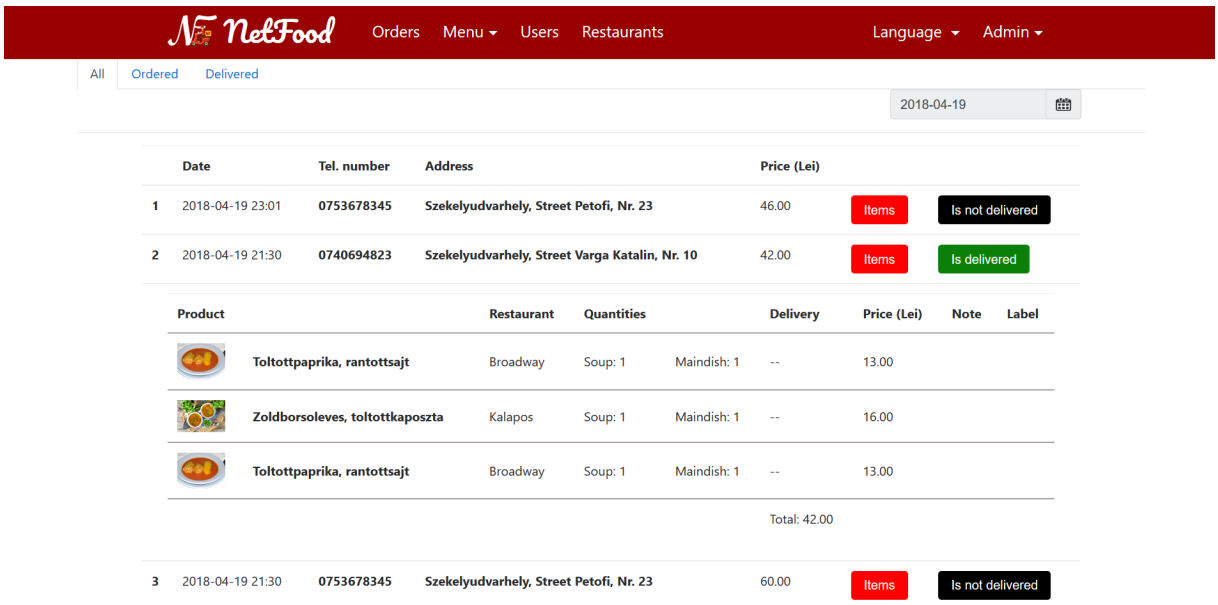
Ha adminisztrátor szerepkörrel rendelkező felhasználó lép be a rendszerbe, számára az ételek helyett a beérkezett rendelések lesznek láthatóak a főoldalon (6. ábra). Az éttermek menüpont alatt, új vendéglőket vihet fel, módosíthatja a már meglévőket vagy törölheti azokat.

Az ételek menüpont alatt a napi menüket, illetve a más kategóriájú ételeket menedzselheti. A napi menük opciót kiválasztva oldalt megjelennek azok az éttermek, amelyeknek a kiválasztott dátumon nincs még feltöltve egyetlen napi menü sem. Egy a nevek mellett megjelenő gomb megnyomásával hozzáadhatja a hiányzó menüt, amely a kiválasztott dátumon lesz rendelhető.

Mindezek mellett egy adminisztrátor megváltoztathatja a felhasználók szerepkörét is a felhasználók menüpont alatt. Egyszerű felhasználóból lehet futár, admininsztrátor vagy étteremtulajdonos, és fordítva.



5. ábra. A csoportos rendelés kosara így néz ki.

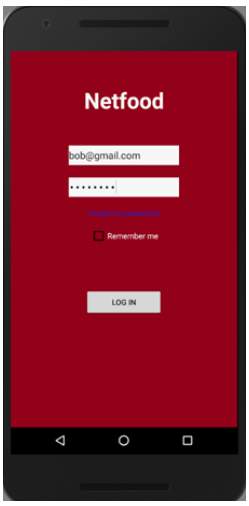


6. ábra. Az adminisztrátornak a főoldalon a rendelések jelennek meg.

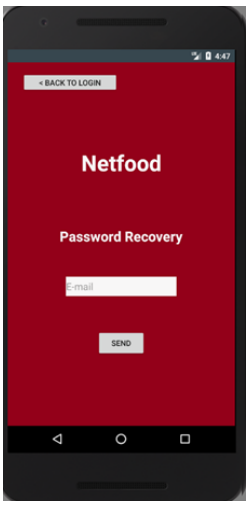
6.2. Az Android kliens használata

Az Android kliensalkalmazásba futár szerepkörrel rendelkező felhasználók tudnak bejelentkezni (7.a ábra). Ha valamelyik kiszállító elfelejtette a jelszavát, akkor igényelhet jelszóemlékeztetőt (7.b ábra).

