Freelancer szállító

UntitledRepo

Konzulens:

Ekler Péter, Forstner Bertalan

Csapattagok:

Kriszt Benedek Miklós	Q0FN13	b3n3d3k97@gmail.com	Frontend / Android / Kotlin
Traxler Bálint	IXNOYB	traxlerbalint@gmail.com	Frontend / Android / Kotlin
Sipos Áron Dávid	EKAUKB	sipisipos@gmail.com	Backend / Spring / Kotlin

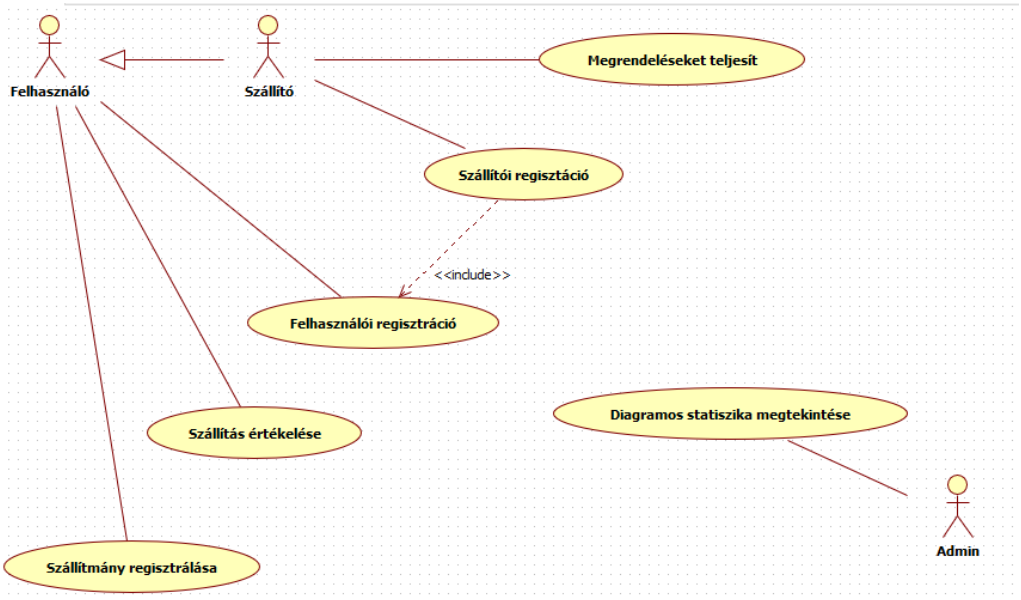
Git: https://github.com/BMKV/UntitledRepo-Temalab

2020.12.10

Specifikáció:

Feladat kiírás

Use Case



Use-case neve	**Szállítmány regisztrálása**
Rövid leírás	Az alkalmazás felhasználói felregisztrálhatnak szállítmányokat, amiket majd a szállítók fognak elszállítani.
Aktorok	Felhasználó
Forgatókönyv	1. A felhasználó megadja a szállítmány adatait (felvevési pont, leadási pont, méret). 1.1 A méretet a felhasználó előre, az alkalmazás által definiált csomagméretben adja meg. 2. A felhasználó véglegesíti a regisztrációt.

Use-case neve	Szállítás értékelése
Rövid leírás	Miután a felhasználóhoz tartozó szállítmány szállítása megtörtént, akkor kitölthet egy értékelő űrlapot.
Aktorok	Felhasználó
Forgatókönyv	1. A felhasználó által regisztrált szállítmány elszállítása megtörtént. 2. A felhasználó értékelheti a kapott szolgáltatást.

Use-case neve	Felhasználói regisztráció
Rövid leírás	A felhasználó létrehoz egy fiókot, amivel hozzáférhet a feladott szállítási kérvényeivel kapcsolatos adataihoz.
Aktorok	Felhasználó
Forgatókönyv	1. A felhasználó megadja az általa későbbiekben használt felhasználónevét, és ahhoz egy jelszót. 2. A felhasználó fiókja létrejön.
Alternatív forgatókönyv	2.A.1 A felhasználó által adatok nem megfelelőek, ezt alkalmazás jelzi a felhasználónak, és kéri a hibás adatok javítását.

Use-case neve	Szállítói regisztráció
Rövid leírás	A szállító beregisztrálja az adatait, mint például, hogy mekkora raktérrel rendelkezik. A szállítónak minden olyan adatot is meg kell adnia, mint ami az egyszerű felhasználónak szükséges, a szállító felhasználóként is tudja használni az alkalmazást (csomagok feladására).
Aktorok	Szállító
Forgatókönyv	1. A regisztráló szállítójelöltnek meg kell adnia a felhasználói adatait. 2. A szállítójelölt megadja a szállításhoz szükséges adatait (pl. rakterének mérete). 2.1 A raktér méretének megadása az alkalmazás által ismert csomagméret mértékegység formájában történik.
Alternatív forgatókönyv	2.A.1 A megadott adatok nem megfelelőek, erről értesítést kap a felhasználó.

Use-case neve	Megrendelés teljesítése
Rövid leírás	A szállító munkája a feladott szállítmányok elfuvarozásának teljesítése.
Aktorok	Szállító
Forgatókönyv	1. A szállító kap egy javaslatot az alkalmazástól, hogy milyen munkákat tud elvégezni. 2. A szállító kiválasztja a megrendelések közül, amelyeket teljesíteni kíván. 3. A szállító elvégzi a fuvarozást. 4. Minden megrendelés teljesítését a szállító jelzi az applikációban.

Use-case neve	Diagramos statisztika megtekintése
Rövid leírás	Az adminnak lehetősége van statisztikákat diagramos formában megtekinteni, amelyekből érdekes információkat szűrhet le.
Aktorok	Admin
Forgatókönyv	1. Az admin kiválasztja, hogy pontosan melyik statisztikára kíváncsi (Leghosszabb/legrövidebb megtett út/szállítási idő, szállítók rangsora átlagos szállítási sebesség alapján). 2. Az admin megtekinti a kívánt statisztikát diagramos formában.

