ETDK-dolgozat

Balázs István-Lehel

Györfi Endre

XXV. reál- és humántudományi Erdélyi Tudományos Diákköri Konferencia (ETDK)

Kolozsvár, 2022. május 19–22.

APIA APP

Támogatásokat kezelő mobilalkalmazás gazdák számára

Szerzők:

Balázs István-Lehel

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar, Informatika szak, III. év

Györfi Endre

Sapientia Erdélyi Magyar Tudományegyetem, Villamosmérnöki Kar, Számítástechnika szak, IV. év

Témavezetők:

dr. Sulyok Csaba, egyetemi adjunktus

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar

dr. Iclănzan David Andrei, docens

Sapientia Erdélyi Magyar Tudományegyetem, Matematika és Informatika Kar

Brassai Beáta, szoftverfejlesztő,

Codespring

Bíró Gellért, szoftverfejlesztő,

Codespring

CODESPRING

OUR CODE YOUR SUCCESS

Kivonat

Románia nagy összegeket kap az Európai Uniótól agrár-környezetvédelem illetve mezőgazdasági termelésben résztvevő személyek támogatására. A Milvus egyesület szerint a hazai gazdáknak gondot jelent a támogatások lehívása, és amennyiben megnyerik a pályázatot, a feltételek betartása.

A jelen dolgozat bemutat egy alkalmazást, melynek célja, hogy segítséget nyújtson a romániai gazdáknak a földterületeikhez tartozó támogatások kiválasztásában és kezelésében.

Az alkalmazásban a gazdáknak lehetőségük nyílik a kiválasztott településben elérhető támogatások böngészésére, ezek feltételeinek megismerésére. Ezen kívül segítséget nyújt a már megnyert támogatások feltételeinek követésében és betartásában az által, hogy értesítések formájában emlékezteti a gazdát.

A dolgozat bemutatja az alkalmazás működését, megvalósításához használt technológiákat, kitérve az architektúrára és a funkcionalitások bemutatására.

Tartalomjegyzék

Bevezető

Napjainkban a román kormány nagy összeget kap az Európai Uniótól a gazdák támogatására, akik pályázni tudnak a saját földterületeikkel vagy állataikkal [12]. Nagy problémát jelent számukra a támogatásokkal kapcsolatos információk beszerzése. Megoldásként az APIA APP összegyűjti a támogatásokról szóló lényeges információkat, csomagokba szervezve ezeket, sokat segítve a gazdáknak.

Manapság az információ beszerzés egy elterjedtebb formája a mobiltelefon, amely mindig az embernél van, használatra kész, ezért egy hatékony módja a kiírt támogatások böngészésére és kezelésére egy Android/iOS alkalmazás.

Az alkalmazás az APIA APP projekt név alatt fut, ez magába zárja az Android és egy leendő iOS mobil alkalmazást.

Az alkalmazást vendég felhasználóként is lehet használni, ebben az esetben nem minden funkcionalitás elérhető, de a támogatások böngészhetők. Bejelentkezve a felhasználó kiválaszthatja a támogatásokat és kezelheti ezeket.

Új támogatást az alkalmazáson belül nem lehet létrehozni, új támogatást csak az adminisztrációs felületen lehet létrehozni.

Az alkalmazás fejlesztése 2021 júniusában kezdődött, a nyári gyakorlat ideje alatt a Codespring Mentorprogram[1] részeként. A Babeș–Bolyai Tudományegyetem csoportos projekt keretén belül a fejlesztésben csatlakozott egy félév erejéig Lázár Dávid, Demian Ervin, Kemény Rudolf és Tőkés Tímea.

A dolgozat bemutatja az alkalmazás funkcionalitásait, a fejlesztési folyamatot, a felhasznált technológiákat, a rendszer architektúráját, s külön kitér a szerveroldalon használt technológiákra.

[1] https://edu.codespring.ro/apia-app/

1. Funkcionalitások

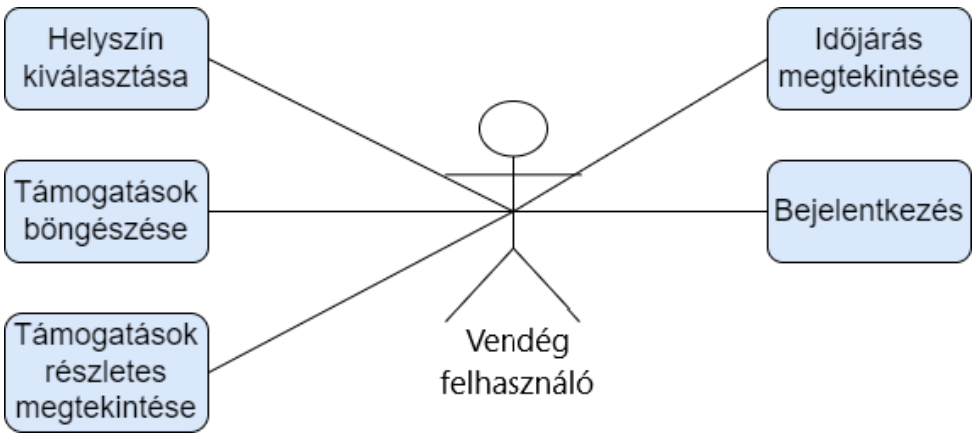
Ebben a fejezetben bemutatásra kerülnek az alkalmazás funkcionalitásai a különböző felhasználó típusok szemszögéből. Az alkalmazás használatához nem szükséges felhasználói fiókot készíteni viszont korlátozva lesznek elérhetőek a funkcionalitások a vendég felhasználó számára.

1.1. A vendég felhasználó funkcionalitásai

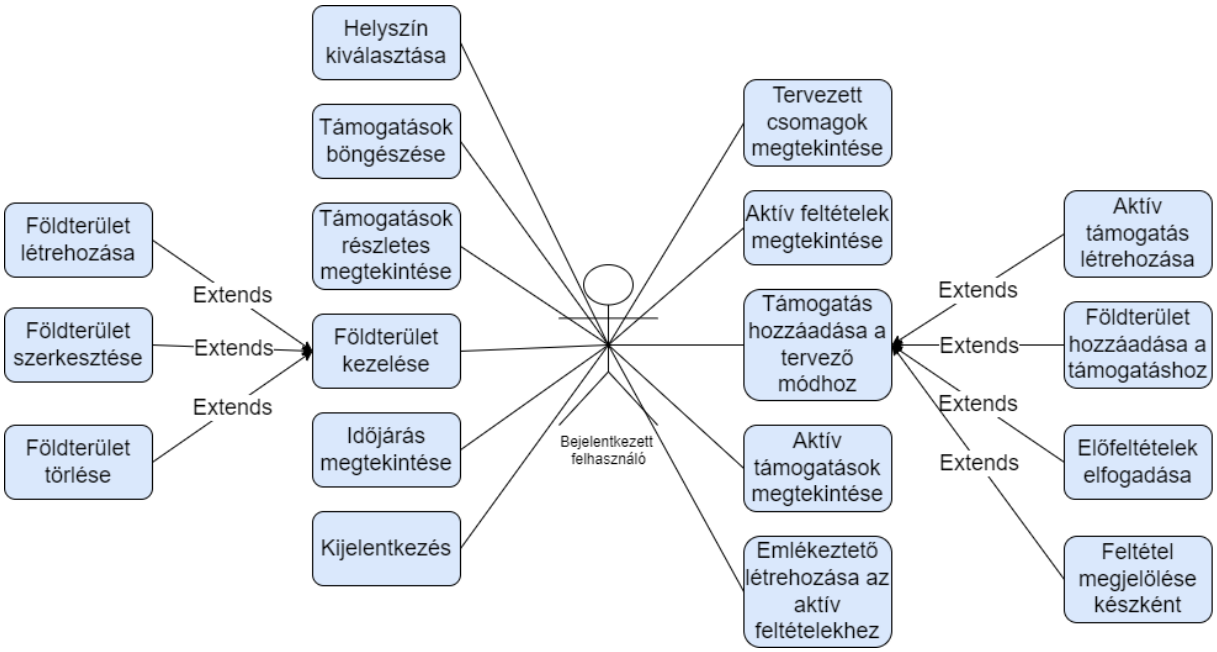
A vendég felhasználó számára a 1. ábrán bemutatott funkcionalitások érhetőek el. Az alkalmazás első megnyitása után láthatóak az országban található megyék, amelyek közül a felhasználó kiválaszthat egyet vagy többet annak függvényében, hogy melyik megyékben szeretné böngészni a kiírt támogatásokat. A kiválasztott megyék után listázásra kerül az ezekben található községek, a felhasználónak majd ezek közül is választania kell. Erre azért van szükség, mert a támogatások listája függ az egyes településektől, mivel a támogatások megye illetve község szinten vannak kiírva [11]. A helység kiválasztása után listázásra kerülnek a támogatások, ezek közt lehetőség van a keresésre. A vendég felhasználónak lehetősége van a pályázatok részleteinek a megtekintésére. Lehetősége van a felhasználónak regisztrálnia Google vagy Facebook fiók segítségével.

