Eötvös Loránd Tudományegyetem

Informatikai Kar

Programozáselmélet és Szoftvertechnológiai Tanszék

Projektmenedzsment eszköz agilis módszertanhoz

Témavezető:
dr. Tejfel Máté
egyetemi docens

Szerző:
Nagy Levente
programtervező informatikus BSc

Budapest, 2020

Az eredeti szakdolgozati / diplomamunka témabejelentő helye.

Tartalomjegyzék

1. fejezet

Bevezetés

A mai világban egyre szélesebb körben alkalmazott az agilis módszertan az informatikai megoldásokkal foglalkozó cégeknél. A jelen dolgozat keretein belül elkészült alkamazás (későbbiekben: *ScrumHelper*) az agilis módszertan szerint működő fejlesztői csapatok (azaz scrum-ok) segítésére született. Manapság sok, különböző ilyen eszköz elérhető, mind fizetős, mind nyílt-forráskódú változatban is. Ezek a projektek egyhamar hatalmasra nőnek, hogy minél több funkciót lássanak el, ezzel bonyolítva kezelésüket.

Ezen alkalmazás kisebb létszámú (például startup) cégek számára lehet előnyös elsősorban, de természetsen bármilyen scrum berendezkedésű fejlesztői csapatnak megfelelő. Az alkalmazást könnyen és egyértelműen lehet kezelni (részletesebb bemutatása felhasználók számára a 2. fejezetben olvasható).

A ScrumHelper a python programozási nyelv segítségével készült el, azon belül is a Django nyílt forráskódú keretrendszerrel, mely webes alkalmazások fejlesztésére szolgál, kihasználva a python nyelv adta sokoldalú lehetőségeket. A teljesség igénye nélkül pár példa ezen előnyök közül:

- rövid és átlátható kódbázis segíti a gyorsabb fejlesztést
- jól dokumentált keretrendszer [1]
- modulokra (applikációkra) bontott szoftver segíti az újra felhasználhatóságot

[1] `https:/docs.djangoproject.com/en/3.0/`

2. fejezet

Felhasználói dokumentáció

Ezen fejezet a felhasználó részletes tájékoztatására szolgál. Az alfejezetek az alkalmazás szükséges előfeltételeit, telepítési és használati információkat tartalmaznak. A program technikai részletekbe menő dokumentációját a 3 . fejezet (Fejlesztői dokumentáció) tartalmazza.

2.1. Röviden az agilis módszertanról és a "scrum"-ról

Mi az agilis módszertan? A szoftverfejlesztési módszerek egy csoportja, ahol a követelmények és megoldások szoros együttműködésén keresztül fejlődnek az önszerveződő és multifonkcionális csapatok. Ez elősegíti a korai szállítást, folytonos továbbfejlesztést és bátorít a változásokra adható gyors és rugalmas válaszokra. [2]

Mi a "scrum"? Az agilis módszertanon belül sokféle irányzat van, ezek egyike a scrum. A scrum középpontjában a kis létszámú, önszerveződő agilis csapatok állnak. Itt, szemben a többi agilis ágazattal, nincsenek általánosan megszabott ,egész rendszert egybefogó szállítási és frissítési időpontok, hanem az aktuális fejlesztések **Sprintekbe** rendeződnek és ezek lejárta után történik az átadás. Ez közvetlenebb visszajelzést biztosít a szállító és a megrendelő között. A scrum szerkezeti felépítése a következő: van egy scrum master, aki egybe fogja

[2]Részletes leírás megtalálható: `https://www.agilealliance.org/agile101/`

a csapat működését, feladata a scrum "menedzselése" gyakorlatilag. Mellette a csapat tartalmaz természetesen fejlesztőket és tesztelőket. A scrum létszáma nincsen hivatalosan meghatározva, de ajánlatos 8-10 főnél nem nagyobbra nőnie, különben veszít hatékonyságából. A csapatok minden nap egy meghatározott időpontban tartanak "stand up"-okat, amelyeken minden tag beszámol feladatairól, haldásáról, így mindenki a csapaton belül képbe kerülhet a **Sprint** aktuális állásáról.

Pár fontosabb fogalom, amelyek a későbbiekben külön fejezetben részletezve lesznek:

- Projekt 2.4.3. fejezet
- Epic, User stories, Task, Issue 2.4.4. fejezet
- Kanban 2.4.6. fejezet

2.2. Rendszerkövetelmények

Minimum követelmények: A *ScumHelper* egy webes alkalmazás. Ez azt jelenti, hogy a felhasználónak csak egy böngészőre van szüksége a számítógépén (például: Google Chrome, Mozilla Firefox, Safari, stb.) és természetesn internet elérésre ahhoz, hogy futtatni tudja a szoftvert. Utóbbi elengedhetetlen, ugyanis csak bejelentkezve lehetséges használni. Internet kapcsolat nélkül a számítógép csak a gyorsírótárában mentett oldalakat tudja megnyitni, de módosításokat nem tudunk végrehajtani a betöltött oldalon és új lapot sem tudunk megnyitni az alkalmazáson belül. Nincsen megkötés operációs rendszer tekintetében, tehát minden, napjainkban használatos rendszeren (Linux, Windows, Mac OS) egyaránt használható.

Ajánlott követelmények: A jobb felhasználói élmény érdekében érdemes legalább 1280x720 felbontású kijelzőn használni és a korábban már felsorolt, jelenleg leggyorsabbnak és legbiztonságosabbnak számító böngészővel megnyitni : Chrome, Mozilla Firefox, Microsoft Edge, valamint érdemes szélessávú internet eléréssel rendelkezni.

2.3. Telepítés

A felhasználói oldalról nem igényel telepítést. Az eléréshez a szerver elérési URL-jére van szükség, illetve egy felhasználó igénylésére az aktuális rendszergazdától. Érdemes az új felhasználóba való belépés után megváltoztatni jelszavunkat biztonsági okokból.

Rendszergazdai oldalról ha még nem rendelkezik felhasználóval, akkor a fejlesztői dokumentációban (3. fejezet) található információ a rendszergazda felhasználó létrehozásának lépéseiről. Ha futtatni szeretné az alkamazásszervert lokálisan, a saját számítógépen, vagy egy kihelyezett szervergépen, akkor azon telepíteni kell a következő programokat: *python* (3.5 vagy később verzió) [3], *django és django-extension* python csomagok[4], valamint egy adatbázis kezelő szoftvert (PostgreSQL, Oracle, MySQL, MariaDB, SQLite).

[3] `https://www.python.org/downloads/`

[4] `https://docs.djangoproject.com/en/3.0/intro/install/`

2.4. Az alkalmazás felépítése

2.4.1. Bejelentkezés, felhasználók

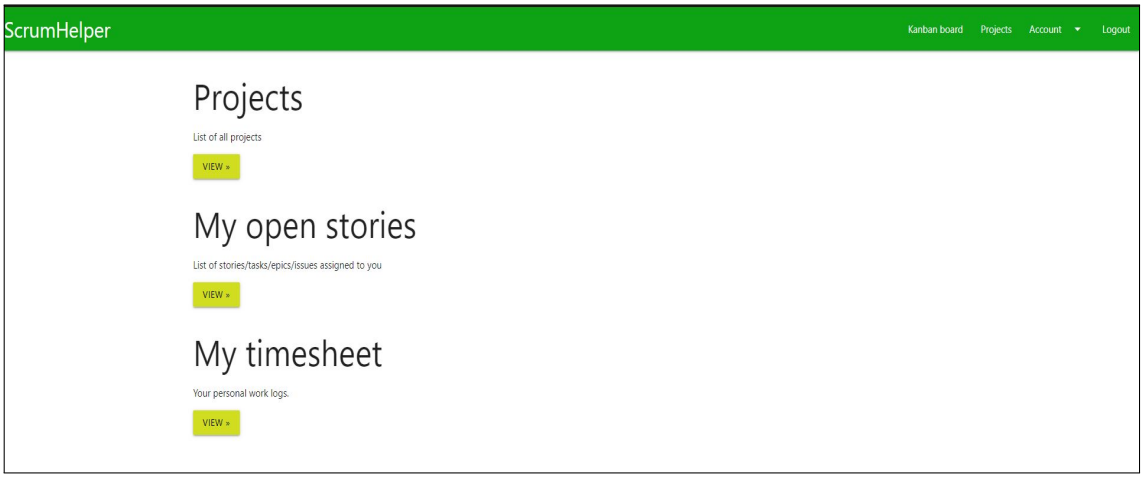
Az alkalmazásban az első képernyő, amely fogadja a felhasználót az a login oldal. A ScrumHelper-t csak sikeres bejelentkezés után lehetséges használni. Ha nem rendelkezik felhasználóval, akkor keresse meg a rendszergazdát, és igényeljen egyet. Az egyes felhasználói fiókok egy-egy jogosultsági csoporthoz kötöttek.

A felhasználói csoportok és jogosultságaik:

Csoport	**Jogosultságok**
Fejlesztő	projekt/epic/story/task/issue létrehozás/szerkeztés, komment létrehozás/törlés, saját munkaidő napló létrehozása/törlése, Dokumentum feltöltés (ha projekt tulajdonos, akkor törlés), jelszóváltás
Tesztelő	projekt/epic/story/task/issue létrehozás/szerkeztés, komment létrehozás/törlés, saját munkaidő napló létrehozása/törlése, Dokumentum feltöltés (ha projekt tulajdonos, akkor törlés), jelszóváltás
Scrum master	projekt/epic/story/task/issue létrehozás/szerkeztés/-törlés, komment létrehozás/törlés, saját munkaidő napló létrehozása/törlése/egész scrum könyvelés megjelenítése, Dokumentum feltöltés (ha projekt tulajdonos, akkor törlés), jelszóváltás
Projekt menedzser	projekt/epic/story/task/issue létrehozás/szerkeztés/-törlés, komment létrehozás/törlés, saját munkaidő napló létrehozása/törlése/egész scrum könyvelés megjelenítése, Dokumentum feltöltés (ha projekt tulajdonos, akkor törlés, jelszóváltás)
Rendszergazda	minden jogosultsággal rendelkezik, létre is hozhat új csoportokat, kezelheti azok jogosultságait, illetve a felhasználókat is tudja szerkezteni/törölni

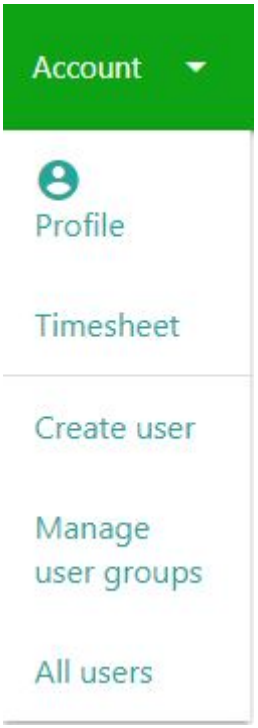
2.1. táblázat. Jogosultsági csoportok és jogosultságaik

2.4.2. Főoldal



2.1. ábra. Főoldal

A 2.1. ábrán látható a főoldal kinézete. A navigációs sávban az alklamazás neve, **Kanban board, Projects, Account** és **Logout** mezőket olvashatjuk. A **Kanban board** elnavigál a scrum Kanban táblájára, ahol az összes felvett feladatot láthatjuk egyben. A **Projects** menüponttal juthatunk el az adatbázisban szereplő projektek listájához. Az **Account** menüpont egy legördülő menü, amelyben a 2.2. ábrán látható tartalom jelenik meg.



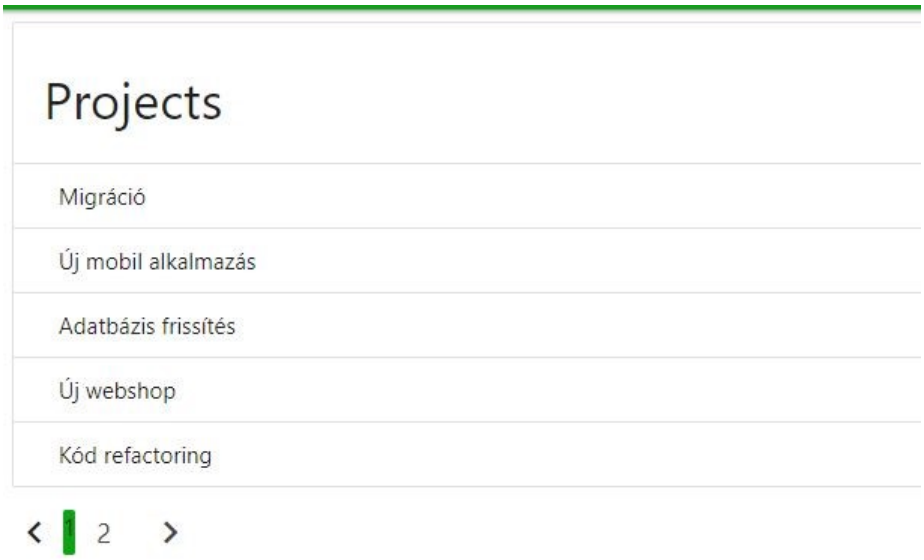
2.2. ábra. Account legördülő menü

Az elválasztó vonal alatti almenük: **Create user, Manage user groups, All users** csak a rendszergazda jogosultságú felhasználók számára láthatóak. A **Logout** menüpont értelemszerűen kijelentkezteti a felhasználót és a bejelentkező oldalra ugrik.