Architektúra

Mobilkliens

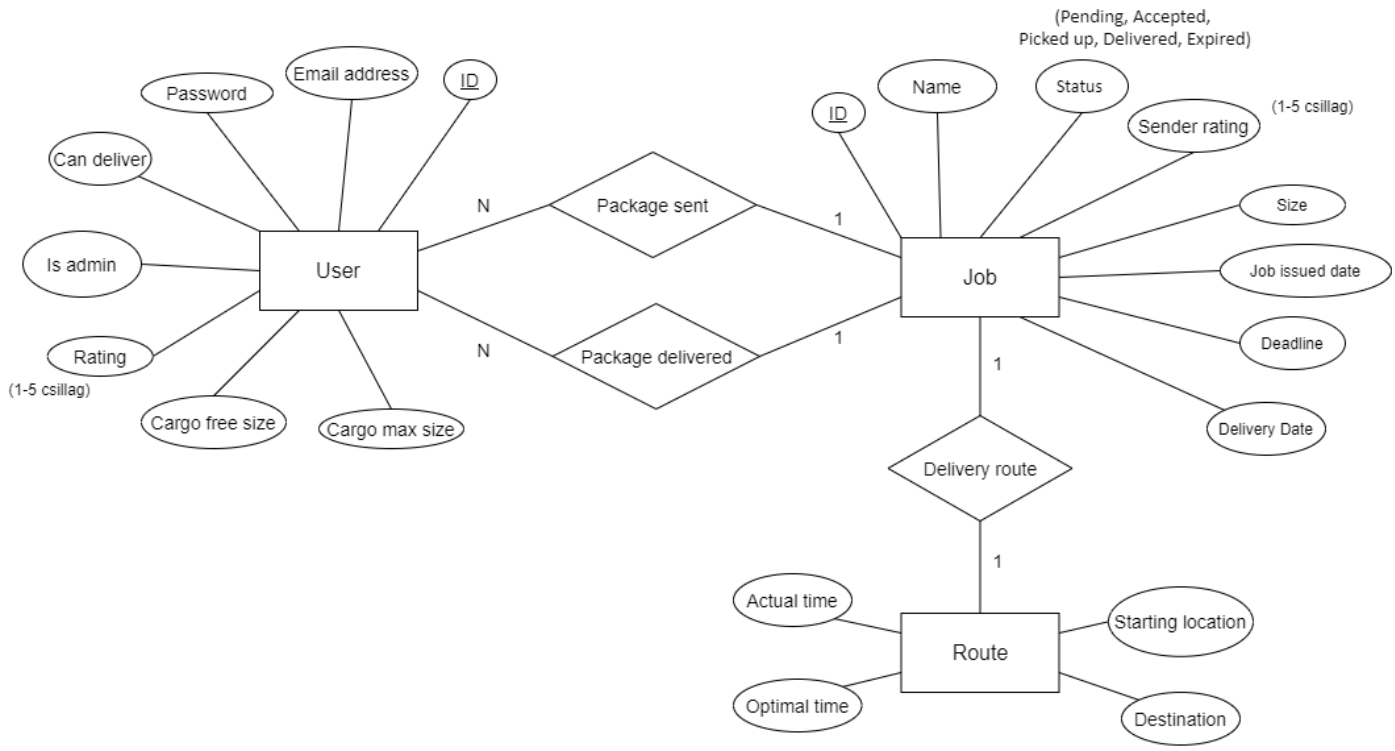
Single Activity Architecture

Az Android Jetpack részét képző Navigation Components alkalmazásának egyik nem titkolt célja, hogy segítségével egyszerűen kialakíthatóvá váljon az alkalmazás tervezésekor a Single Activity Architecture. A Single Activity Architecture lényege, hogy az alkalmazásunk csak egy Activityvel rendelkezik, és benne Fragmenteket váltogatunk. A Navigation Components ehhez biztosít egy jól használható könyvtárat. Ennek az architektúrának egyik nagy előnye, hogy az alkalmazáson belüli navigáció könnyedén átlátható, tervezhető, és tesztelhető. Továbbá segítségével az alkalmazásfejlesztő sokkal nagyobb kontrollt kap az alkalmazás különböző navigálási animációi felett; így a fejlesztő egy kiszámítható, egységes alkalmazásmegjelenést dolgozhat ki.

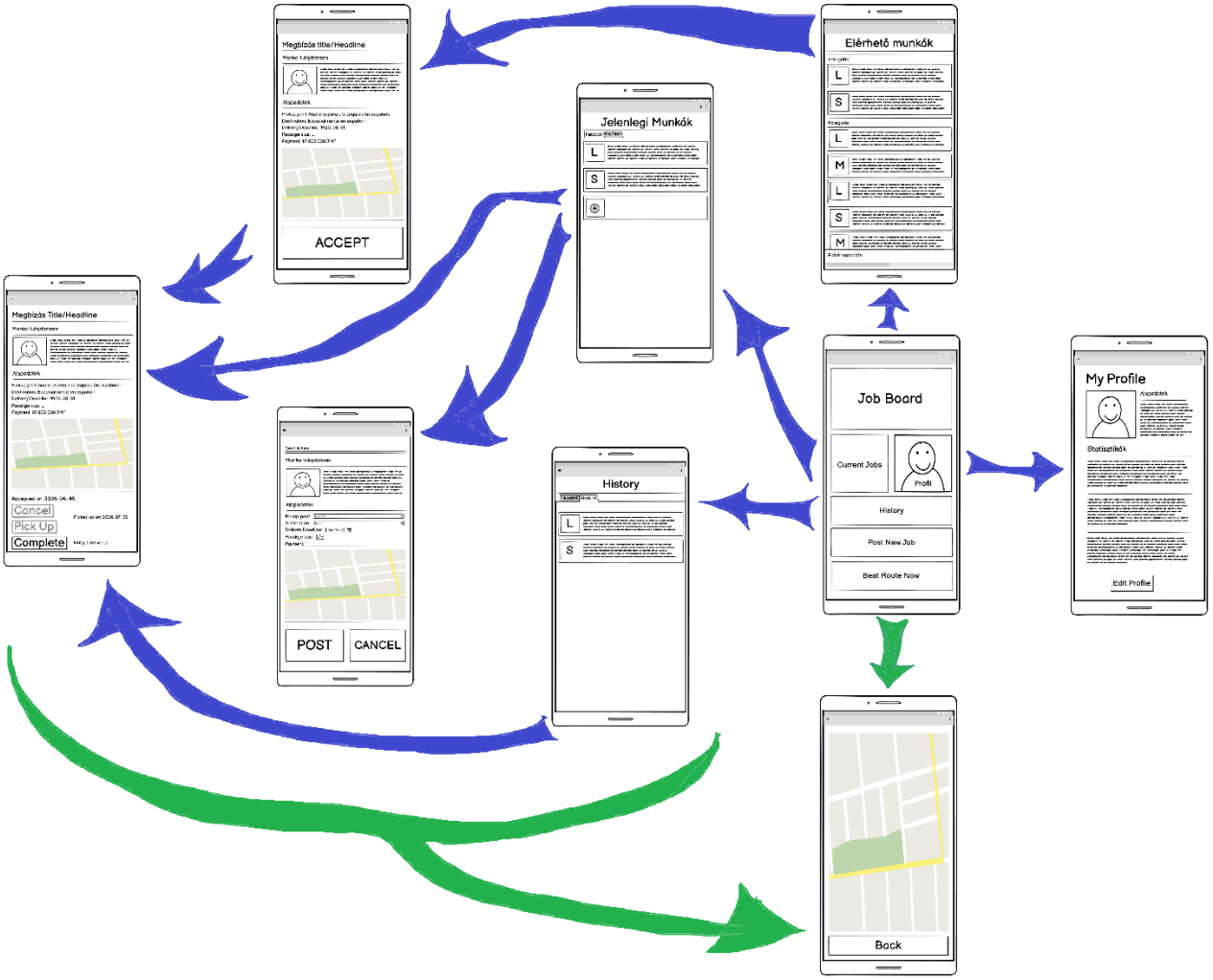
Model-View-ViewModel (MVVM) Architecture

Ahogy már tanultuk, egy szoftver karbantarthatóságának szempontjából egyik legkritikusabb szempont a megfelelő architektúra megválasztása. Android szoftverfejlesztéskor ekkor kiváló támogatást ad nekünk ebben a feladatban az Android Jetpack részét képező Architecture Components. Az architektúra az alkalmazást alapvetően három rétegre bontja: View, ViewModel, Model. A View réteg felelőssége az android specifikus megjelenítés megvalósítása. A View réteg alatt helyezkedik el a ViewModel réteg, amely adatokat szolgáltat a View réteg elemeinek. Ezenkívül a ViewModel effektivitását még az adja, hogy képes megfigyelni a hozzárendelt View rétegbeli komponens életciklusát, így például nagyon effektíven lehet frissíteni vele a View réteg komponenseit. A ViewModel az adatokat a Model réteg elemeitől szerzi. Általában a Model rétegben Repository osztályokat használunk, amelynek felelőssége az lesz, hogy implementálja a konkrét adathozzáférést. (Legyen ez akár valamilyen perzisztens adatforráshoz hozzáférés, vagy hálózati adatlekérés.).

ER-diagram



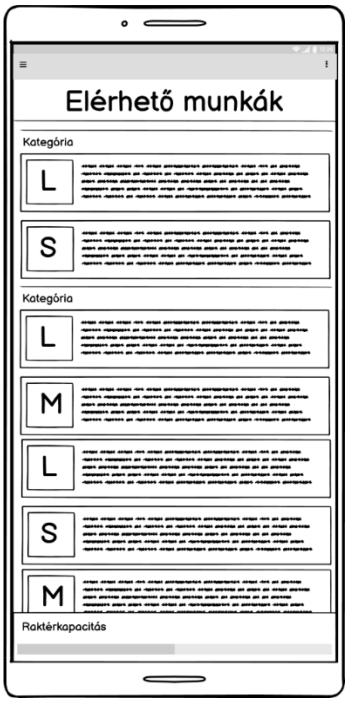
Layout Designok



Profile screen

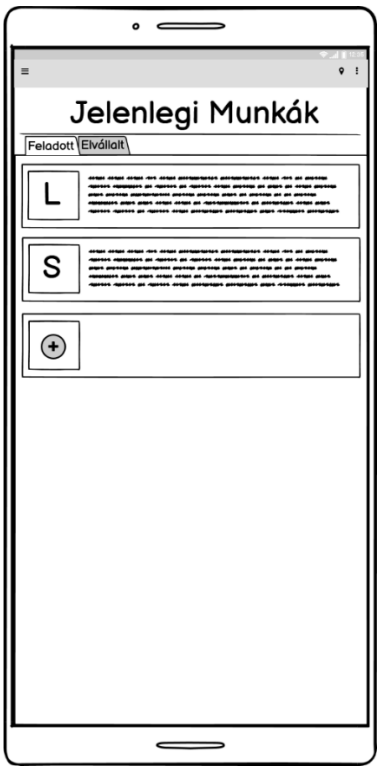
Job board

Listázza az általad felvehető munkákat, a csomagokat rendezve jeleníti meg: kicsi - közepes - nagy. Majd ezeken a kategóráikon belül a felvevési pont közelsége alapján vannak rendezve.



