A megvalósítandó szoftver követelmény leírása

**Áttekintés**

Egy komplex rendszer kifejlesztése hosszú időt, tervezést és odafigyelést igényel, amit befolyásol a projekt mérete és az azon dolgozó csapatok száma. Minél összetettebb a rendszer, minél több fejlesztőcsapat dolgozik a projekten, annál komolyabb munka a csapatok lokális és globális menedzselése. A szoftver minősége és a határidők betartása függ az adott csapat készségétől, a tervezéstől és az előző fejlesztési ciklusok alapján kialakult tanulságoktól, valamint a csapattagok motiváltságától. A szakdolgozat témája egy olyan applikáció elkészítése, amelyben lehetőség nyílik egy agilis módszertan alapján dolgozó fejlesztőcsapat menedzselésére és támogatására, beleértve a munkafolyamatok megtervezésének gördülékennyé tételét, az adott fejlesztési ciklus, azaz ‘sprint’ kiértékelését csapat szinten, ezek visszakereshetőségét és feladatkörökre való osztását, a csapatmorál mérését és metrikák generálását a még hatékonyabb tervezés érdekében, valamint kompetencia mátrix előállítását a csapat tudását illetően.

Cél egy olyan alkalmazás lefejlesztése és prezentálása, ami egy valódi agilis csapatban jelentkező problémák megoldását teszi könnyebbé, gyorsabbá, és átláthatóbbá, a lehető legnagyobb transzparencia mellett. A Célközönség egy agilis módszertan alapján fejlesztő scrum csapat, ami a fejlesztőkből, egy scrum masterből, és egy product owneből áll.

Funkcionalitás szempontjából a legfontosabb, hogy átláthatóak és könnyen kezelhetőek legyenek, amiket a módszertan megkövetel, valamint az agilitáshoz megfelelően maximális rugalmasságot tegyen lehetővé. (A fent említett Scrum módszertanok később kerülnek kifejtésre). Az elkészített szoftver egy a böngészőből elérhető webes alkalmazás.

**Rendszer Funkciói**

Az elkészített web alkalmazás egy a legmodernebb szoftvertechnológiák által ötvözött úgynevezett web-stack. A web-stack szoftverek és technológiák kollekciója, amik egy komplex alkalmazás lefejlesztéséhez szükségesek. A Fő alkotó elemük egy Frontend service egy Backend service egy adatbázis motor, ami általában SQL adatbázis, tehát nem relációs adatbázis, ezek általában JSON alapúak. Ezen szakdolgozat témáját alkotó alkalmazás egy a manapság legmodernebb technológiák felhasználásával alkotott full-stack web alapú szoftver, ami a (MySQL, Express.js, React, Node.js)-ből áll.

* MySQL
  + A MySQ egy [többfelhasználós](https://hu.wikipedia.org/w/index.php?title=T%C3%B6bbfelhaszn%C3%A1l%C3%B3s&action=edit&redlink=1), [többszálú](https://hu.wikipedia.org/w/index.php?title=T%C3%B6bbsz%C3%A1l%C3%BAs%C3%A1g&action=edit&redlink=1), [SQL](https://hu.wikipedia.org/wiki/SQL)-alapú [relációs adatbázis-kezelő](https://hu.wikipedia.org/wiki/Rel%C3%A1ci%C3%B3s_adatb%C3%A1zis-kezel%C5%91) [szerver](https://hu.wikipedia.org/wiki/Szerver).
  + <https://hu.wikipedia.org/wiki/MySQL>
* Express.js
  + Az Express egy gyors, rugalmas, de mégis kis web framework, amiben könnyedén lehet robosztus web alkalmazásokat írni Node.js alapokon.
  + <https://expressjs.com/>
* React
  + A React egy komponens alapú javascript frontend könyvtár, felhasználói interfészek megjelenítésére. <https://reactjs.org/>
* Node.js:
  + A **Node.js** egy szoftverrendszer, melyet [skálázható](https://hu.wikipedia.org/wiki/Sk%C3%A1l%C3%A1zhat%C3%B3s%C3%A1g) internetes alkalmazások, mégpedig [webszerverek](https://hu.wikipedia.org/wiki/Webszerver) készítésére hoztak létre. A programok [JavaScript](https://hu.wikipedia.org/wiki/JavaScript)-ben írhatók, [eseményalapú](https://hu.wikipedia.org/wiki/Esem%C3%A9nyalap%C3%BA_szoftverarchitekt%C3%BAra), aszinkron I/O-val a túlterhelés minimalizálására és a [skálázhatóság](https://hu.wikipedia.org/wiki/Sk%C3%A1l%C3%A1zhat%C3%B3s%C3%A1g) maximalizálására.
  + <https://hu.wikipedia.org/wiki/Node.js>