1.2. Bejelentkezett felhasználó funkcionalitásai

A bejelentkezett felhasználó számára a 2. ábrán látható funkcionalitások érhetőek el. Ezek közül a legfontosabb a támogatások kezelése, amely magába foglal egy tervező módot és a



1. ábra. Vendég felhasználó use case diagram



2. ábra. Bejelentkezett felhasználó use case diagram

már megpályázott támogatások követését. A tervező mód segítségével például a felhasználó könnyebben eloszthatja a földterületeit a megpályázni kívánt támogatások között. Úgyan ezen a tervező felületen az alkalmazás lehetőséget ad a támogatások előfeltételeinek böngészésére. A Romániában meghirdetett támogatások esetén vannak olyan feltételek, amelyek nem a megpályázás idején kell teljesüljenek hanem az év különböző időpontjához kötöttek. Ezeket az úgynevezett utófeltételek követheti a felhasználó az aktív csomagok felületén. A bejelentkezett felhasználó funkcionalitásai részletes bemutatásra kerülnek a 5.1. fejezetben.

Az alkalmazás több nyelven elérhető: magyar, román, angol. A nyelv kiválasztása automatikusan történik, alkalmazkodva az eszközön kiválasztott nyelvhez. Azokon az Android telefonokon, amelyek támogatják a sötét-világos témát (Android 10-től felfele) az alkalmazás az operációs rendszernek megfelelő témát választja ki.

2. Architektúra

Fontos, hogy az alkalmazás könnyen karbantartható, továbbfejleszthető és robosztus legyen. Egy jól szervezett kódbázissal könnyebb dolgozni, könnyebb tesztelni és természetesen könnyebb dokumentálni. Az alkalmazás fejlesztése során MVP (Model-View-Presenter) architektúra került alkalmazásra, amely modularitást, tesztelhetőséget, valamint tisztább és karbantarthatóbb kódbázist biztosít.

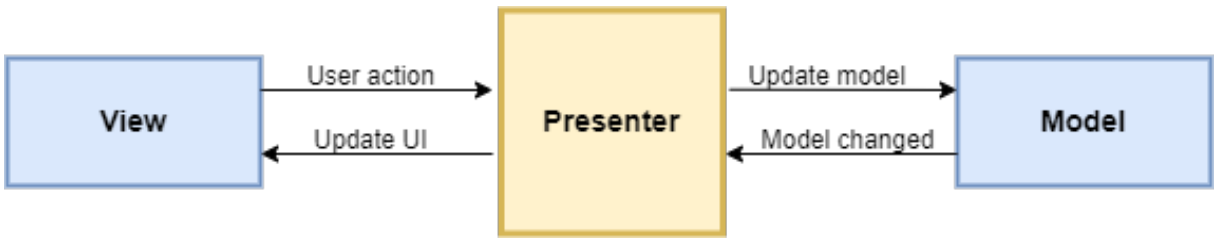
2.1. A Model-View-Presenter architektúra

Az MVP architektúra [10] lényege, hogy elválasztja az alkalmazásban alkalmazott logikát a felhasználói felülettől, ahogyan a 3. ábra is mutatja. A kommunikáció két interfészen keresztül valósul meg. Külön interfész van ha a UI szeretne kommunikálni a logikát implementáló osztályokkal, és külön interfész ha a model osztályok szeretnének kommunikálni a UI-al. Három réteget különböztetünk meg [14], a model réteg az adat tárolásért felelős, a logika megvalósításáért az adatbázis illetve hálózat eléréséért. A view egyetlen szerepe, hogy metódust hívjon meg a presenterből egy felhasználói felületen végrehajtott művelet esetén általában egy activity vagy fragment által van implementálva. A presenter a közvetítő szerepet tölti be a view és modell osztályok között.

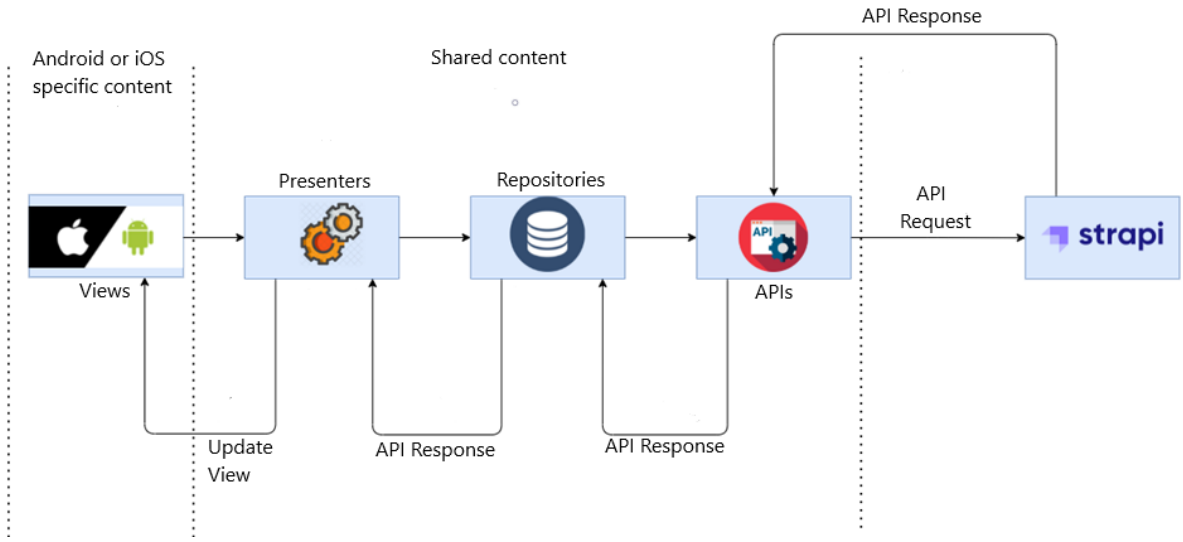
2.2. Adatelérési réteg

Az alkalmazás REST kérések segítségével kommunikál a szerverrel, a kérések az adatelérési rétegben vannak implementálva. A REST API hívásokat a Ktor [9] keretrendszer valósítja meg. A Ktor biztosít egy HTTP klienst, amellyel az alkalmazás kéréseket indíthat és válaszokat fogadhat aszinkron módon. A kommunikáció JSON formátumban valósul meg, a szerializációról a Ktor Serialization Plugin gondoskodik [5].

A 4. ábrán látható az APIA APP Android alkalmazásban a Model-View-Presenter



3. ábra. Általános MVP architektúra



4. ábra. APIA APP alkalmazásban használt MVP architektúra

architektúra alkalmazása. A legfelsőbb szinten a felhasználói felület van, amely figyeli a felhasználó cselekedeteit. A felhasználói felületen történt változásokra reagálva meghívja a presenter függvényeit, amely hatására a model osztályok a Repositories függvényein keresztül megtörténik az APIs osztályokban deklarált HTTP kérések meghívása. Az adatokat Kotlin specifikus úgynevezett data osztályok tárolják [8] amelyekhez a fordító automatikusan kigenerál néhány függvényt. Például: **equals(), hashCode(), toString(), componentN()** és **copy()**. Az adat osztályok csak állapotot tárolnak, nem végeznek műveletet.