My current jobs

Listázza az általad elfogadott vagy kiadott megbízásokat amik aktívak, ezenfelül erről a képernyőről jutunk tovább a csomag feladása és csomag kiszállítása felületekre.

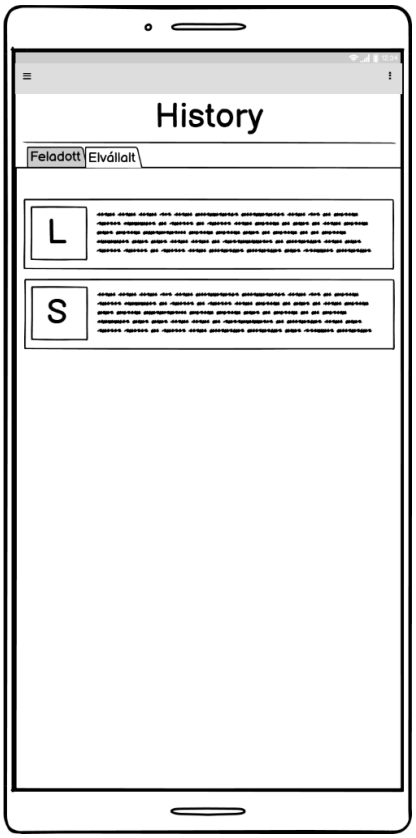


Currently best útvonal

Akkor jelenik meg, érhető el a gomb, ha van legalább egy aktív munka

History

Listázza a már befejezett munkáidat.

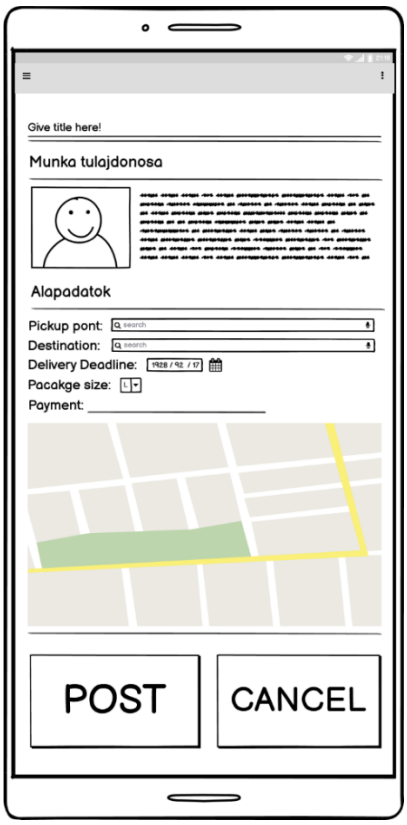


Job details

Részletes infót ad egy megbízásról. Megtalálható rajta: Feladó, csomag felvételi és leadási pontja, a csomag mérete, és az ár.

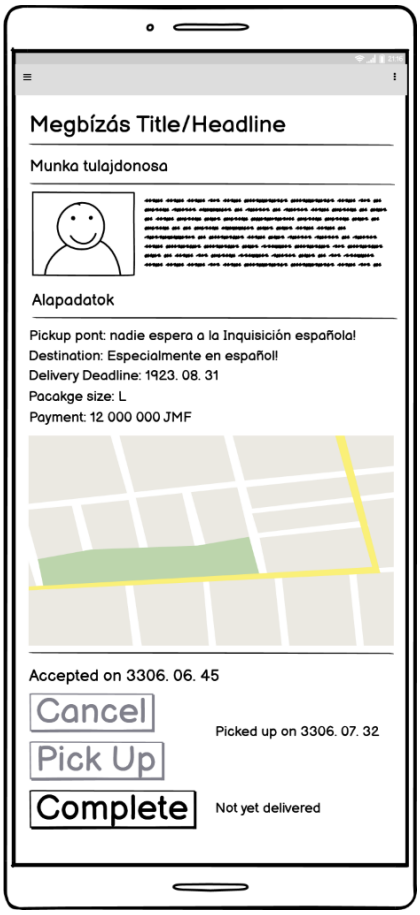
Post parcel

A képernyőn a feladó kitölti az adatokat a csomagról: megadja a mérete és a felvételi ladás pon

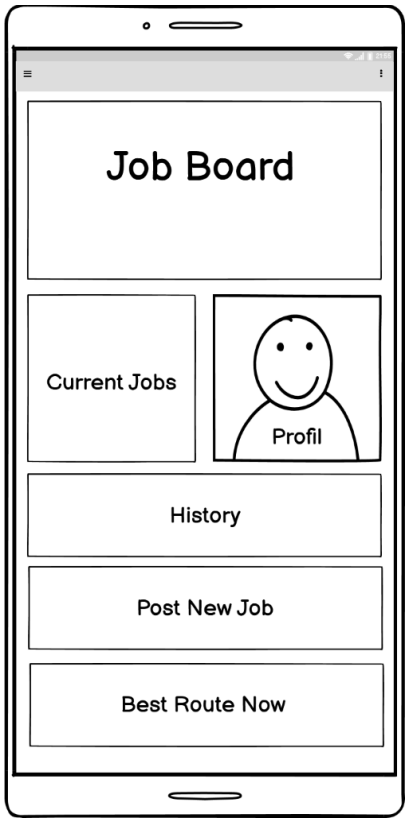


Accept parcel

Check parcel



- Login/Regisztráció screen (mint feladó) -> part 2 of regisztráció pedig szállító reg
- Main screen / navigáció:
    - Win10 csempe stílusú home screen
    - NavigationDrawer



Haladási napló:

4.hét:

Kriszt Benedek Miklós:

- Egyszerű hírfeltöltő alkalmazás

Traxler Bálint:

- Egyszerű hírolvasó alkalmazás elkészítése, ami képes megjeleníteni a backendről lekért adatokat.
- Use-case diagram elkészítése a feladathoz a csapattal megbeszéltek alapján; diagramon szereplő use-casek kifejtése.

Sipos Áron Dávid:

- Egyszerű REST alapján kommunikáló backend alkalmazás elkészítése Spring segítségével. Az alkalmazás Heroku-n fut és adatbázisból olvassa ki és menti le az adatokat.

5.hét:

Kriszt Benedek Miklós:

- GUI tervezés, alkalmazás felépítésének kidolgozása

Traxler Bálint:

- Ezen a héten csak részt vettem az értekezleteken.

Sipos Áron Dávid:

- Backend entitás diagram elkészítése.
- Backend REST API tervezés elkezdése.

6.hét:

Kriszt Benedek Miklós:

- GUI vázlatok korábbi tervek alapján megvalósítása wireframe formátumban

Traxler Bálint:

- Kisebb változtatások a github repositoryn (gitignore hozzáadása, struktúra kisebb változtatása).
- History feature implementálása hálózati bekötés nélkül.
- Kisebb átalakítások az alkalmazásban; Single Activity Architecture implementálására Navigation Components segítségével.

Sipos Áron Dávid:

- Backend REST API tervezés befejezése.
- Backend implementáció elkezdése.
- Az eddig implementált végpontok elérhetővé tétele Heroku segítségével.

7.hét:

Kriszt Benedek Miklós:

- JobDetails képernyők layout
- PostJob layout
- Main Menu layout
- Profile Screen layout

Traxler Bálint:

- További átalakítások Navigation Components segítségével
- History feature hálózati bekötése
- History feature néhány relevánsabb részletének dokumentálása

Sipos Áron Dávid:

- ORM, Hibernate használata az adatbázis rétegben.
- Backend implementáció folytatása.
- Heroku-s git helyett a csapat Git használata backend branchben.