A főoldal törzsében található 3 opció: **Projects** - a **Projects** menüponttal megegyezően a projektek listájára navigál, **My open stories** - a bejelentkezett felhasználóhoz rendelt story-k/task-ok/issue-k tekinthetőek meg és a profil adatok, **My timesheet** - a felhasználó munkaidő könyvelési oldalára kalauzol.

2.4.3. Projektek

A projektek listájához a **Projects** menüpontból, illetve a főoldalról tudunk eljutni. Az oldalon egyszerre 5 projekt jelenik meg, a többit a 2.3.ábra alján látható oldal léptetéssel érhetjük el. A projektek létrehozási dátum szerint a legújabbtól haladva a legrégebbi felé jelennek meg. A projekt oldalak esetében a navigációs sávban is megjelenik egy "Create project" feliratú gomb, mellyel egyből a létrehozás oldalra juthatunk.



2.3. ábra. Projektek listája

Az egyes projektekre kattintva eljutunk a projektek saját oldalára. Itt megtalálhatjuk az alapvető információkat a projektről, a projekthez feltöltött dokumentumokat, valamint a projekthez tartozó összes feladat (epic,story,task,issue) felosolását

egy görgethető listában. A jobb alsó sarokban található egy menü, mely az egyes feladat fajták saját oldalain is megtalálhatóak, természetesen részben eltérő menüpontokkal. A projektek esetében ez tartalmazza a szerkeztés, epic, story, task, issue létrehozásait, illetve a megfelelő jogosultsággal rendelkezőknek a törlés opciót. (2.4. ábra).

2.4. ábra. almenü, mely különböző opciókat tartalmaz projekt, vagy feladat típustól függően

2.4.4. Epic, Story, Task, Issue

(a) Epic (b) Story (c) Task (d) Issue

2.5. ábra. Az egyes feladat típusok ikonjai

Epic: Egy projekten belül levő feladatok gyűjteménye. Mivel projektek több sprinten át élhetnek, ezért rövidebb időszakokban az összefüggő story-k és task-ok összefogására érdemes ezt használni. Létrehozni az adott projekt oldalán, a korábban említett almenüből lehet (2.4. ábra).

Story: Felhasználói feladat. Ha valamilyen jól körülhatárolt fejlesztés van a projekten belül, akkor érdemes ezt használni annak leírására. Létrehozatalkor még nem kötelező hozzárendelni felhasználót, azt lehet később is a story szerkeztésével. Létrehozni a projekt oldalán levő almenüből (2.4. ábra) lehetséges, vagy egy adott epic saját oldalán levő almenüből (story ikonjára kattintva), illetve a navigációs sávban megjelenő **Create story** gombra kattintva. Ez olyankor elérhető, amikor egy meglévő story oldalán vagyunk éppen.

Task: Szintén felhasználói feladat. A task-ot a story-val ellentétben kisebb feladatok leírására érdemes használni. Például, ha valaki dokumentációt ír, vagy meetingekre jár, vagy csak valami apró fejlesztésről van szó, akkor érdemes azt "taskosítani". Létrehozni a projektek, illetve epic-ek és story-k oldalán lehetséges az almenüből (2.4. ábra) a task ikonjára kattintva.

Issue: Az issue kicsit eltér a story-tól és a task-tól. Az issue valamilyen jellegű hiba leírására használható. Például, ha egy elkészült fejlesztésben a teszetelő hibát talál, vagy ha már a leszállított build-ben derül ki valamiféle "bug". Létrehozni a projekt almenüjében lehetséges (2.4. ábra).

A story, task és issue hármas státusszal is rendelkezik. Az aktuális státuszt minden olyan oldalon láthatjuk, ahol valamilyen felsorolásban szerepel a 3 közül bármely feladat típus. A státuszt az adott feladat saját oldalán lehet a jobb felső sarokban látható gombbal állítani, melyen mindig az a státusz olvasható, amely az éppen aktuálisat követi.

2.6. ábra. Példa egy story-ra egy felsorolásban: látható a neve, a projekt kódja, ikonja és aktuális státusza

2.4.5. Egy példa fejlesztési ciklus

Az egyes fogalmak gyorsabb és könnyebb megértése érdekében egy példa fejlesztési folyamat a szemléltetés céljából:

1. A scrum masterhez megérkezik a fejlesztési igény (projekt) a megrendelőtől.
2. Felvesz egy projektet a *ScrumHelper*-ben. Feltölti hozzá az igényhez kapott dokumentumokat, amelyek segítik majd a tervezők munkáját. **Projects** menüpont a navigációs sávban (vagy a Főoldalon) -> **Create project** sárga gomb a navigációs sávban -> A formula helyes kitöltése és **Submit** gomb -> Az új projekt oldalán a **File** gombra kattitnva kiválasztja a fájlt, amit fel akar tölteni, majd rányom a gémkapocs ikonnal rendelkező gombra.

3. Ezt követően kiosztja az igény megtervezését egy tervezőnek (architect). Ehhez task-ot készít, ahol összeírja a teendőket röviden. Az új projekt oldalán a jobb alsó menüből, a **Task** ikonjára kattint (kék alapon fehér könyvjelző) -> Kitölti a **Task** kreáló formulát, az *assignee* mezőben a tervező felhasználóját adja meg -> rányom a **Submit** gombra

4. A tervező megírja az igény specifikációját arról, hogyan lehetne ezt az adott rendszerben implementálni (High Level Solution Design).

5. Utóbbit feltölti a projekt többi dokumentumai közé, majd elkezdi lebontani feladatcsoportokra (epic) és feladatokra (story). Hasonlóan a 2. lépésben leírtakhoz, a projekt oldalán feltölti a fájlt -> Az almenüben az **Epic** ikonjára kattint (lila alapon fehér négyzet), vagy a **Story** ikonjára kattint (zöld alapon fehér diagramm) -> kitölti a formulákat és a **Submit** gombra nyom -> létrejönnek az új feladatok a projekthez.

6. A fejlesztő ezután válogathat a story-k között, melyiket szeretné megcsinálni. Amelyiket kiválasztja azt magához rendeli a story szerkeztői oldalán és IN PROGRESS-be rakja a státuszt. **Projects** menüpontból, avagy a főoldalról elnavigál a projekt oldalára, vagy a **Kanban board** menüponttal megtekinti a kanban táblán az aktuálsi feladatokat -> kiválaszt egyet a listákból és az adott feladat saját oldalán az almenüben a szerkeztés gombra kattint (sárga alapon fehér toll) -> az *assignee* mezőben kiválasztja a saját felhasználóját -> **Submit** gomb. -> a feladat (Story) saját oldalán a státusz gombbal IN PROGRESS státuszba helyezi azt.

7. Amint végzett a fejlesztéssel, TESTING státuszba állítja. Egy tesztelő megkeresi (például a Kanban boardon) és magához rendeli. A fejelsztő a feladat oldalán rányom a **TESTING** feliratú gombra -> a tesztelő a **Kanban board** menüpontbeli kanban táblán megkeresi a TESTING státuszú feladatok között -> rákattintva eljut annak saját oldalára -> a jobbalsó menüben rányom a szerkeztés ikonjára (sárga alapon fehér toll) és az *assignee* mezőt a saját felhasználójára állítja.

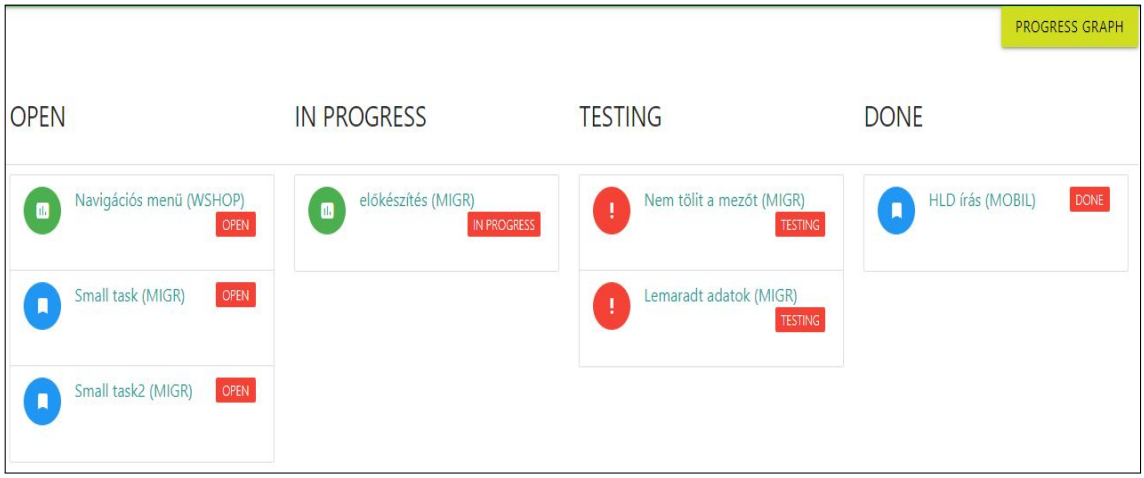
8. Ha tesztelés közben hibát talál, akkor felvesz a projekthez egy issue-t amit az eredeti fejlesztőhöz (vagy aki javítani tudja) rendeli. Érintett projekt saját oldala -> almenüben issue ikonja (piros alapon fehér felkiáltójel) -> Formula megfelelő kitöltése (helyes hibatípus kiválasztása, fejlesztőhöz rendelés) és **Submit** gomb.

9. Amikor már nincsen probléma a story-val, akkor DONE státuszba rakja. **Story** saját oldala -> **DONE** feliratú gomb

10. Egy feladat életciklusának utolsó állomása a CLOSED, amelyre a sprint végén ajánlatos állítani. Ha valami okból újra foglalkozni kell vele, akkor lehetőség van a REOPEN opcióval újra nyitottá tenni. **Story** saját oldala -> **CLOSE** feliratú gomb

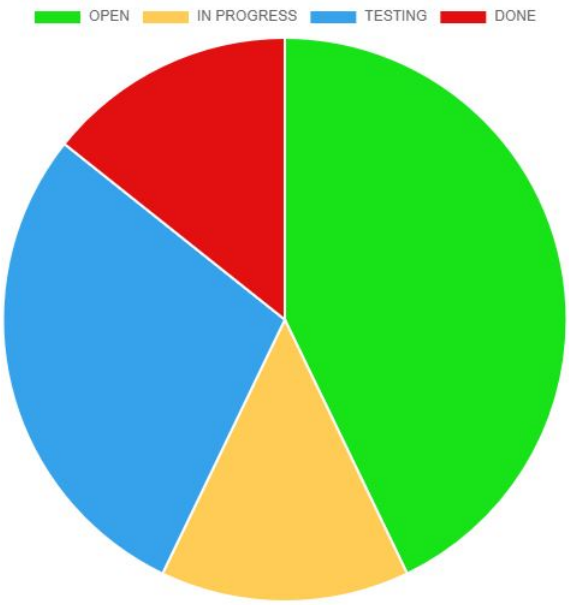
2.4.6. Kanban tábla

A *kanban* szintén egy egységesen használt fogalom (avagy eszköz) a scrum módszertanban. A kanban board - azaz "tábla" - arra szolgál, hogy egybe gyűjtse a scrum összes felvett feladat típusát. A *ScrumHelper*-ben ez a funkció a navigációs sávban a **Kanban board** menüpontra kattintva érhető el. Itt négy státusz alapján négy oszlopra osztva láthatóak az aktuális (DONE státuszban csak a 30 napon belül módosultak maradnak megjelenítve) feladatok: OPEN (nyitott), IN PROGRESS (folyamatban), TESTING (tesztelés alatt), DONE (kész).