3. Felhasznált technológiák

Fejlesztés során a csapat hangsúlyt fektetett arra, hogy egy modern és jó teljesítményű alkalmazást hozzon létre, így ebben a fejezetben részletes bemutatásra kerülnek az általuk használt technológiák.

3.1. Kotlin Multiplatform Mobile

A Kotlin Multiplatform Mobile [16] (KMM) egy szoftverfejlesztő készlet (SDK), mely Android és iOS alkalmazások fejlesztésére szolgál. Megkönnyíti azon csapatok munkáját, akik mindkét platformra szeretnék elkészíteni az alkalmazásukat, mivel a felhasználói felület kivételével egy nyelvben (Kotlin vagy Java) meg tudják írni az alkalmazás üzleti logikáját.

Az APIA app Android alkalmazáson belül a KMM konvenciót követve az osztályok három különböző csomagba vannak csoportosítva: androidApp, iosApp és shared. Az első két csomag (androidApp, iosApp) tartalmazza a nézetekkel kapcsolatos kódrészleteket, mindpéldául: gombok elhelyezése, képek beszúrása, navigáció felépítése képernyők között. A shared csomag magába foglal minden kódrészletet, ami közös mindkét platformon. Tehát itt vannak megírva a különféle algoritmusok, API lekérdezések, adatok tárolásával kapcsolatos függvények és komplex számítások. Erre a projektstruktúrára megfelelő az MVP (Model View Presenter) [[10]] tervezési minta használata, ami a 2.1 fejezetben részletesebb bemutatásra került.

A KMM egyik legfontosabb előnye, hogy a közös kódbázis Androidon Java bájtkóddá, míg iOS-en natív bináris kóddá kompilálódik, így teljesítmény szempontjából egy natívan megírt alkalmazásnak felel meg. Ha a fejlesztés során mégis szükség van platform specifikus implementációra, lehetőség van az expect/actual kulcsszavak használatára[5], azoknál a változóknál, függvényeknél vagy osztályoknál ahol platform specifikus implementációra van szükség.

Sok multiplatform könyvtárral rendelkezik, melyet lehet közösen használni, mint például a Ktor, mellyel hálózati kéréseket tudunk küldeni különböző API-k felé.

3.2. Kotlin vs. Java

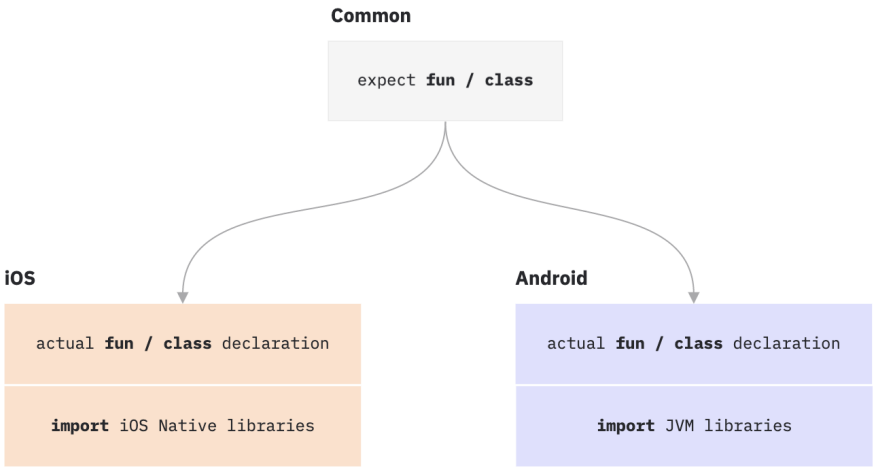
Egy Android alkalmazás tervezésekor a két nyelv közül tudnak választani: Kotlin vagy Java. Mindkettő rendelkezik úgy előnyökkel, mint hátrányokkal ezért érdemes mérlegelni a választásban. Mindkét nyelv interop, tehát létező Java kód meghívható Kotlin-ból, és Kotlin

kód használható Java-ból is. Az APIA App Android alkalmazás fejlesztése során Kotlin nyelv volt használva.

A Kotlin egy programozási nyelv, mely a Jetbrains fejlesztése és 2010-ben jelent meg. [13] Könnyen használható és megtanulható a Java fejlesztők számára, mivel a szintaxisa lényegesen hasonlít a Java-ra, sőt a két nyelv ugyan abban az osztályban is használható. Főbb jellemzői közé tartozik az, hogy null-bíztonságos, ami megelőzi a Java-ból jól ismert NullPointerException-t [15]. Más fő jellemzője az, hogy automatikusan létrehozza getter és setter metódusokat egy osztályon belül, ezzel időt spórolva a fejlesztőknek. A Kotlin nyelv fejlesztői nagy hangsúlyt fektettek a minél átláthatóbb és könnyebben megérthető kód írásának lehetőségére.

A Java egy programozási nyelv, mely a Sun Microsystems által lett létrehozva 1995-ben [13]. Az Android SDK nagy része még mindig Java-ban van megírva, de attól még jól lehet Kotlin mobil alkalmazásokat fejleszteni.

Összehasonlítván a két nyelvet, a Kotlin újabb, könnyebben megtanulható és használható, javasoltabb használni cross-platform rendszerek esetén, viszont kisebb a népszerűsége jelenleg. Ellenben ezzel, a Java-nak több könyvtára és keretrendszere van, amik közül a fejlesztők tudnak választani, de ettől eltekintve, a Google is Android fejlesztésre a Kotlin-t tartja fő nyelvként, mivel segít a fejlesztőknek produktívabbaknak lenni, hibákat megelőzni és aszinkron kód írását is ugyanolyan könnyűvé teszi, mint a szinkronét. [1]



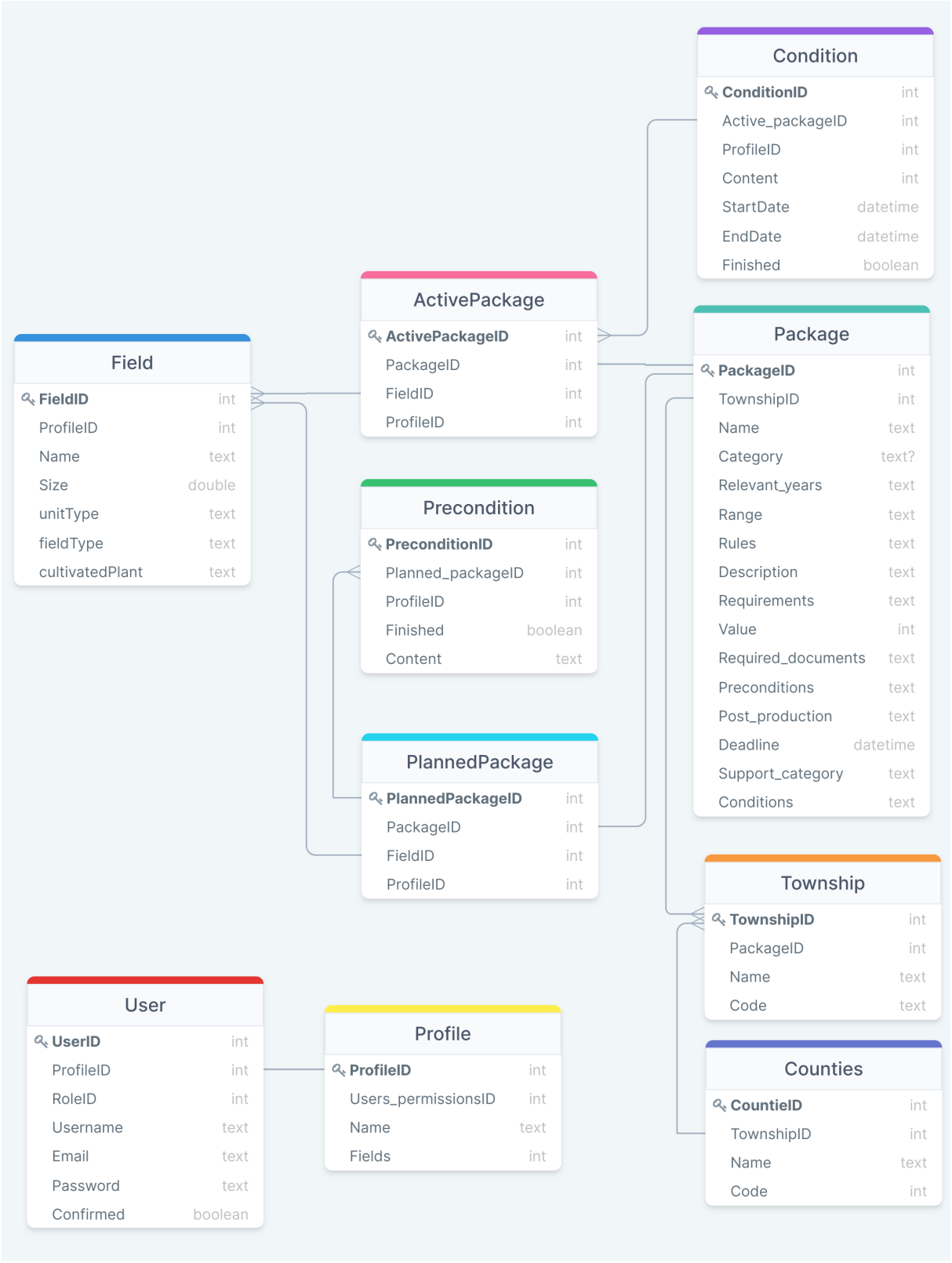
5. ábra. Expect/actual minta [4]