8.hét:

Kriszt Benedek Miklós:

- Navigation Graph összeállítása
- Adattároló osztályok kialakítása, implementálása

Traxler Bálint:

- Hálózatkezelés átalakítása a konzulenssel egyeztetett módon (NetworkManager osztály bevezetése), eszerint kód átszervezése
- History feature UX javítások, kezdetleges hibakezelés
- Navigation Components-cel Munka részletek feature összekötése a History feature-rel

Sipos Áron Dávid:

- Backend végpontok implementálva vannak, innen tesztelni kell őket, hogy az elvárt viselkedést biztosítsák.
- OpenRouteAPI használata backend-en az optimális út számításához.

9.hét:

Kriszt Benedek Miklós:

- Navigation Components implementálásához szükséges átalakítások folytatása

Traxler Bálint:

- Jelenlegi munkák feature elkészítése Repository szintig bezárólag
- Kisebb javítások a History feature-ön
- Munka listázó funkciók általánosítása; közös RecyclerView Adapter ötletének megvalósítása
- Navigáció igazítása az újításokhoz

Sipos Áron Dávid:

- Sprint eltérítés backend-re eső részek megvalósítása:
- OpenAPI dokumentáció frissítése
- ER – diagram frissítése
- Adatbázisban lévő táblák frissítése
- Változtatások implementálása

10.hét:

Kriszt Benedek Miklós:

- Sprint eltérítés implementálása az adattároló osztályokban
- JobDetails és PostJob funkciók kiegészítése a sprint eltérítés funkciójával

Traxler Bálint:

- Jelenlegi munkák feature hálózati bekötése
- Elérhető munkák feature teljes implementálása
- Kisebb változtatások, és alkalmazás általam implementált része felkészítése az első demóra

Sipos Áron Dávid:

- REST végpontok frissítése, hogy kényelmesebb legyen a frontend használata visszajelzés alapján:
- Job végpont adja vissza a feladó felhasználó Id-ját
- Használt Spring deprecated annótációk törlése
- Maven pom.xml feljesztése
- User/register végpont javítása

11.hét:

Kriszt Benedek Miklós:

- Térkép elemek implementálása
- JobDetails és PostJob fragment MotionLayout-os kiegészítése
- MotionLayout animáció implementálása
- JobDetails fragment fragment tranzakcióval kiegészítése elfogadott munkák esetében a státusz változtatásához.

Traxler Bálint:

- Általam eddig elkészített featureök felhasználóbarátabbá tétele (jobb hibakezelés, jobb design, hálózati frissítés gyakoriságának változtatása)
- Clean code elvek szerinti refaktorálás az eddig általam implementált featureökben
- Profil feature kiegészítése MVVM architektúra szerint, és hálózati bekötése (Kivéve a View réteg; azt korábban Benedek készítette el.)

Sipos Áron Dávid:

- Különböző végpontok javítása:
  - user/{user-id}/cargo
  - user/update
  - user/{user-id}
  - user/profile
  - user/register (megint)
- Frontend és backend adat entitások egymással egyenlő állapotba hozása: admin felhasználó, Job entitásnak név

12.hét:

Kriszt Benedek Miklós:

- All Locations fragment létrehozása és beillesztése az alkalmazásba
- JobDetails és PostJob fragment hálózati kommunikációra bekötésének befejezése
- Profile Screen elrendezésének véglegesítése
- Felhasználó által feladott más által elvégzett munkák értékelése funkció létrehozása
- JobDetails és PostJob fragment kiegészítése a térképek nyitását/csukását vezérlő gombbal
- DemoData felszámolása

Traxler Bálint:

- Login feature implementálása
- Register feature implementálása
- Admin statisztika feature implementálása
- Főmenü MVVM architektúra szerinti átalakítása (Kivéve a View réteg; azt korábban Benedek készítette el.)

Sipos Áron Dávid:

- Dátum formátum egyesítése: Java LocalDateTime objektumokra.
- Munkának új státusz: felvett állapot, ezeknek OpenAPI dokumentációja.
- Dokumentáció vázának elkészítése.

13.hét:

Kriszt Benedek Miklós:

- ...

Traxler Bálint:

- Applikáció design kialakítása (Téma, ikon)
- Felhasználó szerepkörének (Küldő, Szállító) megfelelő alkalmazás logika implementálása
- Login feature hibakezelés javítás
- Regisztráció feature módosítása a megbeszéltek szerint; hogy a transport role-ú felhasználó meg tudja adni, hogy maximum mennyi csomagot tud szállítani

Sipos Áron Dávid:

- Felhasználó rakterének backend kezelésének javítása:
  - konzisztens használat a frontend-el.
  - Null pointer exception-ök javítása
- Felhasználó értékelésének számítása implementálva, frissítés minden egyes értékelés után.
- Felhasználó regisztrációnál raktár méretének opcionális megadása implementálva.

14.hét:

Kriszt Benedek Miklós:

- Térkép markerek megvalósítása a JobDetails és PostJob fragmenteken

Traxler Bálint:

- -

Sipos Áron Dávid:

- JWT token autentikáció az API-hoz
- Jelszavak titkosított tárolása

Tapasztalatok az együttműködési platformról:

Sipos Áron Dávid:

Először használtam a Github Projects-et, viszont volt már tapasztalatom Trello-val. Trello után pár funkció nekem hiányzott.:

A Github Projects nem menti el az időt, nem lehet határidőt megadni, amit Trello-val lehetett, így a végén csak a sorrend mutatja az „időt", ami, ha sok kártya nehezen átláthatóvá számomra.

Nem lehet több embert ugyan ahhoz a kártyához rendelni, sőt csak a készítője látszik, ami ilyen kis csapatban nem fontos, de később gond lehet, hiszen így mindenkinek saját felelőssége a kártyák vezetése, amiből akár gond is lehet, ha valaki nem vezeti a kártyáit. A kártyák áthúzása után így el is veszíthetjük az adatokat, hogy ki mit csinált.

Egy pozitívum volt számomra, hogy a kártyákat szöveges formátumban is meg lehet írni, az alapján hozza létre, vagyis elég érteni a szintaxishoz és nem kell külön menüket használni, például listát létrehozhatunk [ ] az elemekkel ami könnyebb volt mint Trello-n

Traxler Bálint:

Én most használtam először Github Projectset; ugyanakkor könnyű kezelhetősége és letisztult működése miatt nagyon jó és effektív eszköznek találtam. Fejlesztés közben a haladást gyakran jeleztem a project boardunkon, (hogy lássák a többiek, hogy az én haladásom esetlegesen hogyan érinti őket,) illetve a megbeszéléseinknek megfelelően minden héten kialakítottam magamnak a megfelelő Noteokat.

Kriszt Benedek

Trello-t használtam korábban, a Github Projects utána egy kicsit fapadosnak érződött, de így is voltak a Trello-nél sokkal kényelmesebb funkciói, például a fent említett szöveges formázási lehetőség.

Az sem rossz, hogy a git repo mellett azonnal elérhető, bár mivel én viszonylag keveset látogattam a projektünk oldalát maga a Projects board is elég kényelmetlenül a mozgáskörömön kívül esett.

Megemlíteném még a Teams felületét, amin a kommunikációnkat kizárólagosan bonyolítottuk:

Kis csapatos együttműködésre szerintem kifejezetten kellemes használni és igényekhez szabható egy bizonyos szintig (pl.: dinamikus, közösen szerkeszthető Notebook fájl fül dokumentációhoz)

Megvalósítás összefoglalása:

Frontend:

1. Applikáció design (Traxler)

Egy alkalmazás kiadásakor szinte a legalapvetőbb feladat a fejlesztőknek, és elvárás a felhasználók részéről, hogy az alkalmazás rendelkezzen valamilyen designnal. Amit legelőször a felhasználó lát az alkalmazásból, az alkalmazás Launcher ikonja. Az ikont az Android Asset Studio segítségével készítettem el (https://romannurik.github.io/AndroidAssetStudio/), és az alábbi képen látható:

Hogy az alkalmazás témája konzisztens, és esztétikus legyen, a Material Components Light témáját alkalmaztam; kisebb változtatásokkal, hogy az alapvető betűtípus Roboto legyen.