2.7. ábra. Kanban tábla

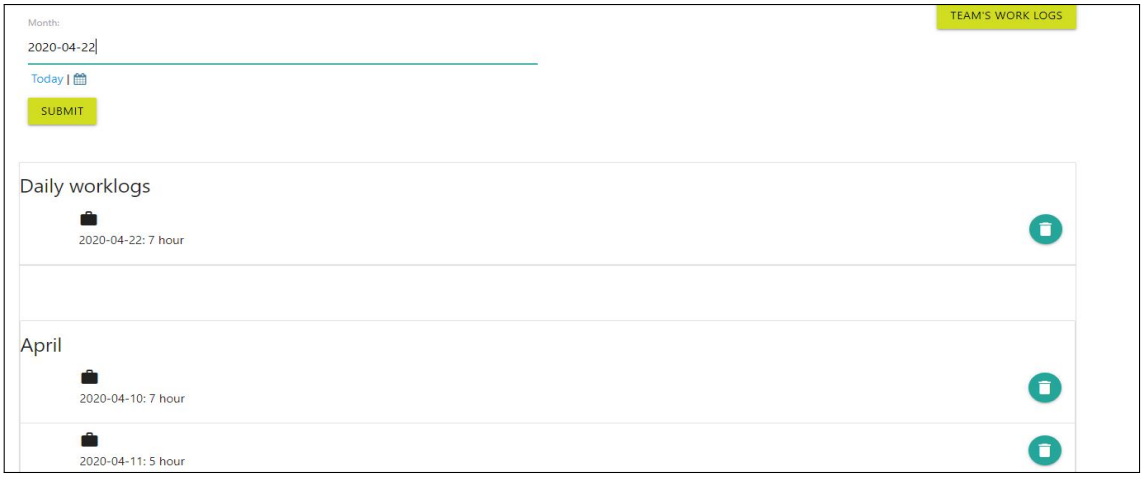
A jobb felső sarokban látható gomb (2.7. ábra "Progress graph") megnyomásával egy körgrafikon segít az aktuális feladatok állásának nyomonkövetésében. Az alábbi ábrán egy példa:



2.8. ábra. Körgrafikon az aktuális feladatok jelen státuszáról

2.4.7. Munkanapló

Az alkalmazás lehetőséget biztosít az egyes feldatokkal eltöltött munkaidő számontartására is. Ehhez el kell navigálniuk az adott feladat (story, task, issue) oldalára és a már korábban többször is említett (2.4. ábra), jobb alsó sarokban található almenüben kiválasztani a munkaidő könyvelése opciót ("Add worklog", ikon: kék körben fehér aktatáska). Ez eljuttatja a felhasználót a munkanapló bejegyzés létrehozó oldalára. Itt a dátum mezőt megfelelő formátumban kitöltve, avagy a naptár ikont használva meg kell adni a munkanapot. Alatta pedig megadni a feladattal töltött munkaórák számát, amely 1 és 8 közé kell essen. Ettől eltérő bementre figyelmeztet az alkalmazás, hogy rossz adatot adott meg. A navigációs sáv **Account** menüpont legördülő menüjében (2.2. ábra) a **Timesheet** opcióra kattintva, avagy a főoldalról a **Timesheet** pontra kattintva tekinthető meg a saját profilunkhoz rögzített összes munkanapló bejegyzés napi és havi bontásban (2.9. ábra).



2.9. ábra. Személyes munkanapló

Kiválasztható a dátum napi pontossággal, akár kézzel beírva a megfelelő formátumban, akár a naptár ikonra kattintva a felugró naptárból. A jobb felső sarokban látható egy "TEAM'S WORK LOGS" feliratú gomb. Erre kattintva megtekinthető az egész scrum havi munkaidő könyvelése (2.10. ábra). Az egyéni naplóhoz hasonlóan, egy megadott dátum alapján az adott hónapra megjeleníti az egyes felhasználók által lekönyvelt össz óraszámát. A jogosultságoknál említettek alapján, csak a scrum masterek, a projektmenedzserek és a rendszergazdák látják ténylegesen minden csapattag óraszámát. A többi csoport csak a sajátját látja összegezve. Törölni

bejegyzést minden felhasználó csak maga tud, nem lehet másét eltávolítani. Ezt a személyes munkanapló oldalon lehet megtenni az egyes bejegyzés melletti kuka ikonra kattintva.

2.10. ábra. Scrum felhasználóinak össz munkaórája az adott hónapban (amit a scrum master lát)

3. fejezet

Fejlesztői dokumentáció

Ez a fejezet fejlesztőknek nyújt segítséget a ScrumHelper feltérképezésében. Az alfejezetek kifejtik az alkalmazás felépítését, részletesen leírják a különböző rétegeinek (Adatbázis-Szerver-Nézet) osztályait és függvényeit. A vizuális segítséghez különböző diagrammokat is tartalmaz (osztály-, csomag-, felhasználói esetek diagram). A fejezet végén egy tesztforgatókönyv ad részletes leírást a teszt esetek és azok elvárt eredményeiről.

3.1. Konfiguráció, fejlesztői környezet

A futtatáshoz szükséges előkövetelmények a felhasználói dokumentáció 2.3. alfejezetében olvashatók. A dolgozat keretein belül csak a lokális szerveren való konfigurálás és futtatás lesz részletezve. A különböző szervergépeken való futtatáshoz részletesebb információt a hivatalos Django dokumentációban lehet találni. Utóbbi eset külön konfigurációt igényel a wsgi.py, illetve asgi.py file-ok és környezeti változók megfelelő beállításának segítségével [5] [6].

Ahhoz, hogy lokálisan futtatni tudjuk a szervert, először is a ScrumHelper/ScrumHelper/setting.py file-ban kell beállítanunk a DATABSES változót. A pontos beállításai eltérőek a különböző adatbázisok esetében, ehhez részletes segítséget nyújt a hivatalos Django dokumentáció. Alább egy PostgreSQL adatbázis konfigurációja látható:

[5] wsgi szerver: `https://docs.djangoproject.com/en/3.0/howto/deployment/wsgi/`

[6] asgi szerver: `https://docs.djangoproject.com/en/3.0/howto/deployment/asgi/`

```
1
2 DATABASES = {
3     'default': {
4         'ENGINE': 'django.db.backends.postgresql',
5         'NAME': 'database name',
6         'USER': 'database user',
7         'PASSWORD': 'password',
8         'HOST': '127.0.0.1',
9         'PORT': '5432',
10    }
11 }
```

3.1. forráskód. Konfiguráció PostgreSQL adatbázis használatához

Ha ez megfelelően van beállítva, akkor a ScrumHelper fő mappába navigálva kell lefuttatni a" python manage.py migrate" parancsot minden első futtatásnál a szükséges adatbázis struktúra kialakításához(illetve ha fejlesztés során olyasmi változik, amely érinti az adatbázis struktúrát, akkor a makemigrations-t is le kell a migrate parancs előtt futtatni). Ezután a "python manage.py runserver" elindítja a lokális szervert a localhost:8000-es portján. Superuser-t, azaz minden jogosultsággal rendelkező felhasználót a szerver futtatása nélkül lehet generálni: "python manage.py createsuperuser" paranccsal, megadva a felhasználónevet és jelszavat utána.

A TIMEZONE változóban adható meg az időzóna. Az egyes projektekhez feltöltött fájlokat a MEDIA_ROOT konfigurációs változóbeli elérési úton elhelyezkedő mappába menti (ez alapból a ScrumHelper/media könyvtárra mutat). Ugyanezen az elven kezeli a statikus fájlokat is a keretrendszer, ennek a környezeti változója a STATIC_ROOT.

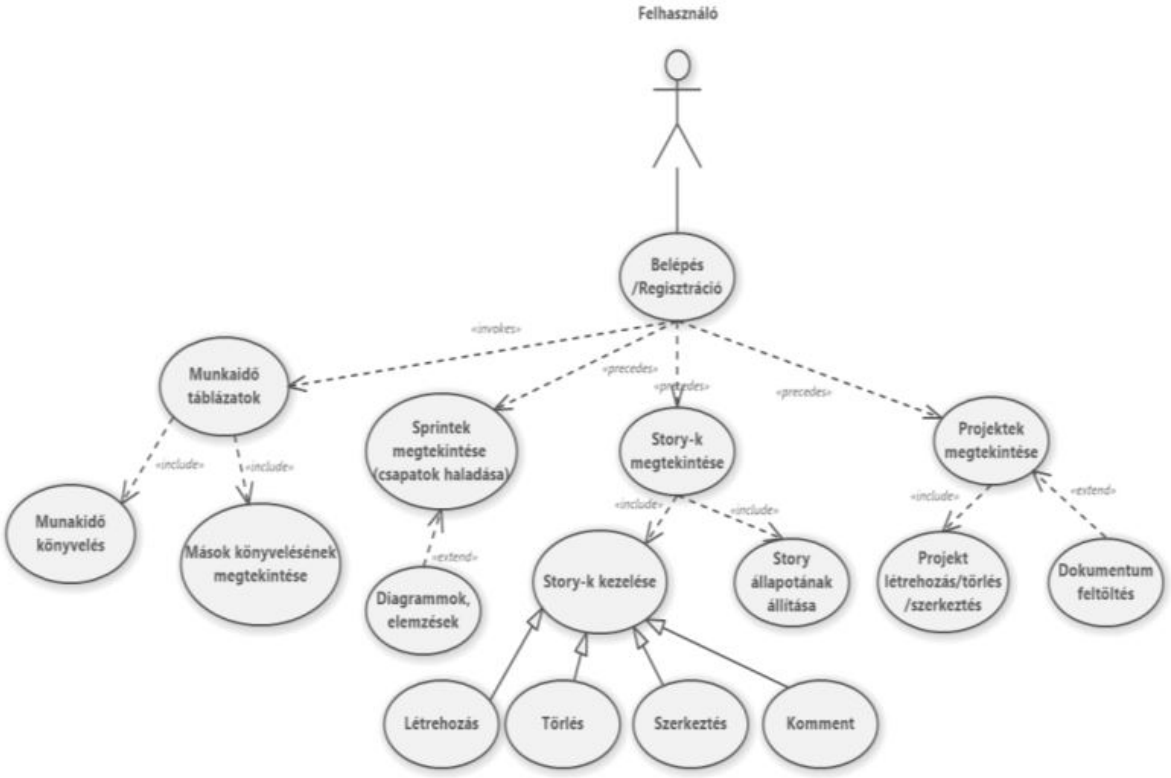
Fontos, hogy mielőtt az alkalmazás valós, produkciós használatba kerülne szükséges a DEBUG változót False-ra, azaz hamisra állítani (biztonásgi okokból). Utóbbi azért is fontos, mert így a szerver nem küld debug információkat a kliensre.

A Django nem rendelkezik saját fejlesztői környezettel, így bármely python fejlesztésre alkalmas IDE használható. Például a Visual Studio Code rendelkezik minden bővítménnyel, amely szükséges lehet egy Django alkalmazás futtatásához, valamint egyéb "kényelmet" segítő funkciókkal is (szintaxis ellenőrzés/kiemelés Pythonhoz/HTML-hez/CSS-hez/JavaScript-hez és a Django template-khez). A

szükséges szoftverek (2.3) megléte mellett akár egy egyszerű szövegszerkeztő alkalmazás is elegendő (de célszerűbb valamilyen integrált fejlesztői környezetet használni).

3.2. Funkcionális terv

Az 3.1. ábrán egy felhasználói eset diagram mutatja be az alkalmazás funkcióit:

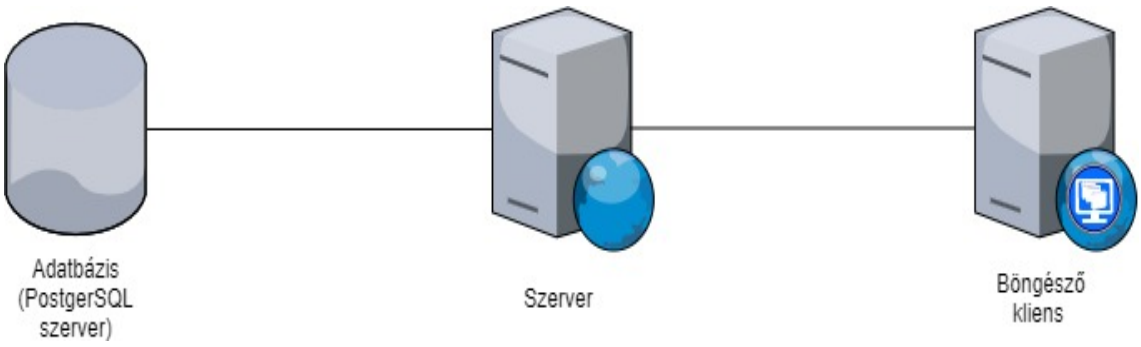


3.1. ábra. Felhasználói eset diagram

3.3. Struktúrális felépítés

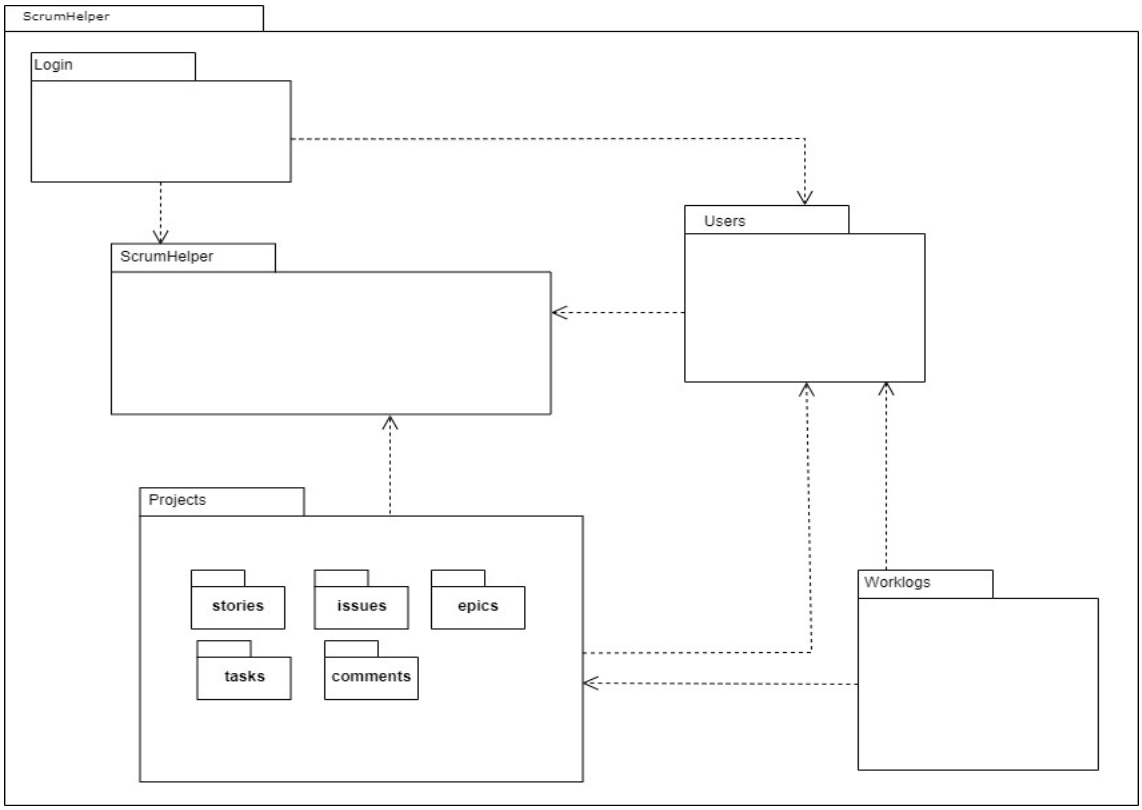
A ScrumHelper egy MVC (Model-View-Controller) alkalmazás, bár a hivatalos Django dokumentációban az MTV (Model-Template-View) kifejezést használják, mivel ezt gondolják pontosabb leírásának. Tekintve, hogy Djangoban írodott az alkalamzás, ezért ebben a dokumentációban is utóbbi analógia az irányadó. Ez azt jelenti, hogy három rétegből tevődik össze: egy Model rétegből, amely az adatbázist hivatott reprezentálni, egy View rétegből, amely az adatot reprezentálja (ez alatt azt értjük, amit az adatbázisből kigyűjtött, nem feltétlenül a felhasználó számára megjelenített adatot), valamint a különböző template-ek renderelését is végzi, to-

vábbá egy Template rétegből, mely definiálja, hogyan legyen az adat megjelenítve a felhasználó számára.



3.2. ábra. Az alkalamzás rétegei: egy adatbázis, egy szerver, ami kinyeri belőle az adatot és továbbítja a nézetnek, valamint a nézet amit a böngészőben láthatunk.

A django alkalmazások úgynevezett applikációkból állnak, amelyek a moduláris felépítést segítik elő. Ezek gyakorlatilag a python nyelvből ismert csomagokkal -package-ekkel- egyeznek meg. A 3.3. ábra szemléletesebben bemutatja ezen applikációkat (modulokat) és azok kapcsolatait:



3.3. ábra. Csomagdiagramm

- Ahogy az ábrán is látható, a fő csomag a ScrumHelper. Ez fogja össze működésben az alkalamzást.
- A legnagyobb applikáció a "Projects", ugyanis ez tartalmazza egyben a "Stories", "Tasks", "Issues", "Comments", "Epics" applikációkat is.
- A felhasználók kezelésével foglalkozó csomag a "Users" és részben a "Login". Utóbbi a ki- és bejelentkeztetésre, valamint a felhasználók regisztrálására és szerkeztésére szolgál.
- Egy kisebb alkalmazás, a "Worklogs" adja a modell szintű reprezentációját a munkaidő naplónak.

Ezek implementáció, részletei a későbbi alfejezetekben olvashatóak.

3.4. Adatbázis réteg

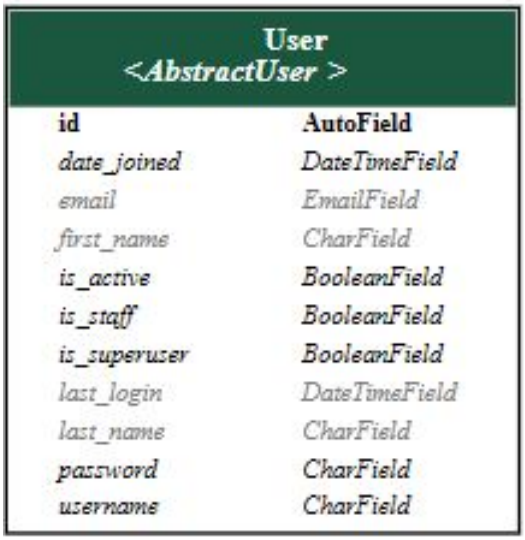
A django keretrendszernek köszönhetően érdemben nem számít, hogy milyen adatbázissal dolgozunk (legalábbis az adatbázis kezelés szempontjából). A ScrumHelper fejlesztése során a nyílt-forráskódú PostgreSQL-re esett a választás, de a python modellek és az adatbázis lekérdezések függetlenek attól, hogy milyen adatbázist használunk. Ha csak egy kis, egyszerű alkalmazás megvalósítása a cél akkor érdemesebb például SQLite adatbázist használni, tekintve, hogy az csak egy lokális fájlban tárolja az adatokat, könnyebb menedzselni. A PostgreSQL (a MySQL, Oracle, MariaDB mellett) jobb döntésnek bizonyul nagyobb méretű alkalmazások esetén.

Az egyes adatbázis táblákat (entitásokat) egy-egy modell reprezentál a forráskódban. Ezek a django.db.models.Model osztályból vannak származtatva. Az egyes modellek attribútumai egy-egy oszlopot reprezentálnak a táblában. A django dokumentációban[7] részletes leírás található arról, mely mezőtípusokhoz milyen paramétereket lehet megadni ahhoz, hogy minél jobban testreszabhassuk a mező tulajdonságait. Van lehetőség modell szintű metódusokat is definiálni ugyanúgy, mint a python osztályoknál (hiszen ezek is python osztályok), de ezek nem lesznek jelen

[7]django modell típusosztályok: `https://docs.djangoproject.com/en/3.0/ref/models/fields/`

az adatbázisban. A modell szinten jellemzően csak egyszerűbb metódusokat szokás definiálni, mint például az *__str__()* felüldefiniálása a felsőbb rétegek segítségéül, vagy az *__init__* inicializációs függényt. Érdemes azonban kerülni az ilyen megoldásokat, hogy minél jobban elszeparálhatóak legyenek az egyes rétegek. A következő alfejezetekben részletes információk találhatóak az egyes modellekről.

3.4.1. Felhasználó kezelés: Users, Login modellek



3.4. ábra. User modell diagrammja

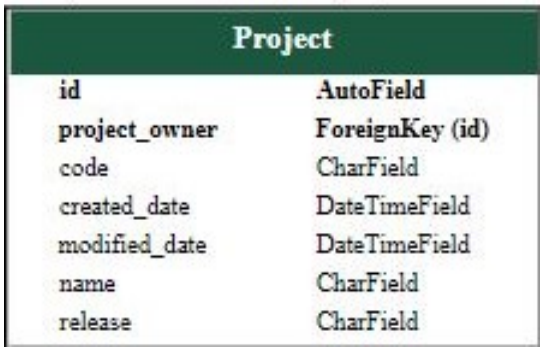
A **django.contrib.auth.models.User** modell reprezentál egy-egy felhasználót. Ez magában a keretrendszerben megtalálható. Elég sokoldalú, de van lehetőség kiegészíteni, esetlegesen helyettesíteni saját megvalósítással is. Kiegészítésre példa a **Profile** modell osztály a users applikációban. Egy **OneToOneField** segítségével és szignálokkal (*create_user_profile* és *save_user_profile*) tudjuk az eredeti **Users** osztályhoz kötni. A szignálok azért szükségesek, mert így tud a modell reagálni az eredeti modell esetleges változásaira (létrehozás, módosítás, törlés). A mezők:

- *id*: mezőazonosító (alapvetően generált)
- *date_joined*: beregisztrálás dátuma
- *email*: email cím
- *first_name*: keresztnév

- *is_ active*: nem zárolt-e a felhasználó (azaz inaktív)
- *is_ staff*: adminisztrátor-e
- *is_ superuser*: Minden jogosultsággal rendelkező felhasználó-e
- *last_ login*: utolsó bejelentkezés dátuma
- *last_ name*: vezeték név
- *password*: jelszó (hashelve, azaz kódolva)
- *username*: felhasználónév

A **Login** modul nem rendelkezik külön adatbázis reprezentációval, mivel a **User** modelljét használja föl.

3.4.2. Project modell



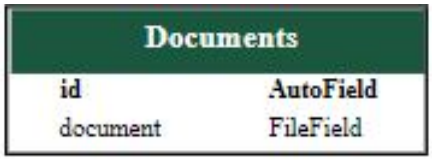
3.5. ábra. Project modell diagrammja

A **Projects** applikáció saját adatbázis modellje a **Project**:

- *id*: mezőazonosító (alapvetően generált)
- *project_ owner*: a projektet létrehozó felhasználó (idegen kulcs a **Users** táblára)
- *code*: a projekt 6 karakter hosszú kódja (egyedi kell legyen)
- *created_ date*: a létrehozás dátuma
- *modified_ date*: az utolsó módosítás dátuma

- *name*: a projekt neve (maximum 50 karakter hosszú)
- *release*: a Release neve (maximum 10 karakter hosszú)

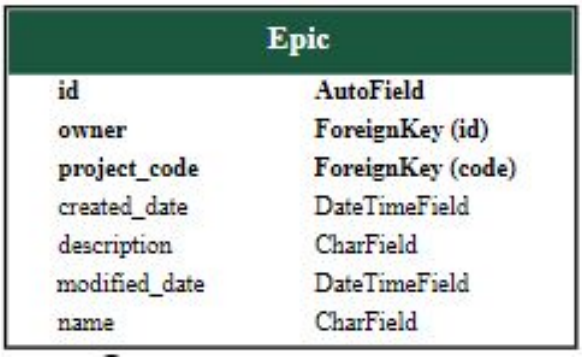
Rendelekezik egy *documents* attribútummal is, mely egy **ManyToManyField** típusú mező, azaz több idegen kulcs kapcsolatot fog egybe (erre a célra a django automatikusan létrehoz egy táblát, amelyben benne lesznek ezek az összekapcsolások) a **Documents** modell táblájának *id* mezőjével. Ennek seígtségével kapcsolódik egy adott projekthez több dokumentum is.



3.6. ábra. Documents modell diagrammja

A **Documents** modell rendelkezik egy *document* attribútummal, mely **FileField** típusú. Ez a típusosztály reprezentálja Djangoban a fájl mezőket az adatbázisban. A fájl relatív elérési útját menti le az adatbázisba. Jelen alkalmazásban a *documents/* mappát használja, de ez is konfigurálható a *projects.models.project_directory_path(instance, filename)* függvény segítségével. A visszatérési értékben (amely egy string) lehet megadni az elérési utat.