3.3. Strapi tartalomkezelő rendszer

A Strapi egy nyílt forráskódú, headless tartalomkezelő rendszer, [17] amely lehetővé teszi a fejlesztők és adminisztrátorok számára, hogy az alkalmazás adminisztrációs felületén keresztül programozási tudás nélkül tudják módosítani az adatokat. A Strapi adminisztrációs panelje és API-ja bővíthető, hogy minden használati eshetőségnek megfeleljen. Mindamellett a strapi rendelkezik egy felhasználói rendszerrel ahol részletesen kezelni lehet, hogy mihez férhetnek hozzá a felhasználók. Feltevődik a kérdés, hogy miért pont a Strapit választotta a fejlesztői csapat az adatok kezelésére. Egy lehetséges alternatíva lett volna nulláról megírni a backend részt amely nem lett volna egy jó megoldás ha nem készítünk egy adminisztrációs felületet ahol képesek az adminisztrátorok adatokat bevinni és kezelni programozási tudás nélkül. A Strapi, fej nélküli CMS elsődleges feladata a tartalomkezelésre összpontosít. Ezt úgy teszi, hogy az adatokat egy adatbázisban tárolja, biztosít egy felhasználói felületet az eltárolt adatok kezelésére, és API-val teszi az adatokat közzé, hogy bármely frontend számára elérhető legyen. Alapértelmezetten a strapi SQLite-ot használ az adatok tárolására de lehetőség van más adatbázis használatára mint például MongoDB, MySQL, MariaDB, PostgreSQL. A Strapi Relations funkció egyszerűvé teszi több kapcsolat létrehozását az adminisztrációs panel használatával amely lehetővé teszi a bejegyzések összekapcsolásához szükséges relációk létrehozását amely lehet egyirányú, egy az egyhez, egy a többhöz, sok a sokhoz vagy a polimorf reláció.

A 6. ábra bemutatja az alkalmazásban használt entitások kapcsolatát. Mint látható a rendszer központi eleme a package tábla amelyhez több község tartozik mivel a támogatások község szinten kerülnek kiírásra. A fenti részben már említésre került, a bejelentkezett felhasználó létrehozhat aktív és tervezett támogatásokat melyek a PlannedPackage és AktivePackage táblák segítségével vannak ábrázolva. Ezen táblák egy az egyhez kapcsolatban állnak a Package táblával és egy ProfileID-val kapcsolódnak a megfelelő felhasználóhoz. Minden Profile-hoz tartozik egy User, tartoznak opcionálisan hozzá adott földterületek és természetesen minden profile-hoz tartozik egy UserPermissions tábla amely magába foglalja egy adott felhasználóhoz tartozó jogokat.

A Strapi lehetőséget ad testreszabott vezérlők használatára, így az alkalmazás frontendjéhez a lehető legideálisabb válaszokat tudjuk visszatéríteni egy API hívás során. A Strapi egy createCoreController beépített funkciót biztosít amely automatikusan generálja a központi vezérlőket és lehetővé teszi egyedi vezérlők létrehozását, illetve a generált vezérlők



6. ábra. Adatbázis diagram

kiterjesztését vagy cseréjét. A vezérlők olyan JavaScript fájlok amelyek több metódust tartalmaznak és egy útvonalhoz vannak csatolva [6]. Minden tartalomtípushoz automatikusan

```
  async findOne(ctx) {
    const user = ctx.state.user;

    if (!user) {
          return ctx.badRequest(null,
          [{
              messages: [{
                  id: 'No authorization header was found.'
              }]
          }]);
    }

    const data = await strapi.services['field'].find({
        profile: user.profile
    });
    if (!data) {
            return ctx.notFound();
    }

    const { id } = ctx.params;
    const newData = data.filter((item) => item.id === parseInt(id))[0];
    if (!newData) {
            return ctx.notFound();
    }
}
```

1. kódrészlet. findOne metódus

létrejönnek alapértelmezett vezérlők és műveletek. Az alapértelmezett vezérlőket arra használják, hogy válaszokat adjanak vissza API kérésekre. Például egy GET kérést elküldve a http://localhost:3001/packages/2 útvonalra, a package tartalomtípus alapértelmezett vezérlőjének a findOne metódusa kerül meghívásra.

Az 1. kódrészletben egy saját személyre szabott vezérlő findOne metódusa van bemutatva, amely a bejelentkezett felhasználók számára id szerint visszatérít egy field (földterület) típust. Természetesen szükséges megírni a további find, create, delete metódusokat is ha szeretnénk egy testreszabott választ kapni egy API hívás esetén.

4. Aszinkron programozás

Az alkalmazás használata során előfordul az, hogy olyan kódrészletek kell fussanak, melyek több időt igényelnek, mint például kommunikáció a szerverrel, különböző adatok betöltése. Ha a fejlesztők mindezeket szinkron módon implementálják, akkor a felhasználói felület akadályozásához vezethet. Ilyenkor ajánlatos használni aszinkron programozást, mely lehetőséget ad optimálisabb kódok fejlesztésére. Ez többféleképpen megoldható, mint például: thread-ekkel és promise-okkal.[2]

A Kotlin-ban az aszinkron programozást úgynevezett *coroutine*-ok által lehet megvalósítani, melyek szünetelhető kódrészlet futását jelenti. Ezek hasonlóak a thread-ekhez, tehát egy kódrészletet hajtanak végre konkurrens módon, viszont a különbség köztük az, hogy a coroutin nincs egy pontosan kiválasztott szálhoz kapcsolva. Ki tudja választani, hogy melyik szálon szeretné folytatni a futását és *suspendable*. Ez azt jelenti, hogy megáll a futása amíg várakozik, de ez nem blokkolja a thread-et amiben éppen fut. [7]

A projektben a coroutine-ok segítségével jön létre a kommunikáció különböző API szerverekkel. Az alábbiakban láthatunk egy aszinkron függvényt, amely az adatelérési rétegbe hív bele, majd válasz érkezésekor elküldi az eredményt a megjelenítési rétegnek.

```
override fun loadPlannedPackagesList() {
        job = CoroutineScope(Dispatchers.Main).launch {
            runCatching(
                block = {
                    packagePlannerView?.setUpLoadingScreen()
                    plannedPackagesRepository.getPlannedPackages()
                        .toMutableList()
                },
                success = {
                    plannedList = it
                    updatePlannedPackagesList()
                },
                error = {
                    packagePlannerView?.setUpErrorScreen(it)
                }
            )
        }
    }
```