2. Login feature (Traxler)

Az alkalmazás indításakor szükséges a felhasználónak autentikálnia magát, ez funkció ebben segít. A beviteli mezőket (email-cím, jelszó) TextInputLayout segítségével implementáltam a látványos, és felhasználóbarát megjelenés érdekében. Az állapot tárolását, a hálózati kommunikáció lebonyolítását az MVVM architektúra alapján történő implementálással oldottam meg. A hibakezelés Snackbar segítségével történik, amin a felhasználó információt kap, hogy a backendtől kapott információ és alkalmazás logika alapján milyen hiba történt; és eligazítja a felhasználót, hogy esetleg mit rontott el.

3. Registration feature (Traxler)

Ha a felhasználó nem rendelkezik még regisztrált fiókkal, itt létrehozhatja itt. A felhasználó email-címét és jelszavát TextInputLayout-ban adhatja meg, és egy RadioGroupban megadhatja a szerepét. Kétféle szerep létezik: feladó és szállító. A feladók csak csomagot tudnak feladni; míg a szállítók feladni és szállítani is tudnak. Ha nem megfelelőek a megadott adatok, vagy valamilyen hiba történik, a felhasználó itt is Snackbaron kap az alkalmazástól információt. Ez a feature is az MVVM architektúra szerint készült el.

4. Főmenü feature (Kriszt, Traxler)

Miután a felhasználó sikeresen bejelentkezett, egy főmenü fogadja, ahonnan könnyedén navigálhat az alkalmazás főbb funkcióihoz:

- kereshet munkákat (ha szállító)
- megtekintheti a munkáival kapcsolatos előzményeit
- megtekintheti, hogy jelenleg milyen jelenleg feladott, illetve elvállalt (ha szállító) munkái vannak
- Megtekintheti profilját
- Hirdethet meg új munkát
- Megtekintheti a jelenlegi munkáihoz kötődő lokációkat egy térképen
- Megtekintheti az admin statisztikákat (Ha rendelkezik admin jogosultsággal)

A főmenü a bejelentkezett felhasználó jogosultságának megfelelő módon működik; csak olyan funkciókat érhet el, ami szerepköréhez illik. A feature implementálása az MVVM architektúra szerint történt.

A főmenü elrendezését a Windows rendszerekben bevezetett csempés elrendezés inspirálta.

5. Profil feature (Kriszt, Traxler)

A felhasználó a profil nézeten megtekintheti a fiókjához tartozó email-címet, és az értékelését. A feature implementálása az MVVM architektúra szerint történt.

6. Job History feature (Traxler)

A felhasználó itt egy listában megtekintheti a korábbi munkáinak adatait. A küldő szerepkörrel regisztrált felhasználók megtekinthetik, hogy a múltban milyen csomagot adtak fel; míg a szállító felhasználók rendelkeznek az előbb felsorolt funkcionalitással kiegészülve azzal, hogy elérik azt is, hogy milyen csomagokat szállítottak. A listák különböző tabokon jelennek meg; ezt a funkcionalitást ViewPager2 segítségével készítettem el. Majd az egyes tabokon RecyclerView listákban jelennek meg az egyes munkák. Ha egy elemre klikkelünk, akkor az alkalmazás betölti a kiválasztott munka részletes adatait. A listaelemeken megjelenik a csomag azonosítója, mérete, állapota és lejárati időpontja.

7. Available jobs (Job board) feature (Traxler)

A szállító felhasználó itt egy listában megtekintheti az elérhető munkák adatait. A felhasználónak különböző tabokon jelennek meg az elérhető munkák méret szerint rendezve; ezt a funkcionalitást ViewPager2 segítségével készítettem el. Majd az egyes tabokon RecyclerView listákban jelennek meg az egyes munkák. Ha egy elemre klikkelünk, akkor az alkalmazás betölti a kiválasztott munka részletes adatait. A listaelemeken megjelenik a csomag azonosítója, mérete, állapota és lejárati időpontja. A képernyő alján látja a felhasználó egy ProgressBaron, hogy jelenleg a csomagtere milyen állapotban van; ezzel is segítve a felhasználó döntését.

8. Current jobs feature (Traxler)

A felhasználó itt egy listában megtekintheti a jelenlegi munkáinak adatait. A küldő szerepkörrel regisztrált felhasználók megtekinthetik, hogy jelenleg milyen feladott csomagjai vannak; míg a szállító felhasználók rendelkeznek az előbb felsorolt funkcionalitással kiegészülve azzal, hogy elérik azt is, hogy milyen csomagok kiszállítására vállalkoztak, amiket a megadott határidőig el kell szállítaniuk. A listák különböző tabokon jelennek meg; ezt a funkcionalitást ViewPager2 segítségével készítettem el. Majd az egyes tabokon RecyclerView listákban jelennek meg az egyes munkák. Ha egy elemre klikkelünk, akkor az alkalmazás betölti a kiválasztott munka részletes adatait. A listaelemeken megjelenik a csomag azonosítója, mérete, állapota és lejárati időpontja. A két tabon található két Floating Action Button is. Azon a tabon, ahol a feladott munkák jelennek meg, a Floating Action Button segítségével könnyedén arra képernyőre navigálhatunk, ahol meghirdethetünk új munkákat. A másik tabon a Floating Action Button pedig az elérhető munkák listájára navigálja a felhasználót.

9. Admin statisztikák feature (Traxler)

Ha a felhasználó rendelkezik admin jogosultsággal, akkor a főmenün megjelenik számára egy Floating Action Button, amire kattintva az admin statisztikák nézetre navigálhat. Itt két tabon (ami szintén ViewPager2 segítségével lett implementálva) talál az admin két különböző statisztikát. Az első oldalon megtekintheti, hogy a felhasználók szerepköreik szerint hogyan oszlanak meg; a második oldalon pedig megtekintheti, hogy a meghirdetett összes munka hogyan oszlik meg állapotuk alapján. A statisztikákat megjelenítő kördiagramok MPieChart könyvtár segítéségével készültek. (https://github.com/PhilJay/MPAndroidChart/)

10. Hálózati kommunikációról (Kriszt, Traxler)

Kliens oldalon a hálózati kommunikációt a Retrofit2 osztálykönyvtár segítségével készítettük el. A Retrofit2 nagyon jól támogatja a HTTP/HTTPS protokoll alapú kommunikációt, így a könyvtár segítségével könnyedén implementáltuk a hálózati kommunikációt a definiált REST API-n keresztül.

11. Új munka meghirdetése funkció (Kriszt)

Az alkalmazás összes felhasználója meghírdethet munkát, amit később mások elvállalhatnak. Ez a feature a jövőbeni munka adatainak megadására ad lehetőséget.
A dátumok felugró dátum választó párbeszédablakban állíthatóak be, a begépelt felvételi és szállítási címek megjelennek a képernyő részét képező térképen.

12. Meghirdetett munka részletek megtekintése funkció (Kriszt)

Az elérhető vagy teljesített munkák listájából választva ez az ablak nyílik meg megjelenítve a kapott azonosító alapján letöltött munka és a feladója adatait.

Annak függvényében, hogy a munka teljesítve van-e vagy még nincs és hogy a felhasználó adta-e fel vagy más a felhasználó különféle módokon léphet interakcióba a képernyővel.
Ha a saját hirdetése és le lett szállítva, akkor értékelheti a szállító munkáját, ha még nem tette meg eddig.

Amennyiben más hirdetése, elvállalhatja a munkát, illetve ezután változtathat az elvállalt munka státuszán.