3.4.3. Epic modell



3.7. ábra. Epic modell diagrammja

Az **Epic** a *project_code* mezővel kapcsolódik a **Project** tábla *code* mezőjéhez. Egy projekthez több több epic is tartozhat, de egy epic csak egy projekthez kapcso-

lódhat. Ha a projekt törlődik, vagy a létrehozó felhasználó, akkor az összes epic is amelyek hozzá tartoznak. A modell mezői:

- *id*: mezőazonosító (alapvetően generált)
- *owner*: az epic-et létrehozó felhasználó (idegen kulcs a **Profile** táblára)
- *project_code*: a projekt 6 karakter hosszú kódja (idegen kulcs, a **Project** tábla *code* mezőjére mutat)
- *created_date*: a létrehozás dátuma
- *modified_date*: az utolsó módosítás dátuma
- *name*: az epic neve (maximum 50 karakter hosszú)
- *description*: egy leírás az epic-ről (maximum 250 karakter hosszú)

3.4.4. Story modell

UserStory	
id	**AutoField**
assignee	**ForeignKey (id)**
epic	**ForeignKey (id)**
owner	**ForeignKey (id)**
project_code	**ForeignKey (code)**
acceptance	CharField
created_date	DateTimeField
description	CharField
importance	CharField
modified_date	DateTimeField
name	CharField
state	CharField

3.8. ábra. Story modell diagrammja

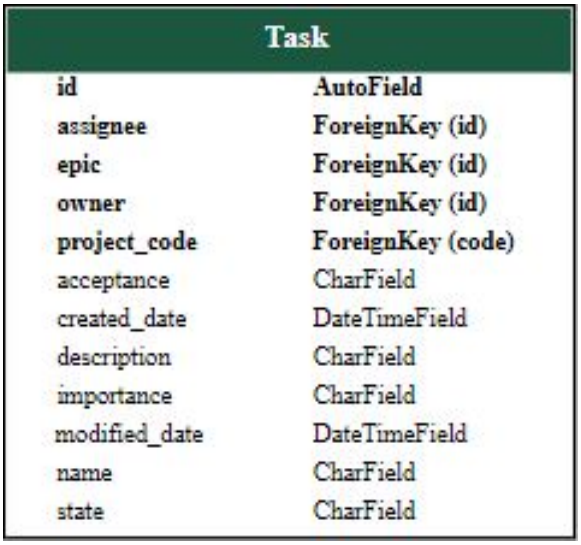
A **UserStory** több táblához is rendelkezik idegen kulccsal: kapcsolódik egyrészt a projekthez, amely alatt létrehozták, kapcsolódik továbbá a felhasználóhoz, aki létrehozta, illetve akihez rendelve van (opcionális), valamint kapcsolódik egy epic-hez (ez is opcionális). Egy story csak addig létezik, amíg a projekt is, amely alatt

létehozták, avagy ki nem törlik. Ha epic-hez van rendelve és töröljük, akkor csak kinullázódik azon mezője. Abban az esetben, ha a felhasználót töröljük, aki létrehozta a story-t, akkor is törlődik. A modell mezői:

- *id*: mezőazonosító (alapvetően generált)
- *assignee*: idegen kulcs a hozzárendelt felhasználóra (lehet üres, **User** tábla)
- *epic*: idegen kulcs az **Epic** táblára (lehet üres)
- *owner*: az story-t létrehozó felhasználó (idegen kulcs a **Profile** táblára)
- *project_ code*: a projekt 6 karakter hosszú kódja (idegen kulcs, a **Project** tábla *code* mezőjére mutat)
- *acceptance*: feladattal kapcsolatos elvárások leírása, amelyeknek teljesülnie kell (például teszt során, maximum 50 karakter)
- *created_ date*: a létrehozás dátuma
- *modified_ date*: az utolsó módosítás dátuma
- *name*: a story neve (maximum 50 karakter hosszú)
- *description*: egy leírás a story-ról (maximum 250 karakter hosszú)
- *importance*: fontosság (Low, Medium, High)
- *state*: aktuális állapot (OPEN, IN PROGRESS, TESTING, DONE, CLOSED)

Rendelkezik egy *comment* mezővel is, amely több kommentet is össze kapcsol egy **UserStory**-val (**ManyToMany** reláció, külön kapcsolati táblával). Ugyanilyen kapcsolatban áll a **Worklog** táblával is a *work_ log* mezőn keresztül.

3.4.5. Task modell



3.9. ábra. Task modell diagrammja

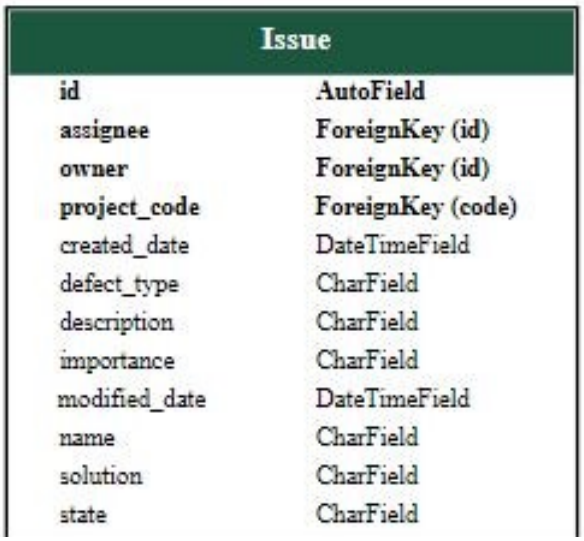
A **Task** modell felépítése szinte megegyezik a **UserStory**-éval. Ugyanúgy egy projekthez, esetlegesen egy epic-hez és a létrehozó felhasználóhoz kapcsolódik (és opcionálisan akihez hozzá van rendelve). Az eltérés a státuszaiban van: csak OPEN (nyitott), DONE(kész) és CLOSED(lezárt) lehet. A mezői:

- *id*: mezőazonosító (alapvetően generált)
- *assignee*: idegen kulcs a hozzárendelt felhasználóra (lehet üres, **User** tábla)
- *epic*: idegen kulcs az **Epic** táblára (lehet üres)
- *owner*: az task-ot létrehozó felhasználó (idegen kulcs a **Profile** táblára)
- *project_ code*: a projekt 6 karakter hosszú kódja (idegen kulcs, a **Project** tábla *code* mezőjére mutat)
- *acceptance*: feladattal kapcsolatos elvárások leírása, amelyeknek teljesülnie kell (például teszt során, maximum 50 karakter)
- *created_ date*: a létrehozás dátuma
- *modified_ date*: az utolsó módosítás dátuma
- *name*: a task neve (maximum 50 karakter hosszú)

- *description*: egy leírás a task-ról (maximum 250 karakter hosszú)
- *importance*: fontosság (Low, Medium, High)
- *state*: aktuális állapot (OPEN, DONE, CLOSED)

Rendelkezik egy *comment* mezővel is, amely több kommentet is össze kapcsol egy **Task**-kal (**ManyToMany** reláció, külön kapcsolati táblával). Ugyanilyen kapcsolatban áll a **Worklog** táblával is a *work_log* mezőn keresztül.

3.4.6. Issue modell



3.10. ábra. Issue modell diagrammja

Az **Issue** modellje már több mezőben is eltér a **Task** és **UserStory**-tól. A plusz mezők a korábbiakhoz képest, a *defect_type* és *solution*. Előbbi a hiba típusát hívatott reprezentálni, utóbbi pedig egy rövid leírás a megoldásról. Ez is kapcsolódik egy projekthez és a létrehozójához, valamint ahhoz akihez hozzárendelik (opcionális).

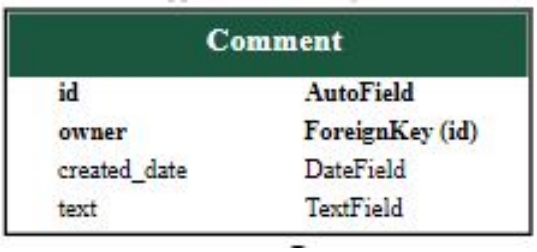
- *id*: mezőazonosító (alapvetően generált)
- *assignee*: idegen kulcs a hozzárendelt felhasználóra (lehet üres, **User** tábla)
- *owner*: az issue-ot létrehozó felhasználó (idegen kulcs a **Profile** táblára)
- *project_code*: a projekt 6 karakter hosszú kódja (idegen kulcs, a **Project** tábla *code* mezőjére mutat)

- *created_ date*: a létrehozás dátuma
- *defect_ type*: a hiba típusa (BUG, SYSTEM DEFECT - rendszer szintű, SPECIFICATION ISSUE - tervezési hiba, DEVELOPMENT ISSUE - fejlesztési hiba)
- *modified_ date*: az utolsó módosítás dátuma
- *name*: az epic neve (maximum 50 karakter hosszú)
- *description*: egy leírás az issue-ról (maximum 250 karakter hosszú)
- *importance*: fontosság (Low, Medium, High)
- *soultion*: a megoldás rövid leírása (maximum 150 karakter)
- *state*: aktuális állapot (OPEN, IN PROGRESS, TESTING, DONE, CLOSED)

Rendelkezik egy *comment* mezővel is, amely több kommentet is össze kapcsol egy **Issue**-val (**ManyToMany** reláció, külön kapcsolati táblával). Ugyanilyen kapcsolatban áll a **Worklog** táblával is a *work_ log* mezőn keresztül.

3.4.7. Comment modell

A **Projects** applikáción belül található a **Comments** modul is. Egy-egy **UserStory, Task, Issue** rendelkezhet kommentekkel különböző felhasználóktól. Ezekhez az egyes modellekben van idegen kulcsos mező, amely mutat egy-egy kommentre (**Many-To-Many** kapcsolat).



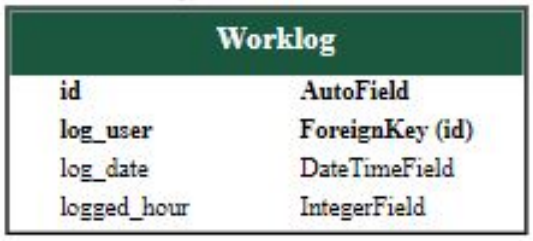
3.11. ábra. Comment modell diagrammja

Mezői:

- *id*: mezőazonosító (alapvetően generált)

- *owner*: a kommentelő felhasználó (idegen kulcs a **User** táblára)
- *created_ date*: a létrehozás dátuma
- *text*: a komment szövege (maximum 250 karakter hosszú)

3.4.8. Worklog modell



3.12. ábra. Worklog modell diagrammja

A munkaidő napló adatbázis reprezentációjául a **Worklog** modell szolgál. Munkaidő napló bejegyzést **UserStory-hoz, Task-hoz és Issue-hoz** tudunk létrehozni. Ezekhez az egyes modellek saját *work_ log* mezőjével kapcsolódik. A mezői:

- *id*: mezőazonosító (alapvetően generált)
- *log_ user*: a létrehozó felhasználója (idegen kulcs a **User** táblára)
- *log_ date*: a létrehozás dátuma
- *logged_ hour*: az órák száma ([0..8] intervallumba kell essen)

3.5. Szerver réteg

A szerver réteg jelen alkalmazásban a szolgáltatások (services) és nézetek (views) adják. Egy-egy applikáció a következő modulokból (gyakorlatilag fájlokból épül fel):

- *admin.py*: A Django biztosít egy admin felületet az alkalmazás adatbázis szintű kezelésére. Egy adott modell akkor szerkeszthető az admin oldalon, ha beregisztráljuk ebben a fájlban.

- *apps.py*: Az egyes applikációk nevét tudjuk itt megadni, amivel tudunk rá hivatkozni a többi alkalmazásból (például: "projects.stories"), valamint egyéb alapkonfigurációkat az applikációról.

- *migrations*: Ez egy almappa, amiben alap és későbbi (fájl nevében időponttal jelölt) adatbázis migrációk találhatók. Ezek a modellek változásaikor az adatbázis reprezentációjukon módosítanak annak megfelelően.

- *templates*: A template-eket, azaz a HTML fájlokat tartalamzza, amik a böngészőben megjelenítendő Nézet rétehet adják. Lehetőség van a fő mappában (jelen esetben *ScrumHelper*) létrehozni egyetlen tempaltes mappát és abban alkalamzásonként almappát, amelybe helyezzük a fájlokat, de az újrafelhasználhatóság érdekében jobb megoldás minden ilyen fájlt és almappát az applikáció saját mappájában elhelyezni.

- *constants.py*: nem kötelező fájl. Applikáció szintű konstansok tárolására használható.

- *forms.py*: Szintén nem kötelező fájl. Ha használunk form-okat (például regisztrációra), akkor érdemes ebbe a fájlba elhelyezni ezeket.

- *services.py*: A szolgáltatások helye. Ez sem kötelező, de ha az alkalamzáshoz például egy API-t (Application Programming Interface) akarunk fejelszteni, akkor helyesebb analógia a view-k helyett service-eket használni.

- *tests.py*: Az egyes applikációk egységteszt modulja.

- *views.py*: A view-k (nézetek) modulja. Ezek jelentik a közvetlen kapcsolatot az adatbázis/szolgáltatás réteg és a template-ek (a megjelenítés) között. A view függvények mindig kapnak egy *request* paramétert bemenetként, amiben a hívás adatai vannak (például form esetén az adatok, vagy fájlok, stb.).

- *urls.py*: Az applikációk szolgáltatásainak és view-inak az elérési útját tartalmazza (azaz a végpontokat, URL címeket).

A következő alfejezetekben az egyes applikációk szerver rétegébe tartozó részei lesznek részletezve (szolgáltatások, nézetek, végpontok, formulák).

3.5.1. Login applikáció

A **Login** applikáció a bejelentkeztetésért felelős. Alapvetően a keretrendszerben implementált bejelentkeztetést használja, illetve azt bővíti ki.