2. kódrészlet. Példa API meghivására aszinkron módon presenter osztályból

4.1. Aszinkron kódrészlet hiba kezelése

A [2] kódrészletet tekintve, észre lehet venni, hogy használva van egy úgynevezett *runCatching* szó, mely a *block*-ban leírt kód helyességét figyeli. Így, ha kliens- vagy szerveroldali hiba merül fel a kommunikáció során, az *error*-ban tartalmazó kód lesz végrehajtva, ellenkező esetben pedig a *success*-ben levő.

```
inline fun <R> runCatching(
    block: () -> R,
    success: (R) -> Unit,
    error: (ApiErrors) -> Unit
    ) {
    return try {
        success(block())
    } catch (ignore: ConnectTimeoutException) {
        error(ApiErrors.CONNECT_ERROR)
    } catch (ignore: RedirectResponseException) {
        error(ApiErrors.REDIRECT_ERROR)
    } catch (ignore: ClientRequestException) {
        error(ApiErrors.CLIENT_ERROR)
    } catch (ignore: ServerResponseException) {
        error(ApiErrors.SERVER_ERROR)
    }
}
```

3. kódrészlet. Aszinkron kód hiba kezelésére alkalmas kód

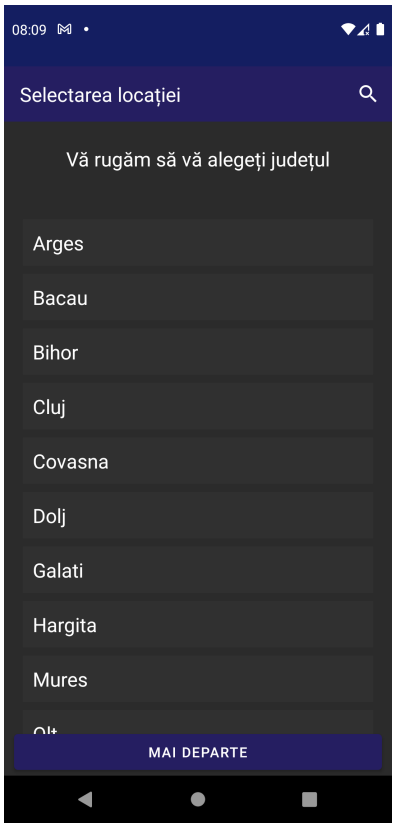
Ezt a kódolási struktúmarát a 3. kódrészlet teszi lehetővé, mely függvény fejlécében definiáltuk, hogy három closure function-t várunk el paraméterként: block, success és error. Ha sikeres a kódunk, akkor végrehajtódik a success kódblokk, ellenben, le van kezelve mindegyik hiba típus az error kódblokkban.

5. Az alkalmazás használata

Ebben a fejezetben bemutatásra kerül az Android alkalmazás használata illetve az adminisztrációs panel kezelése.

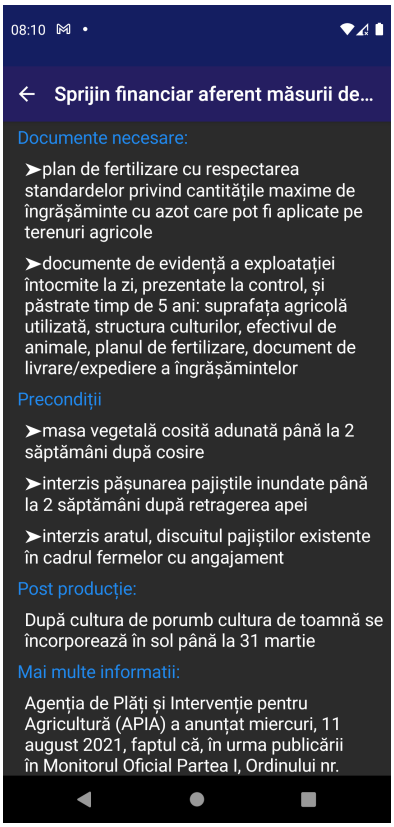
5.1. Android alkalmazás használata

A pályázatokat román nyelven írják ki, hivatalosan nincsenek lefordítva ezért az alkalmazásban románul jelennek meg a támogatások függetlenül attól, hogy az alkalmazás más nyelven fut. Az alkalmazás első megnyitása során megjelennek a megyék (7. ábra) amelyek közül egyet vagy többet ki lehet választani, lehetőség van a keresésre, a kereső ablakba minden betű lenyomása során frissül a megjelenített lista tartalmazva a keresni kívánt szót, szótöredéket. Kiválasztva a megyét a következő gombra lépve megjelennek a kiválasztott megyékhez tartozó községek listája, itt szintén lehetőség van egy vagy több elem kiválasztására. A kiválaszt gombra kattintva megjelennek a kiválasztott községekhez tartozó támogatások listája mint ahogy a 8. ábra mutatja. Ezen az oldalon böngészni lehet a támogatásokat, biztosítva van a keresési opció. Minden támogatásról a listaelemben a legfontosabb jellemzők vannak megjelenítve, támogatás neve, a támogatás potenciális értéke, a támogatás típusa, egy rövid leírás és a támogatás típusa. Egy listaelemre kattintva egy részletesebb nézet tárul elénk (9. ábra) ahol minden további információ részletesen meg van jelenítve.



7. ábra. Megye és község kiválasztása

8. ábra. Támogatások böngészése

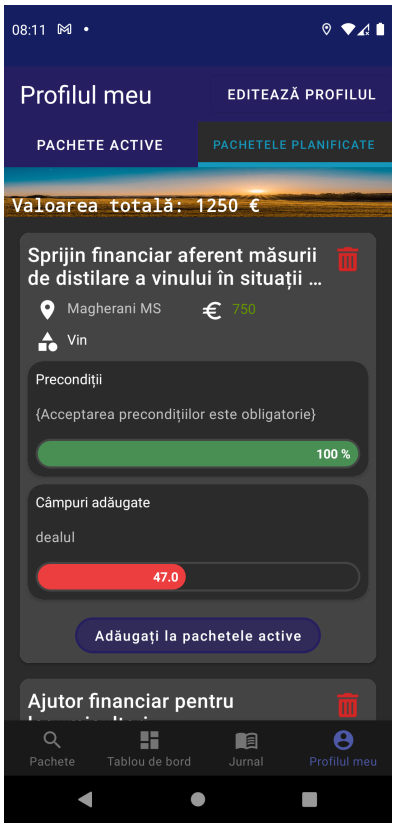


9. ábra. Támogatások részletes nézete

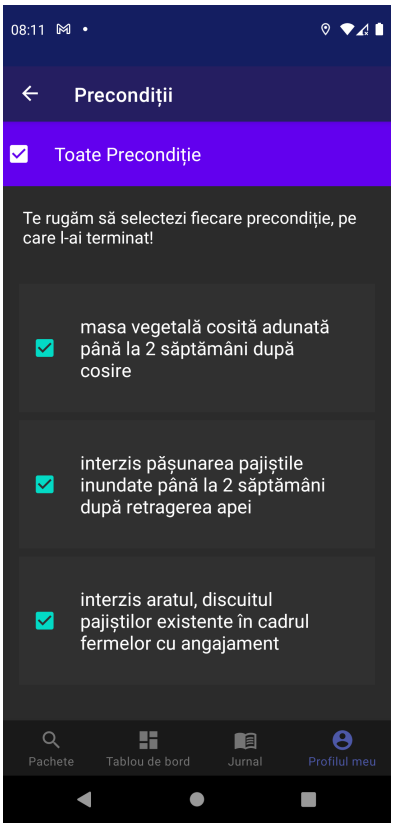
A további funkcionalitások elérése érdekében szükséges regisztrálni és bejelentkezni, amelyet a Saját profil oldal alatt lehet megtenni, regisztrációhoz használni lehet egy Facebook vagy Google fiókot. A bejelentkezés folyamata után a csomag részletes nézetének az oldalán látható, egy gomb a jobb alsó sarokban, amelyet megnyomva az aktuális csomagot hozzá tudjuk adni a tervező módhoz.

Ha szeretnénk más megyét vagy községet kiválasztani, erre lehetőség van a jobb felső sarokban elhelyezett gomb segítségével, amelyre kattintva vissza kerülünk az első oldalra, ahol kezdtük az alkalmazás használatát.