Backend:

1. OpenAPI

Mivel egy csapatban dolgoztunk a kommunikáció nagyon fontos. Mivel a Frontend és Bacend-nek kommunikálnia kell egymással ezért ennek a pontos leírása elengedhetetlen. A REST API leírására az OpenAPI-t használtam. A Swagger nagyon hasznos eszköznek bizonyult: az api leírása egy yaml típusú fájlban található, ami alapján a Swagger generál egy weboldalt, vagy pdf dokumentumot. Ez nagyon kényelmes hiszen gyorsan, könnyen írható le az API, mégis könnyen olvasható volt a többi csapattársam számára. A végpontok megválasztásakor törekedtem arra, hogy minél specifikusabb végpontok legyenek, és lehetőleg kerüljem azokat a végpontokat, amik egy nagyon hosszú json-t adnak vissza, (persze ezáltal kivételek a listák, amik hosszabbak). A rövidség mellett a másik szempont a használat kényeleme volt. Törekedtem arra, hogy Frontend oldal ne kelljen egy dolog miatt 5 hívást elindítani, hogy meg legyenek az kívánt adatok, elég legyen az az 1 hívás is. A feladatban a kihívás számomra a rövidség és kényelem közötti egyensúly megtalálása volt.

2. Spring

A Backend fejlesztése Spring keretrendszer segítségével folyt Kotlin nyelven. Megvalósítás során kényelmes volt az implementáció a Kotlin nyelv elemei és Spring egyszerű használata miatt. A feladat főleg az elején volt nehéz, amikor az architektúra nem volt megfelelő, miután ez javívta lett, a feladatát könnyen ment, a Spring segítségével és az ORM(Hibernate) használatával az adatbázis elérése nagyon könnyű volt, csak az adat osztályokat kellett megírni és CrudRepository-n keresztül használhatóak is voltak már, az adatbázis elérését Spring innen keresztül már megoldotta.

3. Heroku

Backend kapcsán egy fontos kérdés volt az elérhetősége. Végül amelett döntöttem az a legegyszerűbb, ha valamilyen szolgáltatás futtatja azt és azon keresztül elérhető. Ezután esett a választás a Heroku-ra. Használata egyszerű volt, egy Heroku-n lévő git repository-ba kell a forráskódót feltölteni és onnantól a Heroku fordítja futtatja azt. Az alap Hobby csomagot használltuk aminek csak 1 hátránya volt, az is csak kényelmi szempontból a mi számunkra. A probléma az volt, hogy ebben a csomagban, ha nincs kérés az alkalmazás számára akkor azt gyorsan leállítja, így sokszor az első kérés Frontend-en nem működött, mert csak a kérés után indította a programot a Heroku.

4. PostgreSQL

Adatbázisként a PostgreSQL használtuk. Törekedtem arra, hogy a lehető legtöbb helyen legyen validáció, ezért az adattípusokat is így választottam meg az adatbázisban. Ebből sajnos a PostgreSQL sajátossága miatt egy gond született: az enumerációt is képes tárolni az adatbázis, így azt a típust használtam erre. Sajnos nem tudtam előre, hogy ebből gond lesz. A probléma a JDBC és a PostrgeSQL típusok között van. Az objektum, amit a JDBC alapértelmezetten átad az adatbázis felé, az nem olyan típusú objektum, amit az adatbázis elvár, hogy tudjon belőle enumerációt csinálni, konvertálni arra.

Sprint eltérítés:

Sprint eltérítés: csomagokhoz fotó társítás, vagy csomagfelvételi határidő

Csomag felvételi határidő implementálását választottuk:

Frontend:

Traxler Bálint:

A változtatás engem főleg a munka listázó featureben érintett. Átalakítottam ezekben a featureökben (History, Elérhető munkák, Jelenlegi munkák) a megjelenést és az alkalmazás logikát, hogy az eltérítésnek megfelelő módon jelenjenek meg az adatok. Az eltérítés érintette az Admin statisztikák featuret, ami szintén az én feladatom volt; a nézetet, ami a munkák jellegét mutatja az eltérítésnek megfelelően alakítottam ki.

Kriszt Benedek:

A változtatáshoz implementálnom kellett az új funkciót támogató adattagokat az alkalmazás adattároló osztályaiban, illetve megjeleníteni ezeket a lehetőségeket a munkák részleteit mutató képernyőkön, valamint a feladásnál megadhatóvá tenni ezeket.

Backend:

Sipos Áron Dávi:

A határidő implementálása nem okozott nagy gondot: a csomagnak új állapotot kellett felvenni és ezt az állapotot továbbítani a végpontokon. Mivel az állapotok enumerációként vannak implementálva így könnyű volt egy ujjat felvenni. A lejárati dátum frissítéséhez pedig Spring feladatütemező segítségével valósítottam meg, amit a @Scheduled annotációval érhető el. Így a sprint eltérítés nem okozott nagy változást se a meglévő modellen, se a kódban.

Ami nem valósult meg:

Frontend:

…

Backend:

Végül minden megvalósult.

Végső értékelés:

Frontend:

Traxler Bálint:

Nagyon élveztem a munkát a többi csapattársammal, úgy gondolom, hogy jól tudtunk csapatban dolgozni együtt. Nagyon élveztem magát a feladatot is, sok olyan dologgal tudtam a projekt keretein belül foglalkozni, ami már a tárgy felvétele előtt érdekelt (pl. MVVM architektúra). Az általam implementált featureöket igyekeztem a konzulens által említett témák és példakódok alapján kialakítani; így törekedtem a Single Activity Architecture kialakítására, illetve az MVVM architektúra követésére is.

Kriszt Bendek:

Élveztem a munkát a csapattal, sokkal kellemesebb és gördülékenyebb élmény volt, mint a korábbi tárgyak kereteiben végzett csoportmunka.
Szakmai oldalról nem a legkellemesebb élmény volt, mivel többször is olyan dolgokhoz kellett nyúljak, amit nem láttam át rendesen, vagy csak nehezen sikerült megérteni, de összevetve ezeket mindenképpen pozitívan jöttem ki a dologból.

A Navigation Components és a Single Activity Architecture pedig kifejezetten kellemes és kényelmesen alkalmazható eszköz.

Backend:

Sipos Áron Dávid

A feladat nekem tetszett. Sok olyan dologgal találkoztam így amivel lehet csak később találkoztam volna. A munka során sok jó tapaszlatot szereztem: Springben az architektúra felépítését láthattam, hogy működik. Lehetőségem volt "tesztelni" a teljesítményt "rossz"(isten osztályokkal, kód a controllerben) és "jobb" architektúrával is, amiben a szerepek jól el vannak osztva a Service-ek között. Ezek mellett megiskedtem a Herokuval is, amin keresztül a PostrgeSQL adatbázist is használtam. Összefoglalva örülök, hogy ezt a témát választottam, mivel egy kicsit beláthattam hogyan működik egy alkalmazás backend-je és egy jó csapat részeként dolgozhattam rajta.

Részvevők hozzájárulása:

Csapattag	Feladatkör	Hozzájárulás	Feladatok
Kriszt Benedek Miklós	Frontend, Mobilkliens	33.33%	• Egyszerű hírfeltöltő alkalmazás készítése • Új munka hozzáadását megvalósító funkció • Listából kiválasztott munka részleteinek megjelenítése • Kiválasztott munka elfogadása/visszamondása/felvétele/befejezése • Elvégzett munka a feladó által értékelhetővé tétele • Alkalmazás UI-jának megtervezése és felvázolása • Profil feature kezelőfelületének megvalósítása • Főmenü kezelőfelületének megvalósítása
Traxler Bálint	Frontend, Mobilkliens	33.34%	• Bemelegítésként egyszerű hírolvasó app alkészítése (Adatlekérdezés) • Use-case diagram elkészítése; Use-casek kifejtése • Login feature implementálása • Register feature implementálása • Munka történet feature implementálása (History) • Jelenleg felvett munkák feature implementálása • Jelenleg elérhető munkák feature implementálása • Admin statisztikák feature implementálása • Főmenü MVVM architektúra szerinti kialakítása (Kivéve View réteg megjelenése) • Profil feature MVVM architektúra szerinti kialakítása (Kivéve View réteg) • Applikáció design
Sipos Áron Dávid	Backend	33.33%	• Egyszerű hírolvasó alkamzáshoz végpontok készítése. • Backend elkészítése • Backend REST API dokumentáció elkészítése • Backend elérhetővé tétele Heroku segítségével.

Backend API Dokumentáció:

Freelancer Backend API

Overview

Untitled-repo

This is the OpenAPI documentation of our backend API.