- **Forms:**

  SignUpForm: A regisztrációs formula. **UserCreationForm**-ból van származtatva, amely a *django.contrib.auth.forms* modulban van implementálva. A formulának hét mezője van: *username, first_ name, last_ name, email, password1, password2, group*. Vannak megkötések az egyes mezőkre. A *first_ name* csak 30 karakter hosszú lehet, nem kötelező megadni. Ugyanezek igazak a *last_ name* mezőre is. Az *email* mező egy maximum 254 karakter hosszú, email formátumú bemenetet fogad csak el. A *password1* és *password2* mezők jelszó mezők, amiknek egyezniük kell a formula véglegesítésekor. A *group* egy olyan mezőtípus (**ChoiceField**, amely közvetlenül az adatbázisból kérdezi le a választási lehetőségeket (az *auth_ groups* táblából).

  ChangePassword: A jelszóváltoztatáshoz használt formula. Ugyanazokkal a mezőkkel rendelkezik, mint a regisztrációs formula.

- **Nézetek (views):**

  CustomLoginView: Erre azért van szükség, mert itt tudjuk megadni a *redirect_ filed_ name* attribútummal, hogy hova navigáljon bejelenkezés után (jelen esetben: users/index.html).

  index: Az index oldal, azaz a "login/index.html" tartalmát jeleníti meg. **Végpontja:** "/" avagy "/login".

  logout: Meghívja a *django.contrib.auth.login* függvényt és visszanavigál a *login* oldalra. **Végpontja:** "/logout/".

  signup: A regisztrációs formula segítségével biztosítja a felahsználó létrehozását. A "POST" HTTP üzenetet kap bemenetként, akkor ellenőrzi, hogy helyes-e a formula, majd lementi a felahsználót és hozzárendeli a megfelelő csoporthoz. Ezután visszajuttat az aktuálisan bejelentkezett felhasználó

index oldalára. "GET" üzenet esetén csak egy üres forumlát biztosít kitöltésre a *registration/signup.html* számára. **Végpontja:** "registration/signup/".

edit_user: Az egyes felhasználók szerkeztésére szolgál. Ugyanazt a formulát ahsznála, mint a regisztráció, csak itt "GET" üzenet esetén is a az ismert adatokkal kitöltve adja át a tempalte-nek. **Végpontja:** "edit_user/<int:user_id>/", azaz a bemenő paramétere egy felhasználói azonosító.

change_pw: A jelszóváltoztatást valósítja meg. Ehhez szintén egy user_id-t vár bementként és a regisztrációs formulát használja. Azért van szükség mégis külön kezelni, mert nem midne felhasználónak van jogosultsága elérni a szerkeztést. **Végpontja:** "change_pw/<int:user_id>".

3.5.2. Projects applikáció

A **Projects** applikáció a projektek megvalósítása. Ennek almappái a **Stories, Issues, Comments, Tasks, Epics** applikációk, de ezek külön fejezetkben leszek részletezve. A **Comments** applikáció lényegében csak a kommentek adatbázis reprezentációjára szolgál, saját szolgáltatásokkal és nézettel nem rendelkezik, így ez nem lesz részletezve ebben a fejezetben.

- **Forms:**

  CreateProjectForm: A projekt létrehozásához használatos form. Mezői: *name, code, release*, azaz a projekt neve, a projekt 6 karakter hosszú kódja és a release kódja.

  CreateStoryForm: A **Story** létrehozás form-ja. Mezői: *name, project_code, assignee, description, importance, epic*. A *project_code, assignee, epic* mezők **ModelChoiceField** típusúak, tehát egy megadott adathalmazból kínálnak fel opciókat. Az *epic* és *assignee* mezők kiválasztása nem kötelező.

  CommentForm: A kommentek létrehozásához használt form. Csak *text* mezője van a komment szövegének lementéséhez.

CreateEpicForm: Az **Epic** létrehozó formulája. Mezői: *name, project_ code, description*, azaz a neve, a projekt kódja, amelyhez tartozik és egy leírás.

CreateWorklogform: A **Worklog**, azaz munkanapló bejegyzés létrehozásához használt form. Mezők: *log_ date, logged_ hour*. A feladattal végzett munkaidő és maga a munkanap.

CreateTaskForm: A **Task** létrehozás form-ja. Mezői: *name, project_ code, assignee, description, importance, epic*. A *project_ code, assignee, epic* mezők **ModelChoiceField** típusúak, tehát egy megadott adathalmazból kínál fel opciókat. Az *epic* és *assignee* mezők kiválasztása nem kötelező.

CreateIssueForm: Az **Issue** létrehozás form-ja. Mezői: *name, project_ code, assignee, description, importance, epic, defect_ type,solution*. A *project_ code, assignee, epic* mezők **ModelChoiceField** típusúak, tehát egy megadott adathalmazból kínál fel opciókat. Az *epic* és *assignee* mezők kiválasztása nem kötelező. A *defect_ type* mező is egy **ChoiceField**, tehát adott lehetőségekből lehet választani: BUG, SYSTEM DEFECT, DEVELOPMENT, SPECIFICATION ISSUE. Utóbbi értékek egy konstansban találhatóak a *contants.py* modulban, az *ISSUE_ CHOICES* változóban.

- **Szolgáltatások (services):**

get_issues_for_project: Bemeneti paramétere: *project_ id*. A megkapott projekt azonosítóhoz lekérdezi magát a projektet, a hozzátartozó storykat, task-okat, issue-kat, epic-eket és feltöltött dokumentumokat. Ezeket egy *context* nevű, dictionary típusú változóba gyűjti, ami a visszatérési értéke a függvénynek.

delete_project: Bemeneti paraméter: *project_ id*. A megkapott projekt azonosító alapján megkeresi az adatbázisban a projektet és kitörli.

- **Nézetek (views):**

index: Lekérdezi az adatbázisban található összes projektet. A *django.core.paginator* modul beli **Paginator** segítségével oldalakra osztja az eredményeket (jelen beállítás szerint ötösével). Ennek megoldása a

3.2. kódrészletben látható. Az így rendezett projekt listát hozzárendeli a "projects/index.html"-hez. **Végpont:** "projects/".

detail: Bemenő paraméter: *request, project_id.* Utóbbival meghívja a *get_issues_for_project* szolgáltatást és az ebből megszerzett *context* változót hozzárendeli a "projects/details.html" oldalhoz. **Végpont**: "projects/<int:project_id>/".

project_new: Új projekt létrehozását kezelő view. "GET" üzenet esetén biztosítja az üres **CreateProjectForm**-ot, "POST" üzenet esetén ellenőrzi és lementi az adatokat. **Végpont:** "projects/new/".

project_edit: A bemeneti paraméterként megkapott *project_id* alapján megkeresi az adatbázisban a projektet, majd egy **CreateProjectForm**-ot ad vissza az ismert adatokkal kitöltve "GET" üzenet esetén. "POST" üzenet esetén frissíti az adatbázisban a projekt adatait. **Végpont:** "projects/<int:project_id>/edit".

delete: A paraméterül kapott *project_id*-val meghívja a *delete_project* szolgáltatást. **Végpont:** "projects/delete/<int:project_id>/".

upload_doc: Bemeneti paramétere: *project_id.* Az adott projekthez való dokumentum feltöltést végzi. A fájlt a *request* paraméter FILE attribútumában kapja meg. **Végpont:** "projects/<int:project_id>/upload_doc".

delete_doc: Bementben megkapja a *project_id*-ját és a *doc_id*-ját. E kettő segítségével törli az adatbázisból a fájlt. **Végpont:** "projects/<int:project_id>/delete_doc/<int:doc_id>".

kanban_board : Legyűjti a az összes story-t, task-ot, issue-ot a kanban táblához (amelyek az elmúlt 30 napban módosultak). **Végpont:** "projects/-kanban/".

get_chart_data: Összeszedi az adatbázisból a szükséges adatokat a kanban körgrafikonos kimutatásához. **Végpont:** "projects/kanban_chart".

```
1
2 def index(request):
3     projects_list = Project.objects.all().order_by('-created_date')
4     paginator = Paginator(projects_list, 5)
5
6     page_number = request.GET.get('page',1)
7     projects = paginator.get_page(page_number)
8
9     context = {
10         'project_list': projects_list,
11         'projects': projects,
12     }
13     return render(request, 'projects/index.html', context)
```

3.2. forráskód. Az adatbázisban található projektek listájának oldalakra tördelése a Paginator segítségével

3.5.3. Epics applikáció

Az **Epics** applikáció az epic-ek megvalósítása.

- **Szolgáltatások (services):**

  get_epic_object: Visszaadja a paraméterül kapott *epic_id*-hoz tartozó epic-et.

  delete_epic: Kitörli az adatbázisból a paraméterül kapott *epic_id*-hoz tartozó epic-et.

  get_epic_details: Bemeneti paramétrben megkapja az *epic_id*-t, majd legyűjti az ehhez tartozó epic-hez rendelt stroy-kat, task-okat, issue-kat. Visszatérési értéke egy **dictionary** típusú változó.

  get_epics_for_user: Megkapja a *user_id*-t, majd legyűjti az ehhez a felhasználóhoz tartozó epic-eket és visszaadja egy **dictionary** típusú változóban.

- **Nézetek (views):**

detail: Bemeneti paraméter: *epic_id*. Ezzel meghívja a *get_epic_object* szolgáltatást, majd az eredményül kapott listát rendeli a "epics/detail.html" template-hez. **Végpont:** "projects/epics/<int:epic_id>/".

epic_new: Új **Epic** létrehozását végző view, a **CreateEpicForm** form segítségével. "GET" üzenet esetén az üres form-ot adja át a "epics/epic_edit.html" tempalte-nek, "POST" esetén lementi az adatokat. **Végpont:** "projects/epics/new/".

epic_edit: Megkapja bemeneti paraméterül az *epic_id*-t, majd visszaad egy **CreateEpicForm** form-ot kitöltve a szerkeztendő epic adataival az "epics/epic_edit.html" template-nek. "POST" üzenet esetén frissíti az adatbázisban az adatait. **Végpont:** "projects/epics/<int:epic_id>/edit".

delete: Meghívja az *epic_id* paraméterrel a *delete_epic* szolgáltatást. **Végpont:** "projects/epics/delete/<int:epic_id>/".

3.5.4. Stories applikáció

Az **Stories** applikáció az story-k megvalósítása.

- **Szolgáltatások (services):**

  get_story_object: Visszaadja a paraméterül kapott *story_id*-hoz tartozó story-t az adatbázisból.

  delete_story: Kitörli az adatbázisból a paraméterül kapott *story_id*-hoz tartozó story-t. Logikai értékkel tér vissza attól függően, sikerült-e.

  get_story_details: Bemeneti paramétrben megkapja az *story_id*-t, majd legyűjti az ehhez tartozó story-hoz rendelt felhasználót (ha van), a létrehozóját (owner), az epic-et (ha hozzá van rendelve), a kommenteket (ha vannak) és a munkanapló bejegyzéseket (ha van). Visszatérési értéke egy **dictionary** típusú változó, amelybe belepakolja mindet.

  get_stories_for_user: Megkapja a *user_id*-t, majd legyűjti az ehhez a felhasználóhoz rendelt (assignee) story-kat és visszaadja egy **dictionary** típusú változóban.

change_story_state: A paraméterül kapott *story_id*-hoz tartozó story státuszát állítja tovább.

- **Nézetek (views):**

  detail: Bemeneti paraméter: *story_id*. Ezzel meghívja a *get_story_object* szolgáltatást, majd az eredményül kapott listát rendeli a "stories/detail.html" template-hez. **Végpont:** "projects/stories/<int:story_id>/".

  story_new: Új **Story** létrehozását végző view, a **CreateStoryForm** form segítségével. "GET" üzenet esetén az üres form-ot adja át a "stories/story_edit.html" tempalte-nek, "POST" esetén lementi az adatokat. **Végpont:** "projects/stories/new/".

  story_edit: Megkapja bemeneti paraméterül a *story_id*-t, majd visszaad egy **CreateStoryForm** form-ot kitöltve a szerkeztendő story adataival a "stories/story_edit.html" template-nek. "POST" üzenet esetén frissíti az adatbázisban az adatait. **Végpont:** "projects/stories/<int:story_id>/edit".

  delete: Meghívja az *story_id* paraméterrel a *delete_story* szolgáltatást. **Végpont:** "projects/stories/delete/<int:story_id>/".

  create_comment: A *story_id*-hoz tartozó story-hoz kommentek létrehozását biztosítja a **CreateCommentForm** segítségével, illetve "POST" üzenet esetén le is menti azt a story-hoz kapcsolva.**Végpont:** "projects/stories/comment/<int:story_id>/".

  delete_comment: A *story_id*-hoz tartozó story-hoz létrehozott, *comment_id*-val rendelkező kommentet törli. **Végpont** "projects/stories/comment/delete/<int:story_id>/<int:comment_id>".

  change_state: Meghívja a *story_id*-val a *change_story_state* szolgáltatást. **Végpont:** "projects/stories/<int:story_id>/change_state/".

  add_worklog :Biztosít egy **CreateWorklogform** form-ot a "worklogs/log_work.html" tempalte-hez. "POST" üzenet esetén létrehozza a munkanapló bejegyzést a story-hoz. **Végpont:** "projects/stories/<int:story_id>/add_worklog/".