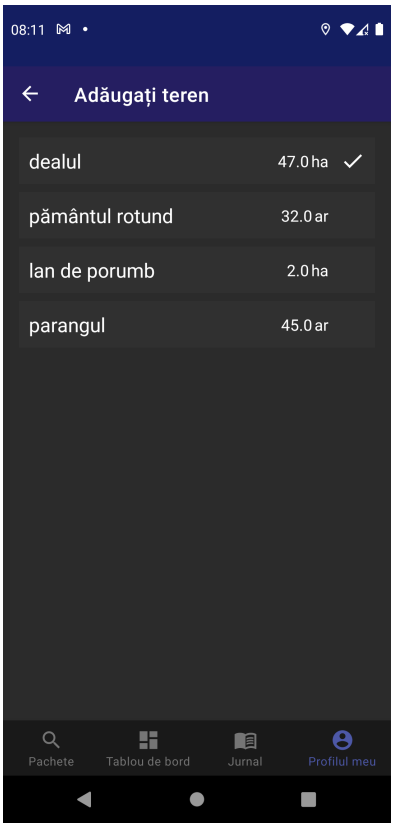
Miután a felhasználó kiválasztotta a neki megfelelő támogatásokat, átlépve a Saját profil oldalra megjelennek a támogatások, amelyeket hozzáadott a tervező módhoz (10. ábra). Itt látható, hogy megjelenik a lap felső részén egy összeg, amely jelöli a támogatások összesített értékét. Minden támogatásnak vannak előfeltételei amelyeknek meg kell felelni, hogy igénybe tudja venni a gazda a támogatást. Az előfeltételek csempére kattintva egy új oldalra navigál az alkalmazás (11. ábra) ahol a felhasználó el tudja fogadni az előfeltételeket egyesével vagy egyszerre az összeset. Mivel a támogatások többsége földterületen alapszik ezért szükséges



10. ábra. Tervezett csomagok



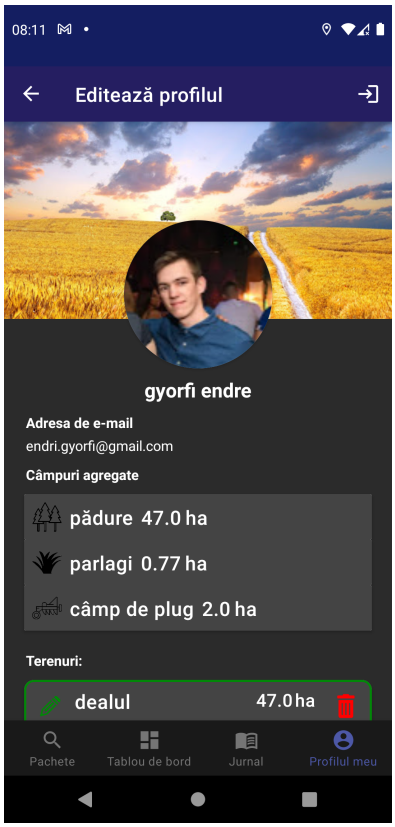
11. ábra. Előfeltételek



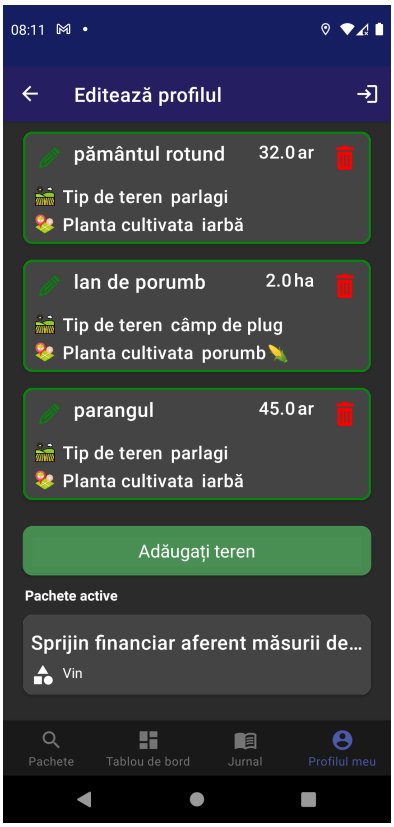
12. ábra. Földterületek

a támogatásokhoz földterületet rendelni. Hasonló módon az előfeltételekhez, a hozzáadott földterületek csempére kattintva egy új oldalon megjelennek a gazda földterületei (12. ábra), amelyeket kiválasztva eleget tud tenni a támogatásnak.

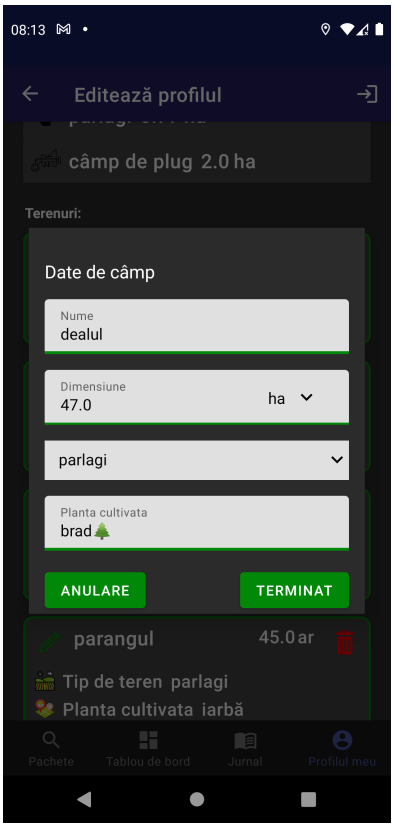
Említve volt, hogy a támogatásokhoz földterületet kell rendelni de hól lehet hozzáadni ezeket a földterületeket? A profil szerkesztése gombra kattintva bejön a profil oldal (13. ábra). Itt a felhasználóról eltárolt fontosabb adatok vannak megjelenítve, e-mail cím, felhasználónév és egy profilkép. Legörgetve látható lesz a gazda minden földterülete (14. ábra), ugyanitt létre tud hozni új földterületet (15. ábra) megadva a terület nevét, méretét, kiválasztva a terület mértékegységet, megadva a földterület típusát (természetvédelmi terület, szántóföld, erdő stb.) és opcionálisan megadva, hogy milyen növény van termesztve a területen. Lehetőség van szerkeszteni a létrehozott területeket illetve opcionálisan törölni is lehet.



13. ábra. Profil szerkesztése oldal

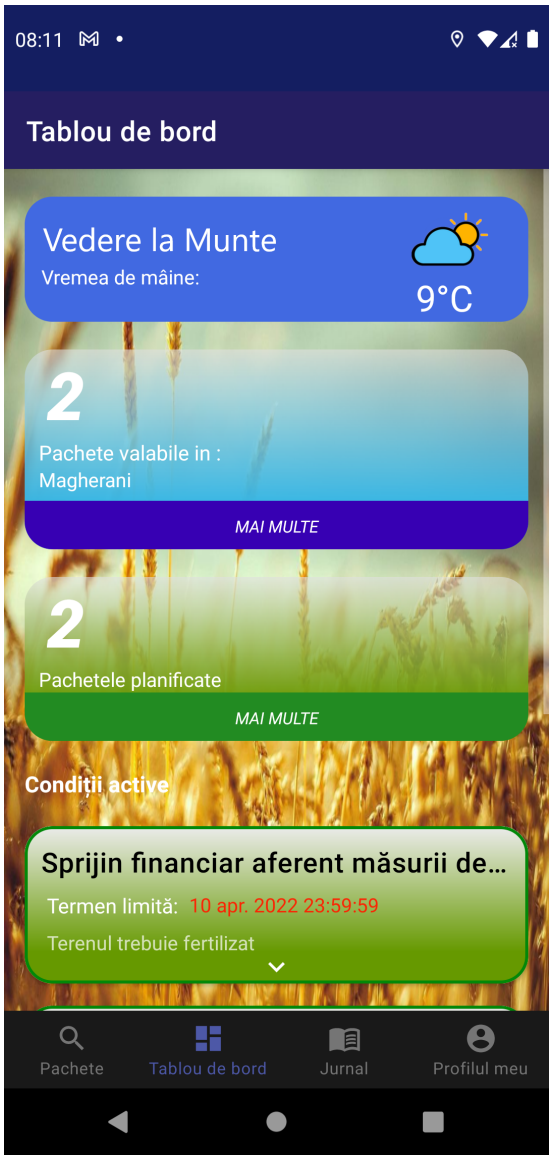


14. ábra. Földterületek és aktív csomagok listája



15. ábra. Földterület létrehozása

Mint látható a 13. ábrán, megjelenik egy összesített nézet, amelyen látható, hogy egy gazdának egy adott típusú fölterületből összesen mennyi van. Most, hogy a tervezett csomaghoz földterületet rendeltünk és elfogadtuk az előfeltételeket, a csomagot aktívvá tudjuk tenni a 10. ábrán látható Hozzáadás az aktív csomagokhoz gombot lenyomva.