Egyéb felhasznált modulok / fejlesztőeszközök / kód könyvtárak:

Ezeket a fejlesztés során külön kell installálni CLI (command line interface) segítségével, az *npm install* paranccsal. Használva a cli adta *--save* kapcsolót lehetőség van a függőségeket nem globálisan, hanem lokálisan az adott projekt mappájába telepíteni, ekkor a projekt *package.json* fájljába kerül elmentésre a *dependencies* kulcsszó alatt a függőség neve, valamint a projekt mappájában létrejövő *node\_modules* mappába történik az installálás. Az így létrejövő projektet, ha szeretnénk egy másikeszközön használni, akkor lehetőségünk nyílik egy paranccsal feltelepíteni az összes függőséget, feltéve, ha mindegyik szerepel a *package.json* fájlban. Ez esetben a megfelelő helyen állva a projekt mappában, kiadva az *npm install* parancsot az npm (node package manager) felolvassa a *package.json dependencies* alatt lévő listát, és feltelepíti mindegyiket.

Backend:

* cors
  + Middleware kiegészítő Express.js-hez, ami lehetővé teszi a CORS (Cross Origin Resource Sharing) módszert, ezáltal a modulok és rétegek közötti adatfolyamot.
  + <https://www.npmjs.com/package/cors>
* passport
  + Express.js kompatibilis middleware felhasználói autentikációra. Különböző előre definiált *Strategy* található meg benne, amikkel elérhetőek a legnépszerűbb közösségi belépési módszerek, mint a Facebook, Google, vagy az Instagram. JWT ( Json Web Token) alapú, ami azt jelenti, hogy ha a belépési kérés *request* sikeres, akkor a rendszer amit használunk a válaszban egy JWT-t küld *response* benne a felhasználó adataival, és a belépésre való jogosultsággal.
  + <https://www.npmjs.com/package/passport>
* chalk
  + Segítségével tudunk különböző színekkel üzeneteket kiírni a terminál képernyőjére, ezáltal elkülöníteni az eltérő üzeneteket.
  + <https://www.npmjs.com/package/chalk>
* bodyParser
  + Átalakítja a szerver felé beérkező kéréseket.
  + <https://www.npmjs.com/package/body-parser>
* Sequelize
  + Többirányú *Promise* alapú ORM (Object Relational Management) kezelő a legelterjedtebb SQL (Structured Query Language) verziókhoz, mint a Posgres, MySQL, MariDB.
* concurrently
  + Segítségével több CLI parancsot tudunk futtatni egyszerre, egyidőben. A szakdolgozat alapján elkészült alkalmazás esetén egyszerre egy parancs segítségével indul mind a server és a kliens alkalmazás is.
  + <https://www.npmjs.com/package/concurrently>
* mysql2
  + Gyorsaságot előnyben részesítő mysql kliens.
  + <https://www.npmjs.com/package/mysql2>
* nodemon
  + Minden a szerver fájlt ért változtatás után újra indítja a szervert az új frissítésekkel.
  + <https://www.npmjs.com/package/nodemon>

Frontend:

* Material-ui
  + Google által fejlesztett React komponensek gyűjteménye.
  + <https://www.npmjs.com/package/@material-ui/core>
* lodash
  + Gyors és könnyű adatkezelés és adat manipuláció.
  + <https://lodash.com/>
* react-router
  + Útvonal meghatározás az oldalak, és komponensek között frontend oldalon.
  + <https://www.npmjs.com/package/react-router>

**Környezet konfigurálása:**

A projekt legelején beállításra kerül, külön a package.json fájlban egy új kulcs érték pár, ami a *’’proxy’’ : ’’http://localhost:5000’’*. Erre azért van szükség, hogy az API hívások esetén a szerver tudja, hogy honnan határozza meg a különböző API kéréseket. Így ha frontend oldalon a *fetch(’/api/team’)* függvény kerül meghívásra, akkor a szerver azt automatikusan átirányítja a *http://localhost:5000/api/team*-re.