3.5.5. Tasks applikáció

Az **Tasks** applikáció az task-ok megvalósítása.

- **Szolgáltatások (services):**

  get_task_object: Visszaadja a paraméterül kapott *task_id*-hoz tartozó task-ot az adatbázisból.

  delete_task: Kitörli az adatbázisból a paraméterül kapott *task_id*-hoz tartozó task-ot. Logikai értékkel tér vissza attól függően, sikerült-e.

  get_task_details: Bemeneti paramétrben megkapja az *task_id*-t, majd legyűjti az ehhez tartozó task-hoz rendelt felhasználót (ha van), a létrehozóját (owner), az epic-et (ha hozzá van rendelve), a kommenteket (ha vannak) és a munkanapló bejegyzéseket (ha van). Visszatérési értéke egy **dictionary** típusú változó, amelybe belepakolja mindet.

  get_tasks_for_user: Megkapja a *user_id*-t, majd legyűjti az ehhez a felhasználóhoz rendelt (assignee) task-okat és visszaadja egy **dictionary** típusú változóban.

  change_task_state: A paraméterül kapott *task_id*-hoz tartozó task státuszát állítja tovább.

- **Nézetek (views):**

  detail: Bemeneti paraméter: *task_id*. Ezzel meghívja a *get_task_object* szolgáltatást, majd az eredményül kapott listát rendeli a "tasks/detail.html" template-hez. **Végpont:** "projects/tasks/<int:task_id>/".

  task_new: Új **Task** létrehozását végző view, a **CreateTaskForm** form segítségével. "GET" üzenet esetén az üres form-ot adja át a "tasks/task_edit.html" tempalte-nek, "POST" esetén lementi az adatokat. **Végpont:** "projects/tasks/new/".

  task_edit: Megkapja bemeneti paraméterül a *task_id*-t, majd visszaad egy **CreateTaskForm** form-ot kitöltve a szerkeztendő task adataival a "tasks/task_edit.html" template-nek. "POST" üzenet

esetén frissíti az adatbázisban az adatait. **Végpont:** "projects/-tasks/<int:task_id>/edit".

delete: Meghívja az *task_id* paraméterrel a *delete_task* szolgáltatást. **Végpont:** "projects/tasks/delete/<int:task_id>/".

create_comment: A *task_id*-hoz tartozó task-hoz kommentek létrehozását biztosítja a **CreateCommentForm** segítségével, illetve "POST" üzenet esetén le is menti azt a task-hoz kapcsolva.**Végpont:** "projects/tasks/-comment/<int:task_id>/".

delete_comment: A *task_id*-hoz tartozó task-hoz létrehozott, *comment_id*-val rendelkező kommentet törli. **Végpont** "projects/tasks/-comment/delete/<int:task_id>/<int:comment_id>".

change_state: Meghívja a *task_id*-val a *change_task_state* szolgáltatást. **Végpont:** "projects/tasks/<int:task_id>/change_state/".

add_worklog :Biztosít egy **CreateWorklogform** form-ot a "worklogs/log_work.html" tempalte-hez. "POST" üzenet esetén létrehozza a munkanapló bejegyzést a task-hoz. **Végpont:** "projects/-tasks/<int:task_id>/add_worklog/".

3.5.6. Issues applikáció

Az **Issues** applikáció az issue-k megvalósítása.

- **Szolgáltatások (services):**

  get_issue_object: Visszaadja a paraméterül kapott *issue_id*-hoz tartozó issue-t az adatbázisból.

  delete_issue: Kitörli az adatbázisból a paraméterül kapott *issue_id*-hoz tartozó issue-t. Logikai értékkel tér vissza attól függően, sikerült-e.

  get_issue_details: Bemeneti paramétrben megkapja az *issue_id*-t, majd legyűjti az ehhez tartozó issue-hoz rendelt felhasználót (ha van), a létrehozóját (owner), az epic-et (ha hozzá van rendelve), a kommenteket (ha vannak) és a munkanapló bejegyzéseket (ha van) és egyéb mezőit.

Visszatérési értéke egy **dictionary** típusú változó, amelybe belepakolja mindet.

get_issues_for_user: Megkapja a *user_id*-t, majd legyűjti az ehhez a felhasználóhoz rendelt (assignee) issue-kat és visszaadja egy **dictionary** típusú változóban.

change_issue_state: A paraméterül kapott *issue_id*-hoz tartozó issue státuszát állítja tovább.

- **Nézetek (views):**

detail: Bemeneti paraméter: *issue_id*. Ezzel meghívja a *get_issue_object* szolgáltatást, majd az eredményül kapott listát rendeli az "issues/detail.html" template-hez. **Végpont:** "projects/issues/<int:issue_id>/".

issue_new: Új **Issue** létrehozását végző view, a **CreateIssueForm** form segítségével. "GET" üzenet esetén az üres form-ot adja át az "issues/issue_edit.html" tempalte-nek, "POST" esetén lementi az adatokat. **Végpont:** "projects/issues/new/".

issue_edit: Megkapja bemeneti paraméterül a *issue_id*-t, majd visszaad egy **CreateIssueForm** form-ot kitöltve a szerkeztendő issue adataival az "issues/issue_edit.html" template-nek. "POST" üzenet esetén frissíti az adatbázisban az adatait. **Végpont:** "projects/issues/<int:issue_id>/edit".

delete: Meghívja az *issue_id* paraméterrel a *delete_issue* szolgáltatást. **Végpont:** "projects/issues/delete/<int:issue_id>/".

create_comment: A *issue_id*-hoz tartozó issue-hoz kommentek létrehozását biztosítja a **CreateCommentForm** segítségével, illetve "POST" üzenet esetén le is menti azt az issue-hoz kapcsolva.**Végpont:** "projects/issues/comment/<int:issue_id>/".

delete_comment: A *issue_id*-hoz tartozó issue-hoz létrehozott, *comment_id*-val rendelkező kommentet törli. **Végpont** "projects/issues/comment/delete/<int:issue_id>/<int:comment_id>".

change_state: Meghívja az *issue_id*-val a *change_issue_state* szolgáltatást. **Végpont:** "projects/issues/<int:issue_id>/change_state/".

add_worklog :Biztosít egy **CreateWorklogform** form-ot a "worklogs/log_work.html" tempalte-hez. "POST" üzenet esetén létrehozza a munkanapló bejegyzést az issue-hoz. **Végpont:** "projects/issues/<int:issue_id>/add_worklog/".

3.5.7. Users applikáció

A **Users**, azaz a felhasználó-, profilkezelés megvalósítása. Nem rendelkezik szolgáltatás réteggel, ugyanis az alkalmazás jelen formájában ezt nem igényli, így példaként szolgál az ilyen típusú megvalósításra. Az egyes view-k (függvények) visszatérési értékét megváltoztatva könnyen szolgáltatássá lehet alakítani ezeket is.

- **Form-ok:**

  SelectMontForm: A munkanapló oldalakon a hónap kiválasztását végző form. Egy naptár **widget**-et is használ az *admin* modulból (megjelenítés).

  AddGroupForm: A csoport létrehozást lehetővé tevő form. Egy szöveges input mezője van.

- **Nézetek (views):**

  index: A felhasználó index oldalát jeleníti meg ("users/index.html"). Gyakorlatilag ez az alkalamzás főoldala. **Végpont:** "users/".

  detail: Egy adott felhasználó profil oldala. Bemeneti paraméterben megkapja a *username*-t, ehhez lekérdezi a felhasználót az adatbázisból, amelynek azonosítójával tovább hívja az egyes applikációk szolgáltatásait (story, issue, task). Ezek visszadják az adott felhasználóhoz rendelt feladatokat, a *detail* nézet ezeket összeszedi egy adatszerkezetbe (**dictionary**), majd ezt továbbítja a template-nek ("users/detail.html"). **Végpont:** "users/<username>/".

  get_users_worklogs: Ez egy hosszabb függvény, amely "GET" üzenet esetén legyűjti az aktuális hónap és nap munkanapló bejegyzéseit a

bemenetül kapott *username* felhasználóhoz. "POST" üzenet esetén a template-en elhelyezett formból kinyeri a beírt dátumot, majd az alapján gyűjti le a napi, illetve havi bejegyzéseket. **Végpont:** "users/<username>/worklogs/".

delete_worklog: A paraméterül kapott *log_id*-val beazonosítja az adatbázisban a munkanapló bejegyzést és kitörli. **Végpont:** "users/delete_worklog/<int:log_id>".

team_worklogs: Szintén egy összetettebb view függvény. Az alapján, hogy a *request*-et küldő felhasználó beletartozik-e a "project_manager" csoportba, avagy *superuser*-e, összeszámolja az egyes felhasználókhoz könyvelt munkanapló bejegyzések óraszámait az adott hónapban, majd táblázatosan megjeleníti. Ha nem igaz egyik sem a felhasználóra, akkor csak a saját munkaidejét számolja ki. **Végpont:** "users/team_worklogs/".

group_list: Legyűjti az összes adatbázisbeli felhasználói csoportot a "users/groups.html" tempalte számára. Kezeli a "POST" üzenetet is, ugyanis a tempalte-en egy input szöveges mező segítségével vehetünk fel új csoportot (feltéve, hogy nem létezik már). **Végpont:** "users/groups/".

group_detail: Egy adott csoportsaját oldalára navigál. Ehhez legyűtji az összes adatbázisbeli jogosultságot (permissions) és az adott csoport (bemenet: *gr_id*) aktuális jogosultságait. **Végpont:** "users/groups/<int:gr_id>".

delete_group: Törli az adatbázisból a kapott *gr_id*-hoz tartozó csoportot. **Végpont:** "users/groups_delete/<int:gr_id>".

add_perm_to_group: Jogosultságot (*p_id*) rendel hozzá egy csoporthoz (*gr_id*). **Végpont:** "users/add_permission/<int:gr_id>/<int:p_id>".

delete_perm_from_group: Eltávolítja az adott jogosultságot (*p_id*) a csoport (*gr_id*) jogosultságai közül. **Végpont** "users/delete_permission/<int:gr_id>/<int:p_id>".

delete_user: Törli a *user_id*-hoz tartozó felhasználót az adatbázisból. **Végpont:** "users/delete_user/<int:user_id>".

list_all_users: Lekérdezi az összes adatbázisbeli felhasználót a "users/all_users.html" template számára. **Végpont:** "users/all_users/".

3.5.8. Worklogs applikáció

A **Worklogs** applikáció csak a munkanapló bejegyzések adatbázis modell reprezentációjára szolgál. Ezért nem rendelkezik külön szolgáltatás és nézet réteggel.

3.6. Megjelenítés (template) réteg

A megjelenítés réteget a HTML tempalte fájlok adják az alkalmazásban. Közvetlen kapcsolatot a nézet (view) függvények jelentenek a szerverrel (adatbázissal). Minden applikáció (modul) rendelkezik saját **templates** mappával, amelyben van egy almappa a saját nevével (például: "projects/tempaltes/projects/"). Utóbbi azért szükséges, mert a keretrendszer így tudja beazonosítani, hogy mely útvonalon találja meg a keresett tempalte-et. A következő felsorolásban össze van gyűjtve, hogy az egyes applikációk milyen tempalte fájlokkal rendelkeznek. Az alapszerkezetüket ezeknek a HTML fájloknak a "ScrumHelper/templates/ScrumHelper/base.html" fájl adja. A Djangoban lehez használni úgynevezett *tag*-eket a HTML kódban, amelynek segítségével lehet sablonokat használni (mint például a *base.html*), valamint ciklusoakt és elágazásokat is írhatunk velük. A főmappában van egy "static/" mappa, ez tartalmazza a CSS és JavaScript állományokat a formázott megjelenítéshez. A *ScrumHelper* a MaterializeCSS könyvtárat használja [8].

[8] `https://materializecss.com/`

```
1
2 {\% extends 'ScrumHelper/base.html' \%}
3 {\% load static \%}
4
5 {\% block title \%} Login {\% endblock \%}
6
7 {\% block login_form \%}
8     {\% if form.errors \%}
9     <p>Your username and password didn't match. Please try again.</
          p>
10    {\% endif \%}
11
12    {\% if next \%}
13        {\% if user.is_authenticated \%}
14        <p>Your account doesn't have access to this page. To
              proceed,
15        please login with an account that has access.</p>
16        {\% else \%}
17 ...
```

3.3. forráskód. Példa a Django tag-ekre a HTML kódban a bejelentkező ablak esetén (login/tempaltes/registration/login.html)

3.6.1. Login

- **login.html**: A bejelentkező ablak. Egy formot jelenít meg, amely bejelentkeztet és jelzi az esetleges hibákat.
- **signup.html**: A regisztrációs form-ot megjelenítő ablak.