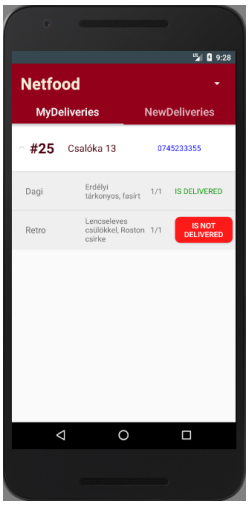
Sikeres bejelentkezés után megjelennek azok a rendelések, amelyek kiszállítását már elvállalta a futár, ezek az applikáció MyDeliveries nézetében tekinthetőek meg (7.c ábra). Ugyanitt



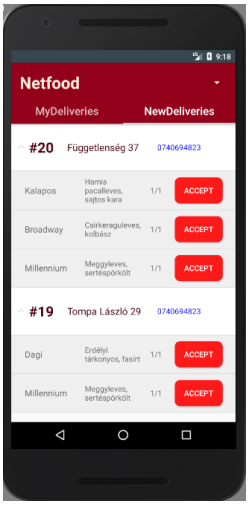
(a) Login felület



(b) Jelszóemlékeztető kérése lehetséges az applikáción belül.



(c) Bejelentkezés után a futár által már elvállalt rendelések jelennek meg, és ha végzett egy rendeléssel bejelölheti, hogy az ki lett szállítva.



(d) A bejelentkezett futár megnézheti az adott napra beérkezett rendeléseket. Minden rendelés lenyitható és azok teljes egészét vagy csak részeit el tudja fogadni.

7. ábra. Android alkalmazás

bejelölhető, ha az adott rendelés kiszállítása megtörtént. Minden rendelés esetében megjelenik a cím, ahova ki kell szállítani az ételeket, illetve a megrendelő telefonszáma, amelyet közvetlenül az alkalmazásból fel is tud hívni a futár. Azok a rendelések, amelyek még nincsenek elfogadva egyetlen futár által sem, az applikáció NewDeliveries nézetében tekinthetőek meg (7.d ábra). Ezeket a futár megnyithatja és kiválaszthatja, hogy mely részeket szeretné elfogadni, ezt követően az elfogadott rendelések megjelennek az elvállaltak között.

Következtetések és továbbfejlesztési lehetőségek

A Netfood projekt keretein belül sikerült egy olyan szoftverrendszert létrehozni, amely segítheti az ételkiszállító cégek munkáját. A webes felületen keresztül a kliensek egyéni, illetve csoportos rendeléseket létrehozva megrendelhetik az általuk kiválasztott ételeket. Az adminisztrátorok menedzselhetik az ételeket, éttermeket, felhasználókat, valamint a rendeléseket is követni tudják.

Az Android alkalmazás segíti a futárok munkáját: azonnal értesülnek a rendelésekről, el tudják vállalni azokat, valamint a kiszállításhoz szükséges információkat is megkapják.

A fejlesztési folyamat közben új funkcionalitások is felmerültek, mint továbbfejlesztési lehetőségek:

- felület az étteremtulajdonosok számára, hogy mindenki maga menedzselhesse a saját étterm ének adatait, ajánlatait;
- lehessen kiválasztani a már felvitt ételek közül, hogy az adott dátumra mi kerül ki napi menü ajánlatként;
- az adminisztrátor tudjon SMS-t küldeni a megrendelőnek egy rendelés beérkezését követően arról, hogy mennyi időt kell várakozzon az ételek megérkezéséig;
- a mobilalkalmazás iOS vátozata;
- a mobilalkalmazás Google Maps segítségével mutassa ki a megrendelő címét, valamint navigálja el a futárt jelenlegi helyzetétől az adott címig;
- különálló, telepíthető mobilalkalmazás a felhasználók számára a rendelések leadásához.

Hivatkozások

[1] Rod Johnson, Juergen Hoeller, Alef Arendsen , Colin Sampaleanu, Rob Harrop, Thomas Risberg, Darren Davison, Dmitriy Kopylenko, Mark Pollack, Thierry Templier, Erwin Vervaet, Portia Tung, Ben Hale, Adrian Colyer, John Lewis, Costin Leau, Rick Evans. *The Spring Framework - Reference Documentation*. URL: http://docs.spring.io/spring-framework/docs/2.0.x/reference/index.html (utolsó elérés dátuma: 2018. márc. 20.)

[2] Phillip Webb, Dave Syer, Josh Long, Stéphane Nicoll, Rob Winch, Andy Wilkinson, Marcel Overdijk, Christian Dupuis, Sébastien Deleuze, Michael Simons, Vedran Pavić, Jay Bryant, Madhura Bhave. *Spring Boot Reference Guide*. 2012-2018.

[3] Oswald Campesato. *ANGULAR 4 Pocket Primer*. 2017.

[4] *Boostrap*. URL: http://getbootstrap.com/ (utolsó elérés dátuma: 2018. ápr. 02.)

[5] *HttpClient*. URL: https://angular.io/guide/http (utolsó elérés dátuma: 2018. ápr. 06.)

[6] *Android hivatalos weboldal*. URL: https://developer.android.com/index.html (utolsó elérés dátuma: 2018. ápr. 13.)

[7] *Gradle hivatalos weboldal*. URL: https://gradle.org (utolsó elérés dátuma: 2018. ápr. 07.)

[8] *Retrofit - Square Open Source*. URL: http://square.github.io/retrofit/ (utolsó elérés dátuma: 2018. ápr. 07.)

[9] *ISO nyelvkódok*. URL: http://www.loc.gov/standards/iso639-2/php/code_list.php (utolsó elérés dátuma: 2018. ápr. 14.)