A rekordok tárolása MySQL adatbázis szerveren történik. Node.js backend környezetben a Sequelize SQL ORM segítségével vannak a táblák és struktúrák definiálva, ez a *db*/*connections.js* fájlban található. Ebben található egy szülő objektum a *Model*, amiből származik az összes JavaScript osztály, ami az SQL táblák struktúráját írja le. Ezek a projekt elején, elkészülnek, ha még nincsenek definiálva, és SQL oldalon létrejön az adatbázis és a hozzá tartozó táblák, a megfelelő attribútumokkal, kezdőértékekkel.

Az elkészült alkalmazás Passport.js-t használ felhasználói autentikációra, valamint verifikációra. Jelen dolgozat tárgyaként szolgáló webes alkalmazás a Passport által elérhető ’stratégiák’ közül a Google közösségi stratégiáját használja. Előkövetelmény, hogy a felhasználóknak legyen Google felhasználói profiljuk, hogy be tudjanak lépni az alkalmazásba.

A környezet beállításához a Google Developer Console felületen regisztrálni kell az elkészített alkalmazást a nevével, és a host címével együtt. Az alkalmazáshoz rendelődik egy egyedi titkosított *ClientId* és egy *ClientSecret*. Erre a regisztrációra azért van szükség, mert így tudja a Google autentikálni az alkalmazást, és a kéréseket kezelni az irányából. Ezután implementálni kell szerver oldalon a *Passport.GoogleStrategy*-t.

passport.use(**new** *GoogleStrategy*({

        clientID: keys.GOOGLE.clientID,

        clientSecret: keys.GOOGLE.clientSecret,

        callbackURL: "/auth/google/callback"