3.6.2. Projects

- **details.html**: A projekt saját oldala, annak minden részletével.
- **index.html**: Az összes projektet listázó oldal (ötösével, oldalakra szedve).
- **kandban_chart.html**: A kanban tábláról készült kördiagrammot megjelenítő oldal. Ehhez egy JSON üzenetben kapja a szervertől az adatokat és a Chart.js (JQuery alapú) könyvtár segítségével jeleníti meg a diagrammot.

- **kanban.html**: A kanban táblát megjelenítő template.
- **proejct_edit.html**: A projekt szerkeztő és létrehozó oldala egyben. A szervertől kapott form-ot jeleníti meg.

3.6.3. Epics

- **detail.html**: Az epic részleteit megjeleíntő oldal tempalte-je.
- **epic_edit.html**: Az epic-et létrehozó, illetve szerkeztő form-ot megjelenítő tempalte. A szervertől kapja a form szerkezetét és adatait.

3.6.4. Stories

- **detail.html**: Az story részleteit megjeleíntő oldal tempalte-je.
- **story_edit.html**: Az story-t létrehozó, illetve szerkeztő form-ot megjelenítő tempalte. A szervertől kapja a form szerkezetét és adatait.

3.6.5. Tasks

- **detail.html** : Az task részleteit megjeleíntő oldal tempalte-je.
- **task_edit.html**: Az task-ot létrehozó, illetve szerkeztő form-ot megjelenítő tempalte. A szervertől kapja a form szerkezetét és adatait.

3.6.6. Issues

- **detail.html**: Az issue részleteit megjeleíntő oldal tempalte-je.
- **issue_edit.html**: Az issue-t létrehozó, illetve szerkeztő form-ot megjelenítő tempalte. A szervertől kapja a form szerkezetét és adatait.

3.6.7. Users

- **all_users.html**: Az összes felahsználó listáját megjelenítő tempalte.
- **group_detail.html**: Az adott felhasználói jogosultságcsoport jogosultságait és az összes jogosultságot jeleníti meg két táblázatban. Előbbiből törölni

lehet(kék alapon fehér kereszt ikon), utóbbiból hozzáadni jogokat a csoporthoz(kék alapon fehér plusz jel).

- **groups.html**: Az összes felhasználói jogosultságcsoportot kilistázó tempalte. Egy szöveges input mező segítségével új csoportot vehetünk fel (ha az még nem létezik, ez esetben hibaüzenetet ad).
- **index.html**: Az alkalamzás főoldalát adó template.
- **personal_issues.html**: A személyes felhasználói profiloldalt adó template. Felhasználónév, email cím, teljes név, hozzárendelt feladatok, jelszóváltás opció, felhasználó szerkeztése és törlése (jogosultsághoz kötött, lásd: jogosultság táblázat).
- **team_worklogs.html**: A scrum felhasználóinak össz munkaóra számát megjelenítő template, havi bontásban. Biztosít egy form-ot a dátum kiválasztásához.
- **worklogs**: A felahsználó munkaidő naplóját jeleníti meg napi és havi bontásban. Biztosít egy form-ot a dátum kiválasztásához.

3.6.8. Worklogs

- **log_work**: Egy form-ot jelenít meg ez a tempalte, amelyen dátumot és munkaórát (1 és 8 között) adhatunk meg.

3.7. Tesztelés

Itt található az alkalamzáshoz készített egység tesztek leírása (hogyan kell futtatni, miket fed le), illetve a különböző funkciókhoz felületi tesztforgatókönyv.

3.7.1. Egység tesztek

A Django keretrendszer lehetőséget biztosít egység tesztek készítésésre és futtatására. Minden applikációban található egy *test.py* fájl. A teszteléshez ebbe teszt osztályokat kell létrehozni, amelyek a **TestCase** leszármazottai kell legyenek. Ezen

belül van lehetőség teszt metódusokat definiálni, amelyek egy-egy tesztesetet fednek le. Lehetőség van a parancssorból lefuttatni az összes tesztesetet egyben, vagy akár csak alkalmazásonként külön-külön. Ehhez a következő parancsot kell kiadni a főmappán belül (ahol a *manage.py* fájl is van): *python manage.py test <app neve>.<esetleg alapplikáció neve>*. A 3.13. ábrán látható a parancssoros kimenet. Ha minen teszteset sikeres, akkor "OK" üzenetet kapunk. Ha van hibás, akkor egy "Error" üzenetben részletesebb információt is kapunk arról, hogy mi az elvárt eredmény az összehasonlításnál és mi az, amit ehelyett kapott.

```
----------------------------------------------------------------------
Ran 56 tests in 28.121s

OK
Destroying test database for alias 'default'...
```

3.13. ábra. A "python manage.py test projects" parancs eredménye.

Az egység tesztekhez a *test* parancs futtatása során létrejön a memóriában egy ideiglenes adatbázis, amely szerkezetileg megegyezik a valós adatbázissal.

A *ScrumHelper* minden applikációjában, ahol van szolgáltatás és/vagy nézet réteg (azaz *services.py vagy views.py* fájlok), ott minden funkcióra (függvényre) vonatkozik egy-egy teszteset. Ahol sajátkezűleg szükséges volt megoldani az esetleges hibák lekezelést (tehát nem a keretrendszer működésén múlik), ott egy pozitív (azaz az helyes működés) és egy negatív (azaz hiba esetén) teszteset vizsgálja az elvárt működést.

3.7.2. Felületi tesztek

Felületi tesztek automatikus elvégzésére nem biztosít lehetőséget a Django, ezért ezeket a teszteket kézzel, az alábbi forgatókönyvet végig játszva lehet eredményesen letesztelni:

1. Bejelentkezés:
    - **helyes adatok:** A program bejelentkezteti a felhasználót és a főoldalra navigálja.

- **helytelen adatok:** Egy hibaüzenet figyelmezteti a felhasználót a hibás adatokra. Az alkalmazás nem léptet tovább.
- **üres mezők:** Figyelmeztető üzenet az üresen hagyott mezőkre.
- **sikeres bejelentkezés nélkül továbblépés:** Ebben az esetben egy hibaüzenet jelenik meg egy linkkel a bejelentkezési oldalra.
- **Logout menüpont a navigációs sávban:** Kijelentkeztet és visszavezeti a felhasználót a bejelentkező oldalra.

2. Főoldal:

- **ScrumHelper szövegre kattintás:** A program bármely oldalról, ha a navigációs sávbeli **Scrumhelper** szövegre kattintunk, akkor visszajuttat erre a főodlalra.
- **Navigációs sáv menüpontjaira kattintás:** Rendre az elvárt oldalakra navigálnak el.
- **A három menüpont alatti View gombok:** Rendre eljutunk a **Projects** index oldalra, a saját munkanapló oldalra, valamint a saját profil oldalra.

3. Projekt index oldal:

- **Listás megjelenítés:** Valóban egyszerre csak öt (avagy a beállított mennyiségű) projekt jelenik meg a listában.
- **Az oldal léptető gombok:** Megfelelő oldalszámot mutat és valóban tovább ugrik a listán.
- **Működő projekt linkek:** A listában szereplő projektek neve melletti nyíl ikon helyesen a projekt saját oldalára mutat.

4. Egyes objektumok részletező oldalai:

- **Felhasználói linkek:** A felhasználóneveket tartalmazó mezők valóban a megfelelő profil oldalára juttatnak el.
- **Dokumentum feltöltés:** Kiválasztó gomb (valamint kiválasztás) és feltöltés működik. Feltöltés után megjelenik a dokumentum a listában.

- **Dokumentum törlés:** A törlés gombra listából eltűnik a dokumentum.
- **Dokumentum linkek:** A dokumentumra kattintva megtekinthetőek a fájlok.
- **Epic, Story, Task, Issue lista:** Az összes feladat típus, ami a projekthez lett létrehozva megjelennek a listában. Az egyes elemre kattitnva eljutunk annak saját oldalára.
- **Almenü (jobb alsó sarok):** Az egeret ráhúzva megjelennek a menüpontok: szerkeztés, issue, story, task és epic létrehozás, munkaidő könyvelés (story, task, issue esetén) valamint megfelelő admin/projekt tulajdonosként törlés.
- **Kommentek létrehozása:** Komment beküldése esetén (ahol erre van lehetőség), a komment megfelelő időponttal, a létrehozó felhasználóhoz kötve (linkkel ellátva), a beírt szöveggel létrejön.
- **Kommentek törlése:** A törlés gombra kattintás határásra a kitörlődik, nem lesz látható.
- **Projekt törlése:** Projekt törlése esetén az összes feladat, amelyet ez alatt hoztak létre törlődik.

5. Az egyes létrehozási oldalak (projekt, story, epic, task, issue)

   - **Üres mezők:** Figyelmeztető üzenet a szükséges mezők kitöltésére. Az opcionálisakra nem.
   - **Formai ellenőrzés:** Az egyes adatmezők formai ellenőrzése megtörténik. Hiba esetén hibaüzenet.
   - **Mentés:** Sikeres kitöltés esetén létrejön az objektum.

6. Munkanapló

   - **Saját munkanapló oldal:** Dátum kiválasztása előtt az aktuálsi hónaphoz és naphoz jelennek meg a bejegyzések (és csak ehhez). Dátum választás után a megfelelő hónaphoz és naphoz.
   - **Hibás input lekérdezésnél:** Hibaüzenet.

- **Hibás input létrehozásnál:** Dátum formátum ellenrőzés, ennek megfelelő hibaüzenet, avagy nem 1 és 8 közé eső szám megadása esetén is figyelmeztetés.
- **Törlés:** A bejegyzés törlés ikonjára kattintva az eltűnik a felületről (és az adatbázisból is).
- **Scrum munkanapló:** Csak rendszergazda, scrum master és projekt menedzser jogosultságú felhasználó látja mindenki kiszámolt óraszámát a kiválasztott hónapra. A sajátját mindenki látja (a többieknél 0 szerepel).

7. Kanban

- **Tartalom:** Az összes olyan story,task,issue, szerepel a táblán, amely nem CLOSED státuszú (és 30 napon belül lett módosítva).
- **Működő linkek:** Az egyes feladatok "kártyájára" kattintva megfelelő linkre jutunk el.
- **Kimutatás:** Megjelenik a kördiagramm és valós arányokat mutat a kanban tábla tekintetében.

8. Jogosultság kezelés

- **Csoport lista:** Megjelenik, oldalakra tördelve (ötösével) az összes felhasználói csoport. A léptetés működik.
- **Csoport létrehozás:** Még nem létező csoport létrehozásakor megjelenik az új csoport neve a listában. Létező esetén hibaüzenet.
- **Csoport törlése:** Eltűnik a listából a csoport a törlés gombra kattintás után. Ugyanezen névvel létre lehet hozni újra.
- **Jogosultság hozzáadás és elvétel:** Az egyes csoprotkra kattintva megjelenik a két lista a jogosultsgákokról: az összes éa a csoport aktuális jogai. A plusz gombra kattintva megjelenik a jogosultság az alsó listában. A törlés gombra kattintva eltűnik onnan.

4. fejezet

Összegzés

Az agilis módszertanban ugyan kisebb létszámú csapatok dolgoznak együtt, mégis rengeteg és szerteágazó feladatokkal találják magukat szemben. A hatékony működés érdekében találták ki ezeket a módszertanokat és azok altípusait (például a scrum-ot).

A gyakorlatban viszont külön eszközökre is szükségük van ezek megvalósításához. A szakdolgozat célja egy olyan alkalmazás elkészítése volt, amely önmagában is alkalmas kisebb scrum-ok fejlesztési munkájának nyomonkövetésére, segítésére.

Természetesen sokféle mód van még az alkalmazás további fejlődésére, ehhez biztosít stabil alapokat maga a keretrendszer is. Pár ötlet ezek közül röviden az alábbi szekcióban olvasható.

1. Egy kereső funkció implemetálása, amely az oldal egész tartalmán végig halad és ezzel megkönnyíti a felhasználóknak az egyes funkciók és objektumok elérését.

2. Még részletesebb statisztikai oldalak implementálása Sprint specifikusabban (grafikonok, diagrammok, szűrések).

3. A nézetek (view-k) áthelyezése teljes mértékben a szolgáltatás rétegbe. Vannak applikációk, ahol ez már megvalósításra került (*service.py* modulok) bizonyos mértékben. Ennek célja egy API (Application Programming Interface) kialakítása, ezzel teljesen szinte teljesen függetlenítve a szerver és kliens réteget. A Django is rendelkezik egy RESTful kommunikációt megvalósító könyvtárral.

Ábrák jegyzéke