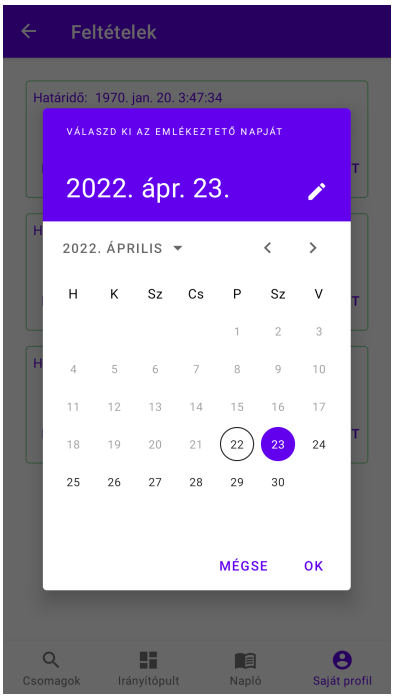
16. ábra. Irányítópult

17. ábra. Feltételek

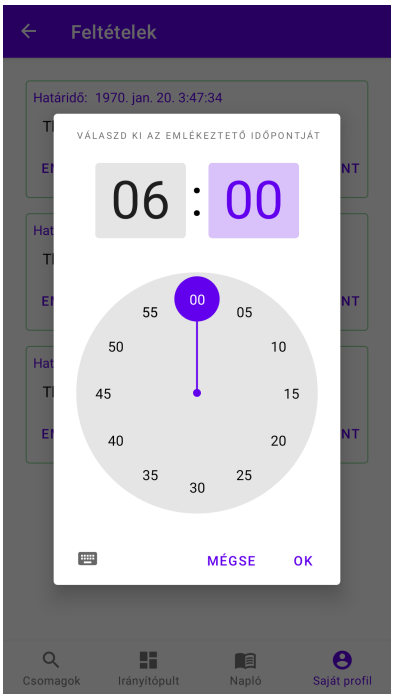
Miután egy csomagot aktívvá tettünk átnavigálva az Irányítópult oldalra (16. ábra) láthatunk egy időjárás jelentést, láthatjuk a kiválasztott helységekben elérhető támogatások számát illetve, hogy hány aktív támogatás van. Minden aktív támogatáshoz tartozhatnak feltételek amelyeket be kell tartani, hogy a gazda megfeleljen a kiírt támogatásnak és megkapj a kiírt összeget. Az Irányítópult oldalon legörgetve láthatóak a feltételek (17) amelyeket a gazda be kell tartson. A feltételeknek van egy határidejük, a feltételek a határidő lejárta szerint csökkenő sorrendben vannak megjelenítve, hogy a gazda mindig azt lássa meg elsőként amelynek a határideje hamarabb lejár.

Minden egyes támogatásnak vannak feltételei, melyeket a gazda el kell fogadjon, amint egy támogatást aktívvá tesz. Ezeknek a feltételeknek van határidejük, és emiatt emlékeztetőket tud

18. ábra. Aktív támogatás feltételei



19. ábra. Emlékeztető dátumának kiválasztása



20. ábra. Emlékeztető idejének kiválasztása

a gazda létrehozni minden egyes feltételhez, ahogyan a (18), (19) és (20) ábrákon láthatjuk. Az alkalmazás emlékeztetőt fog küldeni pár nappal hamarabb a feltétel határidejének lejárta előtt is.

5.2. Adminisztrációs panel használata

A támogatásokat android alkalmazáson belül nem lehet létrehozni csak a Strapi által nyújtott adminisztrációs panelen keresztül. Egy támogatás létrehozásának folyamata a következő módon történik. A Strapi megnyitása után a bal oldalt található menüből ki kell választani a Collection Types alatt található Packages mezőt, ezt követően megjelenik egy táblázat ahol láthatóak a publikált támogatások listája. A táblázat felett megjelenő Add New Packages gombra lépve megjelenik egy felület, itt minden adatot meg lehet adni ami a támogatással kapcsolatos. Kitöltve a szükséges adatokkal a mezőket ki kell választani, hogy milyen helységekben lesz érvényes a támogatás. A Save gombbal elmentve Draft állapotba kerül amelyet a publikálás előtt lehet szerkeszteni, ha minden megfelel a követelményeknek a támogatást publikálni lehet, így minden végfelhasználó számára elérhetővé válik.

6. Fejlesztési eszközök és módszerek

Ebben a fejezetben bemutatásra kerül a fejlesztés folyamata során alkalmazott eszközök használata, amely lényegesen megkönnyítette a csapatban történő fejlesztést.

6.1. Scrum

Az APIA APP projekt fejlesztése a Scrum [18] módszer alapján történt. A projekt fejlesztésében résztvevő csapat a problémákat két hetes sprintekre osztotta fel. Minden sprintet tervezés előzött meg ahol a csapat megbeszélte a funkcionalitásokat, amelyeket a sprint végére meg szeretne valósítani. Minden sprint végén egy megbeszélés keretén belül a csapat bemutatta az aktuális fejlesztéseket az ötletgazdának és a mentoroknak, akik kifejezték véleményüket a Demóban bemutatott haladásról. A Codespring keretein belül rendezett nyári gyakorlat ideje alatt a csapat minden nap tartott egy megbeszélést, amelyben elmondták a résztvevők, hogy mivel foglalkoztak a megbeszélés óta, hol akadtak el és mivel fognak foglalkozni ezt követően. Az egyetem kezdetével a napi megbeszélések átalakultak és heti rendszerességgel találkozott a csapat.

6.2. Git és GitLab

A projekt fejlesztése során Git-et használtunk a verziókövetésre és GitLab-ot a repository menedzsmentre. Szétválasztásra került a backend és az Android alkalmazás forráskódja két különböző repository-ba, megkönnyítve a párhuzamosan történő fejlesztést. Minden feature implementálása külön branchen történt, egy funkcionalitás implementálását követően egy Merge requestet létrehozva a csapat átnézte a kódot majd ha minden megfelelő volt a develop branchen egyesítésre került.

6.3. CI/CD Pipeline

A Continuous Integration (CI) fontos szerepet hordoz a projekt fejlesztésében, mivel biztosítja azt, hogy helyes és a konvencióoknak megfelelő új kód kerül be a már meglévő kódbázisba. Mindez úgy történik, hogy automatikus ellenőrző folyamatokat indít, amikor a fejlesztő módosít valamely részt a projektben és feltölti azt a git tárolóra.

A GitLab CI/CD egy olyan eszköz, mely a CI használatát egyszerűvé teszi, úgynevezett *pipeline*-ok segítségével, amelyek különböző feladatokat hajtanak végre. [3] A projekt

gyökerében található egy *.gitlab-ci.yml* nevezetű fájl, amiben a pipeline-ok működését lehet leírni, majd ezen belül a feladatok futásának sorrendjét és egyéb beállításokat lehet meghatározni. Ebben a fájlban különböző stage-eket tudunk létrehozni, melyek sorrendben futnak le egymás után és egy-egy fontos szerepet hordoznak, mint például kompilálás tesztelése és kód minőségi ellenőrzése. Ezeken belül találhatóak a job-ok, melyek párhuzamosan is tudnak futni és lépéseket fogalmaznak meg a stage-n belül.