    },

    (*accessToken*, *refreshToken*, *profile*, *cb*) => {

        connections.User.findOrCreate({ where: {

            email: profile.emails[0].value

        }, defaults: {

            name: profile.displayName,

            profilePicture: profile.photos[0].value,

        } }).then(([*registeredUser*, *created*]) => {

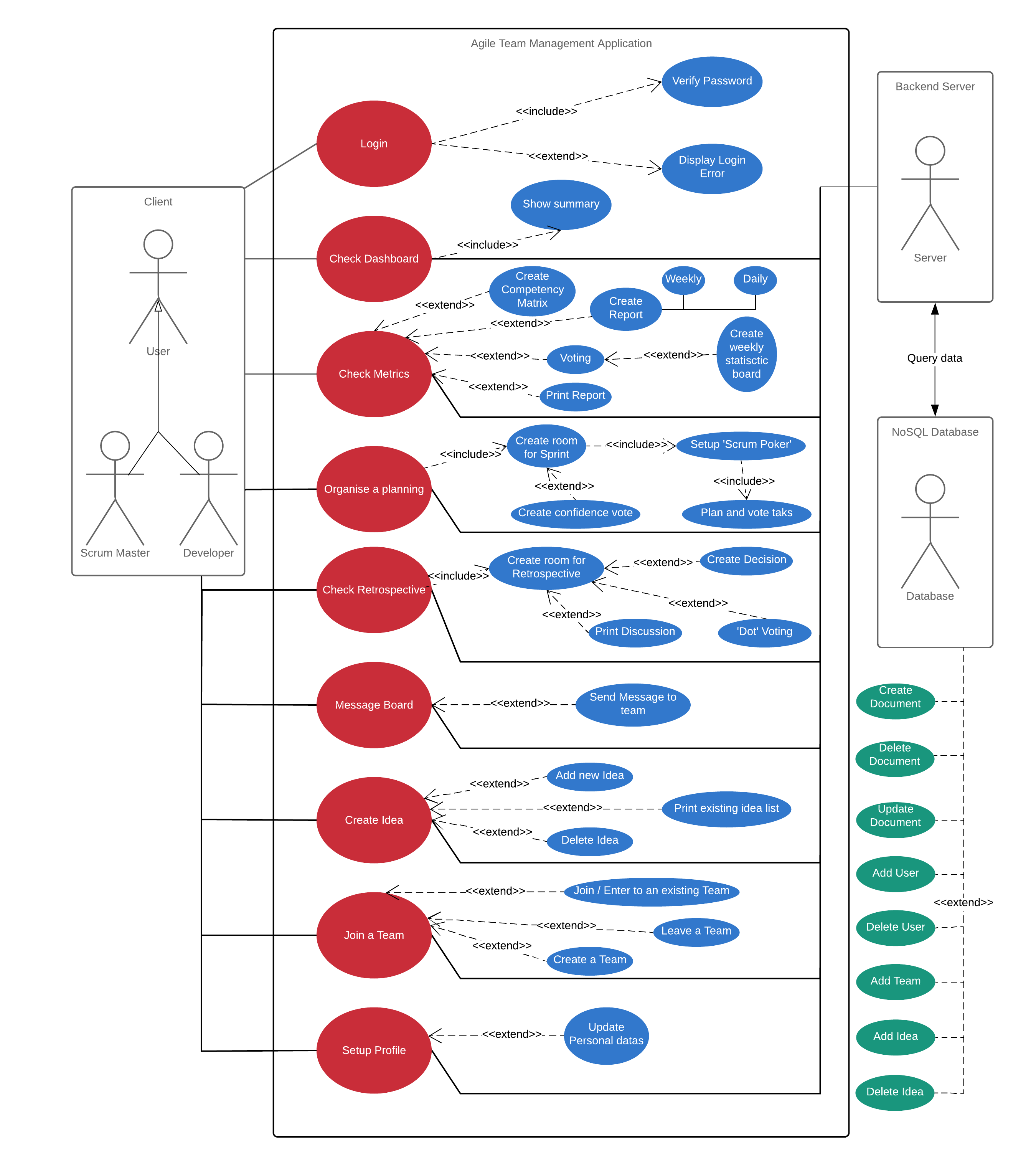
            user = registeredUser.get({ plain: true });

        });

        return cb(null, profile);

    }));

Létrehozunk egy új GoogleStrategy objektumot, a clientId, clientsecret és callbackURL kulcsokkal, amiket a Developer Console-n konfiguráltunk be, ezáltal a két platform tud csatlakozni egymáshoz. Sikeres belépés esetén a szerver visszakap egy JSON objektumot, amiben az autentikált felhasználó adatai lesznek benne, amik, a felhasználónév, id, email cím, profilkép url címe… A szerver létrehoz egy új rekordot az ATM adatbázis User táblájában, a GoogleStrategy eredménye alapján. Mivel a *User* táblába *findOrCreate* metódussal lett elmentve a rekord, így ha nem létezik akkor elmentésre kerül, azután a szerveren lévő kezdetben üres *user* objektum feltöltésre kerül az felhasználó adataival, ha létezik a táblában, vagy most lett beregisztrálva. Ez az objektum tartalmazza szerver oldalon a felhasználó adatait, ami elküldésre kerül API-on keresztül a frontend felé.



Nico-nico:

<https://www.agilealliance.org/glossary/nikoniko/>

Miért érdemes egy Agilis csapatnak Nico-Nico táblát használnia? Ez egy hangulat mutató, ami általában egy sprintet fed le, és mind csapatra, mind az egyénre lehet hosszú és rövidtávú következtetéseket, metrikákat levonni belőle, valamint lokalizálni az esetleges negatív tényezőket. A Nico-Nico táblázat fejléce a napokat tartalmazza, és ahhoz vannak oszlopban felsorolva a csapat tagjai. Általában három fajta mutatóval dolgoznak, ez három emotikon, vagyis a köznyelvben elterjedt ’Smiley’. Három típusa van:

* Szomorú:



* Neutrális: 
* Boldog:



Ezek által mérhető a csapat motivációja és teljesítménye. A Nico-Nico lényege, hogy ne legyen semmi befolyásoló tényező, ezért az aznapi hangulatot jelző kis emotikont a munka befejeztével kell elvégezni, hogy ez legyen az utolsó teendő a munkahelyről való távozás előtt, így semmilyen új input nem fogja befolyásolni az adott személy hangulatát az aznapi munkanapra vetítve. Scrum esetén a Scrum Master felelőssége ezen táblázat kiértékelése, az esetleges negatív dolgok felkutatása, és feloldása. Jelen táblázat hasznosságát más sem bizonyíthatja jobban, mint a Nico-Nico táblázat összehasonlítása az adott sprint sikerességével, mivel ezen két metrika nagyban korrelál egymáshoz. Egy sikertelen sprint esetén a táblázat több ’szomorú’ fejet fog tartalmazni, és egy sikeresnek mondható sprint esetén több ’boldog’ fej lesz, mivel a tervezés minősége és a munka mibenléte motiválta a csapatot. Ha a táblázatban kezdenek felgyűlni a ’szomorú’ smileyk akkor érdemes lehet a scrum masternek organizálni egy retrospective-t a Product Owner felé, hogy a csapat megvitassa a problémát a jelenlegi sprinttel kapcsolatban. Adott esetben ez a sprint újra tervezését vagy a sprint megszakítását eredményezheti.

Kompetencia mátrix:

<http://infografika.ofi.hu/Tud%C3%A1st%C3%A1r/A-diagramok-t%C3%ADpusai/61-sugardiagram>

Egy agilis csapatban szükség van kompetencia mátrix elkészítésére, ami vizualizálja a csapat tudását, a különböző tudásanyagok, technológiák lefedettségi szintjét, ezzel analizálva, és izolálva a hiányzó kompetenciákat a jövőbeni fejlődést elősegítve, ezáltal a csapat hatékonyságát növelve.

A kompetencia mátrix egy sugárdiagram, ami egy itt jön az idézet: *’ a* [*diagram*](http://infografika.ofi.hu/component/rafcloud/1/diagram?searchphrase=exact&ordering=newest) *középpontjából induló és a külső diagramgyűrűn végződő elkülönült tengely mentén ábrázolja az egyes kategóriák értékeit. Az adatsoroknak egy középponthoz, illetve egymáshoz viszonyított változásait vagy gyakoriságát szemlélteti. Minden kategória saját értéktengellyel rendelkezik, amelyek a középpontból sugárirányban ágaznak ki’.*

Jelen szakdolgozat középpontjában álló webes alkalmazás által használt kompetencia mátrix elemei dinamikusan konfigurálhatók a felhasználók által. Ezek programozási nyelvek, valamint technológiák. Ezeket a rendszer az ATM.Competencies táblában tárolja az adatbázisban. Ezek a rekordok törölhetők, valamint megváltoztathatók.

**Mi is az Agilis szoftverfejlesztés?**

Ebben a fejezetben az Agilis szoftverfejlesztés fő elemeit, kialakulást, valamint a Scrum agilis fejlesztési módszertan kerül kifejtésre. Különböző aspektusokból vizsgáljuk meg mi a létjogosultsága az agilis módszertan használatának szoftverfejlesztői környezetben, valamint kifejtésre kerül a Scrum mint módszertan, a csapat felépítése és a módszertan adta események.

**[1] Kiáltvány az agilis szoftverfejlesztésért**

A szoftverfejlesztés hatékonyabb módját tárjuk fel saját tevékenységünk és a másoknak nyújtott segítség útján. E munka eredményeképpen megtanultuk értékelni:

1. Az egyéneket és a személyes kommunikációt a módszertanokkal és eszközökkel szemben
2. A működő szoftvert az átfogó dokumentációval szemben
3. A megrendelővel történő együttműködést a szerződéses egyeztetéssel szemben
4. A változás iránti készséget a tervek szolgai követésével szemben

Azaz, annak ellenére, hogy a jobb oldalon szereplő tételek is értékkel bírnak, mi többre tartjuk a bal oldalon feltüntetetteket.