Tags

user

Operations about Users

jobs

Operations about Jobs

admin

Operations administrators do

Paths

GET `/admin/statistics` Get statistics on the jobs and users

Admin user read statistics

Parameters

Type	Name	Description	Schema
query	**user-id** *required*	The User's ID	integer (int64)

Responses

Code	Description	Links
200	OK *Content* **application/json** **application/xml**	No Links
401	User does not have admin privilege	No Links

POST `/user/register` Register a new User

This can be done by anyone

Responses

Code	Description	Links
200	Successful registration	No Links
405	Invalid input	No Links
409	Email already in use	No Links
422	Email format is not correct	No Links

GET `/user/login` Log's User into the system

Parameters

Type	Name	Description	Schema
query	**password** *required*	The password for login in clear text	string
query	**email** *required*	The User's email for login	string

Responses

<table>
<tr><th>Code</th><th>Description</th><th>Links</th></tr>
<tr><td>200</td><td>successful login

Headers
<table><tr><th>Name</th><th>Description</th><th>Schema</th></tr><tr><td>X-API-Token</td><td>The User’s token for the API</td><td>string (JWT Token)</td></tr><tr><td>X-Expires-After</td><td>Date in UTC when token expires</td><td>string (date-time)</td></tr></table>
Content
application/json
The Users’s id

Example
<pre>1245678900</pre>
application/xml
The User’s id</td><td>No Links</td></tr>
<tr><td>400</td><td>Invalid format</td><td>No Links</td></tr>
<tr><td>409</td><td>Invalid email/password supplied</td><td>No Links</td></tr>
</table>

GET `/user/logout` Log’s out the User with the bearer token

Responses

Code	Description	Links
default	Successful logout	No Links

Type	Name	Scopes
apiKey	bearerAuth	

GET `/user/profile` Return the logged-in User's profile

Parameters

Type	Name	Description	Schema
query	**user-id** *required*	The User's ID	integer (int64)

Responses

Code	Description	Links
200	The matching User's cargo *Content* **application/json** **application/xml**	No Links

Type	Name	Scopes
apiKey	bearerAuth	

DELETE `/user/profile` Delete User

This can only be done by the logged in User.

Parameters

Type	Name	Description	Schema
query	**user-id** *required*	The User's ID	integer (int64)

Responses

Code	Description	Links
200	The API message *Content* **application/json** **application/xml**	No Links

Type	Name	Scopes
apiKey	bearerAuth	

PUT `/user/profile/update/cargo` Update the User's max cargo size

This can only be done by the logged in User.

Parameters

Type	Name	Description	Schema
query	**user-id** *required*	The User's ID	integer (int64)

Responses

Code	Description	Links
200	The API message *Content* **application/json** **application/xml**	No Links

Type	Name	Scopes
apiKey	bearerAuth	

PUT `/user/profile/update/can-deliver` Update the User's deliverer status

This can only be done by the logged in User.

Parameters

Type	Name	Description	Schema
query	**user-id** *required*	The User's ID	integer (int64)

Responses

Code	Description	Links
200	The API message *Content* **application/json** **application/xml**	No Links

Type	Name	Scopes
apiKey	bearerAuth	

PUT `/user/profile/update/email` Update the User's email

This can only be done by the logged in User.

Parameters

Type	Name	Description	Schema
query	**user-id** *required*	The User's ID	integer (int64)

Responses

Code	Description	Links
200	The API message *Content* **application/json** **application/xml**	No Links

Type	Name	Scopes
apiKey	bearerAuth	

PUT `/user/profile/update/password` Update the User's password

This can only be done by the logged in User.

Parameters

Type	Name	Description	Schema
query	**user-id** *required*	The User's ID	integer (int64)

Responses

Code	Description	Links
200	The API message *Content* **application/json** **application/xml**	No Links

Type	Name	Scopes
apiKey	bearerAuth	

GET `/user/{user-id}` Get User by User ID

Parameters

Type	Name	Description	Schema
path	**user-id** *required*	The user's ID	string

Responses

Code	Description	Links
200	Minimal information on the queried User *Content* **application/json** **application/xml**	No Links
400	Invalid User ID supplied	No Links
401	Access token is missing or invalid	No Links
404	User not found	No Links

Type	Name	Scopes
apiKey	bearerAuth	

GET `/user/{user-id}/jobs` Get all Jobs the User sent and delivered

Parameters

Type	Name	Description	Schema
query	**size** *optional*	The filter on the size of the packages	enum (small,medium,large)
path	**user-id** *required*	the User's ID	string
query	**status** *optional*	The filter on the status of the packages	enum (pending,accepted,pickedUp,delivered)

Responses

Code	Description	Links
200	The matching job array *Content* **application/json** **application/xml**	No Links
400	invalid format	No Links
401	Access token is missing or invalid	No Links
404	Jobs not found	No Links

Type	Name	Scopes
apiKey	bearerAuth	

GET `/user/{user-id}/jobs/sent` Get the Jobs the User sent

Parameters

Type	Name	Description	Schema
query	**size** *optional*	The filter on the size of the packages	enum (small,medium,large)
path	**user-id** *required*	the User s ID	string

Type	Name	Description	Schema
query	**status** *optional*	The filter on the status of the packages	enum (pending,accepted,pickedUp,delivered)

Responses

Code	Description	Links
200	The matching job array *Content* **application/json** **application/xml**	No Links
400	Invalid format	No Links
401	Access token is missing or invalid	No Links
404	Jobs not found	No Links

Type	Name	Scopes
apiKey	bearerAuth	

GET `/user/{user-id}/jobs/delivered` Get the Jobs the User delivered

Parameters

Type	Name	Description	Schema
query	**size** *optional*	The filter on the size of the packages	enum (small,medium,large)
path	**user-id** *required*	the User's ID	string
query	**status** *optional*	The filter on the status of the packages	enum (pending,accepted,pickedUp,delivered)

Responses

Code	Description	Links
200	The matching job array *Content* **application/json** **application/xml**	No Links
400	Invalid format	No Links
401	Access token is missing or invalid	No Links
404	Jobs not found	No Links

Type	Name	Scopes
apiKey	bearerAuth	

GET `/user/{user-id}/cargo` Get all information on the User's cargo

Parameters

Type	Name	Description	Schema
path	**user-id** *required*	the User's ID	string

Responses

Code	Description	Links
200	The matching user's cargo *Content* **application/json** **application/xml**	No Links
400	Invalid format	No Links
401	Access token is missing or invalid	No Links
404	User data not found	No Links

Type	Name	Scopes
apiKey	bearerAuth	

GET `/jobs` Lists available Jobs

Lists the available jobs

Parameters

Type	Name	Description	Schema
query	**size** *optional*	The filter on the size of the packages	enum (small,medium,large)

Responses

Code	Description	Links
200	The matching job array *Content* **application/json** **application/xml**	No Links
400	Invalid format	No Links
401	Access token is missing or invalid	No Links
404	No Jobs available	No Links

Type	Name	Scopes
apiKey	bearerAuth	

POST `/jobs` Posts a Job

Parameters

Type	Name	Description	Schema
query	**user-id** *required*	ID of the User that post's the Job	integer (int64)

Responses

Code	Description	Links
201	Created *Headers* **Name** / **Description** / **Schema** Location / / string (uri)	No Links
405	Invalid input	No Links

Type	Name	Scopes
apiKey	bearerAuth	

GET `/jobs/post/{job-id}` Get Job from Job ID

Returns a single Job

Parameters

Type	Name	Description	Schema
path	**job-id** *required*	ID of the Job	integer (int64)

Responses

Code	Description	Links
200	Information on the queried Job *Content* **application/json** **application/xml**	No Links
400	Invalid ID supplied	No Links
401	Access token is missing or invalid	No Links
404	Job not found	No Links

Type	Name	Scopes
apiKey	bearerAuth	

PUT `/jobs/post/{job-id}` Update a Job

Only the User who posted the Job can update it

Parameters

Type	Name	Description	Schema
path	**job-id** *required*	ID of the Job that needs to be updated	integer (int64)
query	**user-id** *required*	ID of the User who posted the Job	integer (int64)

Responses

Code	Description	Links
200	OK	No Links
401	Access token is missing or invalid	No Links
404	Job not found	No Links
405	Invalid input	No Links
409	You couldn’t modify the Job	No Links

Type	Name	Scopes
apiKey	bearerAuth	

DELETE `/jobs/post/{job-id}` Deletes a Job

Only the User who posted the Job can delete it

Parameters

Type	Name	Description	Schema
path	**job-id** *required*	Job ID to delete	integer (int64)

Responses

Code	Description	Links
400	Invalid id supplied	No Links

Code	Description	Links
401	Access token is missing or invalid	No Links
404	Job not found	No Links
409	You couldn’t delete the Job	No Links

Type	Name	Scopes
apiKey	bearerAuth	

POST `/jobs/accept/{job-id}` Accept a Job

The User who posted it can’t accept their own Jobs.