Ha minden stage, ami a pipeline-ban található, sikeresen lefutott, akkor azt jelenti, hogy minden rendben van a kóddal, és nem kell javítani semmit se benne. Ez megkönnyíti a programozók feladatát, mivel nem kell manuálisan megnézzék ők, ha helyes-e minden, így egyből elfogadhatják az újonnan bekerült módosítást.

```
image: androidsdk/android-30

stages:
  - check
  - build

cache:
  key: ${CI_PROJECT_ID}
  paths:
    - .gradle/

check:
  stage: check
  script:
    - ./gradlew detekt

build:
  stage: build
  script:
    - ./gradlew assembleDebug
```

4. kódrészlet. GitLab CI

A projekt *.gitlab-ci.yml* állományában a 4. kódrészlet található, ahol két különböző stage található, melyek kód minősége, illetve build-elési folyamat tesztelésére szolgálnak. A kód ellenőrzését a Detekt [6.5] futtatásával ellenőrzi a pipeline, majd a kód kompilálását az *assembleDebug* task végrehajtásával, mely hibát dob, ha gondok vannak a kódban valahol.

6.4. Gradle

Android Studio környezetben jelen van a Gradle project építő eszköz, amely több funkcionalitással rendelkezik. Az egyik legfontosabb a Gradle funkcionalitásai között a függőség kezelés. Az applikáció fejlesztése során több függőséget használunk, melyek gyakran modulok formájában jelentkeznek. Meg kell adni a Gradle-nek, hogy hol találja ezeket a modulokat, hogy a build felhasználhassa őket. Ezt a helyet tárolónak nevezzük amely lehet lokális vagy távoli. A mi esetünkben a függőségek nagy részét a Google és a Maven Central biztosítja. A több tíz vagy száz függőséggel rendelkező projektek könnyen szenvedhetnek úgynevezett függőségi poklot, ezért szükséges egy eszköz amely folyamatosan ellenőrzi és vizsgálja az esetlegesen tranzitív függőségek használatát.

6.5. Detekt

Egy projekt csapatban történő fejlesztése során szükséges lefektetni olyan szabályokat, melyek segítenek a kód egységes kinézetelében, ezzel elkerülve a potenciális problémákat a teszt kód olvashatóságával és karbantarthatóságával kapcsolatban. A detekt plugin használata lehetővé teszi, hogy általános szabály készletet hozzunk létre a kódbázis kinézetét illetően. Alapvetően a detekt kigenerál egy detekt.yml filet amelybem definiálva vannak az alkalmazni kívánt szabályok amelyek vonatkozhatnak függvényekre, a függvény paramétereinek maximális száma megvan határozva, továbbá a Függvény maximális hossza is megvan határozva, megkötések vannak importokra, változókra, osztályokra és még sok másra.

Következtetések és továbbfejlesztési lehetőségek

A jelen dolgozat bemutatta az APIA APP Android alkalmazás egy prototípusát, a felhasznált technológiákat és az adminisztrációs felületet. A közösen meghatározott és előre kitűzött céloknak megfelelően az alkalmazás összegyűjti a Románia szinten kiírt támogatásokat, amely nagymértékben segíthet a gazdáknak az informálódásban mivel Románia szinten egyedinek tekinthető az alkalmazás.

A projekt jövőbeli célkitűzései közé tartozik a gazdálkodási napló automatikus kitöltése, amely vezetné a gazdák tevékenységeit a támogatás alatt lévő földterületeiken. Lehetőséget teremtve a felhasználónak, hogy az elavult kézzel kitöltendő gazdálkodási napló helyett egy kidolgozott felületen részben automatikusan megtörténjen a tevékenységek rögzítése. Lehetőség lenne az elektronikus napló exportálására az állam által követelt nyomtatott formátumban.

A jövőben szeretnénk implementálni további funkcionalitásokat:

- szeretnénk megvalósítani az alkalmazás iOS-en futó verzióját, amelyhez a Kotlin Multiplatformnak köszönhetően csak az iOS felhasználói felületet szükséges létrehozni;
- a földterületekkel kapcsolatban szeretnénk megvalósítani, hogy a gazda GPS segítségével tudja bejelölni a földterületeit és egy térképen megjeleníteni azokat;
- szükségét érezzük egy offline mód megvalósításának tekintve arra, hogy a gazdák gyakran járhatnak olyan helyeken ahol nem megfelelő az internet kapcsolat;
- értesítéseket szeretnénk küldeni a felhasználónak a támogatások feltételeinek egyszerűbb betartása érdekében
- szeretnénk bevezetni egy felhasználói felületet ahol a gazdák feltehetik a kérdéseiket vagy segítséget kérhetnek bizonyos folyamatok, iratok tisztázásában a támogatáshoz szükséges mappa összeállításához;

Hivatkozások

[1] *Android's Kotlin-first approach*. 2021. URL: https://developer.android.com/kotlin/first (elérés dátuma: 2022. ápr. 13.)

[2] *Asynchronous programming techniques*. 2021. URL: https://kotlinlang.org/docs/async-programming.html (elérés dátuma: 2022. ápr. 22.)

[3] *CI/CD pipelines*. 2022. URL: https://docs.gitlab.com/ee/ci/pipelines/index.html (elérés dátuma: 2022. ápr. 22.)

[4] *Connect to platform-specific APIs*. 2022. URL: https://kotlinlang.org/docs/multiplatform-connect-to-apis.html (elérés dátuma: 2022. ápr. 26.)

[5] *Content negotiation and serialization*. 2022. URL: https://ktor.io/docs/serialization.html (elérés dátuma: 2022. ápr. 26.)

[6] *Controllers*. 2022. URL: https://docs.strapi.io/developer-docs/latest/development/backend-customization/controllers.html#implementation (elérés dátuma: 2022. ápr. 13.)

[7] *Coroutines basics*. 2022. URL: https://kotlinlang.org/docs/coroutines-basics.html (elérés dátuma: 2022. ápr. 22.)

[8] *Data classes*. 2022. URL: https://kotlinlang.org/docs/data-classes.html#copying (elérés dátuma: 2022. ápr. 5.)

[9] *Getting started with a Ktor Client*. 2022. URL: https://ktor.io/docs/getting-started-ktor-client.html (elérés dátuma: 2022. ápr. 1.)

[10] *Getting Started with MVP (Model View Presenter) on Android*. 2022. URL: https://www.raywenderlich.com/7026-getting-started-with-mvp-model-view-presenter-on-android (elérés dátuma: 2022. ápr. 1.)

[11] *Hartile zonelor eligibile pentru M10, M11 și M13*. 2022. URL: https://www.madr.ro/pndr-2014-2020/implementare-pndr-2014-2020/masuri-de-mediu-si-clima/masuri-de-mediu-si-clima-2021/hartile-zonelor-eligibile-pentru-m10-m11-si-m13.html (elérés dátuma: 2022. ápr. 26.)

[12] *IT system - Agency for Payments and Intervention in Agriculture (APIA)*. 2020. URL: https://www.fi-compass.eu/sites/default/files/publications/financial_needs_agriculture_agrifood_sectors_Romania.pdf (elérés dátuma: 2022. ápr. 26.)

[13] *Kotlin vs Java: Which is the Best Choice for Android App Development?* 2021. URL: https://medium.com/javarevisited/kotlin-vs-java-which-is-the-best-choice-for-android-app-development-7c9fc782d2c9 (elérés dátuma: 2022. ápr. 13.)

[14] *MVP (Model View Presenter) Architecture Pattern in Android with Example*. 2022. URL: https://www.geeksforgeeks.org/mvp-model-view-presenter-architecture-pattern-in-android-with-example/ (elérés dátuma: 2022. ápr. 1.)

[15] *NullPointerException?* URL: https://docs.oracle.com/javase/7/docs/api/java/lang/NullPointerException.html (elérés dátuma: 2022. ápr. 26.)

[16] *Share the logic of your iOS and Android apps while keeping the UX native*. 2022. URL: https://kotlinlang.org/lp/mobile/ (elérés dátuma: 2022. ápr. 26.)

[17] *Strapi v4 developer documentation*. 2022. URL: https://docs.strapi.io/developer-docs/latest/getting-started/introduction.html (elérés dátuma: 2022. ápr. 5.)

[18] *What is Scrum?* 2022. URL: https://www.scrum.org/resources/what-is-scrum (elérés dátuma: 2022. ápr. 26.)