A 2001-ben megjelent fent olvasható kiáltvány szembement az addigi fejlesztési módszerekkel, valamint csapat és vállalati struktúrákkal. Az agilis módszertan nagyobb hangsúlyt fektet az egyénre, és az úgynevezett „soft skillek”-re, mint a csapat orientáltság, kommunikációs készség, szervezőképesség, és a rugalmasság. A szoftver minősége elsőbbséget élvez a dokumentáltsággal szemben, valamint folyamatos kapcsolat tartását a vevő és a fejlesztőcsapat között. A kiáltvány magába foglalja az *Agilis szoftverfejlesztés 12 alapelvét*, ami nagyobb szabadsággal látja el a fejlesztőcsapatot, viszont ez a nagyobb szabadság és rugalmasság nagyobb felelősséget is helyez a csapatra. A döntések meghozatalától kezdődően együtt kell működjenek az üzleti szakértők a fejlesztők és a vevők egyaránt, ezzel elősegítve a könnyebb és gyorsabb információ folyamot. A csapat felelőssége a folyamatos kiválóságra való törekedés, a csapat hatékonyságának és tudásának növelésére való igény fenntartása, és a munkakörülmények minél gördülékenyebbé tétele. Végére pedig érdemes megemlíteni az egyik legfontosabb újítást az Agilis módszertan és az addigi bevált módszerek között, ami a gyakoribb szállítás. Ez azt jelenti, hogy a szoftvert kisebb egységekben, általában pár hetenként, vagy havonta szállítani kell a vevő felé. Ezek kisebb újítások. Amivel ez a gyakori szállítás értéket teremt a régebbi módszerekhez képest, hogy a folyamatos kisebb egységenként történő szállítás esetén azonnal lehetősége van a vevőtől visszajelzést kapnia a fejlesztőcsapatnak, így rugalmasan igazodva az esetleges változtatásokhoz, véleményekhez, minthogy egyszer szállítja a teljes egészében elkészült szoftvert, ami potenciális hibákat, vagy a vevői igényektől eltérő dolgokat tartalmazhat. A agilis esetben a végtermék megfelel minden vevői igénynek, és további változtatásra nincs szükség.

**[2] Agilis szoftverfejlesztés 12 alapelve:**

1. Legfontosabbnak azt tartjuk, hogy az ügyfél elégedettségét a működő szoftver mielőbbi és folyamatos szállításával vívjuk ki.
2. Elfogadjuk, hogy a követelmények változhatnak akár a fejlesztés vége felé is. Az agilis eljárások a változásból versenyelőnyt kovácsolnak az ügyfél számára.
3. Szállíts működő szoftvert gyakran, azaz néhány hetenként vagy havonként, lehetőség szerint a gyakoribb szállítást választva.
4. Az üzleti szakértők és a szoftverfejlesztők dolgozzanak együtt minden nap, a projekt teljes időtartamában.
5. Építsd a projektet sikerorientált egyénekre. Biztosítsd számukra a szükséges környezetet és támogatást, és bízz meg bennük, hogy elvégzik a munkát.
6. A leghatásosabb és leghatékonyabb módszer az információ átadásának a fejlesztési csapaton belül, a személyes beszélgetés.
7. A működő szoftver az elsődleges mércéje az előre haladásnak.
8. Az agilis eljárások a fenntartható fejlesztést pártolják. Fontos, hogy a szponzorok, a fejlesztők és a felhasználók folytonosan képesek legyenek tartani egy állandó ütemet.
9. A műszaki kiválóság és a jó terv folyamatos szem előtt tartása fokozza az agilitást.
10. Elengedhetetlen az egyszerűség, azaz az elvégezetlen munkamennyiség maximalizálásának művészete.
11. A legjobb architektúrák, követelmények és rendszertervek az önszerveződő csapatoktól származnak.
12. A csapat rendszeresen mérlegeli, hogy miképpen lehet emelni a hatékonyságot, és ehhez hangolja és igazítja az működését.

[1] Kiáltvány az agilis szoftverfejlesztésért - <https://agilemanifesto.org/iso/hu/manifesto.html>

|  |  |  |
| --- | --- | --- |
| Kent Beck Mike Beedle Arie van Bennekum Alistair Cockburn Ward Cunningham Martin Fowler | James Grenning Jim Highsmith Andrew Hunt Ron Jeffries Jon Kern Brian Marick | Robert C. Martin Steve Mellor Ken Schwaber Jeff Sutherland Dave Thomas |

[2] Agilis szoftverfejlesztés 12 alapelve - <https://agilemanifesto.org/iso/hu/principles.html>

[3]

[4]

[5]

[6]

[7]

[8]

[9]

[10]

[11]

[12]

[13]

[14]

[15]