Parameters

Type	Name	Description	Schema
path	**job-id** *required*	Job ID to delete	integer (int64)
query	**user-id** *required*	The deliverer user’s id	integer (int64)

Responses

Code	Description	Links
400	Invalid id supplied	No Links
401	Access token is missing or invalid	No Links
404	Job not found	No Links
409	Couldn’t accept Job	No Links

Type	Name	Scopes
apiKey	bearerAuth	

DELETE `/jobs/accept/{job-id}` Abandon a Job

The User who currently accepted the Job can abandon it.

Parameters

Type	Name	Description	Schema
path	**job-id** *required*	Job ID to delete	integer (int64)
query	**user-id** *required*	The deliverer user's id	integer (int64)

Responses

Code	Description	Links
400	Invalid id supplied	No Links
401	Access token is missing or invalid	No Links
404	Job not found	No Links

Type	Name	Scopes
apiKey	bearerAuth	

PUT `/jobs/rate/{job-id}` Rate a Job

The User who posted the Job can rate it.

Parameters

Type	Name	Description	Schema
path	**job-id** *required*	Job ID to rate	integer (int64)
query	**user-id** *required*	The sender User's id	integer (int64)
query	**rating** *required*	The new rating of the job	integer (int64)

Responses

Code	Description	Links
200	Job rated	No Links
400	Invalid id supplied	No Links

Code	Description	Links
404	Job not found	No Links

Type	Name	Scopes
apiKey	bearerAuth	

PUT `/jobs/pickup/{job-id}` Pick up a Job

The User who accepted the Job can pick it up.

Parameters

Type	Name	Description	Schema
path	**job-id** *required*	Job's ID to pick up	integer (int64)
query	**user-id** *required*	The deliverer User's id	integer (int64)

Responses

Code	Description	Links
200	Job picked up	No Links
400	Invalid id supplied	No Links
404	Job not found	No Links

Type	Name	Scopes
apiKey	bearerAuth	

PUT `/jobs/deliver/{job-id}` Deliver a Job

The User who accepted and picked up the Job can deliver it.

Parameters

Type	Name	Description	Schema
path	**job-id** *required*	Job's ID to deliver	integer (int64)

Type	Name	Description	Schema
query	**user-id** *required*	The deliverer User's id	integer (int64)

Responses

Code	Description	Links
200	Job delivered	No Links
400	Invalid id supplied	No Links
404	Job not found	No Links

Type	Name	Scopes
apiKey	bearerAuth	

Components

Schemas

ApiResponse

Response of the backend

Properties

Name	Description	Schema
code *required*		integer (int32)
type *required*		string
message *optional*		string

User

A minimal information on a User

Properties

Name	Description	Schema
user-id *required*	The ID of the User	integer (int64)
email *required*	The email address of the User	string (email)
rating *optional*	The rating of the User *nullable* **Maximum**: 5 **Minimum**: 1	number (float)

UserCargo

Information on the User's cargo

Properties

Name	Description	Schema
cargoFreeSize *required*	The actual size of the cargo *nullable*	integer
cargoMaxSize *required*	The maximal size of the cargo *nullable*	integer

UserProfile

Detailed information on a User

Properties

Name	Description	Schema
user-id *required*	The ID of the User	integer (int64)
email *required*	The email address of the User	string (email)
rating *optional*	The rating of the User *nullable* **Maximum**: 5 **Minimum**: 1	number (float)

Name	Description	Schema
canDeliver *required*	The boolean value which says if the User can deliver or not	boolean
cargoFreeSize *optional*	The actual size of the cargo *nullable*	integer
cargoMaxSize *optional*	The maximal size of the cargo *nullable*	integer

UserUpdatePassword

Data for password change

Properties

Name	Description	Schema
user-id *required*	The ID of the User	integer (int64)
current-password *required*	The current password of the User	string (password)
new-password *required*	The new password of the User	string (password)

UserUpdateCanDeliver

Data for changing deliverer status

Properties

Name	Description	Schema
user-id *required*	The ID of the User	integer (int64)
password *required*	The password of the User	string (password)
can-deliver *required*	The new value for the canDeliver variable	boolean

UserUpdateEmail

Data for email change

Properties

Name	Description	Schema
user-id *required*	The ID of the User	integer (int64)
password *required*	The password of the User	string (password)
new-email *required*	The new email address of the User	string (email)

UserUpdateCargoSize

Data for cargo size change

Properties

Name	Description	Schema
user-id *required*	The ID of the User	integer (int64)
password *required*	The password of the User	string (password)
new-cargo-size *required*	The new cargo size of the User	integer (int64)

UserRegistration

Data on the new User

Properties

Name	Description	Schema
email *required*	The email address of the new User	string (email)
password *required*	The password of the new User	string (password)
canDeliver *optional*	The value determines if the new User can accept Jobs	boolean

Name	Description	Schema
cargoSize *optional*	The size of the new User's cargo *nullable*	integer

Job

Detailed information on a Job

Properties

Name	Description	Schema
job-id *required*	The ID of the Job	integer (int64)
name *required*	The name of the Job	string
sender-id *required*	The ID of the sender User	integer (int64)
status *required*	The current status of the Job	enum (pending,accepted,pickedUp,delivered,expired)
senderRating *optional*	The sender rating of the delivery *nullable* **Maximum**: 5 **Minimum**: 1	integer
size *required*	The size of the package	enum (small,medium,large)
payment *required*	The payment for the delivery	integer
jobIssuedDate *required*	The date the sender issued the Job	string (date-time)
deadline *required*	The deadline of the Job	string (date-time)

Name	Description	Schema
deliveryDate *optional*	The day the package was delivered *nullable*	string (date-time)
deliveryRoute *required*		Route

JobRegistration

Data on the new User

Properties

Name	Description	Schema
size *required*	The size of the package	enum (small,medium,large)
name *required*	The name of the Job	string
payment *required*	The payment for the delivery	integer
jobIssuedDate *required*	The date the sender issued the Job	string (date-time)
deadline *required*	The deadline of the Job	string (date-time)
startLocation *required*	The start location of the Route	string
destination *required*	The destination of the Route	string

Route

A Route for a specific Job

Properties

Name	Description	Schema
actualTime *optional*	The time the Route was completed *nullable*	integer (int64)
optimalTime *optional*	The optimal time for the Route *nullable*	integer (int64)
startLocation *required*	The start location of the Route	string
destination *required*	The destination of the Route	string

Statistics

Statistics on User’s and Job’s

Properties

Name	Description	Schema
allUsers *optional*	The number of all User’s	integer (int64)
allSenders *optional*	The number of all sender User’s	integer (int64)
allDeliverers *optional*	The number of all deliverer User’s	integer (int64)
allJobs *optional*	The number of all Job’s	integer (int64)
allJobsPending *optional*	The number of all pending Job’s	integer (int64)
allAccepted *optional*	The number of all accepted Job’s	integer (int64)
allJobsPickedUp *optional*	The number of all picked up Job’s	integer (int64)
allJobsDelivered *optional*	The number of all delivered Job’s	integer (int64)

Name	Description	Schema
allJobsExpired *optional*	The number of all expired Job's	integer (int64)

Responses

Responses

Code	Description	Links
ApiResponse	The API message *Content* **application/json** **application/xml**	No Links
UnauthorizedError	Access token is missing or invalid	No Links
User	Minimal information on the queried User *Content* **application/json** **application/xml**	No Links
UserCargo	The matching user's cargo *Content* **application/json** **application/xml**	No Links
UserProfile	The matching user's cargo *Content* **application/json** **application/xml**	No Links
Job	Information on the queried Job *Content* **application/json** **application/xml**	No Links

Code	Description	Links
JobArray	The matching job array *Content* **application/json** **application/xml**	No Links